COMP 7003
Assignment 2

Design

Nicky Cheng
A01269051
Oct 5th, 2024

Purpose

- This program accepts 3 arguments from the command line:
    - <interface>
    - <capture_filter>
    - <packet_count>
- The program takes <interface>, <capture_filter>, and <packet_count> and sends them to a function. This function uses all the information to packet capture as many packets as <packet_count>. The packets that get captured are then parsed by the program. First, the program detects the port that the packet is in through <capture_filter>. Then the program detects the size of the interface's header to see where the packet header field is. After finding the packer header field, the program parses the field and prints the information on all fields.

Data Types

Arguments

Purpose: To hold the unparsed command-line argument information

Field	Type	Description
argc	integer	The number of arguments
argv	string[]	The arguments
program_name	string	The name of the program
interface	string	The name of the interface the program will read from
capture_filter	string	The user-defined BPF filter
packet_count	int	The amount of packets to catch

Settings

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

Field	Type	Description
interface	string	The name of the interface the program will read from
capture_filter	string	The user-defined BPF filter

packet_count	int	The amount of packets to catch

Context

Purpose: To hold the arguments, settings, packet, and hex data information

Field	Type	Description
arguments	Arguments	The command line arguments
settings	Settings	The parsed command line arguments
packet	Packet	The caught packet
hex_data	String	A string of hex data taken from the packet

Functions

Function	Description
capture_packets	Capture packets based on the provided interface and BPF filter
packet_callback	Converts packet into hex data
parse_header	Detects the packet type
parse_arp_header	Parses the hex data from the ARP header
parse_ipv4_header	Parses the hex data from the IPv4 header
parse_udp_header	Parses the hex data from the UDP header
parse_tcp_header	Parses the hex data from the TCP header

States

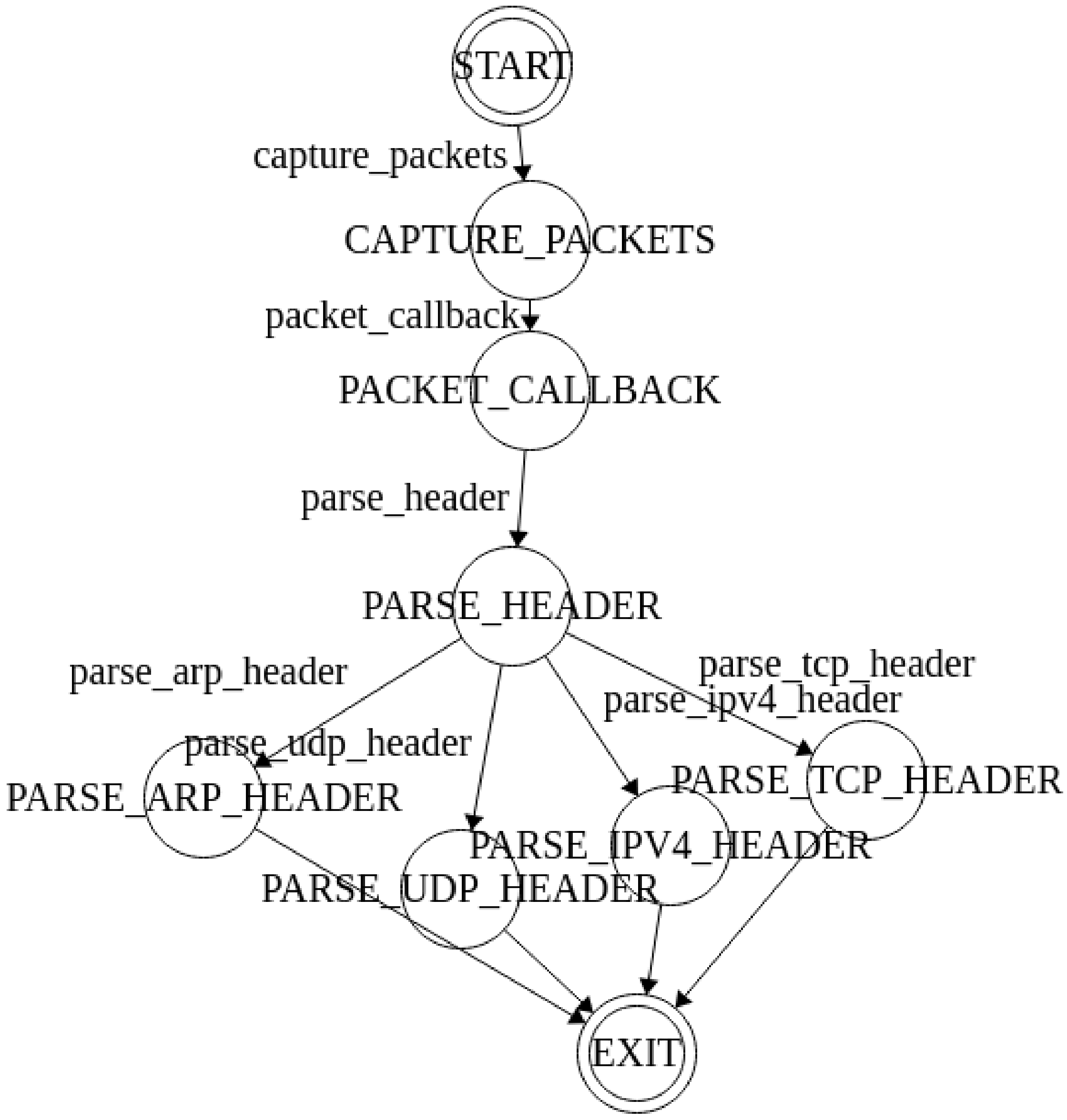
State	Description
CAPTURE_PACKETS	Capture packets based on the provided interface and BPF filter
PACKET_CALLBACK	Converts packet into hex data
PARSE_HEADER	Detects the packet type
PARSE_ARP_HEADER	Parses the hex data from the ARP header

PARSE_IPV4_HEADER	Parses the hex data from the IPv4 header
PARSE_UDP_HEADER	Parses the hex data from the UDP header
PARSE_TCP_HEADER	Parses the hex data from the TCP header

State Table

From State	To State	Function
START	CAPTURE_PACKETS	capture_packets
CAPTURE_PACKETS	PACKET_CALLBACK	packet_callback
PACKET_CALLBACK	PARSE_HEADER	parse_header
PARSE_HEADER	PARSE_ARP_HEADER	parse_arp_header
PARSE_HEADER	PARSE_IPV4_HEADER	parse_ipv4_header
PARSE_HEADER	PARSE_UDP_HEADER	parse_udp_header
PARSE_HEADER	PARSE_TCP_HEADER	parse_tcp_header
PARSE_ARP_HEADER	EXIT	
PARSE_UDP_HEADER	EXIT	
PARSE_TCP_HEADER	EXIT	
PARSE_IPV4_HEADER	EXIT	

State Transition Diagram



Pseudocode

capture_packets

Parameters

Parameter	Type	Description
ctx	Context	The program context

Return

Value	Reason
PACKET_CALLBACK	The packet has been successfully captured

Pseudo Code

```
Store ctx.argv[1] as ctx.settings.interface
Store ctx.argv[2] as ctx.settings.capture_filter
Store ctx.argv[3] as ctx.settings.packet_count

Print indication of packet capturing
packet_capture(ctx.argv[1], ctx.argv[2], packet_callback(ctx.packet),
ctx.argv[3])

return PACKET_CALLBACK
```

packet_callback

Parameters

Parameter	Type	Description
ctx	Context	The program context

Return

Value	Reason
PARSE_HEADER	The args are correct

Pseudo Code

```
Store raw_data as ctx.packet in bytes
Convert raw_data into hexadecimal and store into ctx.hex_data
Print ctx.hex_data
return PARSE_HEADER
```

parse_header

Parameters

Parameter	Type	Description
ctx	Context	The program context

Return

Value	Reason
PARSE_ARP_HEADER	Filter has “arp”
PARSE_IPV4_HEADER	Filter has “ipv4”
PARSE_UDP_HEADER	Filter has “udp”
PARSE_TCP_HEADER	Filter has “tcp”

Pseudo Code

```
Split ctx.settings.capture_filter into list
Loop through each word in filter list
     If word = “arp”
          return PARSE_ARP_HEADER
     If word = “ip”
          return PARSE_IPV4_HEADER
     If word = “udp”
          return PARSE_UDP_HEADER
     If word = “tcp”
          return PARSE_TCP_HEADER
```

parse_arp_header

Parameters

Parameter	Type	Description
ctx	Context	The program context

Return

- EXIT

Pseudo Code

```
Split ctx.hex_data numbers 29-32 to hardware_type
Split ctx.hex_data numbers 33-36 to protocol_type
Split ctx.hex_data numbers 37-38 to hardware_size
Split ctx.hex_data numbers 39-40 to protocol_size
Split ctx.hex_data numbers 41-44 to opcode
Split ctx.hex_data numbers 45-56 to sender_mac_address
Split ctx.hex_data numbers 57-64 to sender_ip_address
Split ctx.hex_data numbers 65-76 to target_mac_address
Split ctx.hex_data numbers 77-84 to target_ip_address

Convert hardware_type into decimal
Convert hardware_size into decimal
Convert protocol_size into decimal
Convert opcode into decimal
Store sender_mac_address into sender_mac_address_readable by
separating each pair of hex numbers by colons
Convert sender_ip_address into decimal and store into
sender_ip_address_readable by adding periods after each decimal
number
Store target_mac_address into target_mac_address_readable by
separating each pair of hex numbers by colons
Convert target_ip_address into decimal and store into
target_ip_address_readable by adding periods after each decimal
number

Print all fields

return EXIT
```

parse_ipv4_header

Parameters

Parameter	Type	Description
ctx	Context	The program context

Return

- EXIT

Pseudo Code

```
Split ctx.hex_data number 29 to version
Split ctx.hex_data number 30 to internet_header_length
Split ctx.hex_data numbers 31-32 to type_of_service
Split ctx.hex_data numbers 33-36 to total_length
Split ctx.hex_data numbers 37-40 to identification
Split ctx.hex_data numbers 41-42 to flags
Split ctx.hex_data numbers 41-44 to fragment_offset
Split ctx.hex_data numbers 45-46 to time_to_live
Split ctx.hex_data numbers 47-48 to protocol
Split ctx.hex_data numbers 49-52 to checksum
Split ctx.hex_data numbers 53-60 to source_address
Split ctx.hex_data numbers 61-68 to destination_address

Convert version into decimal
Convert internet_header_length into decimal
Convert total_length into decimal
Convert flags into binary
Store 2nd bit in flags as dont_fragment
Store 3rd bit in flags as more_fragments
Convert fragment_offset into binary and store all bits after 3rd bit
into fragment_offset
Convert time_to_live into decimal
Convert protocol into decimal
Convert source_address into decimal and store into
source_address_readable by adding periods after each decimal number
Convert destination_address into decimal and store into
destination_address_readable by adding periods after each decimal
number

Print all fields
```

```
return EXIT
```

parse_udp_header

Parameters

Parameter	Type	Description
ctx	Context	The program context

Return

- EXIT

Pseudo Code

```
Split ctx.hex_data numbers 69-72 into source_port
Split ctx.hex_data numbers 73-76 into destination_port
Split ctx.hex_data numbers 77-80 into length
Split ctx.hex_data numbers 81-84 into checksum

Convert source_port into decimal
Convert destination_port into decimal
Convert length into decimal

Print all fields

return EXIT
```

parse_tcp_header

Parameters

Parameter	Type	Description
ctx	Context	The program context

Return

- EXIT

Pseudo Code

```
Split ctx.hex_data numbers 69-72 into source_port
Split ctx.hex_data numbers 73-76 into destination_port
Split ctx.hex_data numbers 77-84 into sequence_number
Split ctx.hex_data numbers 85-92 into acknowledgement_number
Split ctx.hex_data number 93 into data_offset
Split ctx.hex_data numbers 94-96 into flags
Split ctx.hex_data numbers 97-100 into window_size
Split ctx.hex_data numbers 101-104 into checksum
Split ctx.hex_data numbers 105-108 into urgent_pointer

Convert source_port into decimal
Convert destination_port into decimal
Convert sequence_number into decimal
Convert acknowledgement_number into decimal
Convert data_offset into decimal
Convert flags into binary
Store 4th bit of flags into accurate_ecn
Store 5th bit of flags into congestion_window_reduced
Store 6th bit of flags into ecn_echo
Store 7th bit of flags into urgent
Store 8th bit of flags into acknowledgement
Store 9th bit of flags into push
Store 10th bit of flags into reset
Store 11th bit of flags into syn
Store 12th bit of flags into fin
Convert window size into decimal
Convert urgent_pointer into decimal

Print all fields

return EXIT
```