

Wireshark

Wireshark is a packet capture tool, enabling you to see exactly what is going on at the most granular level of your network. By capturing the packets, it provides detailed insight into the communications happening between devices, including which protocols are used and even contents of each packet. Wireshark used to be known as Ethernet but was renamed in 2006 and remains one of the most standard tools for networking analysis across industries.

Key Features of Wireshark

1. **Live Capture and Offline Analysis:** Wireshark captures live network data or reads saved capture files.
2. **Packet Details:** Each captured packet is broken down into its layer details, from the physical to the application layer.
3. **Big Set of Protocol Support:** Wireshark offers dissection and display of over 2,000 protocols including TCP/IP, HTTP, FTP, DNS etc.
4. **Filters and Colouring Rules:** It has robust filtering and colouring features that make it easier to sort and prioritize data.
5. **Cross-Platform Compatibility:** Wireshark is available for Windows, macOS, and Linux.

Understanding Wireshark's Core Components

1. **Packet Capture Library (PCAP):** It depends on the PCAP library, either libpcap on Unix/Linux or WinPcap/Npcap on Windows,

that capture packets. PCAP is the middle layer that interfaces with the system's network adapter to access the packet data.

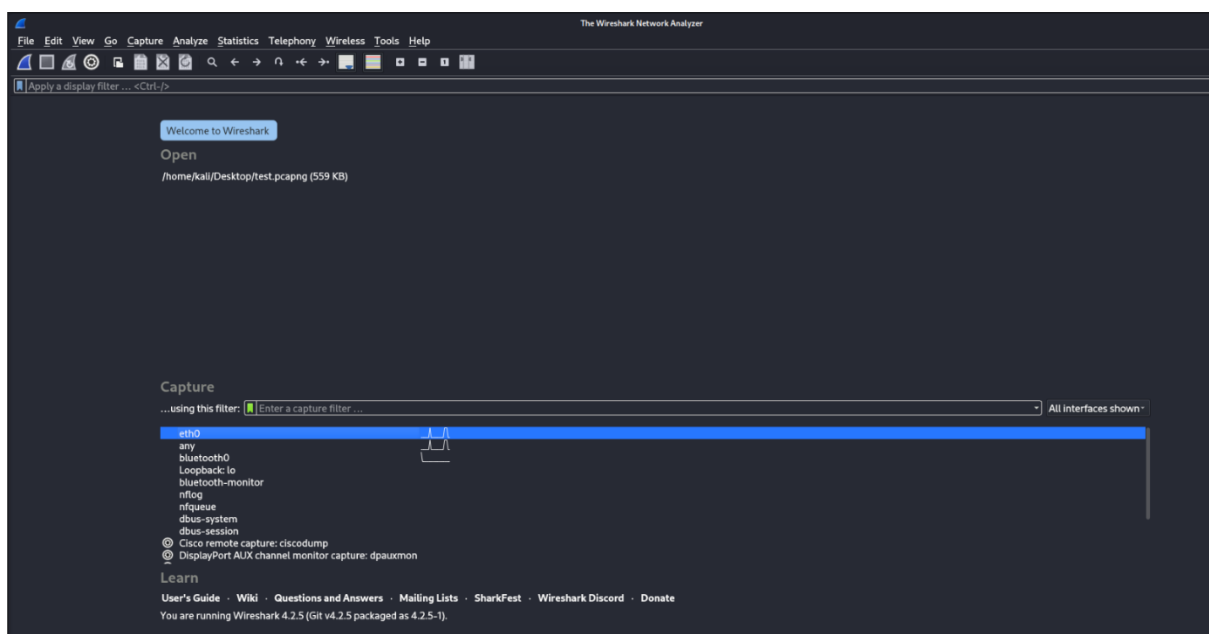
2. **Protocols and Protocol Dissection:** The Wireshark tool can dissect most protocols and structure of each packet, thus revealing what information exists for each layer of protocol.
3. **Filters:** Wireshark uses two types of filters:
 - **Capture Filters:** Defines what data will be captured in a session.
 - **Display Filters:** Narrow down captured data to show only relevant packets for analysis.

How Wireshark Works: Packet Capture Process

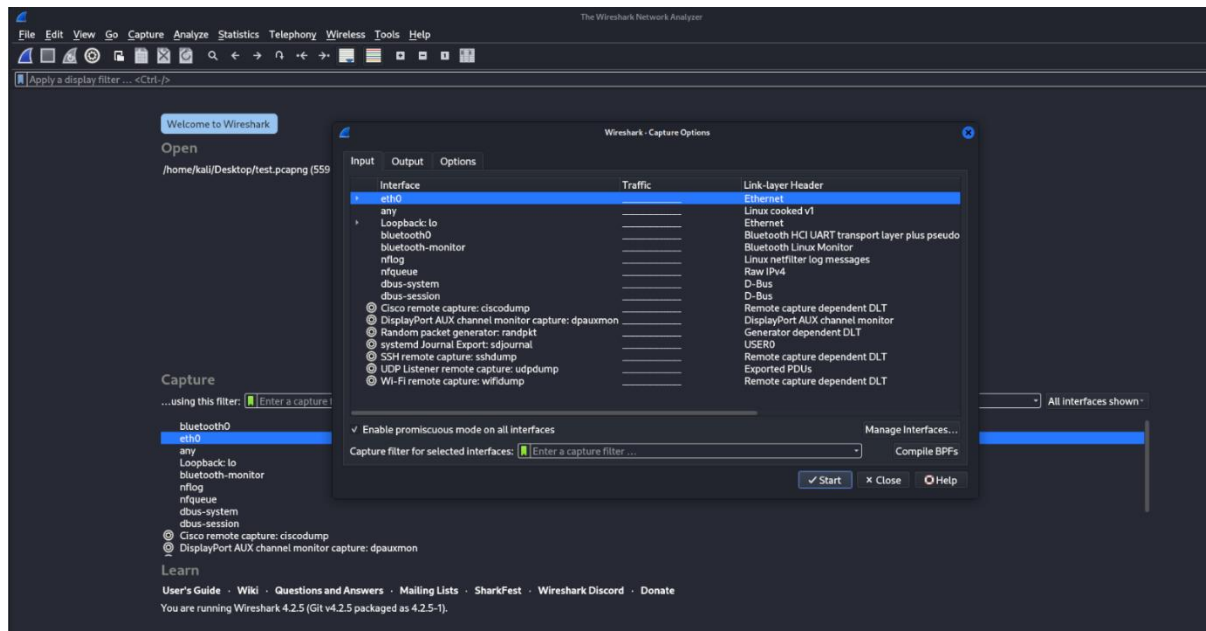
The process of packet capture and analysis in Wireshark involves multiple stages:

1. Packet Capture Initialization

To begin capturing packets, the user selects an active network interface.

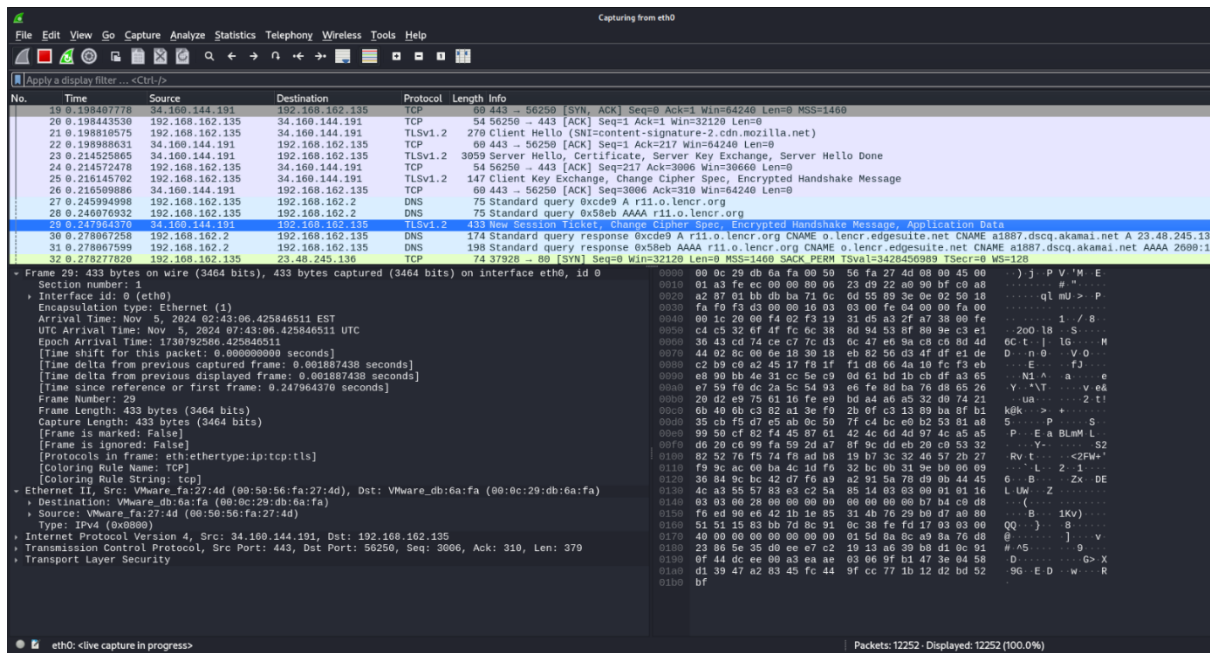


Once an interface is selected, Wireshark taps into the network traffic by putting the chosen interface into **promiscuous mode**; thus, all packets on the given network segment, not only those destined for the user's device.



2. Data Capture with PCAP

Once packet capture is initiated, Wireshark uses the PCAP library to intercept all network traffic on the chosen interface. PCAP works at the Data Link layer, capturing raw frames of data from the network. These frames contain payload data along with headers from the different layers of the OSI model.



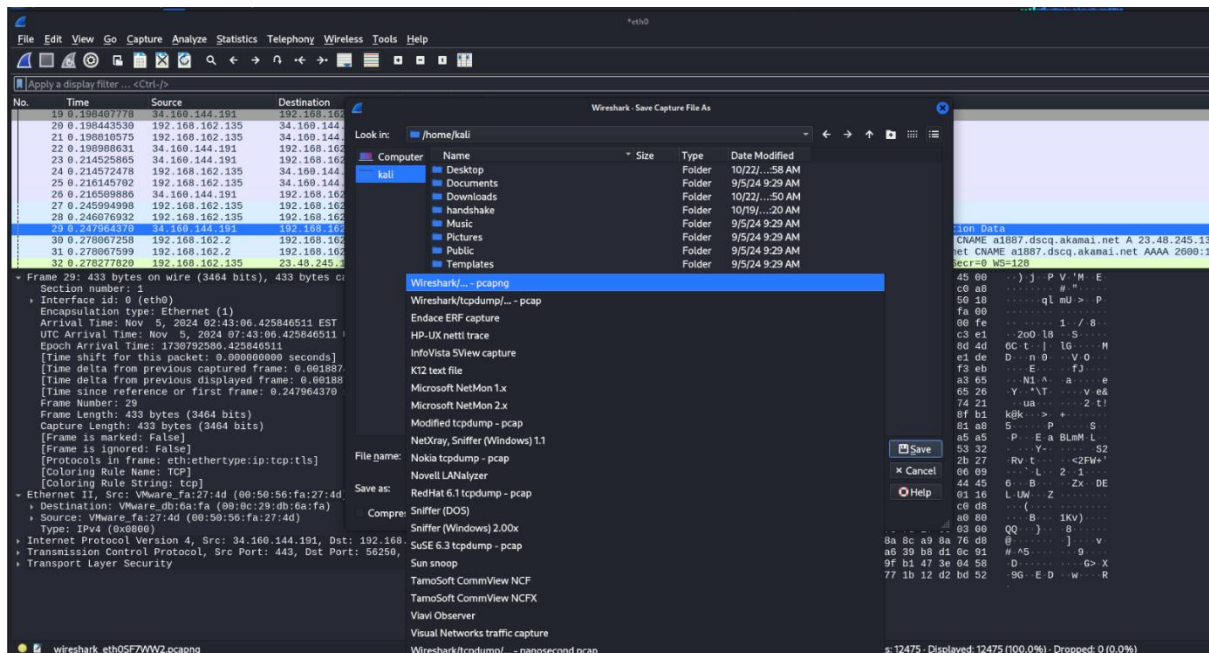
3. Packet Dissection

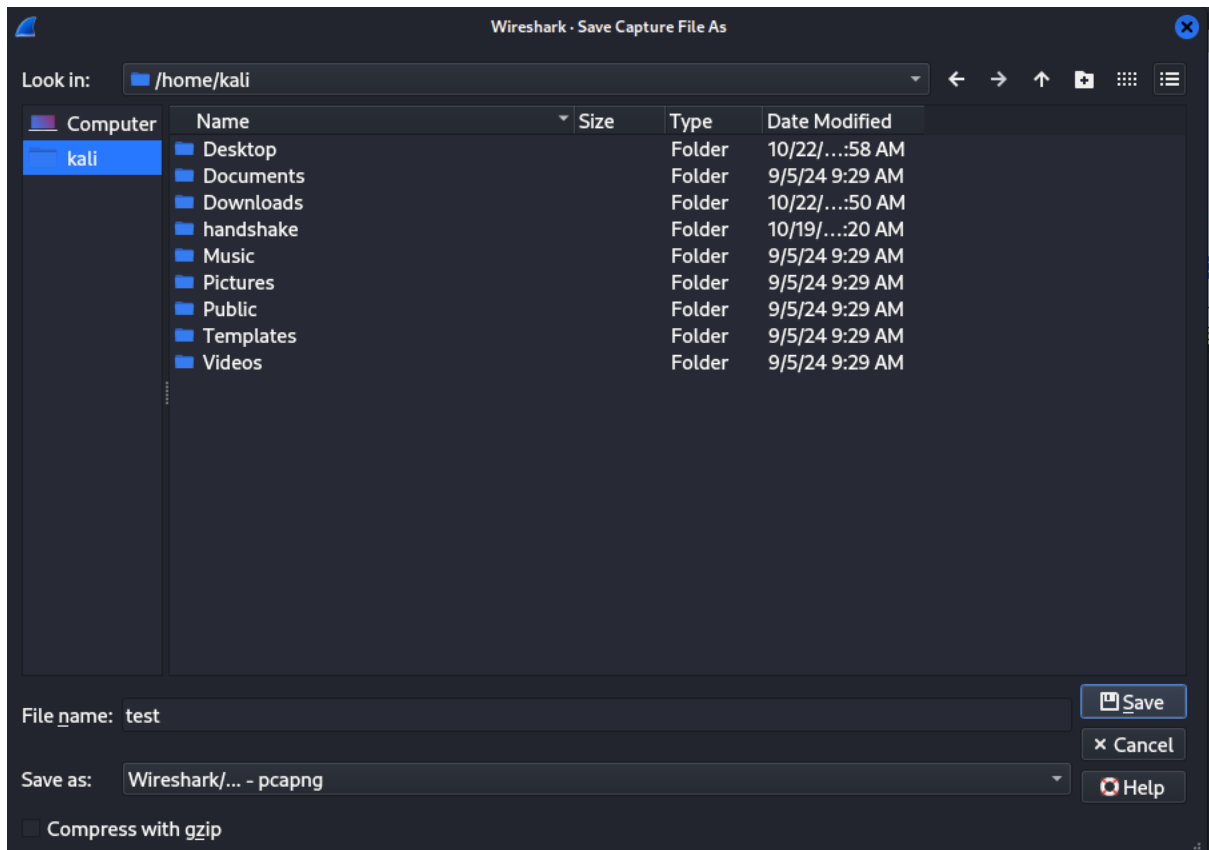
Wireshark's core strength lies in its ability to dissect captured frames. This dissection happens through several steps:

- **Frame:** General information is mentioned regarding the captured packet.
- **Data Link Layer:** It captures the Ethernet frame including MAC addresses, type, and checksum.
- **Network Layer:** Protocols like IP and ARP are analysed to show the source and destination IP addresses, TTL, and routing information.
- **Transport Layer:** Wireshark decodes protocols such as TCP, UDP, and ICMP by showing port numbers, sequence numbers, flags, and checksums.
- **Application Layer:** For protocols such as HTTP, FTP, and DNS, it displays payload content including the GET requests, status codes, DNS query, and responses.

4. Data Storage

The captured data can be saved in various formats, including the .pcap file format, and therefore allowing users to analyse data even offline or share captured data with others for facilitate collaborative analysis.

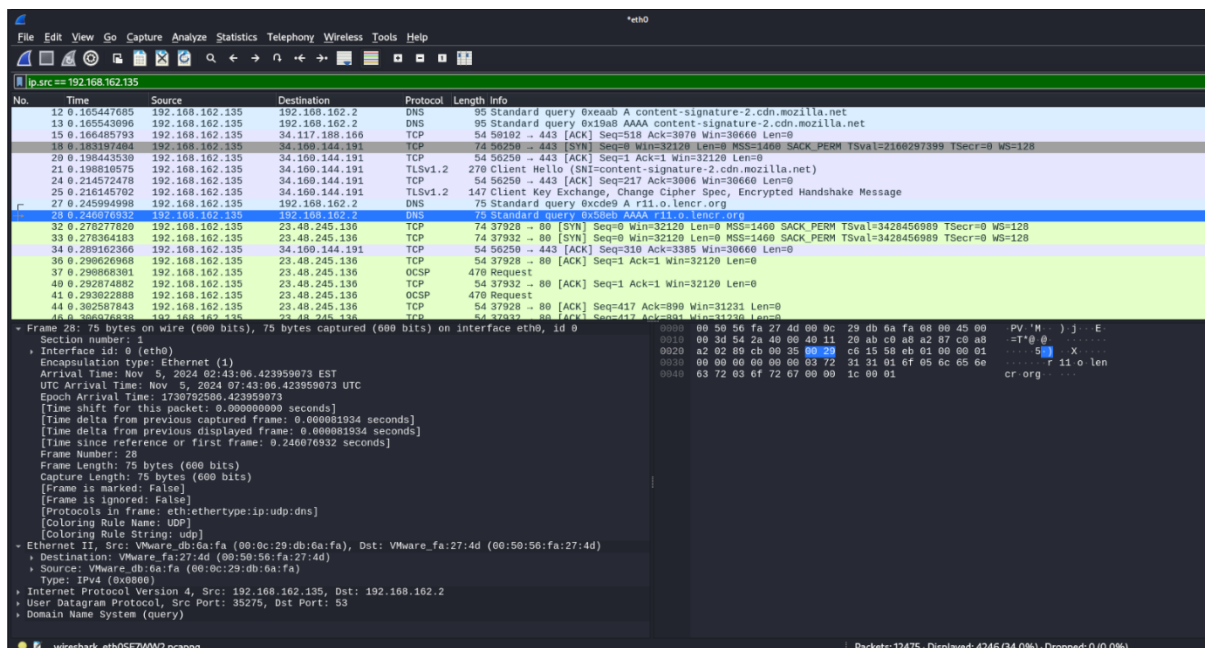




5. Filtering and Analysis

Wireshark provides extensive filtering capabilities to aid in analysis:

- **Display Filters:** These filters help isolate specific packets based on criteria, such as protocol, IP address, or port. For example, `ip.src == 192.168.162.135` will only display packets originating from a particular IP address.



- **Colour Coding:** Different packet types and protocols are colour-coded. Thus, users can easily identify key traffic flows and patterns.

Practical Applications of Wireshark

1. **Network Troubleshooting:** Wireshark proves extremely helpful in diagnosing slow network performance, packet loss, and connectivity issues.
2. **Protocol Analysis:** It explains how protocols work by providing a transparent view of what data each layer is processing.
3. **Security Analysis:** Wireshark will highlight unauthorized access attempts, malware communication, and packet-based attacks.
4. **Compliance and Auditing:** For highly regulated industries, Wireshark will help in auditing protocol usage and network compliance.

5. **Education:** Wireshark is widely used in educational courses where one learns network protocols, packet structures, and trouble shooting.

Security Implications and Considerations

While Wireshark is powerful, it also poses potential security risks if misused. Network administrators and security teams should be mindful of:

- **Access Controls:** Limit Wireshark use to authorized personnel as it can expose sensitive data in plain text.
- **Legal Compliance:** Capturing and analysing network traffic should comply with privacy laws and organizational policies.
- **Promiscuous Mode Caution:** Enabling promiscuous mode can increase visibility but also the risk of sensitive data exposure.

Wireshark is undoubtedly one of the most important tools for in-depth network analysis. It provides full visibility into network traffic at all OSI layers. Due to its ability to dissect packets, apply complex filters, and support a number of protocols, Wireshark has proven invaluable for troubleshooting, security analysis, and many different teaching purposes. Mastering Wireshark helps users detect anomalies and diagnose issues in the network in order to guarantee its reliability and security, making it a must-have in each cybersecurity professional's toolkit.

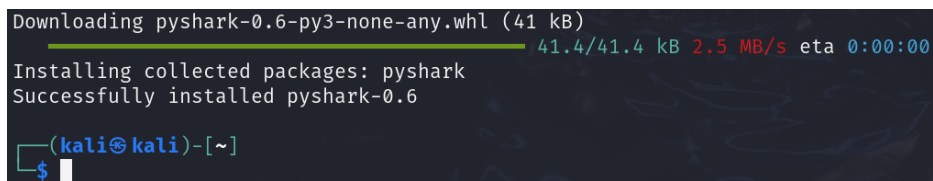
Script for Capturing Network Traffic

Capturing network traffic is important for troubleshooting, analysis, and software and communications protocol development. For the security-minded individual, monitoring network traffic is crucial for detecting malicious activity or policy violation.

PyShark is a Python wrapper for the popular network protocol analyzer **Wireshark**. It allows users to capture, read, and analyze network packets directly in Python, making it useful for network forensics, security analysis, and automation. Unlike traditional packet capture tools, PyShark leverages **tshark** (Wireshark's command-line tool) to process packets in real time or from a saved **PCAP** file, enabling users to extract detailed network insights programmatically.

As a prerequisite, we need to install the Python package – Pyshark. For this enter the command: -

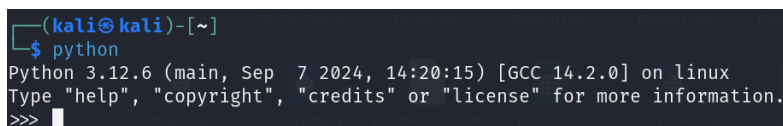
\$ pip install pyshark



```
Downloading pyshark-0.6-py3-none-any.whl (41 kB)
41.4/41.4 kB 2.5 MB/s eta 0:00:00
Installing collected packages: pyshark
Successfully installed pyshark-0.6
(kali@kali)-[~]
$
```

We will run some parts of the script in the Python interpreter. To open it use the command: -

\$ python



```
(kali@kali)-[~]
$ python
Python 3.12.6 (main, Sep 7 2024, 14:20:15) [GCC 14.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The interpreters prompt “>>>” is displayed. Here we will first import the pyshark library: -

```
>>> import pyshark
>>> 
```

```
>>> capture =
pyshark.LiveCapture(interface='eth0')
>>> for packet in
capture.sniff_continuously(packet_count=10):
>>>     print(packet)
```

```
>>> capture = pyshark.LiveCapture(interface='eth0')
>>> for packet in capture.sniff_continuously(packet_count=10):
...     print(packet)
... 
```

We get the output as shown below: -

[illegible]

We have made this code a bit more interactive. The program will keep capturing network packets and send them to a file in the kali/home directory. The code can be copied from here:

[https://github.com/avnishnaithani/pythonforcybersecurity/blob/main/Domain%206/Capture Network Traffic.py](https://github.com/avnishnaithani/pythonforcybersecurity/blob/main/Domain%206/Capture%20Network%20Traffic.py)

Copy this code into the nano editor. Open it using the command: -

```
$ nano Capture_Network_Traffic.py
```

A terminal window with a dark background. The prompt is (kali@kali)-[~]. The command nano Capture_Network_Traffic1.py has been entered and the cursor is at the end of the line.

```
(kali@kali)-[~]  
$ nano Capture_Network_Traffic1.py
```

After copying the code here save it using “CTRL+x” followed by “y” then finally press “Enter”.

Change the file to an executable one using: -

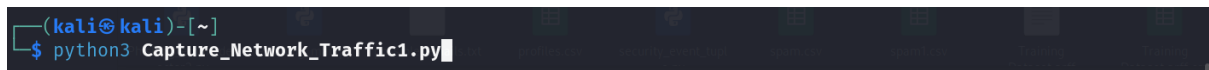
```
$ chmod +x Capture_Network_Traffic1.py
```

A terminal window with a dark background. The prompt is (kali@kali)-[~]. The command chmod +x Capture_Network_Traffic1.py has been entered and the cursor is at the end of the line.

```
(kali@kali)-[~]  
$ chmod +x Capture_Network_Traffic1.py
```

Finally run the program using: -

```
$ Python3 Capture_Network_Traffic1.py
```

A terminal window with a dark background. The prompt is (kali@kali)-[~]. The command python3 Capture_Network_Traffic1.py has been entered and the cursor is at the end of the line.

```
(kali@kali)-[~]  
$ python3 Capture_Network_Traffic1.py
```

Network anomaly detectors also work on this captured data. After feature extraction is done based on this captured network traffic data, we can train and test a Machine Learning Algorithm, analyze the results and perform further training to improve the results of the model. This way we can train the model to identify “Normal” network traffic. If any anomalous traffic is encountered, the machine

learning model can flag it and raise alerts for security analysts to analyze.