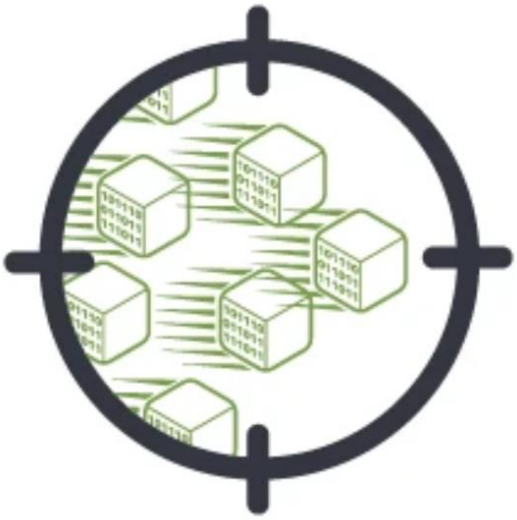
PACKET MONITORING

INTRODUCTION TO PACKET MONITORING

Packet monitoring (also known as packet sniffing or protocol analyzing) is used to intercept and capture live data as it travels over the network (Ethernet or Wi-Fi) in order to understand what is happening in the network.

Packet analysis is done by protocol analyzers such as Wireshark available on the Internet.
It can be used for packet capture, packet drop detection, packet filtering and counting.



A packet analyzer used for intercepting traffic on wireless networks is known as a wireless analyzer or WiFi analyzer. While a packet analyzer can also be referred to as a network analyzer or protocol analyzer these terms can also have other meanings.

As data streams flow across the network, the analyzer captures each packet and, if needed, decodes the packet's raw data, showing the values of various fields in the packet, and analyzes its content.

The applications that are sitting in the corner of your own network and eating the bandwidth. They might be making your network insecure or making it visible to the public network.

Through this unnoticed application, different forms of network traffic can enter without any restrictions.

Packet monitoring can be used for the following aspects:

- To analyze network problems by looking into the packets and their specific details so that you can get a better hold over your network.

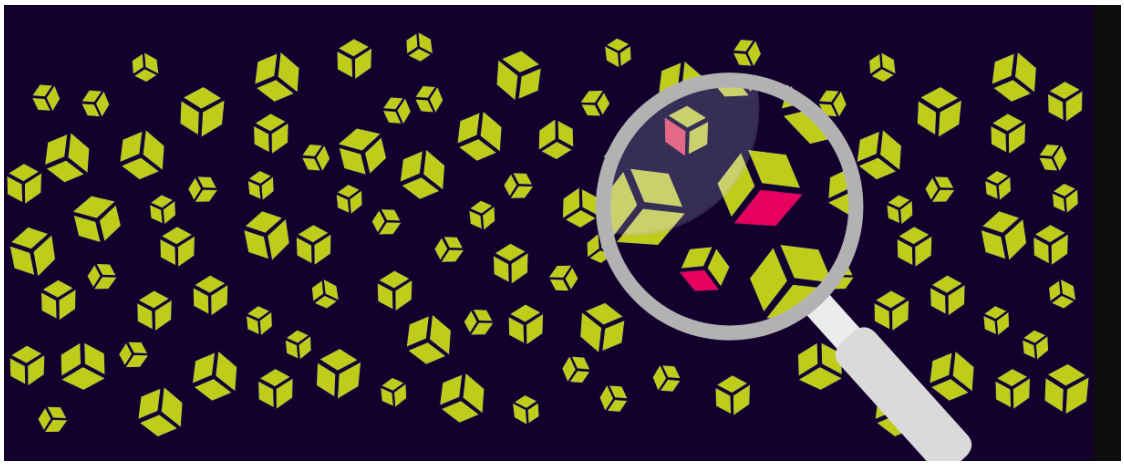
- To detect network intrusion attempts and whether there are any malicious users who are trying to get into your network, or they have already got access to something in your network
- To detect network misuse by internal or external users by establishing firewall rules in your security appliance and then monitoring each of these rules through Wireshark.

- To isolate exploited systems so that the affected system doesn't become a pivot point for your network for malicious users.

- To monitor data in motion once it travels live in your network to have better control over the allowed and restricted categories of data.

- To gather and report network statistics by filtering the most specific packets as per your requirements and then creating specific capture filters for your perusal that can help you in the long run.

To identify possible or malicious attacks that your network can be a victim of, to analyze them, control/supervise them, and make yourself ready for any possible malicious activity.



How to do packet monitoring?

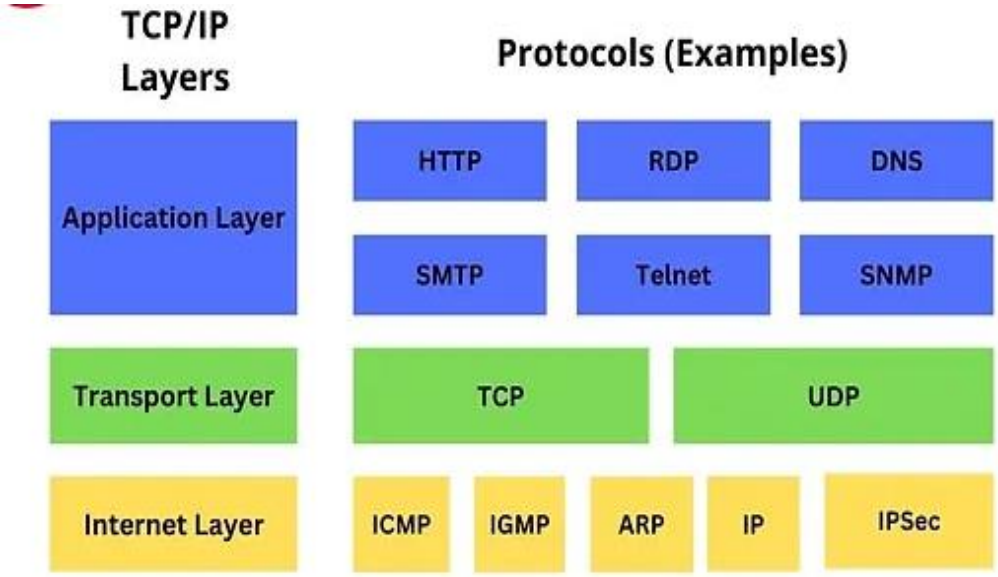
When traffic is captured, either all raw data is captured or only the header data is captured without capturing the total content of the packet.

Captured information is decoded from raw data to a human-readable form, which allows users to understand the exchanged data between the networks in a much more precise manner.

When performing a packet analysis, you should take care of things such as which protocols can be interpreted, which is the best software you can use according to your expertise, which protocol analyzer will best suit your network requirement..

Packet sniffers can interpret common network protocols (such as IP and ICMP), transport layers (such as TCP and UDP), and application protocols (such as DNS and HTTP).

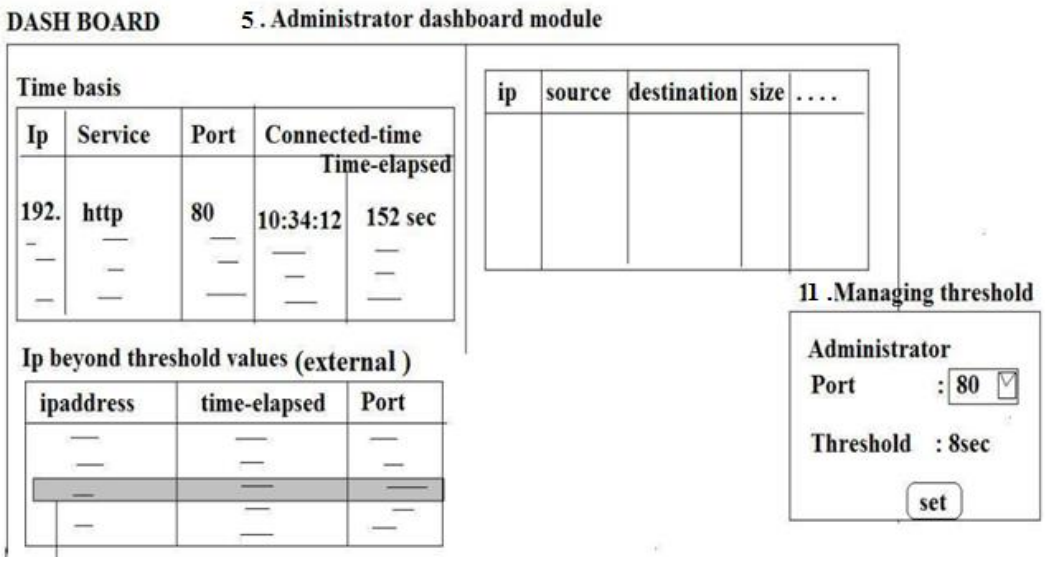
Each of these protocols has different complexity levels depending on how and where they are being implemented. Majorly, all protocols work in the same fashion, where they send a request and wait for the confirmation, and as they receive an acknowledgement, they let the devices communicate.

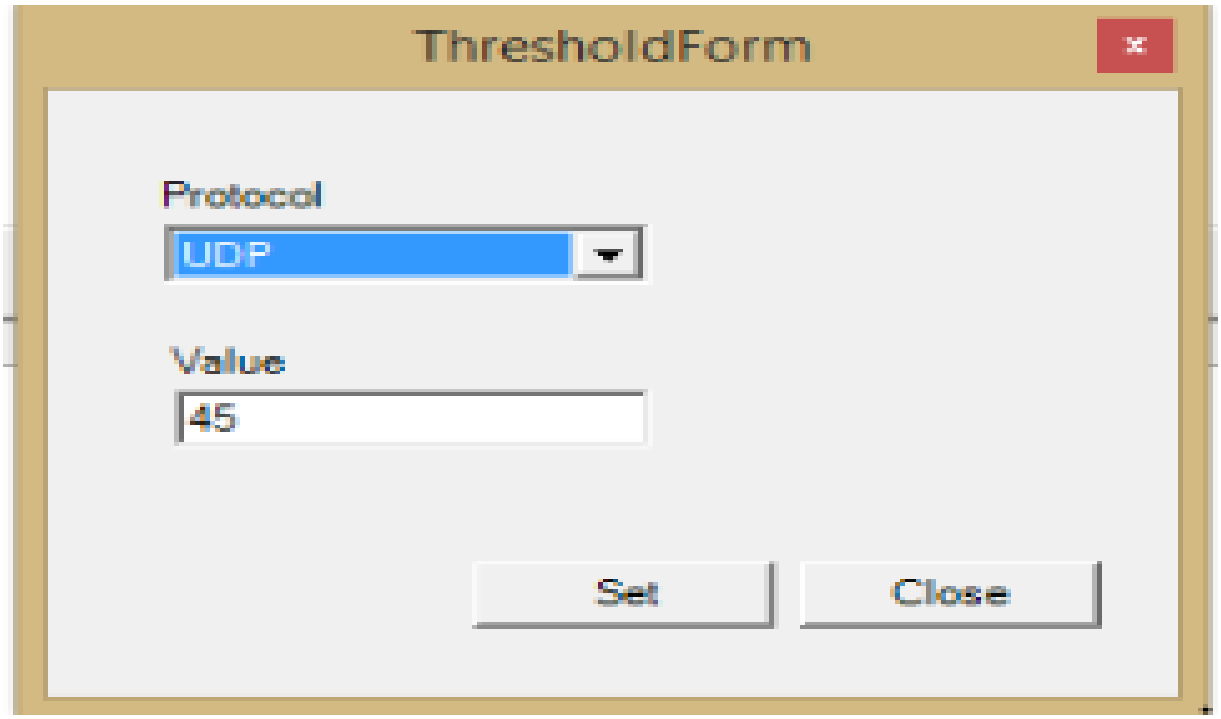


Packet Monitoring module

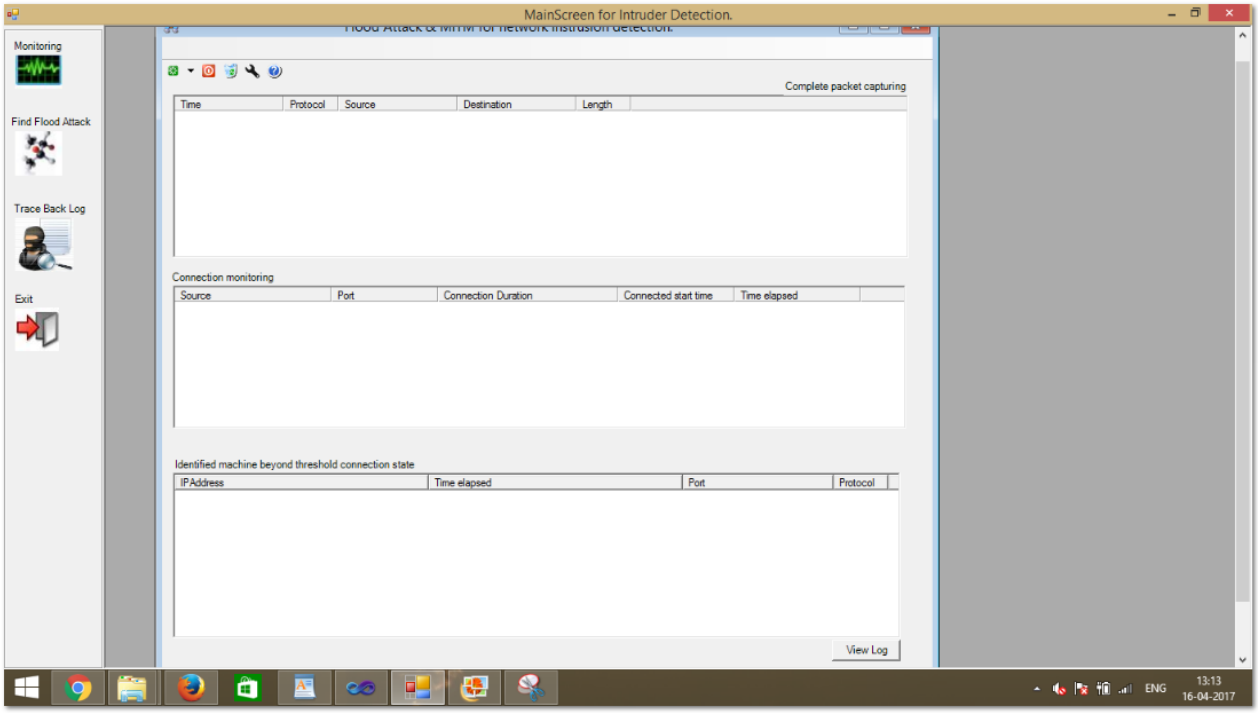
When the admin chooses to monitor the packets that pass through his network, he is directedto PacketMonitor form. This form consists of a tool bar which has the following options:

- Start button and the dropdown menu: This lets the admin select a particular IP address from the list of the IP addresses available. The selected IP will be checked and all the incoming packets will be captured.
- Stop button: This will stop capturing packets that pass through the selected IP ad- dress.
- Clear button: This button will clear all the values displayed on the interface.
- Threshold button: This provides admin to set the threshold values for TCP, UDP and ICMP protocol. Initially the experimental values are set as threshold. For TCP it is 12 for ICMP and 45 for UDP.





UI to set threshhold values



Form where IP's are displayed

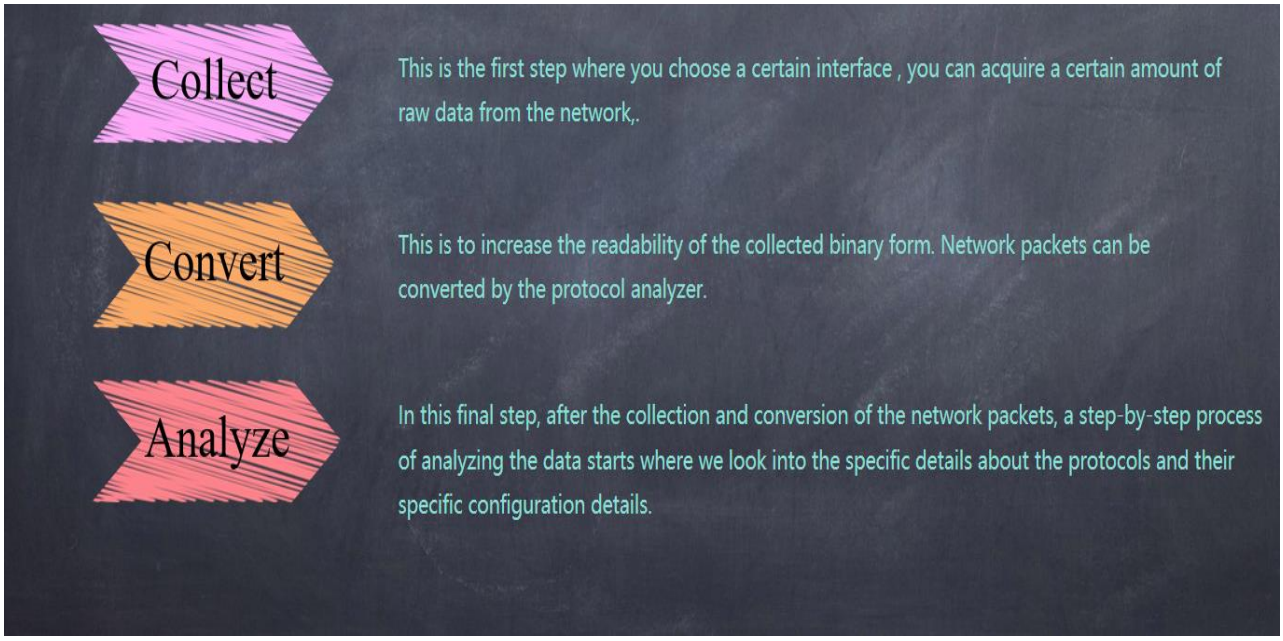
How it works?

There are three common step processes that every protocol analyzer follows: collect, convert, and analyze. These are described as follows:

Collect: This is the first step where you choose a certain interface to listen on, and through this, you can acquire a certain amount of raw data from the network.
which can be achieved by switching your interface into a promiscuous mode so that, after capturing what ever traffic is being broadcasted in your network..

Convert: This is to increase the readability of the collected binary form. Network packets can be converted by the protocol analyzer, such as Wireshark, to simple and easier formats so that people like us can have a better understanding of packets and solve our day-to-day problems easily.

Analyze: In this final step, after the collection and conversion of the network packets, a step-by-step process of analyzing the data starts where we look into the specific details about the protocols and their specific configuration details.
Then, we move on to host and destination addresses and the kind of information they are sharing. Rest of the analysis is left to the user's consent and how they filter and review the collected data.



My role in project

As an intern at BSQ Technologies, our team was tasked with developing a packet display module for a strong room protection project. My role specifically focused on displaying the source and destination addresses from the parsed packets using C# as the backend language.

The strong room protection project aimed to enhance the security of valuable assets stored in secure facilities. It involved various components such as access control, surveillance, and monitoring. The packet display module played a crucial role in monitoring and analyzing network traffic within the strong room.

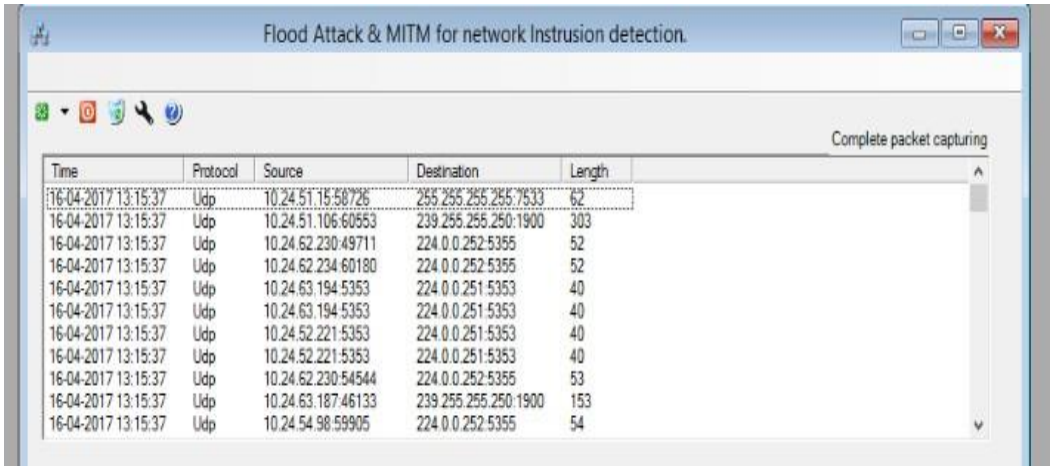
During my internship, I delved into understanding the project requirements and familiarizing myself with the existing infrastructure. I had to capture incoming packets, parse them, and extract important information, specifically the source and destination addresses. These addresses were essential for identifying the origin and destination of the network traffic, ensuring its legitimacy.

To tackle this task, I utilized C# as the backend programming language. C# provided a robust environment for handling network protocols and manipulating data. By leveraging the .NET framework, I integrated libraries and APIs necessary for packet capturing and processing.

One of the main challenges I encountered was optimizing packet capturing and parsing while maintaining system performance. To address this, I implemented multithreading techniques and utilized asynchronous programming patterns. These optimizations ensured that the system could handle a high volume of network traffic without sacrificing responsiveness.

Once the backend development was complete, the next step involved creating a user-friendly interface for visualizing the extracted packet information. Working closely with the UI/UX team, I collaborated on designing an intuitive display that prominently showcased the source and destination addresses. The interface included real-time updates, filtering options, and visual cues to identify suspicious or unauthorized network activity easily.

Throughout my internship, I had the opportunity to collaborate with experienced professionals at BSQ Technologies. The project team consisted of software engineers, network specialists, and security experts. This collaborative environment allowed me to gain valuable insights into various aspects of the project and fostered a supportive learning environment.



Conclusion:

My internship at BSQ Technologies provided me with invaluable hands-on experience working on a packet display module for a strong room protection project. The opportunity to contribute to this project allowed me to enhance my skills in displaying packet information, specifically the source and destination addresses, using C# as the backend language.

Throughout my internship, I worked closely with a talented team at BSQ Technologies, gaining exposure to real-world challenges and collaborating with experienced professionals. This collaborative environment not only helped me develop my technical skills but also provided me with valuable insights into the field of strong room protection and network security.

By successfully implementing the packet display module, I contributed to improving the overall security and monitoring capabilities of the strong room protection system. The displayed source and destination addresses played a critical role in identifying the origin and destination of network traffic, ensuring the legitimacy of the data being transmitted.

The experience of working on this project taught me the importance of attention to detail and efficient programming practices, as capturing and parsing packets while maintaining system performance required careful optimization. Additionally, I learned how to integrate different components of a project, collaborate effectively with cross-functional teams, and deliver a user-friendly interface for visualizing packet information.

I am grateful for the opportunity to work on such a meaningful project during my internship at BSQ Technologies. The skills and knowledge gained will undoubtedly contribute to my future endeavors in the field of network security and software development. I am excited to continue building upon this experience and applying what I have learned to future projects and challenges.
Code with Explanation

Code with Explanation

```
using System;
using System.Windows.Forms;

namespace PacketDisplayApp
{
    public partial class MainForm : Form
    {
        public MainForm()
        {
            InitializeComponent();
        }

        private void DisplayPacket(Packet packet)
        {
            // Display packet information in the UI
            timeLabel.Text = packet.Time.ToString();
            protocolLabel.Text = packet.Protocol;
            sourceLabel.Text = packet.Source;
            destinationLabel.Text = packet.Destination;
            lengthLabel.Text = packet.Length.ToString();
        }

        private void parseButton_Click(object sender, EventArgs e)
        {
            // Simulated parsing of packet
            Packet parsedPacket = ParsePacket();

            // Display the parsed packet information
            DisplayPacket(parsedPacket);
        }

        private Packet ParsePacket()
        {
            // Simulated packet parsing logic
            // Replace with your actual parsing code

            // Sample packet information
            DateTime time = DateTime.Now;
            string protocol = "TCP";
            string source = "192.168.1.100";
            string destination = "10.0.0.1";
            int length = 1024;

            // Create a new packet object with the parsed information
```

```
            Packet packet = new Packet(time, protocol, source, destination,
length);

            return packet;
        }
    }

    public class Packet
    {
        public DateTime Time { get; set; }
        public string Protocol { get; set; }
        public string Source { get; set; }
        public string Destination { get; set; }
        public int Length { get; set; }

        public Packet(DateTime time, string protocol, string source, string
destination, int length)
        {
            Time = time;
            Protocol = protocol;
            Source = source;
            Destination = destination;
            Length = length;
        }
    }
}
```

1. The code starts with the `using` statements that include the necessary namespaces for the application.

2. The `namespace` declaration `PacketDisplayApp` encloses the entire code in a logical grouping.

3. The `MainForm` class is defined and it serves as the main form for the application. It inherits from the `Form` class provided by the Windows Forms framework.

4. The constructor `MainForm()` is defined, which initializes the form by calling the `InitializeComponent()` method. This method is responsible for initializing all the controls and components on the form.

5. The `DisplayPacket(Packet packet)` method is defined. It takes a `Packet` object as a parameter and displays its information in the UI. It updates the text of several `Label` controls on the form to show the packet's time, protocol, source, destination, and length.

6. The `parseButton_Click` event handler method is defined. It is called when the user clicks the "Parse" button on the form. It performs a simulated parsing of a packet by calling the `ParsePacket()` method and then displays the parsed packet information by calling the

`DisplayPacket()` method.

7. The `ParsePacket()` method is defined. It simulates the parsing of a packet and returns a `Packet` object. In the example code, it creates a `DateTime` object for the current time, sets some sample values for the protocol, source, destination, and length, and then creates a new `Packet` object using these values.

8. The `Packet` class is defined. It represents a network packet and has properties for the packet's time, protocol, source, destination, and length. The class also has a constructor that takes these values as parameters and initializes the corresponding properties.

Overall, this code demonstrates a basic implementation of a Windows Forms application that parses and displays information about network packets. It provides a starting point for further development and customization based on the specific requirements of the application.