COMP 7003
Introduction to Information and Network Security

Assignment-02

Design

Anmol Mittal
A01397754
February 2nd, 2025
Course Reference Number (CRN): 91662

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, IPv6, ICMPv6), 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.
parse_ipv6_header	Extracts and displays IPv6 header details.

parse_icmp_v6_header	Extracts and displays ICMPv6 header details.
parse_dns_header	Extracts and displays DNS 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

Packet Sniffer (main.py)

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:

```
  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:
  DEFINE allowed_filters ← ["tcp", "icmp", "udp", "arp", "ip", "ip6", "icmp6", "dns"]

  IF filter is not empty THEN:
    SET filter_lower ← lowercase(filter)
    IF filter_lower not in allowed_filters THEN:
      DISPLAY ERROR
      EXIT

    IF filter_lower = "dns" THEN:
      SET filter ← "udp port 53 or tcp port 53"

prompt_for_filter_if_needed:
  IF filter is empty THEN:
    DISPLAY prompt for filter input (tcp, icmp, udp, arp, ip, ip6, icmp6, dns, or all)
    SET user_input

    IF user_input in allowed_filters THEN:
      SET filter ← user_input
    ELSE
      DISPLAY "Error: Invalid filter '{user_input}'. Allowed filters: {allowed_filters}"
      EXIT

validate_interface:
Professor Provided Function
```

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: "
  DISPLAY “Source Address: "
  DISPLAY “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 IF ether_type = "08dd" THEN:
    CALL parse_ipv6_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: "
  DISPLAY “Protocol Type: "
  DISPLAY “Hardware Size: "
  DISPLAY “Protocol Size: "
  DISPLAY “Operation: "
  DISPLAY “Sender MAC: "
  DISPLAY “Sender IP: "
  DISPLAY “Target MAC: "
  DISPLAY “Target IP: "
```

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: "
  DISPLAY “Header Length: "
  DISPLAY "Total Length: "
  DISPLAY "Identification: "
  DISPLAY "Flags & Fragment Offset: "
  DISPLAY "Reserved Bit: "
  DISPLAY "DF (Do Not Fragment): "
  DISPLAY "MF (More Fragments): "
  DISPLAY "Fragment Offset: "
  DISPLAY "Protocol: "
  DISPLAY "Source IP: "
  DISPLAY "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: "
  DISPLAY "Destination Port: "
  DISPLAY "Length: "
  DISPLAY "Checksum: "

  IF source_port = 53 OR destination_port = 53 THEN:
    CALL parse_dns_header and pass payload
  ELSE:
     DISPLAY 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: "
DISPLAY "Destination Port: "
DISPLAY "Sequence Number: "
DISPLAY "Acknowledgment Number: "
DISPLAY "Data Offset: "
DISPLAY "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: "
DISPLAY "Checksum: "
DISPLAY "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: "
  DISPLAY "Code: "
  DISPLAY "Checksum: "

  IF payload is not empty:
    DISPLAY "ICMP Payload: " payload
  ELSE:
    DISPLAY "No ICMP Payload"

  RETURN icmp_type, code, checksum, payload
```

parse_ipv6_header:

Parameters

Parameter	Type	Description
hex_data	String	The raw packet data in hexadecimal format.

Return

Value	Reason
None	Displays parsed IPv6 header details.

Pseudo Code

```
parse_ipv6_header:
  version ← Convert first character of hex_data to integer (base 16)
  traffic_class ← Convert next 2 characters of hex_data to integer (base 16)
  flow_label ← Convert next 5 characters of hex_data to integer (base 16)
  payload_length ← Convert next 4 characters of hex_data to integer (base 16)
  next_header ← Convert next 2 characters of hex_data to integer (base 16)
  hop_limit ← Convert next 2 characters of hex_data to integer (base 16)
  source_hex ← Extract next 32 characters from hex_data
  dest_hex ← Extract next 32 characters from hex_data
  source_ip ← Format source_hex into colon-separated IPv6 address
  dest_ip ← Format dest_hex into colon-separated IPv6 address

  DISPLAY "IPv6 Header:"
  DISPLAY "Version: "
  DISPLAY "Traffic Class: "
```

```
  DISPLAY "Flow Label: "
  DISPLAY "Payload Length: "
  DISPLAY "Next Header: "
  DISPLAY "Hop Limit: "
  DISPLAY "Source IP: "
  DISPLAY "Destination IP: "

  payload ← Extract remaining data from hex_data

  IF next_header = 6 THEN:
    CALL parse_tcp_header and pass payload
  ELSE IF next_header = 17 THEN:
    CALL parse_udp_header and pass payload
  ELSE IF next_header = 58 THEN:
    CALL parse_icmp_v6_header and pass payload
  ELSE:
    ERROR
```

parse_icmp_v6_header:

Parameters

Parameter	Type	Description
hex_data	String	The raw packet data in hexadecimal format.

Return

Value	Reason
None	Displays parsed ICMPv6 header details.

Pseudo Code

```
parse_icmp_v6_header:
  icmp_v6_type ← Convert first 2 characters of hex_data to integer (base 16)
  icmp_v6_code ← Convert next 2 characters of hex_data to integer (base 16)
  icmp_v6_checksum ← Convert next 4 characters of hex_data to integer (base 16)
  payload ← Extract remaining data from hex_data

  DISPLAY "ICMPv6 Header:"
  DISPLAY "Type: "
  DISPLAY "Code: "
  DISPLAY "Checksum: "
  DISPLAY "ICMPv6 Payload"
```

parse_dns_header:

Parameters

Parameter	Type	Description
hex_data	String	The raw packet data in hexadecimal format.

Return

Value	Reason
None	Displays parsed DNS header details.

Pseudo Code

```
parse_dns_header:
  dns_id ← Convert first 4 characters of hex_data to integer (base 16)
  dns_flags ← Convert next 4 characters of hex_data to integer (base 16)
  dns_question_count ← Convert next 4 characters of hex_data to integer (base 16)
  dns_answer_count ← Convert next 4 characters of hex_data to integer (base 16)
  dns_rr_count ← Convert next 4 characters of hex_data to integer (base 16)
  dns_additional_rr_count ← Convert next 4 characters of hex_data to integer (base 16)

  DISPLAY "DNS Header:"
  DISPLAY "Transaction ID: "
  DISPLAY "Flags: "
  DISPLAY "Questions: "
  DISPLAY "Answer RRs: "
  DISPLAY "Authority RRs: "
  DISPLAY "Additional RRs: "
```