NETWORK PACKET SNIFFER

A PROJECT REPORT

Submitted by

Priyanka Srinivas [Reg No: RA1911026010014]
Abhishek Hindocha [Reg No: RA1911026010027]
Vaibhav Singh [Reg No: RA1911026010042]

Under the Guidance of

Mr P. Saravanan
(Associate Professor, Department of Information Technology)

In partial fulfilment of the Course

18CSC302J - Computer Networks
in
Department of Information Technology

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chengalpattu District - 603203

NOVEMBER 2021

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "**NETWORK PACKET SNIFFER"** is the Bonafide work of **"Priyanka Srinivas [Reg No: RA1911026010014], Abhishek Hindocha [Reg No: RA1911026010027], Vaibhav Singh [Reg No: RA1911026010042],** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Mr. P Saravanan
GUIDE
Assistant Professor
Dept. of Information Technology

Dr. G. VADIVU
HEAD OF THE DEPARTMENT
Dept. of Information Technology

Signature of Internal Examiner

Signature of External Examiner

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honourable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavours. We would like to express our warmth of gratitude to our **Registrar Dr. S. Ponnusamy**, for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V. Gopal**, for bringing out novelty in all executions. We would like to express our heartfelt thanks to the **Chairperson, School of Computing Dr. Revathi Venkataraman**, for imparting confidence to complete my course project.

We wish to express our sincere thanks to the **Department Head Dr. G. Vadivu** for his constant encouragement and support. We are highly thankful to my **Computer Networks Internal Guide Mr. P. Saravanan, Associate Professor, Department of Information Technology**, for his assistance, timely suggestion and guidance throughout the duration of this course project. Finally, we thank our parents and friends who directly and indirectly contributed to the successful completion of our project.

ABSTRACT

In IT operations, ensuring secure and reliable communications over different networks is a crucial requirement. IT administrators have to rely on different protocols, networking best practices, and network monitoring tools to ensure the flow of data in a network meets various standards for security and Quality of Service (QoS). One of these common practices is known as packet sniffing, which helps IT administrators keep track of packets (small formatted units of data) and ensure they're transferred smoothly. While the packet sniffing technique is often associated with cyberattacks, it is commonly used by internet service providers, government agencies, advertisers, and even large organizations for network monitoring.

Keywords - Wireshark, packet sniffing, network security, HTTP, FTP

TABLE OF CONTENTS

ABBREVIATIONS

OS Operating System

IT Information Technology

QoS Quality of Service

FCS Frame Check Sequence

NIC Network Interface Card

TCP/IP Transmission Control Protocol/Internet Protocol

UDP User Datagram Protocol

ICMP Internet Control Message Protocol

IPv4 Internet Protocol version 4

IPv6 Internet Protocol version 6

MAC Media Access Control

UML Unified Modeling Language

CHAPTER 1

INTRODUCTION

1.1. What is a packet sniffer?

A packet sniffer — also known as a packet analyzer, protocol analyzer or network analyzer — is a piece of hardware or software used to monitor network traffic. Sniffers work by examining streams of data packets that flow between computers on a network as well as between networked computers and the larger Internet. These packets are intended for — and addressed to — specific machines, but using a packet sniffer in "promiscuous mode" allows IT professionals, end-users or malicious intruders to examine any packet, regardless of destination.

Packet sniffers come in several forms. Some packet sniffers used by network technicians are single-purpose hardware solutions. In contrast, other packet sniffers are software applications that run on standard consumer-grade computers, using the network hardware provided on the host device to perform packet capture and injection tasks.

1.2. Types of Packet Sniffers

There are two main types of packet sniffers:

a. Hardware sniffers - Hardware packet sniffers are commonly used when network administrators have to analyze or monitor a particular segment of a large network. With a physical connection, these packet sniffers allow administrators to ensure all packets are captured without any loss due to routing, filtering, or any other network issue. A hardware packet sniffer can have the facility to store the packets, or they can be programmed to forward all captured packers to a centralized location for further analysis.

b. Software sniffers - Software Packet Sniffers are the more common type of packet sniffers used by many organizations. Every computer or node connects to the network using a Network Interface Card (NIC), which is generally configured to ignore the packets not addressed to it. However, a Software Packet Sniffer changes this behaviour, so one can receive every bit of network traffic for analysis. The NIC configuration is known as promiscuous mode. The amount of information collected depends on whether the packet sniffer is set in filtered or unfiltered mode. Depending on the size and complexity of a network, multiple packet sniffers might be required to monitor and analyze a network effectively. This is because a network adapter can only collect traffic from one side of a switch or a router. Similarly, in wireless networks, most network adapters can connect to only a single channel at a given time. To capture packets from other channels, one has to install multiple packet sniffers.

It is possible to configure software sniffers in two ways. The first is "unfiltered," meaning they will capture all packets possible and write them to a local hard drive for later examination. Next, is "filtered" mode, meaning analyzers will only capture packets that contain specific data elements. Our unfiltered sniffer contains different modules to extract packets with different types of protocols such as IPv4, TCP, UDP etc. It is constructed using python.

CHAPTER 2

REQUIREMENT ANALYSIS

2.1. Hardware Requirements

Processor: 2.4 GHz Clock Speed

RAM: 1 GB

Hard Disk: 500 MB (Minimum free space)

2.2. Software Requirements

OS: Linux (with admin access)

Compiler: Python 3.9

CHAPTER 3

RELEVANT THEORY

3.1. Packet

In networking, any data that is sent over a network is divided into packets. These packets are then recombined by the device that is supposed to receive them to make sense of the information sent by the source. A packet header is a label of sorts, which provides information about the packet's contents, origin, and destination.

3.2. Ethernet Frame

The encapsulated data defined by the network access layer is called an Ethernet frame. An Ethernet frame starts with a header, which contains the source and destination MAC addresses, among other data. The middle part of the frame is the actual data. The frame ends with a field called Frame Check Sequence (FCS).

Preamble	SFD	Destination MAC	Source MAC	Type	Data and Pad	FCS
7 Bytes	1 Byte	6 Bytes	6 Bytes	2 Bytes	46-1500 Bytes	4 Bytes

3.3. Socket and Port

Socket: Two processes that are running on a computer or running on two different systems can communicate via a socket. A socket works as an inter-process communicator and seen as the endpoint of the process communication. For communication, the socket uses a file descriptor and is mainly employed in client-server applications.

Port: Port is a part of the transport layer and helps in network communication. A port is a logical identifier assigned to a process in order to identify that process uniquely in a network system. When two network devices communicate, they do so by sending packets

to each other. Each packet received by a receiver device contains a port number that uniquely identifies the process where the packet needs to be sent.

3.4. MAC Address

A Media Access Control (MAC) address is a 48-bit (6 bytes) address that is used for communication between two hosts in an Ethernet environment. It is a hardware address, which means that it is stored in the firmware of the network card.
MAC addresses are usually written in the form of 12 hexadecimal digits. For example, consider the following MAC address:
D8-D3-85-EB-12-E3

3.5. Protocols

3.5.1. TCP/IP

The TCP/IP suite is named after its most important protocols, the Transmission Control Protocol (TCP) and the Internet Protocol (IP).
The TCP/IP suite is a set of protocols used on computer networks today (most notably on the Internet). It provides end-to-end connectivity by specifying how data should be packetized, addressed, transmitted, routed and received on a TCP/IP network. This functionality is organized into four abstraction layers and each protocol in the suite resides in a particular layer.

3.5.2. UDP

The User Datagram Protocol is basically a scaled-down version of TCP. Just like TCP, this protocol provides delivery of data between applications running on hosts on a TCP/IP network, but, unlike TCP, it does not sequence the data and does not care about the order in

which the segments arrive at the destination. Because of this, it is considered to be an unreliable protocol.

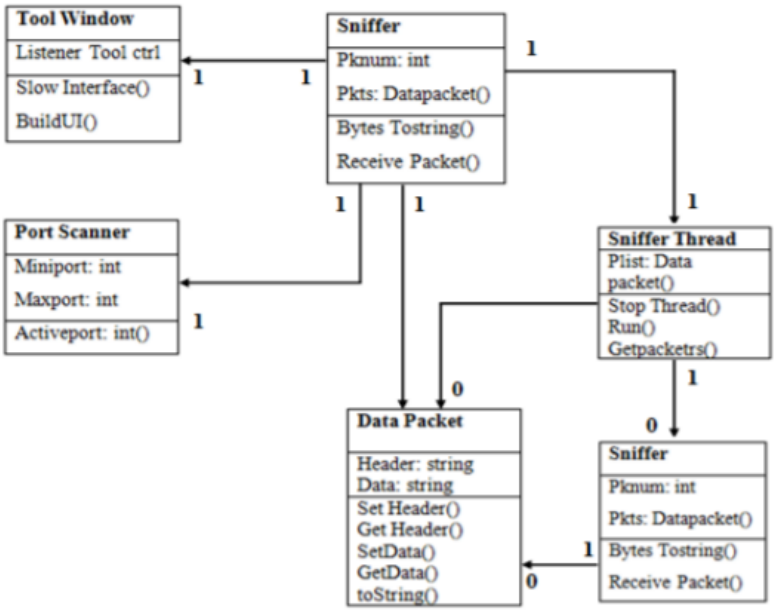
3.5.3. ICMP

ICMP or the Internet Control Message Protocol. It is a network layer protocol. It is used for error handling in the network layer, and it is primarily used on network devices such as routers. As different types of errors can exist in the network layer, ICMP can be used to report these errors and to debug those errors.

CHAPTER 4

ARCHITECTURE AND DESIGN

4.1. Class Diagram

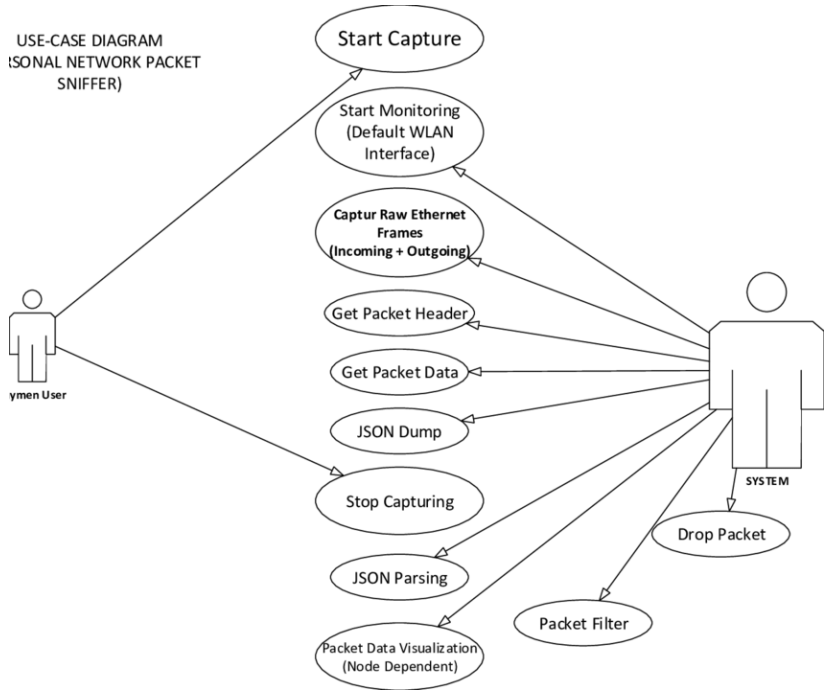


There are 5 main classes implemented in this system: Tool Window, Port Scanner, Data Packer, Sniffer and Sniffer Thread.

a. The Tool Window class is constantly connected to the sniffer, as it provides the interface to display the data collected.
b. The Port Scanner class probes a server or host for open ports. The majority of uses of a port scan are not attacking, but rather simple probes to determine services available on a remote machine.
c. The Sniffer Thread class controls the activity of the program; i.e, it keeps the socket alive by controlling the activity thread.
d. The Sniffer class is constantly connected to a socket that continuously listens to the port for streams of data.

e. The Data Packet class collects the data from Sniffer Thread and formats it to break down the header, protocol and data received.

4.2. Use Case Diagram



A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram does not represent the order of the processes performed. It focuses on the relationships between the actors and the system. In this diagram, the process of capturing a packet to extract the data from it is represented where all the important use cases are highlighted. The two figures on the left and right of the diagram represent the two actors that interact with the system.

The process goes from starting to capture the packet to displaying the data inside the packet. For most of the use cases such as capturing the packet to extract the data, the system interacts heavily with the process to understand and decode the information. The user starts the program and interrupts the capture process as required.

CHAPTER 5

IMPLEMENTATION

5.1. Packages

This network sniffer has been implemented completely in python. It requires the following packages, which usually come pre-installed with python. They are:

1. socket - The socket module provides various objects, constants, functions and related exceptions for building full-fledged network applications including client and server programs. A sample client program using this library's basic objects and functions is as follows:

```
#import the socket module
import socket

#Create a socket instance
socketObject = socket.socket()

#Using the socket connect to a server...in this case localhost
socketObject.connect(("localhost", 80))
print("Connected to localhost")

#Send a message to the webserver to supply a page as given by
Host param of GET request

HTTPMessage = "GET / HTTP/1.1\r\nHost: localhost\r\n Connection:
close\r\n\r\n"
bytes = str.encode(HTTPMessage)

socketObject.sendall(bytes)

#Receive the data
while(True):
      data = socketObject.recv(1024)
      print(data)

      if(data==b''):
      print("Connection closed")
      break

socketObject.close()
```

The output is as follows:

```
Connected to localhost

b'HTTP/1.1 200 OK\r\nDate: Thu, 26 Jan 2019 17:40:19 GMT\r\nServer: Apach
OpenSSL/0.9.8za\r\nLast-Modified: Thu, 26 Jan 2019 17:40:12 GMT\r\nETag:
bytes\r\nContent-Length: 165\r\nContent-Type: text/html\r\n\r\n<html>\n\n
</Title>\n\t<link rel="stylesheet" href="MyIST.css">\n\n</head>\n\n<body>
</h1></center>\n</body>\n\n</html>\n\n'

b''

Connection closed
```

2. struct - The struct module in Python is used to convert native Python data types such as strings and numbers into a string of bytes and vice versa. It is used mostly for handling binary data stored in files or from network connections, among other sources.

3. textwrap - The textwrap module can be used for wrapping and formatting plain text. This module provides formatting of text by adjusting the line breaks in the input paragraph. It contains many attributes such as width, tab space, indent etc which help with formatting. In the sniffer, since long streams of data are collected, they are formatted using the textwrap function into human-readable text.

5.2. Display

```
"""
Sample data display -

Ethernet:
    IPv4:
        abc213
        def124
        ...
        TCP:
            ghi12432
            jkl54768
"""
```

The data collected will be displayed in this manner. The ethernet frame will hold the type of address (IPv4 or IPv6) and its header, protocol number etc. Additionally, the protocol details (TCP, UDP etc) and the data gathered will be displayed.

5.3. Modules

There are 6 main working modules in this implementation:

1. def ethernet_frame(data)

```
# Unpack ethernet frame to find the data
def ethernet_frame(data):
    dest_mac, src_mac, proto = struct.unpack('! 6s 6s H', data[:14])
    return get_mac_addr(dest_mac), get_mac_addr(src_mac), socket.htons(proto), data[14:]
```

The first 14 bytes of an ethernet frame capture the destination address, source address and protocol number. The unpack function separates the destination MAC address, the source MAC address and the protocol number according to the byte fields provided after the exclamation.

For further interpretation of protocol, we need to format the MAC address in a readable way.

2. def get_mac_addr(bytes_str)

```
# Return properly formatted MAC address (i.e. AA:BB:CC:DD:EE:FF)
def get_mac_addr(bytes_addr):
    bytes_str = map('{:02x}'.format, bytes_addr)
    mac_addr = ':'.join(bytes_str).upper()
    return mac_addr
```

The above module formats the MAC address into a human-readable address.

3. def ipv4_packet(data)

```
# Unpack IPv4 packet
def ipv4_packet(data):
    version_header_length = data[0] # contains version and header length
    version = version_header_length >> 4 # bit shifting by 4 to get the version
    header_length = (version_header_length & 15) * 4
    ttl, proto, src, target = struct.unpack('! 8x B B 2x 4s 4s', data[:20])
    return version, header_length, ttl, proto, ipv4(src), ipv4(target), data[header_length:]
```

This module captures the ipv4 packet to break down data into the following variables: version (version), header length (header_length), time to live (ttl), protocol (proto), source address (src) and destination address (target).

The version and header length are unpacked as the first data point in the structure. Hence, they need to be separated through the extraction of separate elements. This extraction is performed by bitwise operations. By bit-shifting by 4, we push the version number out of the data sequence that contains both version and header length (version_header_length). Similarly, we perform bitwise AND (&) to extract the header length.

4. def icmp_packet(data)

```
# Unpack ICMP packet
def icmp_packet(data):
    icmp_type, code, checksum = struct.unpack('! B B H', data[:4])
    return icmp_type, code, checksum, data[4:]
```

The Internet Control Message Protocol (ICMP) has many messages that are identified by a "type" field. The icmp_type variable captures this field. For example, types 1 and 2 indicate that the destination is unassigned, whereas type 3 states that the destination is unreachable. Likewise, there are around 255 types of ICMP messages.

Most internet protocols use a common checksum algorithm to validate the integrity of the packets that they exchange. The checksum in ICMP is allotted 2 bytes.

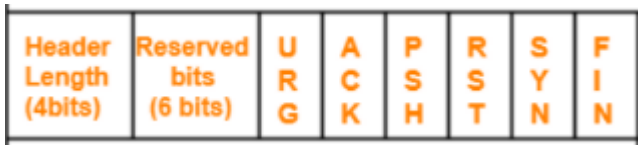
5. def tcp_packet(data)

```
# Unpack TCP segment
def tcp_packet(data):
    (src_port, dest_port, seq, ack, offset_reserved_flags) = struct.unpack('! H H L L H', data[:14])
    offset = (offset_reserved_flags >> 12) * 4
    flag_urg = (offset_reserved_flags & 32) >> 5
    flag_ack = (offset_reserved_flags & 16) >> 4
    flag_psh = (offset_reserved_flags & 8) >> 3
    flag_rst = (offset_reserved_flags & 4) >> 2
    flag_syn = (offset_reserved_flags & 2) >> 1
    flag_fin = (offset_reserved_flags & 1)

    return src_port, dest_port, seq, ack, flag_urg, flag_ack, flag_psh, flag_rst, flag_syn, flag_fin, data[offset:]
```

The TCP packet is the most common protocol captured. There are 6 main parameters in a TCP packet: source port number, destination port number, sequence number, and acknowledgement, along with offset and flags.

From bit-shifting in a particular capacity, we come across different types of flags.



6. def udp_segment(data)

```
# Unpack UDP segment
def udp_segment(data):
    src_port, dest_port, size = struct.unpack('! H H 2x H', data[:8])
    return src_port, dest_port, size, data[8:]
```

From a UDP packet, we can extract the source and destination address, along with the size of the packet.

Additionally, there are 2 supporting functions:

1. def ipv4(unformatted_addr)

```
# Return properly formatted IP address (i.e. 127.0.0.1)
def ipv4(unformatted_addr):
    return '.'.join(map(str, unformatted_addr))
```

The above function helps with

2. def format_multi_line(prefix, string, size=80)

```
# Format multi-line data
def format_multi_line(prefix, string, size=80):
    size -= len(prefix)
    if isinstance(string, bytes):
        string = ''.join(r'\x{:02x}'.format(byte) for byte in string)
        if size % 2:
            size -= 1
    return '\n'.join([prefix + line for line in textwrap.wrap(string, size)])
```

Lastly, there is the main() function which supports the running of the rest of the modules.

CHAPTER 6

RESULT AND ANALYSIS

6.1. OUTPUT

```
Ethernet frame:
        - Destination: 00:50:56:F9:7F:1D, Source: 00:0C:29:E5:2F:9E, Protocol: 8
        - IPv4 Packet:
                - Version: 4, Header Length: 20, TTL: 64,
                - Protocol: 6, Source: 192.168.11.128, Target: 35.244.247.133
        - TCP Packet:
                - Source Port: 60116, Destination Port: 443
                - Sequence: 1220008083, Acknowledgement: 1035868485
                - Flags:
                        - URG: 0, ACK: 1, PSH: 0, RST: 0, SYN: 0, fin: 0
                - Data:


Ethernet frame:
        - Destination: 00:50:56:F9:7F:1D, Source: 00:0C:29:E5:2F:9E, Protocol: 8
        - IPv4 Packet:
                - Version: 4, Header Length: 20, TTL: 64,
                - Protocol: 17, Source: 192.168.11.128, Target: 91.189.89.199
        - UDP Packet:
                - Source Port: 50039, Destination Port: 123, Length: 33270
                - Data:
                        \x23\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
                        \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
                        \x00\x00\xe5\x3b\xa5\xf6\x15\x3f\xc7\x74
```

```
Ethernet frame:
        - Destination: 00:0C:29:E5:2F:9E, Source: 00:50:56:F9:7F:1D, Protocol: 8
        - IPv4 Packet:
                - Version: 4, Header Length: 20, TTL: 128,
                - Protocol: 6, Source: 35.244.247.133, Target: 192.168.11.128
        - TCP Packet:
                - Source Port: 443, Destination Port: 60116
                - Sequence: 1035868439, Acknowledgement: 1220008083
                - Flags:
                        - URG: 0, ACK: 1, PSH: 1, RST: 0, SYN: 0, fin: 0
                - Data:
                        \x17\x03\x03\x00\x29\x00\x00\x00\x00\x00\x00\x05\x9b\x5a\x03\xd4\x8b\xd5
                        \xfb\x57\xa1\x3e\xf3\x0a\xb7\x1f\x97\xd6\x91\xc6\xcc\x2f\x5f\xd9\x9c\xd5\x2f
                        \xe8\xf4\x25\xf7\xd1\x75\xaa\xbf

Ethernet frame:
        - Destination: 00:50:56:F9:7F:1D, Source: 00:0C:29:E5:2F:9E, Protocol: 8
        - IPv4 Packet:
                - Version: 4, Header Length: 20, TTL: 64,
                - Protocol: 6, Source: 192.168.11.128, Target: 35.244.247.133
        - TCP Packet:
                - Source Port: 60116, Destination Port: 443
                - Sequence: 1220008083, Acknowledgement: 1035868485
                - Flags:
                        - URG: 0, ACK: 1, PSH: 0, RST: 0, SYN: 0, fin: 0
                - Data:
```

```
Ethernet frame:
        - Destination: 33:33:00:00:00:FB, Source: 00:50:56:C0:00:08, Protocol: 56710

Ethernet frame:
        - Destination: FF:FF:FF:FF:FF:FF, Source: 00:50:56:C0:00:08, Protocol: 8
        - IPv4 Packet:
                - Version: 4, Header Length: 20, TTL: 128,
                - Protocol: 17, Source: 192.168.11.1, Target: 192.168.11.255
        - UDP Packet:
                - Source Port: 57621, Destination Port: 57621, Length: 13264
                - Data:
                        \x53\x70\x6f\x74\x55\x64\x70\x30\x31\xdf\x71\x3e\x16\x0b\xe6\x1a\x00\x01\x00
                        \x04\x48\x95\xc2\x03\x5f\x60\x3c\x60\xd5\x85\xc3\x3d\xb1\x53\x2c\x15\x0d\xf8
                        \xa7\xf2\xc0\x12\xb6\xf3

Ethernet frame:
        - Destination: 00:50:56:F9:7F:1D, Source: 00:0C:29:E5:2F:9E, Protocol: 8
        - IPv4 Packet:
                - Version: 4, Header Length: 20, TTL: 64,
                - Protocol: 6, Source: 192.168.11.128, Target: 35.244.247.133
        - TCP Packet:
                - Source Port: 60116, Destination Port: 443
                - Sequence: 1220008083, Acknowledgement: 1035868485
                - Flags:
                        - URG: 0, ACK: 1, PSH: 1, RST: 0, SYN: 0, fin: 0
                - Data:
                        \x17\x03\x03\x00\x29\x00\x00\x00\x00\x00\x00\x00\x08\xf9\x87\xed\xa0\xed\x84
                        \xc3\x61\xd9\xde\xd8\xca\x54\xb7\x23\x7e\xdf\x8a\xab\x33\x29\x86\x2e\xfa\xe5
                        \x29\xf9\x4f\x55\x96\xeb\x76\xb4

Ethernet frame:
        - Destination: 00:0C:29:E5:2F:9E, Source: 00:50:56:F9:7F:1D, Protocol: 8
        - IPv4 Packet:
                - Version: 4, Header Length: 20, TTL: 128,
                - Protocol: 6, Source: 35.244.247.133, Target: 192.168.11.128
        - TCP Packet:
                - Source Port: 443, Destination Port: 60116
                - Sequence: 1035868485, Acknowledgement: 1220008129
                - Flags:
                        - URG: 0, ACK: 1, PSH: 0, RST: 0, SYN: 0, fin: 0
                - Data:
                        \x00\x00\x00\x00\x00\x00
```

6.2. Analysis and Conclusion

The data is formatted in a manner where individual frames are distinguishable from each other. The most common protocols for all packets is TCP, followed by UDP. The data is captured in hexadecimal, and can be further processed into human-readable data. There are different source and destination addresses for all packets, hence they are being captured from different ports.

For all TCP packets, the presence of different flags has been detected. Overall, the data has been displayed neatly with the help of a formatting function.

CHAPTER 7

REFERENCES

7.1. Online Resources

1. https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml

7.2. Research Papers

1. An Enhanced Sniffing Tool for Network Management [↗]
2. Packet Sniffing and Sniffing Detection [↗]