**Step 1: Set Up Your Environment**

First, you'll need to install Scapy:

Scapy is a powerful packet manipulation tool used for network security and penetration testing. Below are the steps to install Scapy on Kali Linux:

sudo apt install python3-scapy -y

Then, verify the installation by running:

**Scapy**

**Step 2: Create a Basic Packet Sniffer**

* Here's a simple script to capture packets:

from scapy.all import \*

def packet\_callback(packet):

print(packet.summary())

# Print source and destination IP if present

if IP in packet:

print(f"Source IP: {packet[IP].src}")

print(f"Destination IP: {packet[IP].dst}")

# Print TCP/UDP port information if present

if TCP in packet:

print(f"Source Port: {packet[TCP].sport}")

print(f"Destination Port: {packet[TCP].dport}")

elif UDP in packet:

print(f"Source Port: {packet[UDP].sport}")

print(f"Destination Port: {packet[UDP].dport}")

print("-" \* 40)

# Start sniffing

print("Starting packet capture. Press Ctrl+C to stop.")

sniff(prn=packet\_callback, store=0, count=10) # Capture 10 packets

* sudo nano basic\_packet\_sniffer.py
* sudo python3 basic\_packet\_sniffer.py

A screenshot of a computer program

AI-generated content may be incorrect.

**Step 3: Add Filtering Capabilities**

Let's enhance our sniffer with some basic filtering:

from scapy.all import \*

import argparse

def packet\_callback(packet):

print(packet.summary())

if IP in packet:

print(f"Source IP: {packet[IP].src}")

print(f"Destination IP: {packet[IP].dst}")

if TCP in packet:

print(f"Source Port: {packet[TCP].sport}")

print(f"Destination Port: {packet[TCP].dport}")

# Check for HTTP traffic

if packet[TCP].dport == 80 or packet[TCP].sport == 80:

print("HTTP Traffic Detected!")

if Raw in packet:

print(f"Payload: {packet[Raw].load}")

print("-" \* 40)

def main():

parser = argparse.ArgumentParser(description='Simple Network Sniffer')

parser.add\_argument('-i', '--interface', help='Network interface to use')

parser.add\_argument('-f', '--filter', help='BPF filter to apply')

parser.add\_argument('-c', '--count', type=int, default=10, help='Number of packets to capture')

args = parser.parse\_args()

print(f"Starting capture of {args.count} packets...")

sniff(

iface=args.interface,

filter=args.filter,

prn=packet\_callback,

store=0,

count=args.count

)

if \_\_name\_\_ == "\_\_main\_\_":

main()

sudo nano basic\_packet\_sniffer\_filter.py

**Step 4: Run Your Network Sniffer**

Run the script with admin/root privileges:

sudo python basic\_packet\_sniffer\_filter.py -i eth0 -f "tcp port 80" -c 20

This will:

* Use the eth0 interface
* Filter for HTTP traffic (TCP port 80)
* Capture 20 packets

A screenshot of a computer program

AI-generated content may be incorrect.

**Step 5: Add Protocol Analysis**

Let's enhance our sniffer to analyze common protocols:

from scapy.all import \*

from scapy.layers.http import HTTP

import argparse

import time

from collections import defaultdict

# Statistics tracking

stats = {

'packet\_count': 0,

'protocols': defaultdict(int),

'ip\_sources': defaultdict(int),

'ip\_destinations': defaultdict(int),

'start\_time': time.time()

}

def analyze\_packet(packet):

"""Analyze a packet and extract useful information"""

stats['packet\_count'] += 1

packet\_info = {

'timestamp': time.time(),

'summary': packet.summary(),

'protocol': 'Unknown'

}

# Identify Ethernet layer

if Ether in packet:

packet\_info['src\_mac'] = packet[Ether].src

packet\_info['dst\_mac'] = packet[Ether].dst

stats['protocols']['Ethernet'] += 1

# Identify IP layer

if IP in packet:

packet\_info['src\_ip'] = packet[IP].src

packet\_info['dst\_ip'] = packet[IP].dst

packet\_info['protocol'] = 'IP'

stats['protocols']['IP'] += 1

stats['ip\_sources'][packet[IP].src] += 1

stats['ip\_destinations'][packet[IP].dst] += 1

# Identify IPv6 layer

elif IPv6 in packet:

packet\_info['src\_ip'] = packet[IPv6].src

packet\_info['dst\_ip'] = packet[IPv6].dst

packet\_info['protocol'] = 'IPv6'

stats['protocols']['IPv6'] += 1

# Identify TCP layer

if TCP in packet:

packet\_info['src\_port'] = packet[TCP].sport

packet\_info['dst\_port'] = packet[TCP].dport

packet\_info['protocol'] = 'TCP'

stats['protocols']['TCP'] += 1

# Check for HTTP

if packet[TCP].dport == 80 or packet[TCP].sport == 80:

packet\_info['protocol'] = 'HTTP'

stats['protocols']['HTTP'] += 1

# Check for HTTPS

elif packet[TCP].dport == 443 or packet[TCP].sport == 443:

packet\_info['protocol'] = 'HTTPS'

stats['protocols']['HTTPS'] += 1

# Identify UDP layer

elif UDP in packet:

packet\_info['src\_port'] = packet[UDP].sport

packet\_info['dst\_port'] = packet[UDP].dport

packet\_info['protocol'] = 'UDP'

stats['protocols']['UDP'] += 1

# Check for DNS

if packet[UDP].dport == 53 or packet[UDP].sport == 53:

packet\_info['protocol'] = 'DNS'

stats['protocols']['DNS'] += 1

if DNSQR in packet:

packet\_info['dns\_query'] = packet[DNSQR].qname.decode()

# Check for ICMP

elif ICMP in packet:

packet\_info['protocol'] = 'ICMP'

stats['protocols']['ICMP'] += 1

packet\_info['icmp\_type'] = packet[ICMP].type

packet\_info['icmp\_code'] = packet[ICMP].code

# Extract payload if available

if Raw in packet:

packet\_info['payload'] = packet[Raw].load

packet\_info['payload\_len'] = len(packet[Raw].load)

return packet\_info

def print\_packet\_info(packet\_info):

"""Print formatted packet information"""

print("\n" + "=" \* 60)

print(f"PACKET: {stats['packet\_count']}")

print("=" \* 60)

# Print basic information

print(f"Protocol: {packet\_info['protocol']}")

print(f"Summary: {packet\_info['summary']}")

# Print MAC addresses if available

if 'src\_mac' in packet\_info:

print(f"Source MAC: {packet\_info['src\_mac']}")

print(f"Destination MAC: {packet\_info['dst\_mac']}")

# Print IP addresses if available

if 'src\_ip' in packet\_info:

print(f"Source IP: {packet\_info['src\_ip']}")

print(f"Destination IP: {packet\_info['dst\_ip']}")

# Print port information if available

if 'src\_port' in packet\_info:

print(f"Source Port: {packet\_info['src\_port']}")

print(f"Destination Port: {packet\_info['dst\_port']}")

# Protocol-specific details

if packet\_info['protocol'] == 'DNS' and 'dns\_query' in packet\_info:

print(f"DNS Query: {packet\_info['dns\_query']}")

if packet\_info['protocol'] == 'ICMP':

icmp\_types = {0: "Echo Reply", 8: "Echo Request"}

icmp\_type = icmp\_types.get(packet\_info['icmp\_type'], f"Type {packet\_info['icmp\_type']}")

print(f"ICMP: {icmp\_type}, Code {packet\_info['icmp\_code']}")

# Print payload information if available

if 'payload' in packet\_info:

print(f"Payload Length: {packet\_info['payload\_len']} bytes")

if packet\_info['payload\_len'] < 100: # Only show short payloads

try:

print(f"Payload (ASCII): {packet\_info['payload'].decode('ascii', errors='replace')}")

except:

print("Payload: (Binary data)")

def print\_statistics():

"""Print capture statistics"""

duration = time.time() - stats['start\_time']

print("\n" + "#" \* 70)

print(f"CAPTURE STATISTICS (Duration: {duration:.2f} seconds)")

print("#" \* 70)

print(f"Total Packets: {stats['packet\_count']}")

if stats['packet\_count'] > 0:

print("\nProtocol Distribution:")

for protocol, count in sorted(stats['protocols'].items(), key=lambda x: x[1], reverse=True):

percentage = (count / stats['packet\_count']) \* 100

print(f" {protocol}: {count} packets ({percentage:.1f}%)")

print("\nTop Source IPs:")

for ip, count in sorted(stats['ip\_sources'].items(), key=lambda x: x[1], reverse=True)[:5]:

print(f" {ip}: {count} packets")

print("\nTop Destination IPs:")

for ip, count in sorted(stats['ip\_destinations'].items(), key=lambda x: x[1], reverse=True)[:5]:

print(f" {ip}: {count} packets")

print(f"\nPacket Rate: {stats['packet\_count'] / duration:.2f} packets/second")

def packet\_callback(packet):

"""Process each captured packet"""

packet\_info = analyze\_packet(packet)

print\_packet\_info(packet\_info)

def main():

parser = argparse.ArgumentParser(description='Network Traffic Analyzer')

parser.add\_argument('-i', '--interface', help='Network interface to use')

parser.add\_argument('-f', '--filter', help='BPF filter to apply')

parser.add\_argument('-c', '--count', type=int, default=100, help='Number of packets to capture')

parser.add\_argument('-o', '--output', help='Output file for packet capture (pcap format)')

args = parser.parse\_args()

print("Network Traffic Analyzer")

print("-" \* 30)

print(f"Interface: {args.interface or 'default'}")

print(f"Filter: {args.filter or 'none'}")

print(f"Packet limit: {args.count}")

print("-" \* 30)

print("Starting capture... Press Ctrl+C to stop.")

try:

# Capture packets

packets = sniff(

iface=args.interface,

filter=args.filter,

prn=packet\_callback,

store=1 if args.output else 0,

count=args.count

)

# Save capture if output file specified

if args.output and packets:

wrpcap(args.output, packets)

print(f"\nCaptured packets saved to {args.output}")

# Print final statistics

print\_statistics()

except KeyboardInterrupt:

print("\nCapture stopped by user.")

print\_statistics()

except Exception as e:

print(f"\nError: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Understanding the Code**

1. **Packet Capture**: The sniff() function from Scapy captures network packets.
2. **Packet Analysis**: Each captured packet is processed by the packet\_callback function, which:

* Extracts layer information (Ethernet, IP, TCP, UDP)
* Identifies protocols (HTTP, DNS, ICMP)
* Collects statistics on traffic patterns

1. **Protocol Detection**:

* TCP port 80 → HTTP
* TCP port 443 → HTTPS
* UDP port 53 → DNS

1. **Statistics Tracking**:

* Protocol distribution
* Top source and destination IPs
* Packet rate

**Running the Advanced Sniffer**

* To monitor all traffic on your default interface

sudo python3 network\_sniffer.py -c 50

* To monitor specific traffic

sudo python3 network\_sniffer.py -i eth0 -f "port 80 or port 443" -c 25 -o capture.pcap

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.