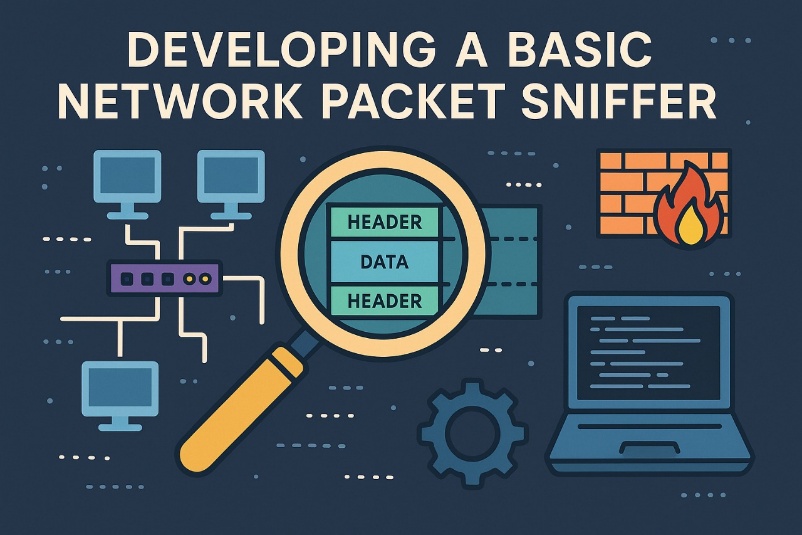
***Developing a Basic Network Packet Sniffer***



To the analysis and troubleshooting of network traffic, network packet sniffers are vital tools. They give you important information about how devices communicate with one another by enabling you to record and examine packets as they pass through a network interface. We'll look at how to use Python and the Scapy module to create a basic network packet sniffer in this tutorial.

**Contents**

[***Developing a Basic Network Packet Sniffer*** 1](#_Toc196142764)

[**1. A network packet sniffer: what is it?** 1](#_Toc196142765)

[**2. Getting Started with Python and Scapy** 2](#_Toc196142766)

[**3. Research and Plan** 2](#_Toc196142767)

[**4. The Packet Sniffer Development** 3](#_Toc196142768)

[**5. Advance option** 6](#_Toc196142769)

[**6. Code break down** 12](#_Toc196142770)

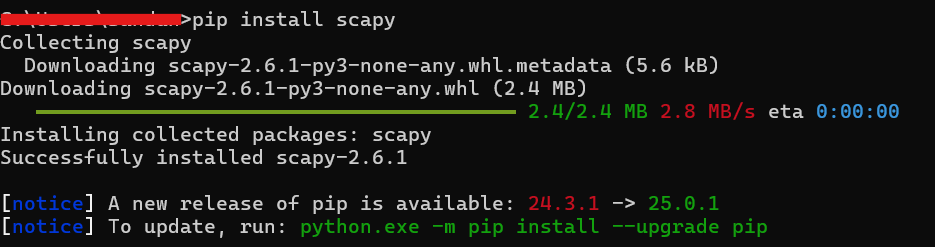
[**7. Conclusion** 17](#_Toc196142771)

## **1. A network packet sniffer: what is it?**

A software program that records and intercepts network data moving via a particular network interface is called a network packet sniffer. It can extract information like sources and destination IP addresses, ports, and packet payloads by analyzing different protocol headers. It collects packets at the data link layer. Network administrators, security experts, and developers frequently employ packet sniffers for activities including security analysis, performance monitoring, and network troubleshooting.

## **2. Getting Started with Python and Scapy**

Scapy is a powerful Python library for packet manipulation and network analysis. It provides a high-level interface for crafting and dissecting network packets, making it an ideal choice for building network utilities such as packet sniffers. Before we begin, ensure that you havve Python and Scapy installed on your system. You can install Scapy using pip:



## **3. Research and Plan**

**Learn Packet Sniffing**: Packets are units of data transmitted over a network, containing headers (metadata) and payloads (data). A sniffer captures these packets for analysis.

**Scapy Basics**: Scapy allows you to capture, dissect, and display packet details. Key functions include sniff() for capturing packets and packet. Summary () for displaying them.

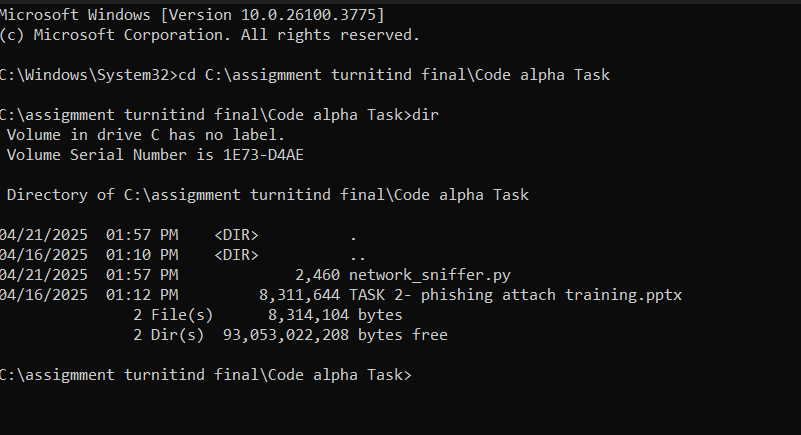
**Key Features for the Sniffer:**

* Capture packets on a specified network interface (e.g., eth0 on Linux, Wi-Fi on Windows).
* Display basic packet details: source IP, destination IP, protocol, and port numbers.
* Filter specific traffic (e.g., HTTP, TCP).
* Save captured packets for later analysis.

## **4. The Packet Sniffer Development**

The Python script that follows builds a simple network sniffer using Scapy. It records packets, checks for TCP traffic, and shows information about protocols, ports, and IP addresses. For clarity, the script has comments and error handling.

|  |
| --- |
| import scapy.all as scapy  from scapy.layers.inet import IP, TCP, UDP, ICMP  import sys  def get\_interface():  """Prompt user to select a network interface by index."""  try:  interfaces = scapy.get\_if\_list()  print("Available interfaces:")  for idx, iface in enumerate(interfaces):  print(f"{idx}: {iface}")  choice = int(input("Select the interface number: "))  return interfaces[choice]  except Exception as e:  print(f"Error retrieving interfaces: {e}")  sys.exit(1)  def process\_packet(packet):  """Analyze and display packet details."""  try:  if packet.haslayer(IP):  src\_ip = packet[IP].src  dst\_ip = packet[IP].dst  protocol = packet[IP].proto  proto\_name = "Unknown"  if protocol == 6 and packet.haslayer(TCP):  proto\_name = "TCP"  src\_port = packet[TCP].sport  dst\_port = packet[TCP].dport  elif protocol == 17 and packet.haslayer(UDP):  proto\_name = "UDP"  src\_port = packet[UDP].sport  dst\_port = packet[UDP].dport  elif protocol == 1 and packet.haslayer(ICMP):  proto\_name = "ICMP"  src\_port = dst\_port = "N/A"  else:  src\_port = dst\_port = "Unknown"  print(f"Source: {src\_ip}:{src\_port} -> Destination: {dst\_ip}:{dst\_port} | Protocol: {proto\_name}")  if packet.haslayer(TCP) and packet[TCP].dport == 80 and packet.haslayer(scapy.Raw):  print(f"HTTP Payload: {packet[scapy.Raw].load[:50]}...")  except Exception as e:  print(f"Error processing packet: {e}")  def start\_sniffer(interface, packet\_count=10):  """Start sniffing packets on the specified interface."""  try:  print(f"Starting sniffer on {interface}... Press Ctrl+C to stop.")  scapy.sniff(iface=interface, prn=process\_packet, filter="ip", count=packet\_count)  except KeyboardInterrupt:  print("\nSniffer stopped by user.")  except Exception as e:  print(f"Error during sniffing: {e}")  sys.exit(1)  def main():  """Main function to run the sniffer."""  print("=== Basic Network Sniffer ===")  interface = get\_interface()  try:  packet\_count = int(input("Enter number of packets to capture (default 10): ") or 10)  except ValueError:  packet\_count = 10  start\_sniffer(interface, packet\_count)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



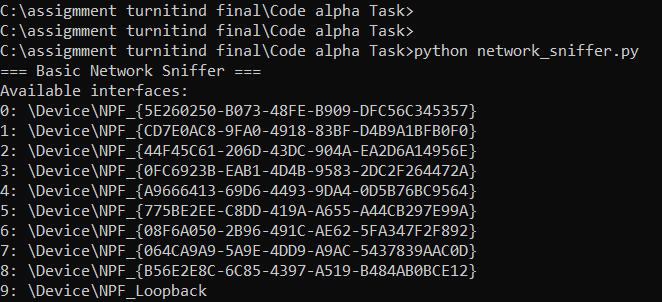
Save the file and navigate the file location through terminal.

**Run the Script**

Execute the below command

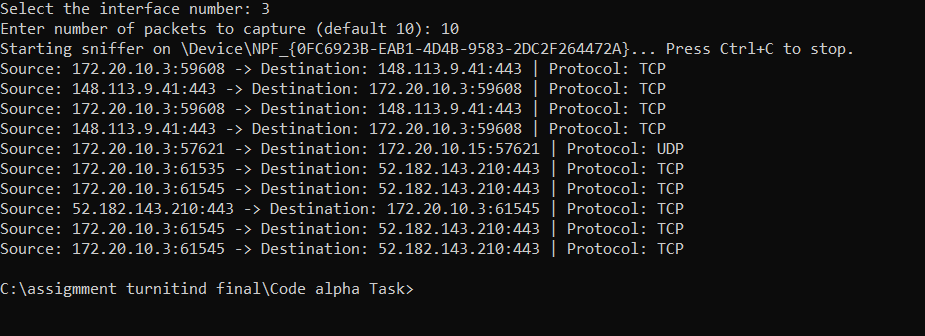
|  |
| --- |
| python network\_sniffer.py |

**Output**



After Just enter a number from the list – Ex -3

**Results**



## **5. Advance option**

 Packet **Filtering Menu**:

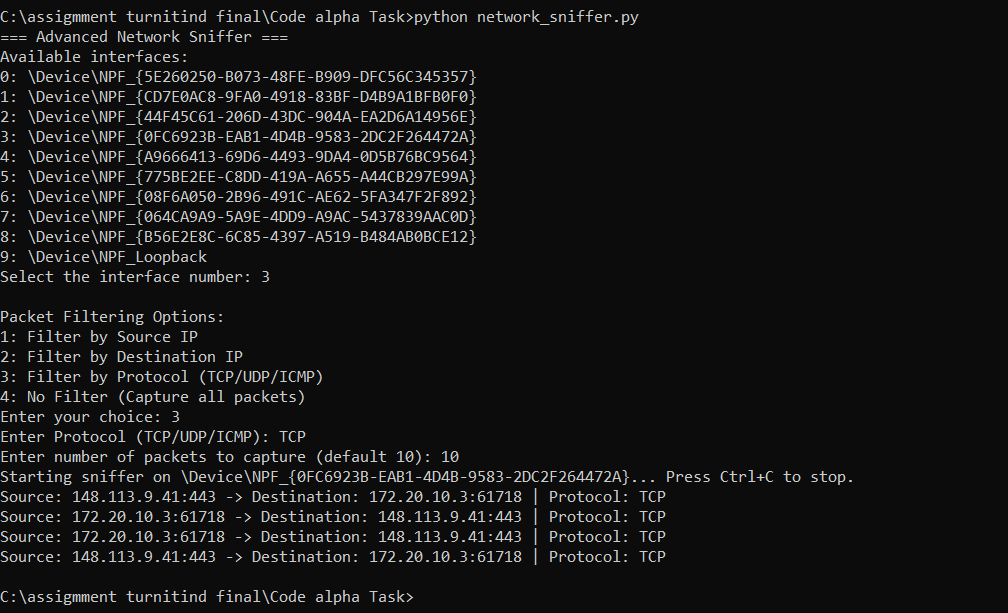
* User can filter packets based on source IP, destination IP, or protocol (TCP/UDP/ICMP).
* The filter option is passed to scapy.sniff() as the filter argument.

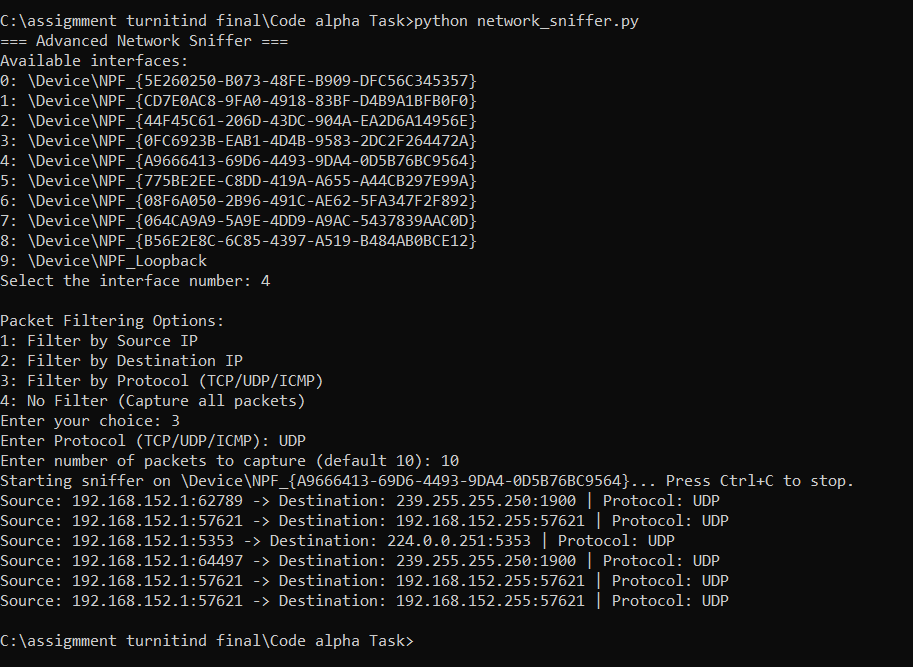
 Suspicious **Packet Detection**:

* This is a basic form of packet inspection for suspicious traffic, such as large HTTP payloads or unusual DNS requests. The system alerts and logs these findings.

 Logging **Capability**:

* Captured packet details, along with suspicious packet alerts, are logged in the logs folder as packet\_logs.txt.





**Log files.**



**Final code**

|  |
| --- |
| import scapy.all as scapy  from scapy.layers.inet import IP, TCP, UDP, ICMP  import sys  import os  import time  # Function to create logs folder if it doesn't exist  def create\_log\_folder():  log\_folder = "C:\\assigmment turnitind final\\Code alpha Task\\logs"  if not os.path.exists(log\_folder):  os.makedirs(log\_folder)  # Function to write logs to file  def write\_log(log\_message):  log\_folder = "C:\\assigmment turnitind final\\Code alpha Task\\logs"  log\_file = os.path.join(log\_folder, "packet\_logs.txt")  with open(log\_file, "a") as log:  log.write(log\_message + "\n")  # Function to get the network interface  def get\_interface():  """Prompt user to select a network interface by index."""  try:  interfaces = scapy.get\_if\_list()  print("Available interfaces:")  for idx, iface in enumerate(interfaces):  print(f"{idx}: {iface}")  choice = int(input("Select the interface number: "))  return interfaces[choice]  except Exception as e:  print(f"Error retrieving interfaces: {e}")  sys.exit(1)  # Function to add a Packet Filtering Menu  def packet\_filtering\_menu():  """Allow the user to set packet filters."""  print("\nPacket Filtering Options:")  print("1: Filter by Source IP")  print("2: Filter by Destination IP")  print("3: Filter by Protocol (TCP/UDP/ICMP)")  print("4: No Filter (Capture all packets)")  filter\_choice = input("Enter your choice: ")  if filter\_choice == "1":  src\_ip = input("Enter Source IP to filter: ")  return f"src host {src\_ip}"  elif filter\_choice == "2":  dst\_ip = input("Enter Destination IP to filter: ")  return f"dst host {dst\_ip}"  elif filter\_choice == "3":  proto = input("Enter Protocol (TCP/UDP/ICMP): ").upper()  if proto == "TCP":  return "tcp"  elif proto == "UDP":  return "udp"  elif proto == "ICMP":  return "icmp"  else:  print("Invalid Protocol")  return ""  elif filter\_choice == "4":  return "" # No filter  else:  print("Invalid choice. Capturing all packets.")  return ""  # Function to detect suspicious packets (basic IDS)  def detect\_suspicious(packet):  """Detect suspicious packets, like abnormal ports."""  try:  if packet.haslayer(TCP):  if packet[TCP].dport == 80 and len(packet[scapy.Raw].load) > 1500:  return "Suspicious HTTP traffic: Large payload"  elif packet[TCP].dport == 53: # DNS port, look for unusual DNS queries  return "Suspicious DNS request"  elif packet.haslayer(ICMP):  return "Suspicious ICMP packet detected"  return None  except Exception as e:  print(f"Error in suspicious packet detection: {e}")  return None  # Function to process each packet  def process\_packet(packet):  """Analyze and display packet details."""  try:  log\_message = ""  if packet.haslayer(IP):  src\_ip = packet[IP].src  dst\_ip = packet[IP].dst  protocol = packet[IP].proto  proto\_name = "Unknown"  src\_port = dst\_port = "Unknown"  if protocol == 6 and packet.haslayer(TCP):  proto\_name = "TCP"  src\_port = packet[TCP].sport  dst\_port = packet[TCP].dport  elif protocol == 17 and packet.haslayer(UDP):  proto\_name = "UDP"  src\_port = packet[UDP].sport  dst\_port = packet[UDP].dport  elif protocol == 1 and packet.haslayer(ICMP):  proto\_name = "ICMP"  src\_port = dst\_port = "N/A"  else:  src\_port = dst\_port = "Unknown"  log\_message = f"Source: {src\_ip}:{src\_port} -> Destination: {dst\_ip}:{dst\_port} | Protocol: {proto\_name}"  # Log the HTTP payload if available  if packet.haslayer(TCP) and packet[TCP].dport == 80 and packet.haslayer(scapy.Raw):  log\_message += f" | HTTP Payload: {packet[scapy.Raw].load[:50]}..."  # Check for suspicious packets  suspicious\_message = detect\_suspicious(packet)  if suspicious\_message:  log\_message += f" | ALERT: {suspicious\_message}"  # Print and log the packet details  print(log\_message)  write\_log(log\_message)  except Exception as e:  print(f"Error processing packet: {e}")  # Function to start the sniffer with the selected interface and filter  def start\_sniffer(interface, packet\_count=10, filter=""):  """Start sniffing packets on the specified interface."""  try:  print(f"Starting sniffer on {interface}... Press Ctrl+C to stop.")  scapy.sniff(iface=interface, prn=process\_packet, filter=filter, count=packet\_count)  except KeyboardInterrupt:  print("\nSniffer stopped by user.")  except Exception as e:  print(f"Error during sniffing: {e}")  sys.exit(1)  # Main function to run the sniffer  def main():  """Main function to run the sniffer."""  print("=== Advanced Network Sniffer ===")  # Create log folder if not exists  create\_log\_folder()  interface = get\_interface()  # Get packet filter from the user  filter = packet\_filtering\_menu()  try:  packet\_count = int(input("Enter number of packets to capture (default 10): ") or 10)  except ValueError:  packet\_count = 10  # Start the packet sniffer  start\_sniffer(interface, packet\_count, filter)  if \_\_name\_\_ == "\_\_main\_\_":  main() |

## **6. Code break down**

**1. MODULE IMPORTS**

|  |
| --- |
| import scapy.all as scapy  from scapy.layers.inet import IP, TCP, UDP, ICMP  import sys  import os  import time |

 Scapy: The main Python library used to capture and analyze packets.

 IP, TCP, UDP, ICMP: Protocol-specific layers from Scapy.

 Sys: Used to exit the program on critical errors.

 Os: Used to create folders or access the file system.

 Time: Available for timestamping (optional).

**2. LOG WRITER**

|  |
| --- |
| def write\_log(log\_message):  log\_folder = "C:\\assigmment turnitind final\\Code alpha Task\\logs"  log\_file = os.path.join(log\_folder, "packet\_logs.txt")  with open(log\_file, "a") as log:  log.write(log\_message + "\n") |

 Appends each captured packet detail (or alert) to a packet\_logs.txt file.

 helps with offline analysis or evidence collection.

**3. SELECT NETWORK INTERFACE**

|  |
| --- |
| # Function to get the network interface  def get\_interface():  """Prompt user to select a network interface by index."""  try:  interfaces = scapy.get\_if\_list()  print("Available interfaces:")  for idx, iface in enumerate(interfaces):  print(f"{idx}: {iface}")  choice = int(input("Select the interface number: "))  return interfaces[choice]  except Exception as e:  print(f"Error retrieving interfaces: {e}")  sys.exit(1) |

 lists all available network interfaces (Ethernet, WiFi, etc.).

 the user selects which interface to sniff from by entering the index number.

**4. PACKET FILTERING MENU**

|  |
| --- |
| def packet\_filtering\_menu():  """Allow the user to set packet filters."""  print("\nPacket Filtering Options:")  print("1: Filter by Source IP")  print("2: Filter by Destination IP")  print("3: Filter by Protocol (TCP/UDP/ICMP)")  print("4: No Filter (Capture all packets)")  filter\_choice = input("Enter your choice: ")  if filter\_choice == "1":  src\_ip = input("Enter Source IP to filter: ")  return f"src host {src\_ip}"  elif filter\_choice == "2":  dst\_ip = input("Enter Destination IP to filter: ")  return f"dst host {dst\_ip}"  elif filter\_choice == "3":  proto = input("Enter Protocol (TCP/UDP/ICMP): ").upper()  if proto == "TCP":  return "tcp"  elif proto == "UDP":  return "udp"  elif proto == "ICMP":  return "icmp"  else:  print("Invalid Protocol")  return ""  elif filter\_choice == "4":  return "" # No filter  else:  print("Invalid choice. Capturing all packets.")  return "" |

 Displays 4 options for packet filtering:

* Source IP
* Destination IP
* Protocol (TCP/UDP/ICMP)
* No filter (capture everything)

 Returns a string that is used by scapy.sniff() as a BPF (Berkeley Packet Filter).

**5. SUSPICIOUS PACKET DETECTION (Basic IDS)**

|  |
| --- |
| def detect\_suspicious(packet):  """Detect suspicious packets, like abnormal ports."""  try:  if packet.haslayer(TCP):  if packet[TCP].dport == 80 and len(packet[scapy.Raw].load) > 1500:  return "Suspicious HTTP traffic: Large payload"  elif packet[TCP].dport == 53: # DNS port, look for unusual DNS queries  return "Suspicious DNS request"  elif packet.haslayer(ICMP):  return "Suspicious ICMP packet detected"  return None  except Exception as e:  print(f"Error in suspicious packet detection: {e}")  return None |

 Basic rules to detect:

* Large HTTP payloads (could be data exfiltration)
* DNS requests on TCP port 53 (usually not normal)
* ICMP messages (potential ping flood attacks)

 Returns a string alert message if suspicious activity is found.

**6. PROCESSING PACKETS**

|  |
| --- |
| def process\_packet(packet):  """Analyze and display packet details."""  try:  log\_message = ""  if packet.haslayer(IP):  src\_ip = packet[IP].src  dst\_ip = packet[IP].dst  protocol = packet[IP].proto  proto\_name = "Unknown"  src\_port = dst\_port = "Unknown"  if protocol == 6 and packet.haslayer(TCP):  proto\_name = "TCP"  src\_port = packet[TCP].sport  dst\_port = packet[TCP].dport  elif protocol == 17 and packet.haslayer(UDP):  proto\_name = "UDP"  src\_port = packet[UDP].sport  dst\_port = packet[UDP].dport  elif protocol == 1 and packet.haslayer(ICMP):  proto\_name = "ICMP"  src\_port = dst\_port = "N/A"  else:  src\_port = dst\_port = "Unknown"  log\_message = f"Source: {src\_ip}:{src\_port} -> Destination: {dst\_ip}:{dst\_port} | Protocol: {proto\_name}"  # Log the HTTP payload if available  if packet.haslayer(TCP) and packet[TCP].dport == 80 and packet.haslayer(scapy.Raw):  log\_message += f" | HTTP Payload: {packet[scapy.Raw].load[:50]}..."  # Check for suspicious packets  suspicious\_message = detect\_suspicious(packet)  if suspicious\_message:  log\_message += f" | ALERT: {suspicious\_message}"  # Print and log the packet details  print(log\_message)  write\_log(log\_message)  except Exception as e:  print(f"Error processing packet: {e}") |

Uses scapy.sniff() to start sniffing:

* iface: selected interface
* prn: function to call for each packet (process\_packet)
* filter: optional BPF string from the filtering menu
* count: how many packets to capture before stopping

## **7. Conclusion**

This enhanced Python-based network sniffer uses Scapy to capture and analyze packets in real time. It allows the user to select a network interface and apply custom filters (like source/destination IP or protocol type). It includes a basic intrusion detection system (IDS) to flag suspicious activity such as large HTTP payloads, unusual DNS behavior, or ICMP floods. All packet details and alerts are logged into a file under a predefined logs folder for later review. The script is fully interactive, user-friendly, and ideal for basic network monitoring and security analysis. Using Python and Scapy, this sophisticated network sniffer software offers a useful and effective tool for basic intrusion detection and real-time packet monitoring. It provides a strong basis for comprehending network behavior and raising cybersecurity awareness with features including interface selection, packet filtering, suspicious activity identification, and logging capabilities. Because of its straightforward yet effective implementation, it may be used for testing, teaching, or light professional applications in network analysis and security research.