

# **Assignment 2 Report**

## Design

Richard Ohata  
British Columbia Institute of Technology  
A01274710  
COMP 7003: Intro to Info & Network Security  
3 Oct 2025

<b>Purpose</b>	<b>5</b>
<b>Data Types</b>	<b>5</b>
Arguments	5
Settings	5
Context	6
Functions	6
States	7
State Table	7
<b>State Transition Diagram</b>	<b>8</b>
<b>Pseudocode</b>	<b>8</b>
Display_hex_dump	8
Parameters	8
Return	8
Pseudo Code	9
Packet_callback	9
Parameters	9
Return	9
Pseudo Code	9
Interface_is_loopback	10
Parameters	10
Return	10
Pseudo Code	10
has_global_ip	10
Parameters	10
Return	11
Pseudo Code	11
Capture_packets	11
Parameters	11
Return	11
Pseudo Code	12
capture_on_all_interfaces	12
Parameters	12
Return	12
Pseudo Code	12
__main__	13
Parameters	13
Return	13
Pseudo Code	13
parse_ethernet_header	14
Parameters	14
Return	14

Pseudo Code	14
parse_IPV4	15
Parameters	15
Return	15
Pseudo Code	15
parse_IPV6	16
Parameters	16
Return	16
Pseudo Code	16
parse_arp_header	16
Parameters	16
Return	17
Pseudo Code	17
parse_udp_header	17
Parameters	17
Return	17
Pseudo Code	17
parse_tcp_header	18
Parameters	18
Return	18
Pseudo Code	18
parse_icmp_header	18
Parameters	18
Return	19
Pseudo Code	19
parse_icmpv6_header	19
Parameters	19
Return	19
Pseudo Code	19
parse_dns	20
Parameters	20
Return	20
Pseudo Code	20

## Purpose

This assignment demonstrates capturing network packets from networks using Python and Scapy, including header parsing, and displaying the information of the result of parsing.

This program accepts the arguments from the command line:

- -i any -c 1 -f <protocol>  
(Goes in depth in [User Guide](#))

Where the protocol can be specified with:

- ARP
- UDP
- TCP
- ICMP

The result of this displays the hex dump of the packet and also parses the packet headers to display each in a human readable format.

## Data Types

### Arguments

Purpose: To hold the unparsed command-line argument information

Field	Type	Description
program_name	string	Name of program
interface	string	Network interface to capture packets
count	integer	Number of packets to capture
protocol	string	Protocol to filter for (ex. ARP, UDP...)

### Settings

Purpose: To hold the settings the program needs to run.

Field	Type	Description
global_packet_limit	integer	Max number of packets to capture
capture_filter	string	Optional filter for packet capture

## Context

Purpose: To hold the arguments, settings, and exit information.

Field	Type	Description
arguments	argument	The raw command-line argument passed to the program
settings	settings	Parsed & validated arguments used by program to manage packet capture
exit_code	integer	Exit code returned by program
exit_message	string	Error or message to display before exiting

## Functions

Field	Description
display_hex_dump	Print raw hexadecimal dump of packet
packet_callback	Callback for each captured packet
parse_ethernet_header	Parse Ethernet frame from capture packet
parse_IPV4	Parse IPV4 header from packet
parse_IPV6	Parse IPV6 header from packet
parse_arp_header	Parse ARP packet header
parse_udp_header	Parse UDP packet header
parse_tcp_header	Parse TCP packet header
parse_icmp_header	Parse ICMP packet header
parse_icmpv6	Parse ICMPv6 packet header

_header	
parse_dns	Parses DNS packet header
interface_is_loopback	Check if interface is loopback
has_global_ip	Check if interface has IPV4 or IPV6 address
capture_packets	Captures packets using Scapy
capture_on_all_interfaces	Iterates over all interfaces

### States

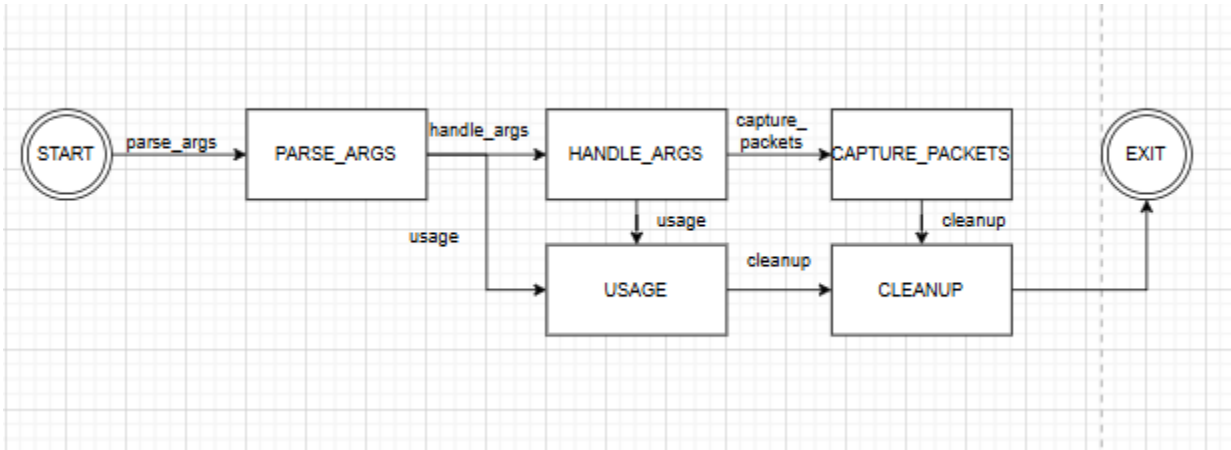
State	Description
PARSE_ARGS	Parse command line arguments
HANDLE_ARGS	Validate command line arguments
USAGE	Display error message if invalid arguments
CAPTURE_PACKETS	Capture packets on specified interface
CLEANUP	Perform cleanup and exit program

### State Table

From State	To State	Function / Action
START	PARSE_ARGS	Parse command line arguments
PARSE_ARGS	HANDLE_ARGS	Validate command line arguments
PARSE_ARGS	USAGE	Detect invalid arguments
HANDLE_ARGS	CAPTURE_PACKETS	Begin packet capture & parsing
HANDLE_ARGS	USAGE	Invalid arguments detected

USAGE	CLEANUP	Display error message and exit
CAPTURE_PACKETS	CLEANUP	Finished packet capture
CLEANUP	EXIT	Exit program

State Transition Diagram



Pseudocode

Display\_hex\_dump

Parameters

Parameter	Type	Description
hex_data	String	Packet data in hexadecimal format

Return

Value	Reason
None	Prints hex dump of packet

## Pseudo Code

set bytes\_line to 16

for i from 0 to length of hex\_data in steps of bytes\_line\*2

    set offset to i / 2

    print offset in hex format

    set hex\_line to empty string

    for j from i to min(i + bytes\_line\*2, length of hex\_data) in steps of 2

        append hex\_data[j:j+2] + " " to hex\_line

    print hex\_line

## Packet\_callback

### Parameters

Parameter	Type	Description
packet	Packet	Captures packet object

### Return

Value	Reason
None	Updates global packet counter & print packet info

## Pseudo Code

acquire counter\_lock

if packet\_counter < global\_packet\_limit

    increment packet\_counter

    print "Captured Packet" + packet\_counter

    convert packet to bytes -> raw\_data

    convert raw\_data to hex -> hex\_data

    call display\_hex\_dump(hex\_data)

    call parse\_ethernet\_header(hex\_data) -> ether\_type, payload

if packet\_counter >= global\_packet\_limit



```
    set stop_event
release counter_lock
```

## Interface\_is\_loopback

### Parameters

Parameter	Type	Description
interface	String	Name of network interface

### Return

Value	Reason
True	Interface is a loopback
False	Interface not a loopback

### Pseudo Code

```
get network addresses -> addrs
if interface in addrs
    for each addr in addrs[interface]
        if addr.family is IPv4 or IPv6 and addr.address is 127.0.0.1 or ::1
            return True
return False
```

## has\_global\_ip

### Parameters

Parameter	Type	Description
interface	String	Name of network interface

## Return

Value	Reason
True	Interface has global IP
False	Interface has no global IP

## Pseudo Code

```
get network addresses -> addrs
if interface in addrs
    for each addr in addrs[interface]
        if addr.family is IPv4 and not startswith "169.254"
            return True
        if addr.family is IPv6 and not startswith "fe80"
            return True
return False
```

## Capture\_packets

### Parameters

Parameter	Type	Description
interface	String	Interface to capture packets on
capture_filter	String	Filter to apply

## Return

Value	Reason
None	Capture packets & calls packet_callback for each

## Pseudo Code

```
print "Starting packet capture on interface with filter"
```

```
try
```

```
    create AsyncSniffer with interface, filter, prn=packet_callback
```

```
    start sniffer
```

```
    while not stop_event set
```

```
        pass
```

```
    if sniffer is running
```

```
        stop sniffer
```

```
catch KeyboardInterrupt
```

```
    print "Packet capture stopped"
```

```
catch Exception e
```

```
    print error
```

```
else
```

```
    print "Packet capture completed"
```

## capture\_on\_all\_interfaces

### Parameters

Parameter	Type	Description
capture_filter	String	Filter to apply
packet_count	Integer	Number of packets to capture

### Return

Value	Reason
None	Captures packets on all valid interfaces

## Pseudo Code

```
set global_packet_limit to packet_count
```

```
get all interfaces -> interfaces
```

```
print available interfaces
```

```
for each interface in interfaces
    if interface_is_loopback(interface) continue
    if not has_global_ip(interface) continue
    start new Thread to capture_packets(interface, capture_filter)
    add thread to threads list
```

```
try
    for each thread in threads
        join thread
catch KeyboardInterrupt
    print "Packet capture interrupted"
    set stop_event
    for each thread in threads
        join thread
```

**\_\_main\_\_**

## Parameters

Parameter	Type	Description
-----------	------	-------------

## Return

Value	Reason
HANDLE_ARGS	Arguments parsed & handled successfully

## Pseudo Code

```
initialize ArgumentParser
add argument -i/--interface required
add argument -f/--filter optional
add argument -c/--count required integer

parse arguments -> args

if args.interface is "any"
    call capture_on_all_interfaces(args.filter, args.count)
```

```

else
    if has_global_ip(args.interface)
        try
            call capture_packets(args.interface, args.filter)
        catch Exception e
            print error
    else
        print "Interface does not have global IP"

print hex_line

```

## parse\_ethernet\_header

### Parameters

Parameter	Type	Description
hex_data	String	Ethernet frame in hexadecimal

### Return

Value	Reason
Ether_type, payload	EtherType of frame & payload data

### Pseudo Code

```

extract dest_mac from hex_data[0:12]
extract source_mac from hex_data[12:24]
extract ether_type from hex_data[24:28]
print Ethernet header info

```

```

set payload to hex_data[28:]

```

```

if ether_type == "0806" # ARP
    call parse_arp_header(payload)
elif ether_type == "0800" # IPv4
    get protocol from payload[18:20]

```

```

call parse_IPV4(payload)
if protocol == 6
    call parse_tcp_header(payload)
elif protocol == 17
    call parse_udp_header(payload)
elif protocol == 1
    call parse_icmp_header(payload)
elif ether_type == "86dd" # IPv6
    call parse_IPV6(payload)
else
    print unknown EtherType message

return ether_type, payload

```

## parse\_IPV4

### Parameters

Parameter	Type	Description
hex_data	String	IPV4 packet data in hex

### Return

Value	Reason
None	Prints parsed IPV4 header fields

### Pseudo Code

```

extract version and header_length from hex_data[0:2]
extract total_length from hex_data[4:8]
extract flags and fragment offset from hex_data[12:16]
extract protocol from hex_data[18:20]
extract source_ip from hex_data[24:32]
extract dest_ip from hex_data[32:40]

```

```

print IPv4 header fields

```

## parse\_IPV6

### Parameters

Parameter	Type	Description
hex_data	String	IPV6 packet data in hex

### Return

Value	Reason
None	Prints parsed IPV6 header fields

### Pseudo Code

```
extract first_word from hex_data[0:8]
extract version, traffic_class, flow_label from first_word
extract payload_length, next_header, hop_limit from hex_data[8:16]
extract source_ip from hex_data[16:48]
extract dest_ip from hex_data[48:80]
```

```
print IPv6 header fields
```

```
set transport_payload to hex_data[80:]
if next_header == 6
    call parse_tcp_header(transport_payload)
elif next_header == 17
    call parse_udp_header(transport_payload)
elif next_header == 58
    call parse_icmpv6_header(transport_payload)
```

## parse\_arp\_header

### Parameters

Parameter	Type	Description
hex_data	String	ARP packet data in hex

### Return

Value	Reason
None	Prints ARP header fields

### Pseudo Code

extract hardware\_type, protocol\_type, hardware\_size, protocol\_size, opcode from hex\_data

extract sender\_mac and sender\_ip from hex\_data

extract target\_mac and target\_ip from hex\_data

print ARP header fields

## parse\_udp\_header

### Parameters

Parameter	Type	Description
hex_data	String	UDP packet data in hex

### Return

Value	Reason
None	Prints UDP header fields and parses DNS as well if it contains it



## Pseudo Code

extract source\_port, dest\_port, length, checksum from hex\_data  
set payload to hex\_data[56:]

print UDP header fields

if source\_port == 53 or dest\_port == 53  
    call parse\_dns(payload)

## parse\_tcp\_header

### Parameters

Parameter	Type	Description
hex_data	String	TCP packet data in hex

### Return

Value	Reason
None	Prints TCP header fields and parses DNS as well if it contains it

## Pseudo Code

extract source\_port, dest\_port, seq\_number, ack\_number from hex\_data  
extract tcp\_byte and tcp\_flags  
extract data\_offset, reserved\_flag, NS/CWR/ECE/URG/ACK/PSH/RST/SYN/FIN flags  
extract window\_size, checksum, urgent\_pointer  
compute payload\_index = 40 + data\_offset\*8  
set payload = hex\_data[payload\_index:payload\_index+64]

print TCP header fields

if source\_port == 53 or dest\_port == 53  
    call parse\_dns(payload)

## parse\_icmp\_header

### Parameters

Parameter	Type	Description
hex_data	String	ICMP packet data in hex

### Return

Value	Reason
None	Prints ICMP header fields

### Pseudo Code

```
extract icmp_type, icmp_code, icmp_checksum from hex_data  
set payload = hex_data[48:]
```

```
print ICMP header fields
```

## parse\_icmpv6\_header

### Parameters

Parameter	Type	Description
hex_data	String	ICMPv6 packet data in hex

### Return

Value	Reason
None	Prints ICMPv6 header fields

## Pseudo Code

extract icmp\_type, icmp\_code, checksum from hex\_data

print ICMPv6 header fields

## parse\_dns

### Parameters

Parameter	Type	Description
hex_data	String	DNS payload data in hex

### Return

Value	Reason
None	Prints DNS header fields

## Pseudo Code

if length of hex\_data < 24

    print "DNS payload too short"

    return

extract transaction\_id, flags from hex\_data

extract qdcount, anccount, nscount, arcount from hex\_data

print DNS header fields

