**COMP 7003**

**Introduction to Information and Network Security**

*Assignment-02*

*Design*

##### Anmol Mittal A01397754

##### February 2nd, 2025

##### Course Reference Number (CRN): 91662

[**Purpose** 3](#_Toc189331324)

[**Functions** 3](#_Toc189331325)

[Packet Sniffer (main.py) 3](#_Toc189331326)

[Packet Parser (packet\_parsers.py) 3](#_Toc189331327)

[**States** 4](#_Toc189331328)

[Packet Sniffer (main.py) 4](#_Toc189331329)

[**State Table** 4](#_Toc189331330)

[Client 4](#_Toc189331331)

[**State Transition Diagram** 4](#_Toc189331332)

[**Pseudocode** 4](#_Toc189331333)

[Packet Sniffer (main.py) 4](#_Toc189331334)

[Main Execution Block 4](#_Toc189331335)

[capture\_on\_all\_interfaces: 5](#_Toc189331336)

[capture\_packets: 6](#_Toc189331337)

[packet\_callback: 6](#_Toc189331338)

[interface\_is\_loopback: 6](#_Toc189331339)

[has\_global\_ip: 6](#_Toc189331340)

[Packet Parser (packet\_parsers.py) 7](#_Toc189331341)

[parse\_ethernet\_header: 7](#_Toc189331342)

[parse\_arp\_header: 7](#_Toc189331343)

[parse\_ipv4\_header: 7](#_Toc189331344)

[parse\_udp\_header: 8](#_Toc189331345)

[parse\_tcp\_header: 8](#_Toc189331346)

[parse\_icmp\_header: 8](#_Toc189331347)

# **Purpose**

The purpose of the program is to capture and analyze network traffic at the packet level using Python and Scapy. It will filter packets by protocol (Ethernet, IPv4, ICMP, TCP, UDP, DNS), convert raw packet data into hexadecimal dumps, and parse the packet headers to extract and display key fields such as source/destination MAC and IP addresses, protocol-specific details, and port numbers. The program aims to provide a clear, structured, and human-readable output of packet information.

The program accepts the command line argument as follows:

* sudo python3 main.py -i <interface> -f <filter> -c <count>

-i or --interface: Specifies the network interface to capture packets on. Use any to capture on all available interfaces.

-f or --filter: Specifies the BPF to apply. Common filters include tcp, udp, icmp, and arp. If no filter is provided, the program will capture all packets.

-c or --count: Specifies the number of packets to capture. Default is 1.

# **Functions**

## Packet Sniffer (main.py)

|  |  |
| --- | --- |
| **Function** | **Description** |
| main | Parses command-line arguments and starts the packet capture. |
| capture\_packets | Captures packets on a specific interface. |
| capture\_on\_all\_interfaces | Captures packets on all available interfaces. |
| packet\_callback | Processes each captured packet and extracts header information. |
| interface\_is\_loopback | Check if an interface is a loopback interface. |
| has\_global\_ip | Determines if an interface has a global IP address. |

## Packet Parser (packet\_parsers.py)

|  |  |
| --- | --- |
| **Function** | **Description** |
| parse\_ethernet\_header | Extracts and displays Ethernet header details. |
| parse\_arp\_header | Extracts and displays ARP header details. |
| parse\_ipv4\_header | Extracts and displays IPv4 header details. |
| parse\_udp\_header | Extracts and displays UDP header details. |
| parse\_tcp\_header | Extracts and displays TCP header details. |
| parse\_icmp\_header | Extracts and displays ICMP header details. |

# **States**

## Packet Sniffer (main.py)

|  |  |
| --- | --- |
| **State** | **Description** |
| START | Initialize arguments and setup capture parameters. |
| SELECT | Choose an interface or capture on all interfaces. |
| CAPTURING | Capture and process packets. |
| STOPPED | Stop capture and processing after reaching the packet limit. |

# **State Table**

## Client

|  |  |  |
| --- | --- | --- |
| **From State** | **To State** | **Function** |
| START | SELECT | main |
| SELECT | CAPTURING | capture\_packets |
| CAPTURING | STOPPED | packet\_callback |

# **State Transition Diagram**

# **Pseudocode**

## Packet Sniffer (main.py)

### Main Execution Block

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| arguments | list | Command-line arguments passed to the program. |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | The function initializes and starts execution. |

#### Pseudo Code

IF program is executed directly THEN:

ADD argument "-i" or "--interface" with:

- Default value: "any"

- Help message: "The interface to capture packets on (e.g., eth0, wlan0, any) (default: any)"

ADD argument "-f" or "--filter" with:

- Help message: "BPF filter to apply (e.g., 'tcp, udp, arp, icmp'). If not provided, captures all packets."

ADD argument "-c" or "--count" with:

- Data type: Integer

- Default value: 1

- Help message: "Number of packets to capture (default: 1)"

Parse command-line arguments

IF count < 0 THEN:

PRINT "Error: The packet count (-c) cannot be negative."

EXIT

IF filter is NOT provided THEN:

Ask the user to provide

IF user input is in ["tcp", "icmp", "arp", "udp"] THEN:

filter ← user\_input

ELSE IF user\_input is in ["", "any", "none"] THEN:

Proceeding with capturing all packets.

ELSE:

INVALID FILTER

EXIT

IF interface converted = "any" THEN:

CALL capture\_on\_all\_interfaces

ELSE:

ERROR

### capture\_on\_all\_interfaces:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| filter | string | The BPF filter to apply to the packet capture. |
| packet\_count | interger | The number of packets to capture before stopping. |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Captures packets on all interfaces. |

#### Pseudo Code

Professor Provided Function

### capture\_packets**:**

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| interface | String | The network interface to capture packets from. |
| filter | String | The BPF filter to apply to the packet capture. |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Captures packets until the stop condition is met. |

#### Pseudo Code

Professor Provided Function

### packet\_callback**:**

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| packet | Object | The captured packet object. |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Processes a captured packet |

#### Pseudo Code

Professor Provided Function

### interface\_is\_loopback:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| interface | String | The name of the interface to check. |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| Boolean | Returns True if the interface is a loop back, else False. |

#### Pseudo Code

Professor Provided Function

### has\_global\_ip:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| interface | String | The name of the interface to check. |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| Boolean | Returns True if the interface has a global IP, else False. |

#### Pseudo Code

Professor Provided Function

## Packet Parser (packet\_parsers.py)

### parse\_ethernet\_header:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| hex\_data | String | The raw packet data in hexadecimal format. |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Displays parsed Ethernet header details. |

#### Pseudo Code

### parse\_arp\_header:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| hex\_data | String | The raw packet data in hexadecimal format. |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Displays parsed ARP header details. |

#### Pseudo Code

### parse\_ipv4\_header:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| hex\_data | String | The raw packet data in hexadecimal format. |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Displays parsed IPv4 header details. |

#### Pseudo Code

### parse\_udp\_header:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| hex\_data | String | The raw packet data in hexadecimal format. |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Displays parsed UDP header details. |

#### Pseudo Code

### parse\_tcp\_header:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| hex\_data | String | The raw packet data in hexadecimal format. |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Displays parsed TCP header details. |

#### Pseudo Code

### parse\_icmp\_header:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| hex\_data | String | The raw packet data in hexadecimal format. |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Displays parsed ICMP header details. |

#### Pseudo Code