Mini Project Report: Basic Network Sniffer in Python

**What is a Network Sniffer?**

A network sniffer is a tool used to capture, monitor, and analyze data packets travelling over a network. It helps in understanding how data moves from one host to another, the structure of various network packets (IP, TCP, UDP, ICMP), and what type of traffic is flowing in/out of a system.

**Network sniffers are essential for:**

- Network troubleshooting

- Security monitoring (SOC work)

- Learning how protocols operate

**Objective of the Project:**

To develop a simple yet functional network sniffer using Python and Scapy, running in a Linux environment (Ubuntu in VirtualBox), which:

- Captures live network packets

- Analyzes basic details (IP addresses and protocols)

- Filters only TCP and ICMP traffic in an upgraded version

**Tools & Environment:**

OS: Ubuntu (running in Oracle VirtualBox)

Language: Python 3.12

Library: Scapy

Environment: Python Virtual Environment (venv)

Privileges: Root (sudo) required for packet sniffing

**Step-by-Step Implementation:**

**Step 1: Install VirtualBox Guest Additions**

cd /media/$USER/VBox\_GAs\*

sudo ./VBoxLinuxAdditions.run

**Step 2: Create and activate a Python virtual environment**

python3 -m venv scapy-env

source scapy-env/bin/activate

**Step 3: Install Scapy inside the virtual environment**

pip install Scapy

**Step 4: Create the sniffer Python script sniffer.py**

Place this script in your home directory (/home/Ray/).

[**Note:** The Python script used in this project was adapted from publicly available educational resources and tutorials for learning purposes.]

**Version 1: All Traffic (No Filter)**

from scapy.all import sniff

def process\_packet(packet):

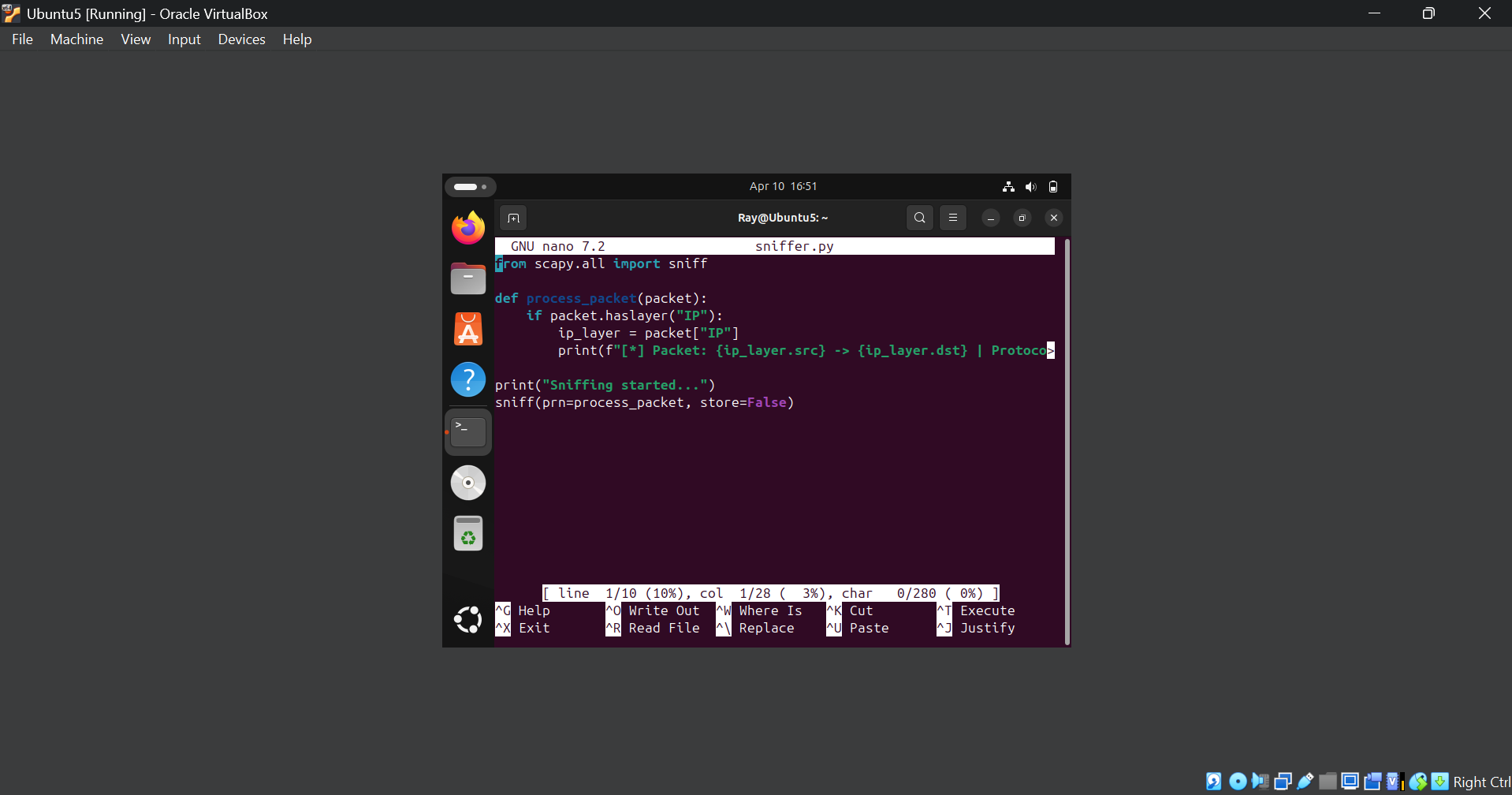
if packet.haslayer("IP"):

ip\_layer = packet["IP"]

print(f"[\*] Packet: {ip\_layer.src} -> {ip\_layer.dst} | Protocol: {ip\_layer.proto}")

print("Sniffing started...")

sniff(prn=process\_packet, store=False)



**Sample Output:**

[\*] Packet: 10.0.2.6 -> 224.0.0.251 | Protocol: 17

**Output Explanation:**

VM (10.0.2.6) sent a UDP multicast packet to 224.0.0.251 (used by mDNS).

Protocol: 17 means UDP (from the IP protocol number list).

This proves that the sniffer worked and detected live traffic on the network.

**Version 2: Filtered Sniffer (TCP & ICMP only)**

**[Note: The Python script used in this project was adapted from publicly available educational resources and tutorials for learning purposes.]**

from scapy.all import sniff

def process\_packet(packet):

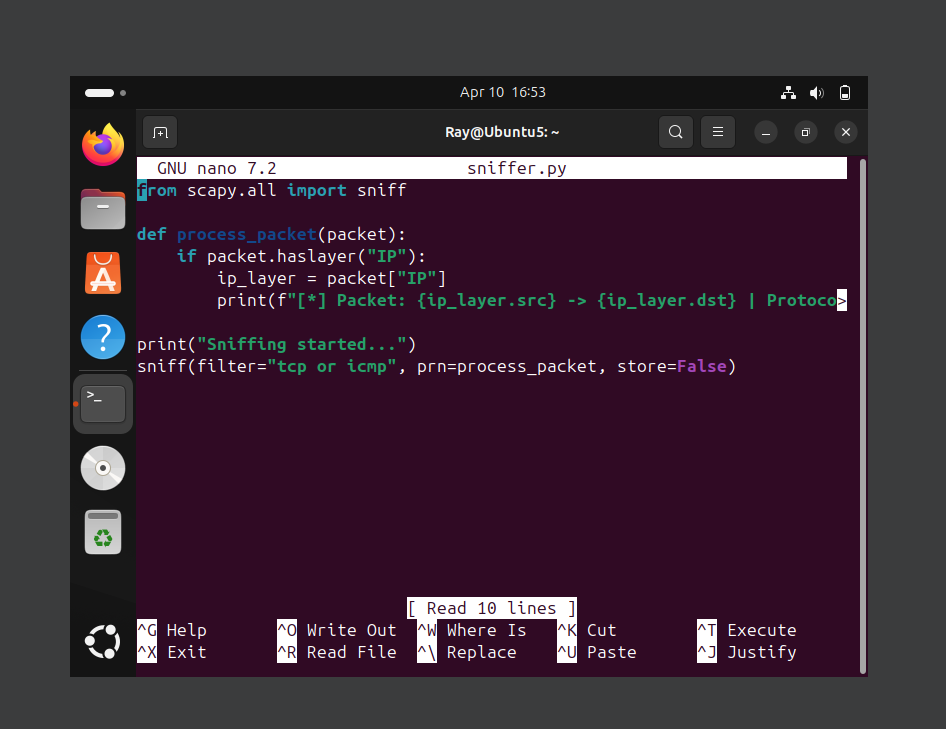
if packet. haslayer("IP"):

ip\_layer = packet["IP"]

print(f"[\*] Packet: {ip\_layer.src} -> {ip\_layer.dst} | Protocol: {ip\_layer. proto}")

print ("Sniffing started...")

sniff (filter="tcp or icmp", prn=process\_packet, store=False)



**Sample Output:**

[\*] Packet: 10.0.2.6 -> 91.189.91.97 | Protocol: 6

[\*] Packet: 91.189.91.97 -> 10.0.2.6 | Protocol: 6

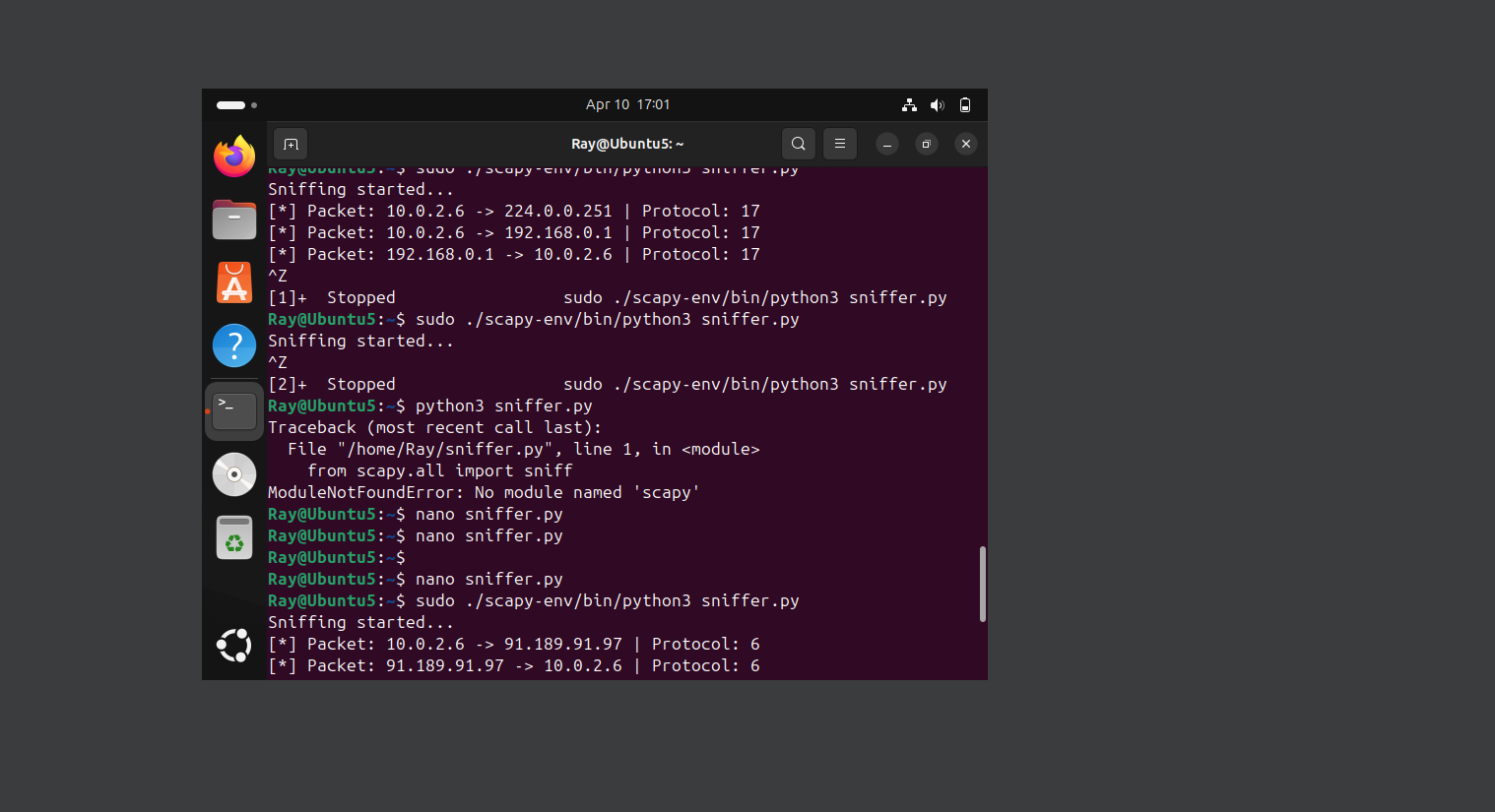
**Explanation:**

Traffic to/from Ubuntu's update server (91.189.91.97) was captured.

Protocol: 6 refers to TCP (e.g., HTTP or HTTPS).

Since ICMP packets (like ping) were not being sent during testing, none were shown.

The filter ensures that only relevant protocol types are captured.



**Learning Outcomes**

- Understood how to use Scapy for packet sniffing.

- Learned about IP protocols and how to identify them using protocol numbers.

- Experienced in setting up a Python environment in Linux.

- Gained confidence in capturing and analyzing live network traffic.

- Understood why root privileges are needed for raw socket access.

**Conclusion**

This project demonstrates how Python and Scapy can be used to build a simple but functional network sniffer. You successfully captured both general and filtered traffic, understood basic packet structures, and got hands-on experience with real network data inside a virtualized Linux environment.