EXP:14	Write a code using RAW sockets to implement packet sniffing

Aim

To write a program using **RAW sockets** to implement a basic **packet sniffer** and capture network traffic.

Algorithm / Procedure

1. **Understand** the concept of raw sockets, which allow access to the data link layer or network layer headers.
2. **Create** a raw socket that can capture all incoming packets on a specified interface.
3. **Implement** a loop to continuously receive packets from the raw socket.
4. **Parse** the captured raw data to extract and display key information from the Ethernet, IP, and TCP/UDP headers (e.g., Source/Destination MAC, Source/Destination IP, Source/Destination Port).
5. **Test** the sniffer by generating traffic (e.g., browsing, pinging) and observing the captured packet details.

Code:

```
import socket
import struct
import textwrap
import time
import sys
import ctypes

# Windows-only constants
SIO_RCVALL = 0x98000001
RCVALL_ON = 1
RCVALL_OFF = 0

def mac_addr(bytes_addr):
    return ':'.join('{:02x}'.format(b) for b in bytes_addr)

def ipv4(addr_bytes):
    return '.'.join(map(str, addr_bytes))

def hexdump(src, length=16): results
    = []
    digits = 2
    for i in range(0, len(src), length):
        chunk = src[i:i+length]
        hexa = ' '.join(f'{b:02x}' for b in chunk)
        text = ''.join((chr(b) if 32 <= b < 127 else '.') for b in chunk)
```

```
        results.append(f'{i:04x}  {hexa:<{length*(digits+1)}}  {text}')
    return '\n'.join(results)


def parse_ip_header(data):
    # Unpack first 20 bytes of IP header
    iph = struct.unpack('!BBHHHBBH4s4s', data[:20])
    version_ihl = iph[0]
    version = version_ihl >> 4
    ihl = (version_ihl & 0xF) * 4
    tos = iph[1]
    total_length = iph[2]
    identification = iph[3]
    flags_offset = iph[4]
    ttl = iph[5]
    protocol = iph[6]
    checksum = iph[7]
    src = ipv4(iph[8])
    dst = ipv4(iph[9])
    return {
        'version': version,
        'ihl': ihl,
        'tos': tos,
        'total_length': total_length,
        'id': identification,
        'flags_offset': flags_offset,
        'ttl': ttl,
        'protocol': protocol,
        'checksum': checksum,
        'src': src,
        'dst': dst,
        'payload': data[ihl:total_length]
    }

def protocol_name(p):
    return {1: 'ICMP', 6: 'TCP', 17: 'UDP'}.get(p, str(p))

def main(listen_addr='0.0.0.0'):
    print(f'[*] Starting sniffer. Listening on {listen_addr}. Press Ctrl+C to stop.') try:
        # Create RAW socket
        s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_IP)
    except PermissionError:
        print('ERROR: Must run as Administrator to create raw socket.')
        sys.exit(1)

    # Bind to interface
    s.bind((listen_addr, 0))

    # Include IP header in captured packets s.setsockopt(socket.IPPROTO_IP,
    socket.IP_HDRINCL, 1)
```

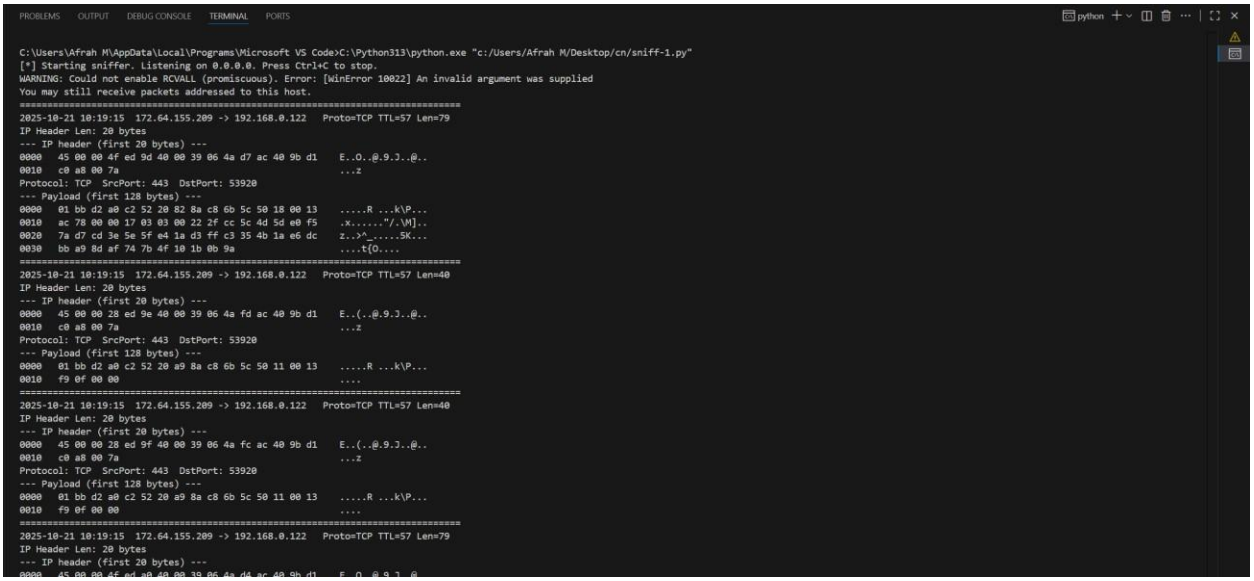
```
    # Enable promiscuous mode (Windows-specific) #
    Use ioctls via socket.ioctl
    try:
      s.ioctl(SIO_RCVALL, RCVALL_ON)
    except Exception as e:
      print('WARNING: Could not enable RCVALL (promiscuous). Error:', e)
      print('You may still receive packets addressed to this host.')

    try:
      while True:
        raw_data, addr = s.recvfrom(65535)
        ts = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime()) ip
        = parse_ip_header(raw_data)
        print('='*80)
        print(f'{ts}  {ip["src"]} -> {ip["dst"]}  Proto={protocol_name(ip["protocol"])}
TTL={ip["ttl"]} Len={ip["total_length"]}')
        # Show first 64 bytes of payload as hex dump
        header_len = ip['ihl']
        print(f'IP Header Len: {header_len} bytes')
        print('--- IP header (first 20 bytes) ---')
        print(hexdump(raw_data[:header_len], length=16))
        # Determine protocol and print simple port info for TCP/UDP if
        ip['protocol'] == 6 and len(ip['payload']) >= 4:  # TCP
          src_port, dst_port = struct.unpack('!HH', ip['payload'][:4])
          print(f'Protocol: TCP  SrcPort: {src_port}  DstPort: {dst_port}')
        elif ip['protocol'] == 17 and len(ip['payload']) >=4:  # UDP src_port,
          dst_port = struct.unpack('!HH', ip['payload'][:4]) print(f'Protocol:
          UDP  SrcPort: {src_port}  DstPort: {dst_port}')
        else:
          print(f'Protocol: {protocol_name(ip["protocol"])}')


        # Show payload (first 128 bytes)
        payload = ip['payload'][:128]
        if payload:
          print('--- Payload (first 128 bytes) ---')
          print(hexdump(payload, length=16))
        else:
          print('No payload (or payload skipped).')
    except KeyboardInterrupt:
      print('\n[*] Stopping sniffer.') finally:
      # Turn off promiscuous
      try:
        s.ioctl(SIO_RCVALL, RCVALL_OFF)
      except Exception:
        pass
      s.close()
```

```
if __name__== '__main__':
    # If you want to listen on a specific local IP, pass it here; default is 0.0.0.0
    main()
```

Output:



Result

A basic packet sniffer was successfully implemented using raw sockets. The program demonstrated the ability to capture and parse raw network packets, extracting header information from different protocol layers