COMP 7003
Assignment 2

Design

Towa Quimbayo
A01086002
Feb 3rd, 2025

Purpose

This program accepts 3 arguments from the command line:

- -i <inteface>: The network interface to capture packets on.
- -c <count>: Number of packets to capture.
- -f <filter> (optional): The BPF filter to apply.

If the <interface> has the value `any` then it will initially print the list of available network interfaces. Then, it would start to capture the packet information and print the Ethernet Header and the selected <filter> header information to the console for <count> times.

Global Variables

The program utilizes several global variables to coordinate packet capture across threads and to manage the overall packet processing flow. These variables are defined at the beginning of the `main.py` file and play a significant role as described below.

Field	Type	Description
packet_counter	Integer	The number of packets that have been captured.
counter_lock	threading.Lock	A threading lock that synchronizes access to the `packet_counter` variable.
stop_event	threading.Event	A threading event flag that signals when packet capture should stop.
global_packet_limit	Integer	The maximum number of packets to capture.

Functions

Function	Description
packet_callback	Callback function to process each captured packet.
interface_is_loopback	Checks if the specified interface is a loopback interface (i.e., 'lo', 'lo0') based on the IP address assigned to it.
has_global_ip	Display a usage message when the command line argument has an issue
capture_packets	Capture packets on a specific interface using the AsyncSniffer from Scapy and a packet callback function to process each packet.
capture_on_all_interf	Capture packets on all available interfaces with global IP addresses

aces	by starting a separate capture thread for each interface.
parse_ethernet_header	Parses the Ethernet header from a hex string of data and prints the results and routes the payload to the corresponding parser based on the EtherType field (ARP or IPv4).
parse_arp_header	Parses the ARP header from a hex string of data and prints the results.
parse_ipv4_header	Parses the IPv4 header from a hex string of data and prints the results and routes the payload to the corresponding parser based on the Protocol field (UDP, TCP, ICMP).
parse_icmp_header	Parses the ICMP header from a hex string of data and prints the results.
parse_tcp_header	Parses the TCP header from a hex string of data and prints the results.
parse_udp_header	Parses the UDP header from a hex string of data and prints the results.
parse_dns_header	Parses the DNS header from a hex string of data and prints the results.

Pseudocode

packet_callback

Parameters

Parameter	Type	Description
packet	scapy.Packet	The captured packet to process.

Return

Type	Reason
None	No value is to be returned.

Pseudo Code

```
FUNCTION packet_callback(packet):
acquire the counter_lock
if packet_counter is less than global_packet_limit
```

```
     increment packet_counter by 1
     call parse_ethernet_header with packet data in hex format

if packet_counter is greater than or equal to global_packet_limit
     stop the packet capture
release the counter_lock
```

interface_is_loopback

Parameters

Parameter	Type	Description
interface	String	he interface name to check.

Return

Type	Reason
boolean	True if the interface is a loopback interface, False otherwise.

Pseudo Code

```
FUNCTION interface_is_loopback(interface):
get the network interfaces and their addresses
if the interface is in the network interfaces
     loop through the addresses of the interface
          if the address family is IPv4 or IPv6 and the address
          is equal to "127.0.0.1" or "::1"
               return True
return False otherwise
```

has_global_ip

Parameters

Parameter	Type	Description
interface	String	The interface name to check.

Return

Type	Reason

boolean	True if the interface has a global IP address, False otherwise.

Pseudo Code

```
FUNCTION has_global_ip(interface):
get the network interfaces and their addresses
if the interface is in the network interfaces
     loop through the addresses of the interface
          if the address family is IPv4 and the address does not
     start
          with "169.254"
               return True
          if the address family is IPv6 and the address does not
          start with "fe80"
               return True
return False otherwise
```

capture_packets

Parameters

Parameter	Type	Description
interface	String	The interface to capture packets on.
capture_filter	String	The BPF filter to apply to the capture.

Return

Type	Reason
None	No value is to be returned.

Pseudo Code

```
FUNCTION capture_packets(interface, capture_filter):
Initialize the packet capture sniffer with the interface and filter
start the packet capture sniffer
loop until the stop_event is set
if the sniffer is still running
     stop the packet capture sniffer
```

capture_on_all_interfaces

Parameters

Parameter	Type	Description
capture_filter	String	The BPF filter to apply to each interface.
packet_count	Integer	The number of packets to capture on each interface.

Return

Type	Reason
None	No value is to be returned.

Pseudo Code

```
FUNCTION capture_on_all_interfaces(capture_filter, packet_count):
set the global packet limit to the specified packet count
get the list of available network interfaces
loop through the interfaces

if the interface is a loopback interface or does not have a global IP
address
     continue to the next interface

create a thread to capture packets on the interface
start the thread
loop through the threads and wait for each thread to complete
```

parse_ethernet_header

Parameters

Parameter	Type	Description
hex_data	String	The hex string of data to parse representing the Ethernet segment.

Return

Type	Reason
Tuple	A tuple containing the EtherType and the payload data.

Pseudo Code

```
FUNCTION parse_ethernet_header(hex_data):
parse the Ethernet header segments from the hex data including the
destination MAC, source MAC, and EtherType
print the parsed Ethernet header segments

route the payload based on the EtherType
if the EtherType is ARP
     call parse_arp_header with the payload
else if the EtherType is IPv4
     call parse_ipv4_header with the payload
else
     print "No parser available for this EtherType."

return the EtherType and the payload
```

parse_arp_header

Parameters

Parameter	Type	Description
hex_data	String	The hex string of data to parse representing the ARP segment.

Return

Type	Reason
None	No value is to be returned.

Pseudo Code

```
FUNCTION parse_arp_header(hex_data):
parse the ARP header segments from the hex data including the
hardware type, protocol type, hardware size, protocol size,
operation, sender MAC, sender IP, target MAC, and target IP
print the parsed ARP header segments
```

parse_ipv4_header

Parameters

Parameter	Type	Description

hex_data	String	The hex string of data to parse representing the IPv4 segment.

Return

Type	Reason
None	No value is to be returned.

Pseudo Code

```
FUNCTION parse_ipv4_header(hex_data):
parse the IPv4 header segments from the hex data including the
version, header length, total length, flags, fragment offset,
protocol, source IP, and destination IP
print the parsed IPv4 header segments

route the payload based on the Protocol
if the Protocol is ICMP
     call parse_icmp_header with the payload
else if the Protocol is TCP
     call parse_tcp_header with the payload
else if the Protocol is UDP
     call parse_udp_header with the payload
else
     print "No parser available for this Protocol."
```

parse_icmp_header

Parameters

Parameter	Type	Description
hex_data	String	The hex string of data to parse representing the ICMP segment.

Return

Type	Reason
None	No value is to be returned.

Pseudo Code

```
FUNCTION parse_icmp_header(hex_data):
parse the ICMP header segments from the hex data including the type,
code, checksum, and payload
```

```
print the parsed ICMP header segments
```

parse_tcp_header

Parameters

Parameter	Type	Description
hex_data	String	The hex string of data to parse representing the TCP segment.

Return

Type	Reason
None	No value is to be returned.

Pseudo Code

```
FUNCTION parse_tcp_header(hex_data):
parse the TCP header segments from the hex data including the source port, destination port, sequence number, acknowledgment number, data offset, reserved, flags, window size, checksum, urgent pointer, and payload
print the parsed TCP header segments
```

parse_udp_header

Parameters

Parameter	Type	Description
hex_data	String	The hex string of data to parse representing the UDP segment.

Return

Type	Reason
None	No value is to be returned.

Pseudo Code

```
FUNCTION parse_udp_header(hex_data):
parse the UDP header segments from the hex data including the source port, destination port, length, checksum, and payload
print the parsed UDP header segments
```

parse_dns_header

Parameters

Parameter	Type	Description
hex_data	String	The hex string of data to parse representing the DNS segment.

Return

Type	Reason
None	No value is to be returned.

Pseudo Code

```
FUNCTION parse_dns_header(hex_data):
parse the DNS header segments from the hex data including the flags,
transaction ID, QD count, AN count, NS count, AR Count, and payload
print the parsed DNS header segments
```