Basic Network Sniffer

Requirements for Building a Network Sniffer in Python on Linux:

Software Requirements

1. Operating System:
- Linux (any distribution, e.g., Ubuntu, Debian, Fedora, etc.)

2. Python:
- Python 3.x (Ensure you have Python 3 installed)

3. Python Libraries:
- **Scapy:** A powerful library used for network packet manipulation and analysis.
- **pip:** Python package installer to install Scapy.

Hardware Requirements

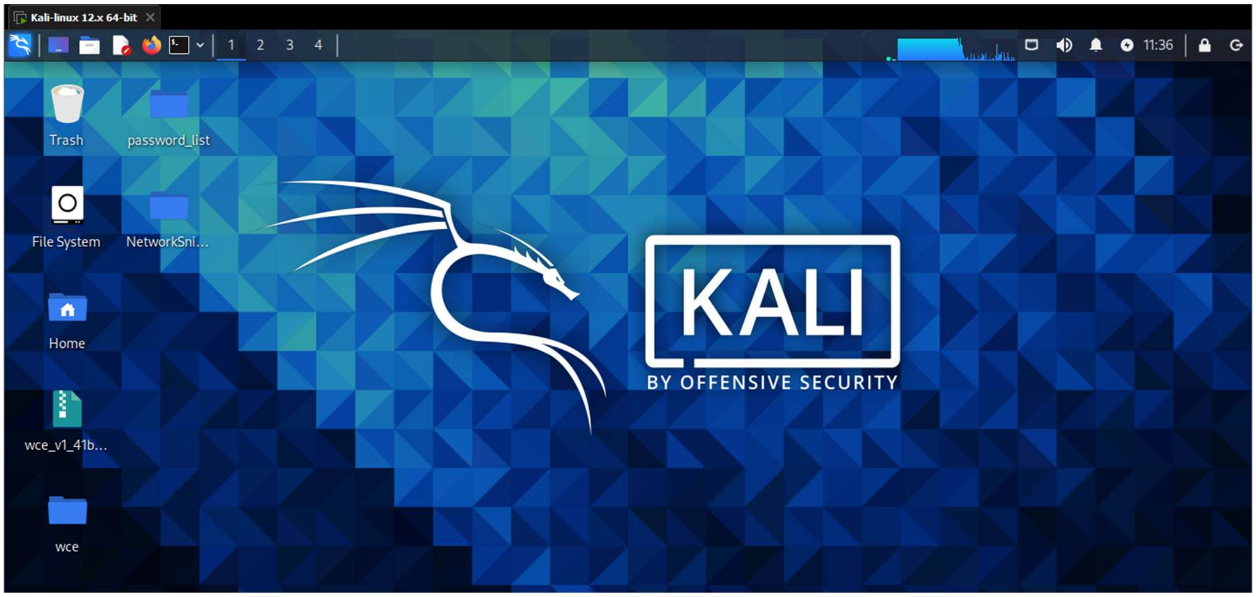
1. Network Interface:
- A network interface (wired or wireless) to capture network packets.

2. Sufficient Permissions:
- Root or administrative privileges are required to capture network packets.

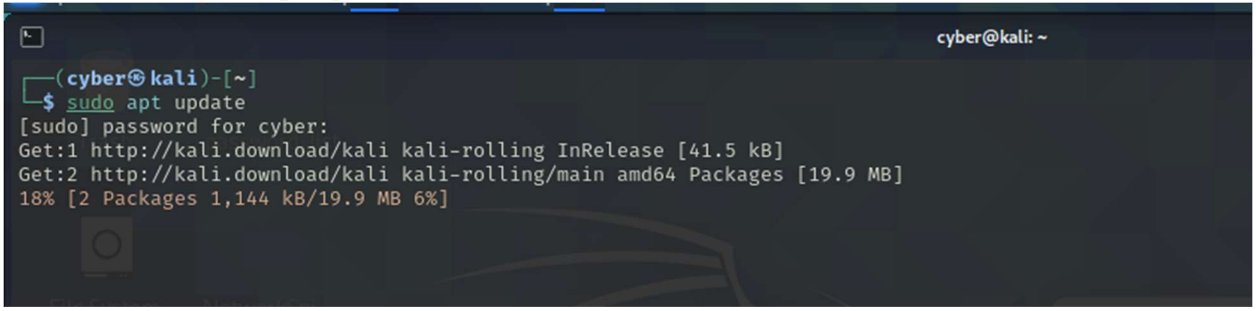
Step-by-Step Setup Guide

1. Install Python and pip

Most Linux distributions come with Python pre-installed. Ensure you have Python 3.x installed First Run kali linux and open terminal. Now type following commands:

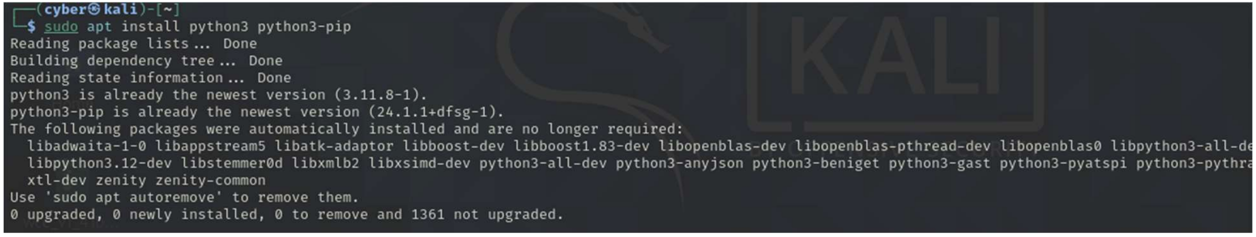


Command: sudo apt update



```
┌──(cyber㉿kali)-[~]
└─$ sudo apt update
[sudo] password for cyber:
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [19.9 MB]
18% [2 Packages 1,144 kB/19.9 MB 6%]
```

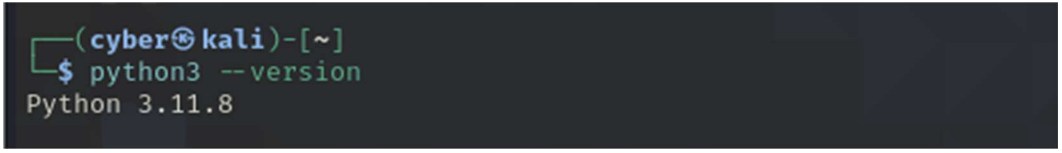
Command: sudo apt install python3 python3-pip



```
┌──(cyber㉿kali)-[~]
└─$ sudo apt install python3 python3-pip
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
python3 is already the newest version (3.11.8-1).
python3-pip is already the newest version (24.1.1+dfsg-1).
The following packages were automatically installed and are no longer required:
  libadwaita-1-0 libappstream5 libatk-adaptor libboost-dev libboost1.83-dev libopenblas-dev libopenblas-pthread-dev libopenblas0 libpython3-all-de
  libpython3.12-dev libstemmer0d libxmlb2 libxsimd-dev python3-all-dev python3-anyjson python3-beniget python3-gast python3-pyatspi python3-pythra
  xtl-dev zenity zenity-common
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 1361 not upgraded.
```

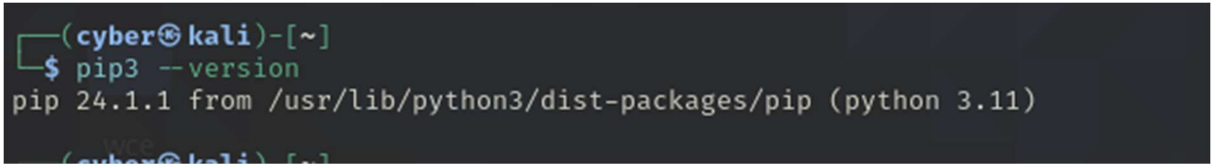
Verify the installation:

Command:	python3 –version



```
┌──(cyber㉿kali)-[~]
└─$ python3 --version
Python 3.11.8
```

Command:	pip3 –version



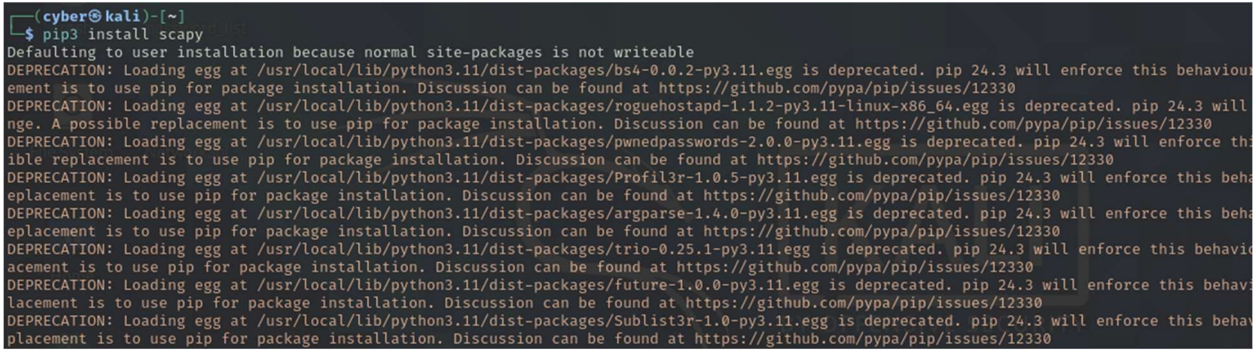
```
┌──(cyber㉿kali)-[~]
└─$ pip3 --version
pip 24.1.1 from /usr/lib/python3/dist-packages/pip (python 3.11)

┌──(cyber㉿kali)-[~]
```

2. Install Scapy:

Install Scapy using following pip command:

Command:	pip3 install scapy



```
┌──(cyber㉿kali)-[~]
└─$ pip3 install scapy
Defaulting to user installation because normal site-packages is not writeable
DEPRECATION: Loading egg at /usr/local/lib/python3.11/dist-packages/bs4-0.0.2-py3.11.egg is deprecated. pip 24.3 will enforce this behaviour
ement is to use pip for package installation. Discussion can be found at https://github.com/pypa/pip/issues/12330
DEPRECATION: Loading egg at /usr/local/lib/python3.11/dist-packages/roguehostapd-1.1.2-py3.11-linux-x86_64.egg is deprecated. pip 24.3 will
nge. A possible replacement is to use pip for package installation. Discussion can be found at https://github.com/pypa/pip/issues/12330
DEPRECATION: Loading egg at /usr/local/lib/python3.11/dist-packages/pwnedpasswords-2.0.0-py3.11.egg is deprecated. pip 24.3 will enforce thi
ible replacement is to use pip for package installation. Discussion can be found at https://github.com/pypa/pip/issues/12330
DEPRECATION: Loading egg at /usr/local/lib/python3.11/dist-packages/Profil3r-1.0.5-py3.11.egg is deprecated. pip 24.3 will enforce this beha
eplacement is to use pip for package installation. Discussion can be found at https://github.com/pypa/pip/issues/12330
DEPRECATION: Loading egg at /usr/local/lib/python3.11/dist-packages/argparse-1.4.0-py3.11.egg is deprecated. pip 24.3 will enforce this beha
eplacement is to use pip for package installation. Discussion can be found at https://github.com/pypa/pip/issues/12330
DEPRECATION: Loading egg at /usr/local/lib/python3.11/dist-packages/trio-0.25.1-py3.11.egg is deprecated. pip 24.3 will enforce this behavic
acement is to use pip for package installation. Discussion can be found at https://github.com/pypa/pip/issues/12330
DEPRECATION: Loading egg at /usr/local/lib/python3.11/dist-packages/future-1.0.0-py3.11.egg is deprecated. pip 24.3 will enforce this behavi
lacement is to use pip for package installation. Discussion can be found at https://github.com/pypa/pip/issues/12330
DEPRECATION: Loading egg at /usr/local/lib/python3.11/dist-packages/Sublist3r-1.0-py3.11.egg is deprecated. pip 24.3 will enforce this behav
placement is to use pip for package installation. Discussion can be found at https://github.com/pypa/pip/issues/12330
```

3. Create the Network Sniffer Script

Create a file named `network_sniffer.py` and add the following code

This Script with Logging and Filtering
Here's the complete script with filtering and logging:

Script:

```
from scapy.all import sniff, Raw
from scapy.layers.inet import IP, TCP, UDP, ICMP

def packet_callback(packet):
    with open("packet_log.txt", "a") as f:
        if IP in packet:
            ip_src = packet[IP].src
            ip_dst = packet[IP].dst
            protocol = packet[IP].proto
            f.write(f"IP Packet: {ip_src} -> {ip_dst} (Protocol: {protocol})\n")
            if TCP in packet:
                tcp_sport = packet[TCP].sport
                tcp_dport = packet[TCP].dport
                f.write(f"TCP Packet: {ip_src}:{tcp_sport} -> {ip_dst}:{tcp_dport}\n")
            elif UDP in packet:
                udp_sport = packet[UDP].sport
                udp_dport = packet[UDP].dport
                f.write(f"UDP Packet: {ip_src}:{udp_sport} -> {ip_dst}:{udp_dport}\n")
            elif ICMP in packet:
                icmp_type = packet[ICMP].type
                f.write(f"ICMP Packet: {ip_src} -> {ip_dst} (Type: {icmp_type})\n")
            if Raw in packet:
                raw_data = packet[Raw].load
                f.write(f"Raw Data: {raw_data}\n")

if __name__ == "__main__":
    print("Starting the packet sniffer...")
    # Filter for only TCP packets and start the sniffer
    sniff(prn=packet_callback, store=0, filter="tcp")
```

Here are the full detail about script:

Import Necessary Libraries:

Python

```
from scapy.all import sniff, Raw
from scapy.layers.inet import IP, TCP, UDP, ICMP
```

-This line imports the required libraries:

- scapy.all: Provides functions for packet manipulation, sniffing, and sending.
- scapy.layers.inet: Defines network layer protocols like IP, TCP, UDP, and ICMP.

Packet Callback Function:

Python

```
def packet_callback(packet):
    # ...
```

-This function is called for each captured packet. It processes the packet and writes information to a file.

File Handling:

Python

```
with open("packet_log.txt", "a") as f:
    # ...
```

-This line opens a file named "packet_log.txt" in append mode ("a"). Any data written to this file will be added to the end of the existing content.

Packet Analysis:

Python

```
if IP in packet:
    # ...
```

-Checks if the captured packet is an IP packet. If it is, it extracts IP-related information.

Python

```
ip_src = packet[IP].src
ip_dst = packet[IP].dst
protocol = packet[IP].proto
f.write(f"IP Packet: {ip_src} -> {ip_dst} (Protocol: {protocol})\n")
```

-Extracts the source and destination IP addresses and the protocol type from the IP header and writes them to the file.

Protocol-Specific Information:

Python
```
if TCP in packet:
    # ...
elif UDP in packet:
    # ...
elif ICMP in packet:
    # ...
```

Checks the protocol type and extracts additional information based on the protocol:

- **TCP:** Extracts source and destination port numbers.
- **UDP:** Extracts source and destination port numbers.
- **ICMP:** Extracts the ICMP type.

The extracted information is written to the file.

Raw Data Extraction:

Python
```
if Raw in packet:
    raw_data = packet[Raw].load
    f.write(f"Raw Data: {raw_data}\n")
```

Checks if there's raw data in the packet and writes it to the file.

Main Execution:

Python
```
if __name__ == "__main__":
    print("Starting the packet sniffer...")
    sniff(prn=packet_callback, store=0, filter="tcp")
```

This block starts the packet sniffing process:

- Prints a message indicating the start of the sniffer.
- Calls the `sniff` function to capture packets.
    - `prn=packet_callback`: Specifies the `packet_callback` function to be called for each captured packet.

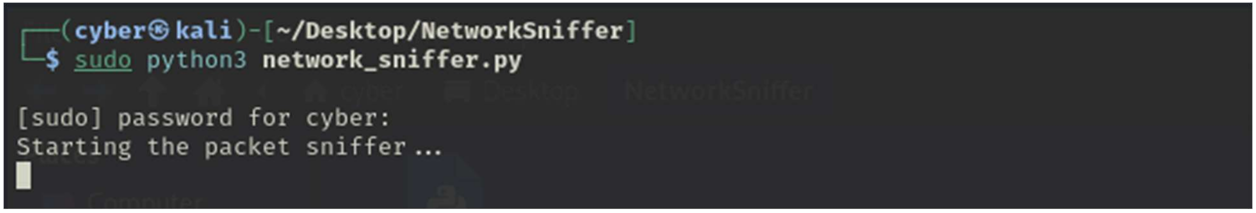
- `store=0`: Prevents Scapy from storing captured packets in memory.
- `filter="tcp"`: Filters for only TCP packets.

Code Functionality:

This code captures network packets, extracts information about IP, TCP, UDP, and ICMP packets, and writes the extracted data to a text file. It focuses on TCP packets due to the filter applied.

-Let's Try this python Script:

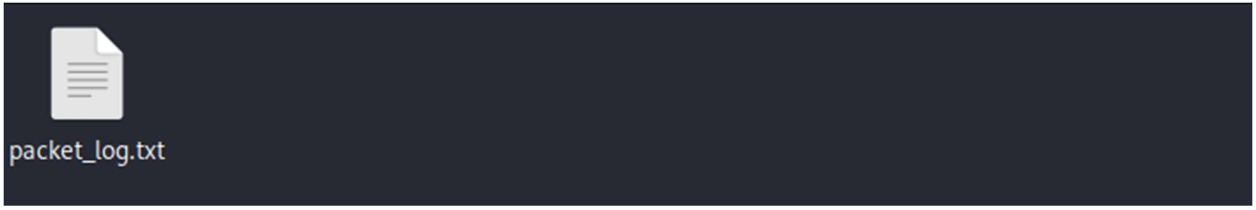
Command: sudo python3 network_sniffer.py



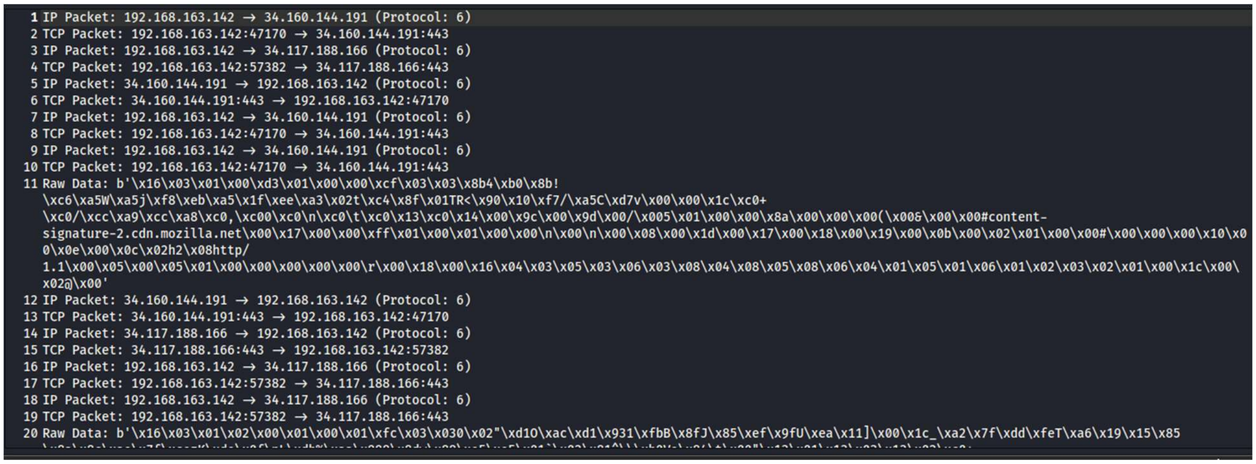
```
┌──(cyber㉿kali)-[~/Desktop/NetworkSniffer]
└─$ sudo python3 network_sniffer.py

[sudo] password for cyber:
Starting the packet sniffer...
```

-It will sniff the packets and create a separate file.



-Let's check this file



```
IP Packet: 192.168.163.142 → 34.160.144.191 (Protocol: 6)
TCP Packet: 192.168.163.142:47170 → 34.160.144.191:443
IP Packet: 192.168.163.142 → 34.117.188.166 (Protocol: 6)
TCP Packet: 192.168.163.142:57382 → 34.117.188.166:443
IP Packet: 34.160.144.191 → 192.168.163.142 (Protocol: 6)
TCP Packet: 34.160.144.191:443 → 192.168.163.142:47170
IP Packet: 192.168.163.142 → 34.160.144.191 (Protocol: 6)
TCP Packet: 192.168.163.142:47170 → 34.160.144.191:443
IP Packet: 192.168.163.142 → 34.160.144.191 (Protocol: 6)
TCP Packet: 192.168.163.142:47170 → 34.160.144.191:443
Raw Data: b'\x16\x03\x01\x00\xd3\x01\x00\x00\xcf\x03\x03\x8b4\xb0\x8b!
\xc6\xa5W\xa5j\xf8\xeb\xa5\x1f\xee\xa3\x02t\xc4\x8f\x01TR<\x90\x10\xf7/\xa5C\xd7v\x00\x00\x1c\xc0+
\xc0/\xcc\xa9\xcc\xa8\xc0,\xc00\xc0\n\xc0\t\xc0\x13\xc0\x14\x00\x9c\x00\x9d\x00/\x005\x01\x00\x00\x8a\x00\x00\x00(\x00&\x00\x00#content-
signature-2.cdn.mozilla.net\x00\x17\x00\x00\xff\x01\x00\x01\x00\x00\n\x00\n\x00\x08\x00\x1d\x00\x17\x00\x18\x00\x19\x00\x0b\x00\x02\x01\x00\x00#\x00\x00\x00\x10\x0
0\x0e\x00\x0c\x02h2\x08http/
1.1\x00\x05\x00\x05\x01\x00\x00\x00\x00\x00\r\x00\x18\x00\x16\x04\x03\x05\x03\x06\x03\x08\x04\x08\x05\x08\x06\x04\x01\x05\x01\x06\x01\x02\x03\x02\x01\x00\x1c\x00\
x02@\x00'
IP Packet: 34.160.144.191 → 192.168.163.142 (Protocol: 6)
TCP Packet: 34.160.144.191:443 → 192.168.163.142:47170
IP Packet: 34.117.188.166 → 192.168.163.142 (Protocol: 6)
TCP Packet: 34.117.188.166:443 → 192.168.163.142:57382
IP Packet: 192.168.163.142 → 34.117.188.166 (Protocol: 6)
TCP Packet: 192.168.163.142:57382 → 34.117.188.166:443
IP Packet: 192.168.163.142 → 34.117.188.166 (Protocol: 6)
TCP Packet: 192.168.163.142:57382 → 34.117.188.166:443
Raw Data: b'\x16\x03\x01\x02\x00\x01\x00\x01\xfc\x03\x030\x02"\xd10\xac\xd1\x931\xfbB\x8fJ\x85\xef\x9fU\xea\x11]\x00\x1c_\xa2\x7f\xdd\xfeT\xa6\x19\x15\x85
```

Our Script is running perfectly and capture packets.