COMP 7003
Assignment 2
Design

Andy Tran
A01266629
Oct 2nd, 2024

Purpose

- This program captures and analyzes network packets similar to WireShark. Allows user to have custom BPF to specify a network interface , filter to capture(TCP, UDP, ARP and IP) and number of packets to capture up to 100.
    - <interface>
    - <filter>
    - <packetcount>
- Packet will be processed and displayed in Hex, Decimal and Binary.

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
interface	string	Network interface to capture packets on
filter	string	The capture filter (ARP, UDP, TCP, IP)
count	integer	Number of packets up to 100

Settings

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

Field	Type	Description
count	unsigned integer	The number of times to display the message
filter	string	The type of packet filter used (TCP, UDP, ARP, IP)

Context

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

Field	Type	Description

Functions

Function	Description
validate_arguments	Verifies that no unknown, duplicate, or excess arguments are present
validate_interface	Verifies that the network interface exists and is active
validate_filter	Verifies that the provided filter is valid (e.g., TCP, UDP, ARP)
validate_count	Verifies that the packet count is valid and within the acceptable range
capture_packets	Initiates the packet capture on the specified interface and filter
parse_ethernet_header	Parses and displays the Ethernet header
parse_arp_header	Parses and displays the ARP header
parse_ipv4_header	Parses and displays the IPv4 header
parse_tcp_header	Parses and displays the TCP header
parse_udp_header	Parses and displays the UDP header
packet_callback	Handles each captured packet, determines its type, and processes it
check_privledges	Checks privledges and handles no permission

Pseudocode

validate_arguments

Parameters

Parameter	Type	Description
expected_args	list	The expected argument flags

Return

Value	Reason
none	none

Pseudo Code

validate_Arguments(expected_args)

```
set known_args to expected_args

If sys.argv is more than expected argos
     Print “Error too many argos”

For each arg in sys.argv
     If ! start w -- or exist in known_Args
          Print “ error unrecognized arg”
```

validate_interface

Parameters

Parameter	Type	Description
interface	string	Network interface to validate

Return

Value	Reason
none	none

Pseudo Code

validate_interface(interface)

```
If interface is empty
      Prnt error “network interface cant be empty”
      Exit
Try open file at "/sys/class/net/<interface>/operstate"
      Read content of file
      If state is not up
            Prnt “interface is not active or is down”
If file does not exist
      Prnt “interace does not exist”
      exit
```

validate_filter

Parameters

Parameter	Type	Description
filter	string	Filter to validate ex TCP ARP IP UDP

Return

Value	Reason
none	none

Pseudo Code

validate_filter(filter)

```
Set valid_filters to [TCP, UDP, IP, ARP]
if filter is not in valid_filters
     Print error invalid BPD filter
     Exit
```

validate_count

Parameters

Parameter	Type	Description
count	integer	Number of packets to capture hardcoded limit of 100

Return

Value	Reason
none	none

Pseudo Code

validate_count(count)

```
Set maxcount to 100

If count is less than or 0
     Print “error packet count must be positive non zero number”
     exit

If count is more than maxcount
     Print “ error packet exceeds limit of {maxcount}”
     exit
```

parse_ethernet _header

Parameters

Parameter	Type	Description
hex_Data	string	Raw packet data in hex

Return

Value	Reason
ether_type	The ethertype from the packet for future protocol processeisng

Pseudo Code

parse_etherent_header(hex_data)

```
Extract dest_mac from bytes 0 to 6
Extract source_mac from bytes 6 to 12
Extract ether_type from bytes 12 to 14

Convert dest_mac to readable format
Convert source_mac to readable format

Print dest_mac, source_mac, and ether_type

Return ether_type
```

parse_arp_header

Parameters

Parameter	Type	Description
hex_Data	string	Raw packet data in hex

Return

Value	Reason
none	none

Pseudo Code

```
parse_arp_header(hex_data)

If length of hex_data is less than 28 bytes:
     Print "error: ARP packet is too short"
     Return

Extract hardware_type from bytes 14 to 16
Extract protocol_type from bytes 16 to 18
Extract hardware_size from bytes 18 to 19
Extract protocol_size from bytes 19 to 20
Extract opcode from bytes 20 to 22

Extract sender_mac from bytes 22 to 28
Extract sender_ip from bytes 28 to 32
Extract target_mac from bytes 32 to 38
Extract target_ip from bytes 38 to 42

Convert sender_mac, target_mac, sender_ip, and target_ip to readable
format

Print ALL extracted fields in decimal and hex.
```

parse_ipv4_header

Parameters

Parameter	Type	Description
hex_Data	string	Raw packet data in hex

Return

Value	Reason
none	none

Pseudo Code

```
parse_ipv4_header(hex_data)

If length of hex_data is less than 20 bytes:
     Print "Error: IPv4 packet is too short"
     Return

Extract version_ihl from bytes 14 to 15
Set version to the first 4 bits of version_ihl
Set ihl to the last 4 bits of version_ihl using bitmask

Extract tos from bytes 15 to 16
Extract total_length from bytes 16 to 18
Extract identification from bytes 18 to 20
Extract flags_fragment from bytes 20 to 22
Extract ttl from bytes 22 to 23
Extract protocol from bytes 23 to 24
Extract header_checksum from bytes 24 to 26

Extract src_ip from bytes 26 to 30
Extract dst_ip from bytes 30 to 34

Set flags to the first 3 bits of flags_fragment
Set fragment_offset to the last 13 bits of flags_fragment

Print all extracted fields in decimal and hex. Binary for flags
```

parse_tcp_header

Parameters

Parameter	Type	Description
hex_Data	string	Raw packet data in hex

Return

Value	Reason
none	none

Pseudo Code

```
parse_tcp_header(hex_data)

If length of hex_data is less than 20 bytes:
     Print "Error: TCP packet is too short"
     Return

Extract src_port from bytes 34 to 36
Extract dst_port from bytes 36 to 38
Extract sequence_number from bytes 38 to 42
Extract ack_number from bytes 42 to 46
Extract flags from bytes 46 to 47

Print all extracted fields in decimal and hex. Binary for flags
```

parse_udp_header

Parameters

Parameter	Type	Description
hex_Data	string	Raw packet data in hex

Return

Value	Reason
none	none

Pseudo Code

```
If length of hex_data is less than 8 bytes:
     Print "Error: UDP packet is too short"
     Return

Extract src_port from bytes 34 to 36
Extract dst_port from bytes 36 to 38
Extract length from bytes 38 to 40
Extract checksum from bytes 40 to 42

Print all extracted fields in decimal and hex. Binary for flags
```

packet_callback

Parameters

Parameter	Type	Description
packet	bytes	Captured packet

Return

Value	Reason
none	none

Pseudo Code

```
packet_callback(packet)

If packet is empty:
    Print "Error: Received an empty or null packet"
    Return

Convert packet to hex format (hex_data)

Call parse_ethernet_header(hex_data) to get ether_type

If ether_type is equal to '0806':
    Call parse_arp_header(hex_data)

Else if ether_type is equal to '0800':
    Call parse_ipv4_header(hex_data)
    Extract protocol from hex_data (bytes 23 to 24)
    If protocol is equal to '06':
        Call parse_tcp_header(hex_data)
    Else if protocol is equal to '11':
        Call parse_udp_header(hex_data)

Else:
    Print "Error: Unsupported EtherType"
```

capture_packets

Parameters

Parameter	Type	Description
interface	string	Network interface to capture from
capture_filter	string	BPF filter used to capture packets
packet_count	integer	Number of packets to capture

Return

Value	Reason
none	none

Pseudo Code

```
capture_packets(interface, capture_filter, packet_count)

Print "Starting packet capture on interface with filter"

Call sniff with:

iface set to interface
filter set to capture_filter
prn set to packet_callback
count set to packet_count
```

check_privileges

Parameters

Parameter	Type	Description

Return

Value	Reason
none	none

Pseudo Code

```
check_privileges()

if os.geteuid() != 0:
     print("Error: This program requires root privileges to capture
packets. Please run with 'sudo'.")
     exit
```