Basic Network Sniffer – Code Explanation

```
from scapy.all import sniff, IP, Ether, TCP, UDP, ICMP
```

Explanation:

Imports: This line imports specific functions and classes from the Scapy library:

- `sniff`: A function for capturing network packets.
- `IP`, `Ether`, `TCP`, `UDP`, `ICMP`: Classes for handling different types of network layers and protocols.

```
def sniff_packets(packet):
```

Explanation:

Function Definition: Defines a function named sniff_packets that will be called for each packet captured by Scapy.

```
  if IP in packet:
```

Explanation:

IP Layer Check: Checks if the packet contains an IP layer. If it does, it proceeds to extract and print IP-related information.

```
    ip_src = packet[IP].src
        ip_dst = packet[IP].dst
        proto = packet[IP].proto
        ttl = packet[IP].ttl
        flags = packet[IP].flags
        frag_offset = packet[IP].frag
        length = len(packet)
        time = packet.time
```

Explanation:

- Extract IP Information:
  - `ip_src`: Source IP address.
  - `ip_dst`: Destination IP address.
  - `proto`: Protocol used (e.g., TCP, UDP, ICMP).
  - `ttl`: Time-to-Live value of the packet.
  - `flags`: IP flags (e.g., DF, MF).
  - `frag_offset`: Fragment offset of the packet (used in fragmented packets).
  - `length`: Length of the packet.
  - `time`: Timestamp when the packet was captured.

```
        print(f"Time: {time}, Source IP: {ip_src}, Destination IP: {ip_dst}, Protocol:
{proto}, Length: {length}")
        print(f"    TTL: {ttl}, Flags: {flags}, Fragment Offset: {frag_offset}")
```

Explanation:

Print IP Information: Prints the extracted information in a human-readable format.

```
        if Ether in packet:
            src_mac = packet[Ether].src
            dst_mac = packet[Ether].dst
            print(f"    Source MAC: {src_mac}, Destination MAC: {dst_mac}")
```

Explanation:

Ethernet Layer Check: Checks if the packet contains an Ethernet layer and extracts MAC addresses:

- `src_mac`: Source MAC address.
- `dst_mac`: Destination MAC address.
- Prints these MAC addresses.

```
        if proto == 6 and TCP in packet:
            sport = packet[TCP].sport
            dport = packet[TCP].dport
            seq = packet[TCP].seq
            ack = packet[TCP].ack
            flags = packet[TCP].flags
            payload = packet[TCP].payload
            print(f"    TCP Source Port: {sport}, Destination Port: {dport}")
            print(f"    Sequence Number: {seq}, Acknowledgment Number: {ack}, Flags:
{flags}")
            print(f"    Payload: {payload}")
```

Explanation:

TCP Protocol Check: If the IP protocol number is `6` (indicating TCP) and the packet contains a TCP layer, it extracts and prints:

- `sport`: Source port.
- `dport`: Destination port.
- `seq`: Sequence number.
- `ack`: Acknowledgment number.
- `flags`: TCP flags (e.g., SYN, ACK).
- `payload`: TCP payload.

```
        elif proto == 17 and UDP in packet:
            sport = packet[UDP].sport
            dport = packet[UDP].dport
            payload = packet[UDP].payload
            print(f"    UDP Source Port: {sport}, Destination Port: {dport}")
            print(f"    Payload: {payload}")
```

Explanation:

UDP Protocol Check: If the IP protocol number is `17` (indicating UDP) and the packet contains a UDP layer, it extracts and prints:

- `sport`: Source port.
- `dport`: Destination port.
- `payload`: UDP payload.

```
    elif proto == 1 and ICMP in packet:
        icmp_type = packet[ICMP].type
        icmp_code = packet[ICMP].code
        payload = packet[ICMP].payload
        print(f"    ICMP Type: {icmp_type}, Code: {icmp_code}")
        print(f"    Payload: {payload}")
```

Explanation:

ICMP Protocol Check: If the IP protocol number is `1` (indicating ICMP) and the packet contains an ICMP layer, it extracts and prints:

- `icmp_type`: ICMP type (e.g., echo request, echo reply).
- `icmp_code`: ICMP code (e.g., network unreachable).
- `payload`: ICMP payload.

```
else:
        print("Other protocol or unrecognized packet")
```

Explanation:

Default Case: If the packet does not match any of the above protocols, it prints a message indicating the packet's protocol is not recognized.

```
if __name__ == "__main__":
    sniff(prn=sniff_packets, store=0)
```

Explanation:

Main Check: Ensures the script is being run directly (not imported as a module). It calls the `sniff` function to start capturing packets.

- `prn=sniff_packets`: Specifies the callback function (`sniff_packets`) to process each captured packet.
- `store=0`: Specifies not to store packets in memory, which helps to reduce memory usage.

This script is a basic network sniffer that captures and analyzes network packets. It extracts and displays detailed information about IP, Ethernet, TCP, UDP, and ICMP layers of the packets, allowing you to understand the structure and content of network traffic. You can run this script with superuser privileges to capture network traffic on the specified network interface.