**COMP 7003**

**Introduction to Information and Network Security**

*Assignment-02*

*Design*

##### Anmol Mittal A01397754

##### February 2nd, 2025

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

[**Purpose** 3](#_Toc189345469)

[**Functions** 3](#_Toc189345470)

[Packet Sniffer (main.py) 3](#_Toc189345471)

[Packet Parser (packet\_parsers.py) 3](#_Toc189345472)

[**States** 4](#_Toc189345473)

[Packet Sniffer (main.py) 4](#_Toc189345474)

[**State Table** 4](#_Toc189345475)

[Client 4](#_Toc189345476)

[**Pseudocode** 4](#_Toc189345477)

[Packet Sniffer (main.py) 4](#_Toc189345478)

[Main Execution Block 4](#_Toc189345479)

[capture\_on\_all\_interfaces: 5](#_Toc189345480)

[capture\_packets: 6](#_Toc189345481)

[packet\_callback: 6](#_Toc189345482)

[interface\_is\_loopback: 6](#_Toc189345483)

[has\_global\_ip: 6](#_Toc189345484)

[Packet Parser (packet\_parsers.py) 7](#_Toc189345485)

[parse\_ethernet\_header: 7](#_Toc189345486)

[parse\_arp\_header: 7](#_Toc189345487)

[parse\_ipv4\_header: 8](#_Toc189345488)

[parse\_udp\_header: 9](#_Toc189345489)

[parse\_tcp\_header: 10](#_Toc189345490)

[parse\_icmp\_header: 11](#_Toc189345491)

# **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 |

# **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

parse\_arguments:

Initialize parser with description "Packet sniffer using Scapy with manual HEX parsing"

Add argument for interface with default "any"

Add argument for filter with no default

Add argument for count with default 1

Parse arguments

validate\_count:

IF count < 0 THEN:

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

EXIT

validate\_filter:

IF filter is not empty THEN:

IF filter not in allowed\_filters THEN:

DISPLAY "Error: Invalid filter '{filter}'. Allowed filters: {allowed\_filters}"

EXIT

ELSE:

DISPLAY prompt for filter input (tcp, icmp, udp, arp, or all)

IF user input is valid THEN:

SET filter to user input

ELSE:

DISPLAY ERROR

EXIT

validate\_interface:

IF interface is "any" THEN:

CALL capture\_on\_all\_interfaces with filter and count

ELSE:

IF interface has global IP THEN:

TRY:

CALL capture\_packets with interface, filter

EXCEPT Exception:

DISPLAY ERROR

ELSE:

DISPLAY 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\_header:

destination\_address ← Extract first 12 characters from packet\_data

source\_address ← Extract next 12 characters from packet\_data

ether\_type ← Extract next 4 characters from packet\_data

DISPLAY " Destination Address: " destination\_address

DISPLAY " Source Address: " source\_address

DISPLAY " Protocol Identifier: " protocol\_identifier

payload\_data ← Extract remaining data from packet\_data

IF ether\_type corresponds to 0806:

CALL parse\_arp\_header and pass on the payload data

ELSE IF ether\_type = "0800" THEN:

CALL parse\_ipv4\_header and pass on the payload data

ELSE:

ERROR

### 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\_arp\_header:

hardware\_type ← Convert first 4 characters of hex\_data to integer

protocol\_type ← Convert next 4 characters of hex\_data to integer

hardware\_size ← Convert next 2 characters of hex\_data to integer

protocol\_size ← Convert next 2 characters of hex\_data to integer

operation\_code ← Convert next 4 characters of hex\_data to integer

sender\_mac\_address ← Extract and format characters 16 to 28 as MAC address

sender\_ip\_address ← Extract and format characters 28 to 36 as IPv4 address

target\_mac\_address ← Extract and format characters 36 to 48 as MAC address

target\_ip\_address ← Extract and format characters 48 to 56 as IPv4 address

DISPLAY "ARP Header:"

DISPLAY " Hardware Type: " hardware\_type

DISPLAY " Protocol Type: " protocol\_type

DISPLAY " Hardware Size: " hardware\_size

DISPLAY " Protocol Size: " protocol\_size

DISPLAY " Operation: " operation\_code

DISPLAY " Sender MAC: " sender\_mac\_address

DISPLAY " Sender IP: " sender\_ip\_address

DISPLAY " Target MAC: " target\_mac\_address

DISPLAY " Target IP: " target\_ip\_address

### 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\_ipv4\_header:

version ← Convert first character of hex\_data to integer

internet\_header\_length ← Convert second character of hex\_data to integer

type\_of\_service ← Convert next 2 characters of hex\_data to integer

total\_length ← Convert next 4 characters of hex\_data to integer

identification ← Convert next 4 characters of hex\_data to integer

flags\_and\_fragment\_offset ← Convert next 4 characters of hex\_data to integer

CONVERT flags\_and\_fragment\_offset to binary

reserved\_bit ← Extract first bit

df\_bit ← Extract second bit

mf\_bit ← Extract third bit

fragment\_offset ← Convert remaining 13 bits to integer

protocol ← Convert next 2 characters of hex\_data to integer

source\_ip ← Extract and format characters 24 to 32 as IPv4 address

destination\_ip ← Extract and format characters 32 to 40 as IPv4 address

DISPLAY " Version: " version

DISPLAY " Header Length: " internet\_header\_length

DISPLAY " Total Length: " total\_length

DISPLAY " Identification: " identification

DISPLAY " Flags & Fragment Offset: " flags\_and\_fragment\_offset

DISPLAY " Reserved Bit: " reserved\_bit

DISPLAY " DF (Do Not Fragment): " df\_bit

DISPLAY " MF (More Fragments): " mf\_bit

DISPLAY " Fragment Offset: " fragment\_offset

DISPLAY " Protocol: " protocol

DISPLAY " Source IP: " source\_ip

DISPLAY " Destination IP: " destination\_ip

IF protocol corresponds to UDP:

CALL parse\_udp\_header and pass remaining payload data

ELSE IF protocol corresponds to TCP:

CALL parse\_tcp\_header and pass remaining payload data

ELSE IF protocol corresponds to ICMP:

CALL parse\_icmp\_header and pass remaining payload data

ELSE:

ERROR

### 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\_udp\_header:

source\_port ← Convert first 4 characters of hex\_data to integer

destination\_port ← Convert next 4 characters of hex\_data to integer

length ← Convert next 4 characters of hex\_data to integer

checksum ← Convert next 4 characters of hex\_data to integer

payload ← Extract remaining data from hex\_data

DISPLAY "Parsing UDP Header"

DISPLAY " Source Port: " source\_port

DISPLAY " Destination Port: " destination\_port

DISPLAY " Length: " length

DISPLAY " Checksum: " checksum

IF payload is not empty:

DISPLAY "UDP Payload: " payload

ELSE:

No UDP Payload

### 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\_tcp\_header:

source\_port ← Convert first 4 characters of hex\_data to integer

destination\_port ← Convert next 4 characters of hex\_data to integer

sequence\_number ← Convert next 8 characters of hex\_data to integer

ack\_number ← Convert next 8 characters of hex\_data to integer

data\_offset ← Convert next 1 character of hex\_data to integer

reserved ← Convert next 1 character of hex\_data to integer

flags ← Convert next 2 characters of hex\_data to integer

window ← Convert next 4 characters of hex\_data to integer

checksum ← Convert next 4 characters of hex\_data to integer

urgent\_pointer ← Convert next 4 characters of hex\_data to integer

payload\_offset ← data\_offset \* 4 # Convert from words to bytes

payload ← Extract remaining data starting from payload\_offset \* 2

DISPLAY "Parsing TCP Header"

DISPLAY " Source Port: " source\_port

DISPLAY " Destination Port: " destination\_port

DISPLAY " Sequence Number: " sequence\_number

DISPLAY " Acknowledgment Number: " ack\_number

DISPLAY " Data Offset: " data\_offset

DISPLAY " Reserved: " reserved

DISPLAY " Flags: " flags (binary representation)

DISPLAY " NS: " flags[3]

DISPLAY " CWR: " flags[4]

DISPLAY " ECE: " flags[5]

DISPLAY " URG: " flags[6]

DISPLAY " ACK: " flags[7]

DISPLAY " PSH: " flags[8]

DISPLAY " RST: " flags[9]

DISPLAY " SYN: " flags[10]

DISPLAY " FIN: " flags[11]

DISPLAY " Window: " window

DISPLAY " Checksum: " checksum

DISPLAY " Urgent Pointer: " urgent\_pointer

IF payload is not empty:

DISPLAY "TCP Payload: " payload

ELSE:

No TCP Payload

### 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

parse\_icmp\_header:

icmp\_type ← Convert first 2 characters of hex\_data to integer

code ← Convert next 2 characters of hex\_data to integer

checksum ← Convert next 4 characters of hex\_data to integer

payload ← Extract remaining data from position 48 onward

DISPLAY "Parsing ICMP Header"

DISPLAY " Type: " icmp\_type

DISPLAY " Code: " code

DISPLAY " Checksum: " checksum

IF payload is not empty:

DISPLAY "ICMP Payload: " payload

ELSE:

DISPLAY "No ICMP Payload"

RETURN icmp\_type, code, checksum, payload