Assignment 2

Design

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

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>
  (More 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_messag e	string	Error or message to display before exiting

Functions

Field	Description
display_hex_d ump	Print raw hexadecimal dump of packet
packet_callbac k	Callpack for each captured packet
parse_etherne t_header	Parse Ethernet frame from capture packet
parse_IPV4	Parse IPV4 header from packet
parse_IPV6	Parse IPV6 header from packet
parse_arp_hea der	Parse ARP packet header
parse_udp_he ader	Parse UDP packet header
parse_tcp_hea der	Parse TCP packet header
parse_icmp_h eader	Parse ICMP packet header
parse_icmpv6 _header	Parse ICMPv6 packet 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

These states conceptually represent the states the program can be in.

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

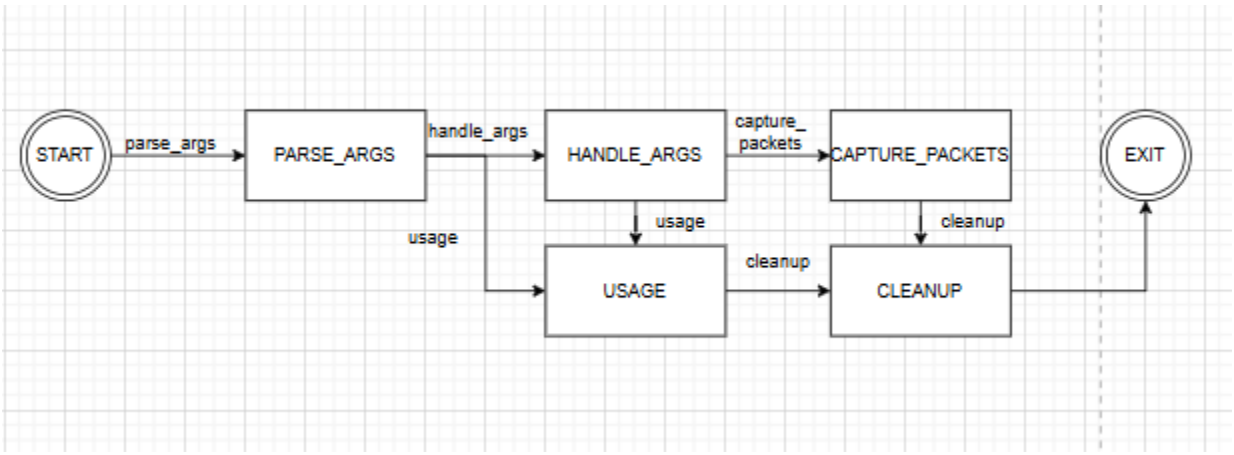
From State	To State	Function / Action
START	PARSE_ARGS	Parse command line arguments
PARSE_ARGS	HANDLE_ARGS	Validate command line arguments
PARSE_ARGS	USAGE	Detect invalid arguments
HANDLE_ARGS	CAPTURE_PACKETS	Begin packet capture & parsing
HANDLE_ARGS	USAGE	Invalid arguments detected
USAGE	CLEANUP	Display error message and exit
CAPTURE_P	CLEANUP	Finished packet capture

ACKETS		
CLEANUP	EXIT	Exit program

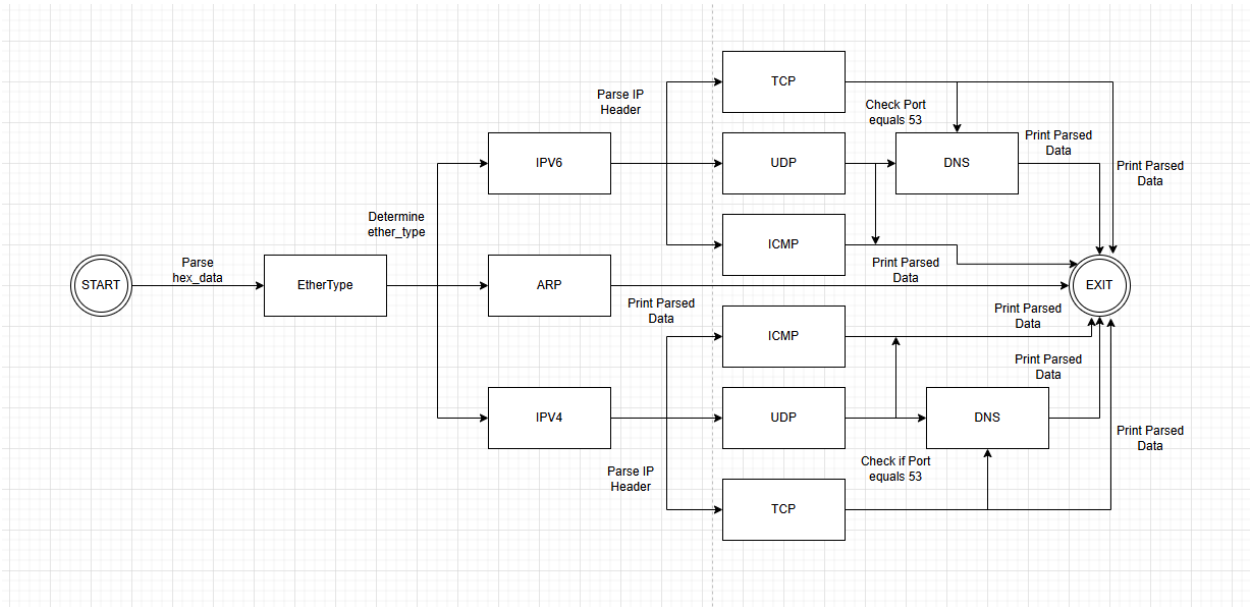
State Transition Diagram

These state transition diagrams conceptually represent the flow of states within the program (main.py & packet_parsers.py)

main .py



packet_parsers.py



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, ancount, nscount, arcount from hex_data

print DNS header fields
```