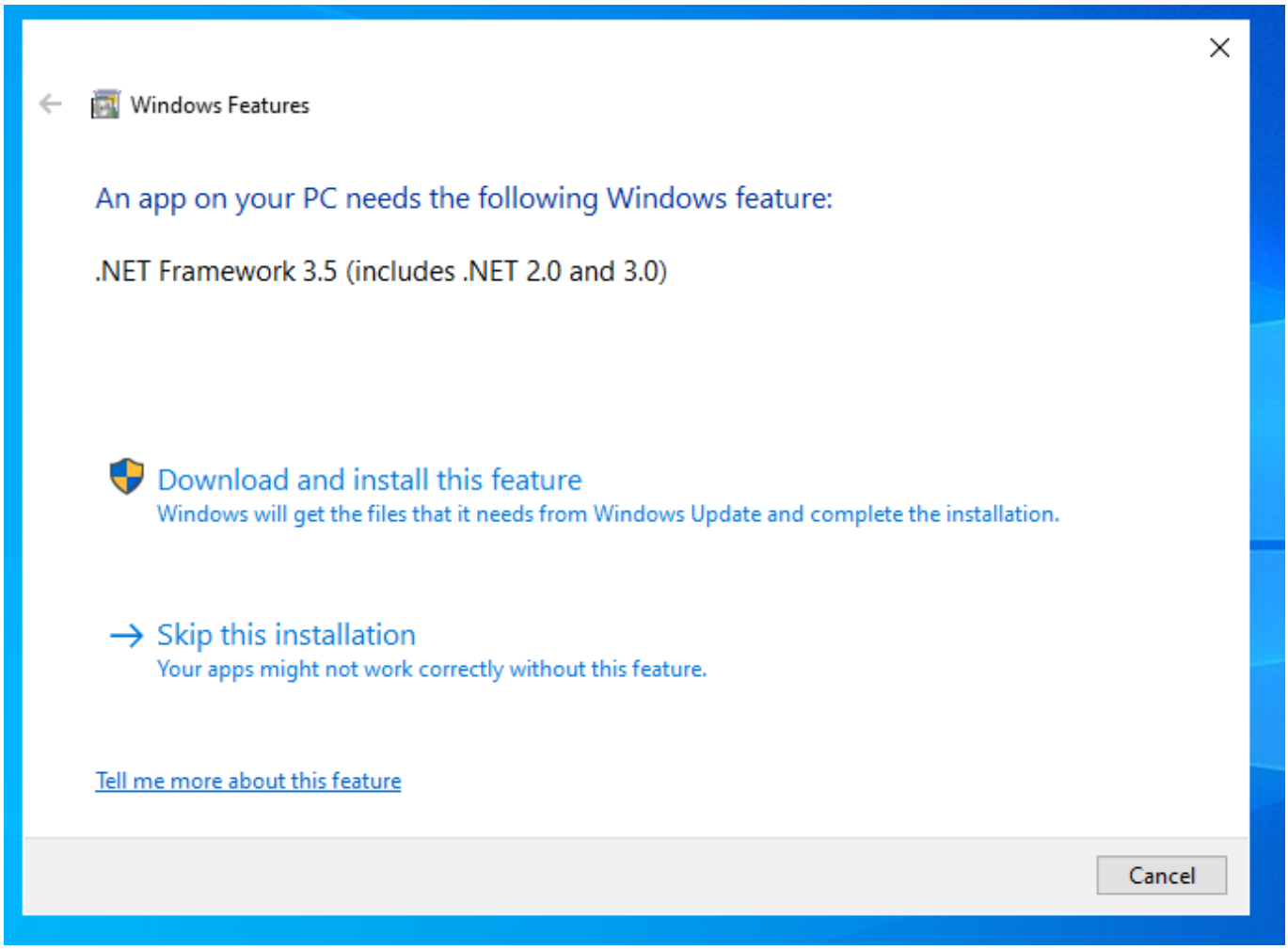
C#: Packet Sniffer w C#

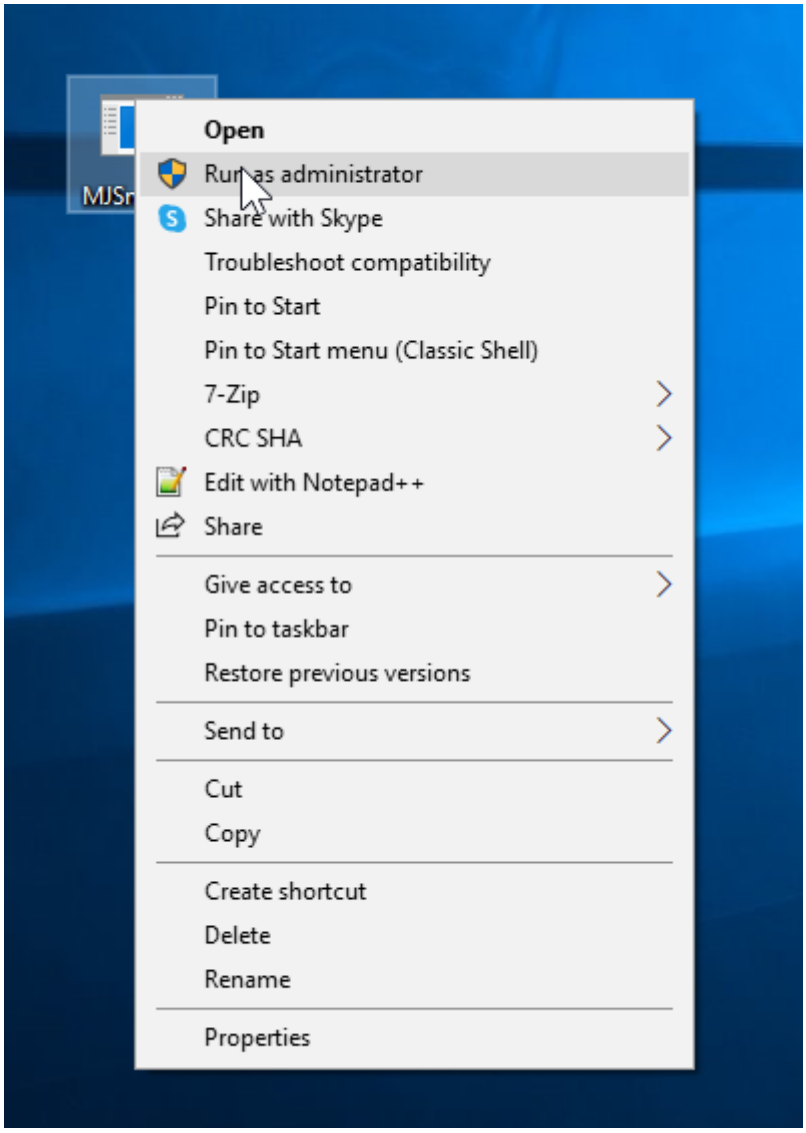
sniffer.exe

Poniżej jest instrukcja jak skorzystać z aplikacji

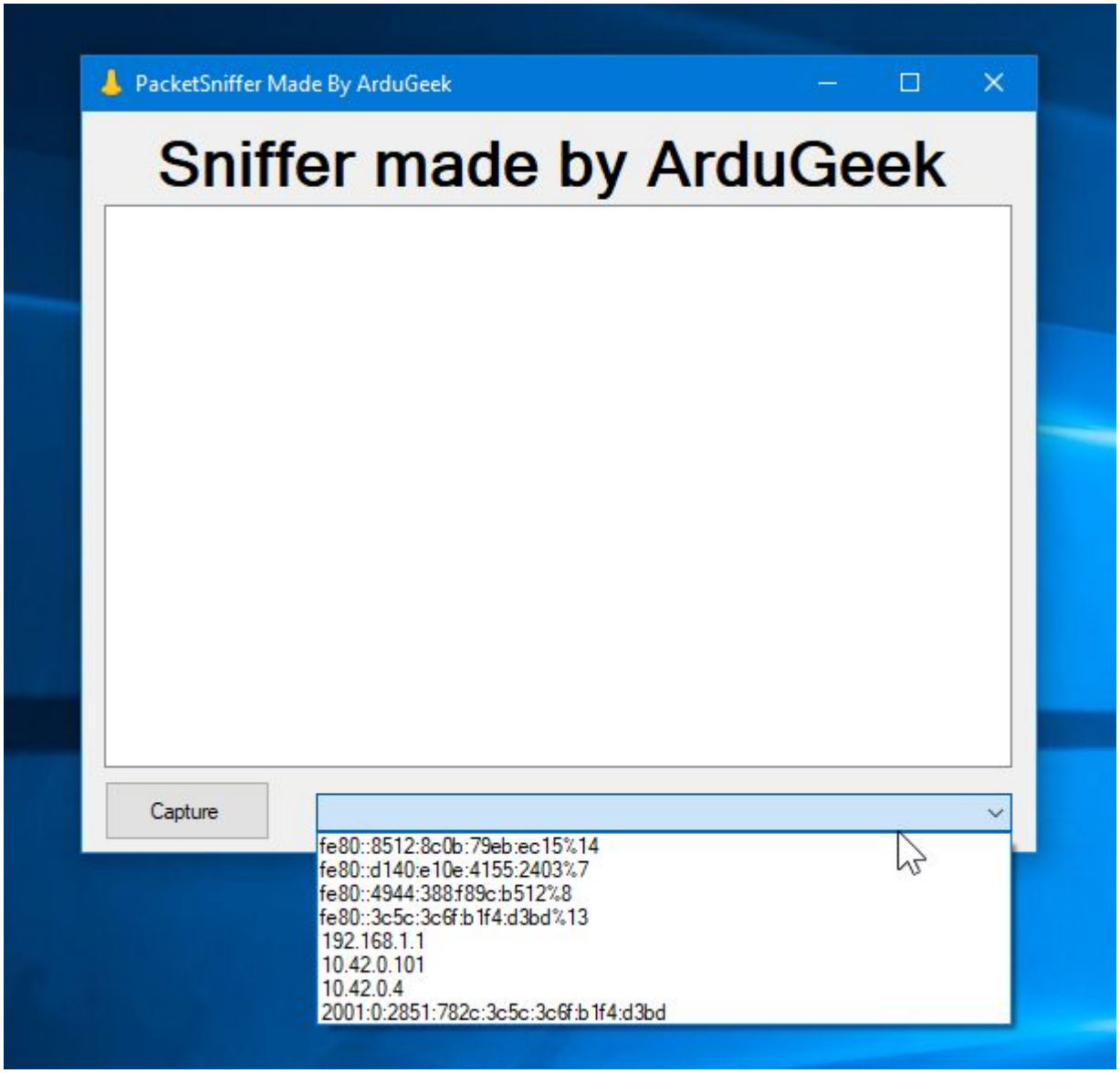


przed uruchomieniem może nam się wyświetlić takie okno,

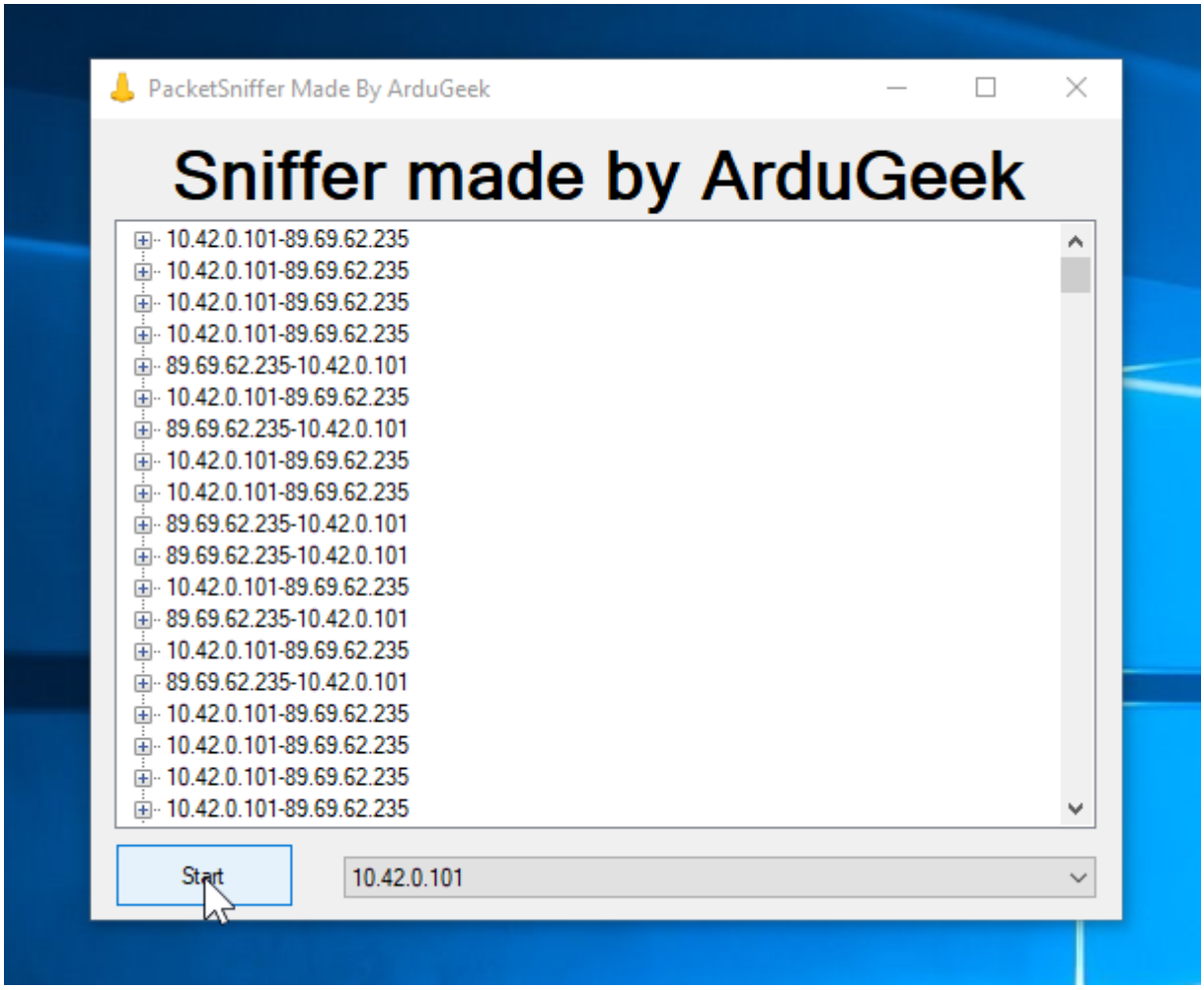
jeżeli tak się stanie to pobieramy framework a potem idziemy dalej



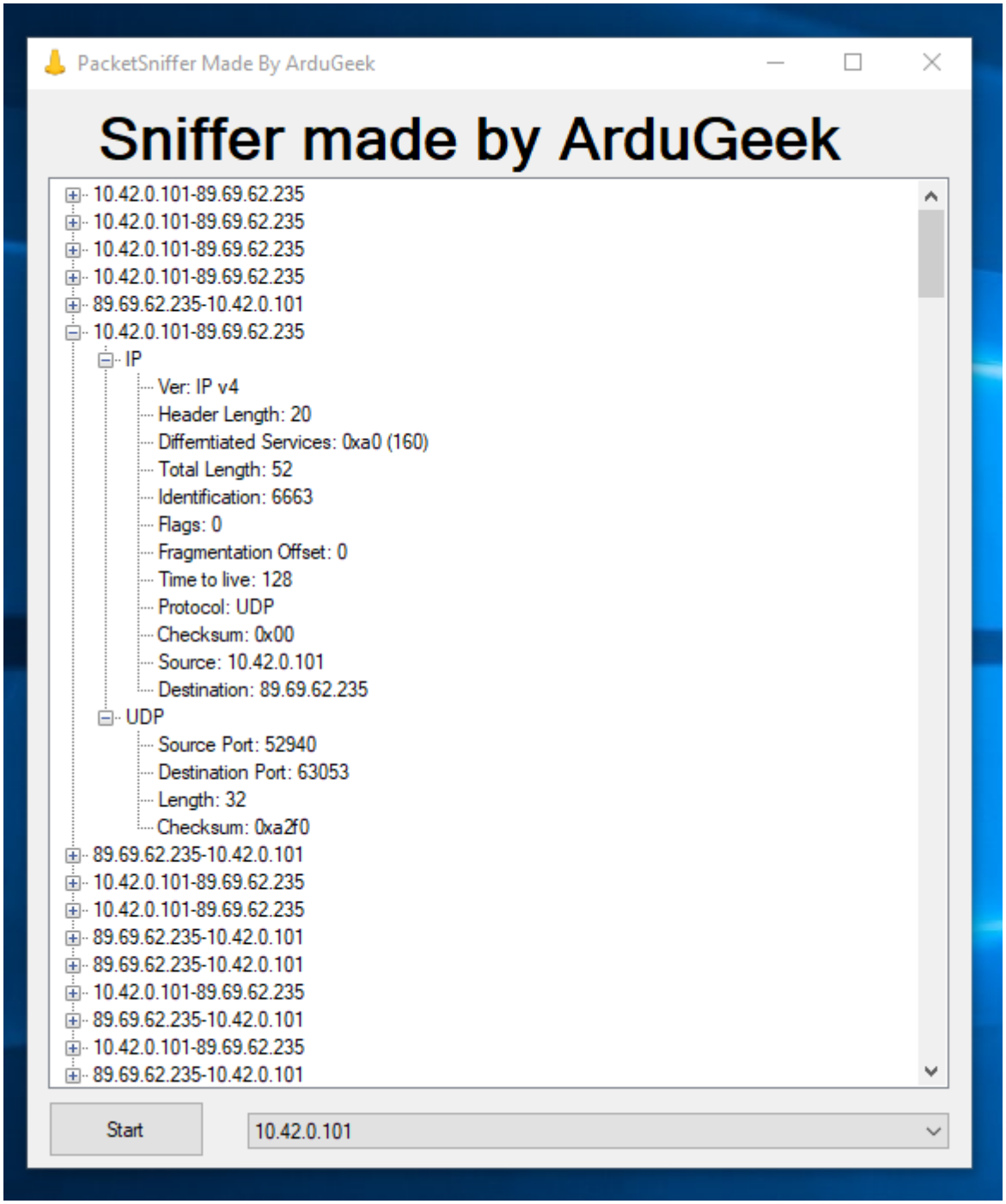
uruchamiamy aplikację z uprawnieniami administratora



następnie wybieramy adres IP na którym będziemy nasłuchiwać



naciskamy przycisk start



a potem oglądamy ruch sieciowy aplikacja powinna wyświetlać UDP TCP DNS

main.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;
using System.Net;
```

```
namespace MJsniffer
{
    public enum Protocol
    {
        TCP = 6,
        UDP = 17,
        Unknown = -1
    };

    public partial class MJsnifferForm : Form
    {
        private Socket mainSocket; // Socket który przechwytuje
wszystkie pakiety
        private byte[] byteData = new byte[4096];
        private bool bContinueCapturing = false; // flaga która
sprawdza czy pakiety zostały złapane poprawnie
        private delegate void AddTreeNode(TreeNode node);

        public MJsnifferForm()
        {
            InitializeComponent();
        }

        private void btnStart_Click(object sender, EventArgs e)
        {
            if (cmbInterfaces.Text == "")
            {
                MessageBox.Show("Select an Interface to capture the
packets.", "Sniffer",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            try
            {
                if (!bContinueCapturing)
                {
                    btnStart.Text = "&Stop";
                    bContinueCapturing = true;

                    mainSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Raw, ProtocolType.IP);
                    mainSocket.Bind(new
IPEndPoint(IPAddress.Parse(cmbInterfaces.Text), 0));

                    mainSocket.SetSocketOption(SocketOptionLevel.IP,
SocketOptionName.HeaderIncluded, true);

                    byte[] byTrue = new byte[4] { 1, 0, 0, 0 };
                    byte[] byOut = new byte[4] { 1, 0, 0, 0 };
```

```
                    mainSocket.IOControl(IOControlCode.ReceiveAll,
byTrue, byOut);

                    mainSocket.BeginReceive(byteData, 0,
byteData.Length, SocketFlags.None,
                        new AsyncCallback(OnReceive), null);
                }
                else
                {
                    btnStart.Text = "&Start";
                    bContinueCapturing = false;
                    mainSocket.Close();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Sniffer",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        private void OnReceive(IAsyncResult ar)
        {
            try
            {
                int nReceived = mainSocket.EndReceive(ar);
                ParseData(byteData, nReceived);

                if (bContinueCapturing)
                {
                    byteData = new byte[4096];
                    mainSocket.BeginReceive(byteData, 0,
byteData.Length, SocketFlags.None,
                        new AsyncCallback(OnReceive), null);
                }
            }
            catch (ObjectDisposedException) { }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Sniffer",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        private void ParseData(byte[] byteData, int nReceived)
        {
            TreeNode rootNode = new TreeNode();
            IPHeader ipHeader = new IPHeader(byteData, nReceived);
            TreeNode ipNode = MakeIPTreeNode(ipHeader);
            rootNode.Nodes.Add(ipNode);
```

```
            switch (ipHeader.ProtocolType)
            {
                case Protocol.TCP:
                    TCPHeader tcpHeader = new TCPHeader(ipHeader.Data,
ipHeader.MessageLength);
                    TreeNode tcpNode = MakeTCPTreeNode(tcpHeader);
                    rootNode.Nodes.Add(tcpNode);

                    if (tcpHeader.DestinationPort == "53" ||
tcpHeader.SourcePort == "53")
                    {
                        TreeNode dnsNode =
MakeDNSTreeNode(tcpHeader.Data, (int)tcpHeader.MessageLength);
                        rootNode.Nodes.Add(dnsNode);
                    }
                    break;

                case Protocol.UDP:
                    UDPHeader udpHeader = new UDPHeader(ipHeader.Data,
(int)ipHeader.MessageLength);
                    TreeNode udpNode = MakeUDPTreeNode(udpHeader);
                    rootNode.Nodes.Add(udpNode);

                    if (udpHeader.DestinationPort == "53" ||
udpHeader.SourcePort == "53")
                    {
                        TreeNode dnsNode =
MakeDNSTreeNode(udpHeader.Data,
                            Convert.ToInt32(udpHeader.Length) - 8);
                        rootNode.Nodes.Add(dnsNode);
                    }
                    break;

                case Protocol.Unknown:
                    break;
            }

            AddTreeNode addTreeNode = new AddTreeNode(OnAddTreeNode);
            rootNode.Text = ipHeader.SourceAddress.ToString() + " - " +
ipHeader.DestinationAddress.ToString();
            treeView.Invoke(addTreeNode, new object[] { rootNode });
        }

        private TreeNode MakeIPTreeNode(IPHeader ipHeader)
        {
            TreeNode ipNode = new TreeNode("IP");
            ipNode.Nodes.Add("Ver: " + ipHeader.Version);
            ipNode.Nodes.Add("Header Length: " +
ipHeader.HeaderLength);
            ipNode.Nodes.Add("Differentiated Services: " +
ipHeader.DifferentiatedServices);
```

```
            ipNode.Nodes.Add("Total Length: " + ipHeader.TotalLength);
            ipNode.Nodes.Add("Identification: " +
ipHeader.Identification);
            ipNode.Nodes.Add("Flags: " + ipHeader.Flags);
            ipNode.Nodes.Add("Fragmentation Offset: " +
ipHeader.FragmentationOffset);
            ipNode.Nodes.Add("Time to live: " + ipHeader.TTL);

            string protocolStr = ipHeader.ProtocolType switch
            {
                Protocol.TCP => "TCP",
                Protocol.UDP => "UDP",
                _ => "Unknown"
            };
            ipNode.Nodes.Add("Protocol: " + protocolStr);

            ipNode.Nodes.Add("Checksum: " + ipHeader.Checksum);
            ipNode.Nodes.Add("Source: " +
ipHeader.SourceAddress.ToString());
            ipNode.Nodes.Add("Destination: " +
ipHeader.DestinationAddress.ToString());

            return ipNode;
        }

        private TreeNode MakeTCPTreeNode(TCPHeader tcpHeader)
        {
            TreeNode tcpNode = new TreeNode("TCP");
            tcpNode.Nodes.Add("Source Port: " + tcpHeader.SourcePort);
            tcpNode.Nodes.Add("Destination Port: " +
tcpHeader.DestinationPort);
            tcpNode.Nodes.Add("Sequence Number: " +
tcpHeader.SequenceNumber);

            if (!string.IsNullOrEmpty(tcpHeader.AcknowledgementNumber))
                tcpNode.Nodes.Add("Acknowledgement Number: " +
tcpHeader.AcknowledgementNumber);

            tcpNode.Nodes.Add("Header Length: " +
tcpHeader.HeaderLength);
            tcpNode.Nodes.Add("Flags: " + tcpHeader.Flags);
            tcpNode.Nodes.Add("Window Size: " + tcpHeader.WindowSize);
            tcpNode.Nodes.Add("Checksum: " + tcpHeader.Checksum);

            if (!string.IsNullOrEmpty(tcpHeader.UrgentPointer))
                tcpNode.Nodes.Add("Urgent Pointer: " +
tcpHeader.UrgentPointer);

            return tcpNode;
        }
```

```
        private TreeNode MakeUDPTreeNode(UDPHeader udpHeader)
        {
            TreeNode udpNode = new TreeNode("UDP");
            udpNode.Nodes.Add("Source Port: " + udpHeader.SourcePort);
            udpNode.Nodes.Add("Destination Port: " +
udpHeader.DestinationPort);
            udpNode.Nodes.Add("Length: " + udpHeader.Length);
            udpNode.Nodes.Add("Checksum: " + udpHeader.Checksum);
            return udpNode;
        }

        private TreeNode MakeDNSTreeNode(byte[] byteData, int nLength)
        {
            DNSHeader dnsHeader = new DNSHeader(byteData, nLength);
            TreeNode dnsNode = new TreeNode("DNS");
            dnsNode.Nodes.Add("Identification: " +
dnsHeader.Identification);
            dnsNode.Nodes.Add("Flags: " + dnsHeader.Flags);
            dnsNode.Nodes.Add("Questions: " +
dnsHeader.TotalQuestions);
            dnsNode.Nodes.Add("Answer RRs: " +
dnsHeader.TotalAnswerRRs);
            dnsNode.Nodes.Add("Authority RRs: " +
dnsHeader.TotalAuthorityRRs);
            dnsNode.Nodes.Add("Additional RRs: " +
dnsHeader.TotalAdditionalRRs);
            return dnsNode;
        }

        private void OnAddTreeNode(TreeNode node)
        {
            treeView.Nodes.Add(node);
        }

        private void SnifferForm_Load(object sender, EventArgs e)
        {
            string strIP = null;
            IPHostEntry hostEntry =
Dns.GetHostEntry(Dns.GetHostName());

            if (hostEntry.AddressList.Length > 0)
            {
                foreach (IPAddress ip in hostEntry.AddressList)
                {
                    strIP = ip.ToString();
                    cmbInterfaces.Items.Add(strIP);
                }
            }
        }

        private void SnifferForm_FormClosing(object sender,
```

```
FormClosingEventArgs e)
        {
            if (bContinueCapturing)
            {
                mainSocket.Close();
            }
        }

        private void treeView_AfterSelect(object sender,
TreeViewEventArgs e) { }

        private void label1_Click(object sender, EventArgs e) { }
    }
}
```