Packet Sniffer w C#

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

{

//label1.Text = string.Parse(cmbInterfaces.Text) ;

private Socket mainSocket; //Socket który przechwytuje wszystkie
pakiety
```

```
private byte[] byteData = new byte[4096];

private bool bContinueCapturing = false; //flaga która sprawdza czy
pakiety zostaly zlapane poprawnie


private delegate void AddTreeNode(TreeNode node);


public MJsnifferForm()

{

InitializeComponent();

}


private void btnStart_Click(object sender, EventArgs e) // przy
nacisnieciu przycisku

{

if (cmbInterfaces.Text == "")

{

MessageBox.Show("Select an Interface to capture the packets.",
"Sniffer",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

if (!bContinueCapturing)

{

//Rozpocznij przechwytywanie


btnStart.Text = "&Stop";


bContinueCapturing = true;
```

```
//zeby przechwytywaæ pakiety z socketu musi to byæ tzw raw socket,

//w tym przypadku potrzebujemy interfejsu który ma adres IP jest w
podsieci i mozemy //przechwytywac z niego tylko dane z warstwy 3


mainSocket = new Socket(AddressFamily.InterNetwork,

SocketType.Raw, ProtocolType.IP);

//podwi¹zanie socketa do konretnego adresu

mainSocket.Bind(new IPEndPoint(IPAddress.Parse(cmbInterfaces.Text),
0));


//kilka opcji

mainSocket.SetSocketOption(SocketOptionLevel.IP, //szuka pakietów IP

SocketOptionName.HeaderIncluded, //szuka naglowka

true); //potwierdzenie ze to chcemy


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

byte[] byOut = new byte[4]{1, 0, 0, 0}; //przechwytywanie wychodzacych
pakietów


//Socket.IOControl jest to biblioteka która dziala podobnie do
biblioteki WinSock przez co //mozna latwiej programowac

mainSocket.IOControl(IOControlCode.ReceiveAll,

byTrue,

byOut);


//Odbieranie pakietów asynchronicznie

mainSocket.BeginReceive(byteData, 0, byteData.Length, SocketFlags.None,

new AsyncCallback(OnReceive), null);

}
```

```
else

{

btnStart.Text = "&Start";

bContinueCapturing = false;

//po zakonczeniu przechwytywania zamknij socket

mainSocket.Close ();

}

}

catch (Exception ex) //je¿eli bed¹ jakieœ b³êdy to wyœwietlij okienko

{

MessageBox.Show(ex.Message, "Sniffer", MessageBoxButtons.OK,
MessageBoxIcon.Error);

}

}


private void OnReceive(IAsyncResult ar)

{

try

{

int nReceived = mainSocket.EndReceive(ar);


//Analiza przychodzacych bajtów

ParseData (byteData, nReceived);


if (bContinueCapturing)

{

byteData = new byte[4096];

//kolejne przyzwanie begin receive zebyœmy nie przerywali sluchania
```

```
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();


//wszystkie protokoly sa enkasulowane w pakiecie IP

//wiec zaczynamy parsowac header IP

IPHeader ipHeader = new IPHeader(byteData, nReceived);


TreeNode ipNode = MakeIPTreeNode(ipHeader);

rootNode.Nodes.Add(ipNode);


//potem parsujemy pole dane datagramu ip

switch (ipHeader.ProtocolType)

{
```

```
case Protocol.TCP:


TCPHeader tcpHeader = new TCPHeader(ipHeader.Data, //IPHeader.Data
przechowuje //dane z pakietu IP

ipHeader.MessageLength);//dlugosc sekcji danych


TreeNode tcpNode = MakeTCPTreeNode(tcpHeader);


rootNode.Nodes.Add(tcpNode);


//jezeli port to 53 to mamy do czynienia z DNSem

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, //IPHeader.Data
przechowuje //dane z pakietu IP

(int)ipHeader.MessageLength);//dlugosc sekcji danych


TreeNode udpNode = MakeUDPTreeNode(udpHeader);


rootNode.Nodes.Add(udpNode);


//jezeli port to 53 to mamy do czynienia z DNSem

if (udpHeader.DestinationPort == "53" || udpHeader.SourcePort == "53")
```

```
{

TreeNode dnsNode = MakeDNSTreeNode(udpHeader.Data,

//dlugosc naglowka UDP zawsze 8 bajtów wiec odejmujemy //to od

//dlugosci calych danych w pakiecie

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();


//to powoduje te dane sa wyswietlane w postaci takiego wielo zlozonego
drzewka

treeView.Invoke(addTreeNode, new object[] {rootNode});

}


//fukcja która zwraca dane z pakietu IP w formie drzewka

private TreeNode MakeIPTreeNode(IPHeader ipHeader)

{

TreeNode ipNode = new TreeNode();
```

```
ipNode.Text = "IP";

ipNode.Nodes.Add ("Ver: " + ipHeader.Version);

ipNode.Nodes.Add ("Header Length: " + ipHeader.HeaderLength);

ipNode.Nodes.Add ("Differntiated 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);

switch (ipHeader.ProtocolType)

{

case Protocol.TCP:

ipNode.Nodes.Add ("Protocol: " + "TCP");

break;

case Protocol.UDP:

ipNode.Nodes.Add ("Protocol: " + "UDP");

break;

case Protocol.Unknown:

ipNode.Nodes.Add ("Protocol: " + "Unknown");

break;

}

ipNode.Nodes.Add("Checksum: " + ipHeader.Checksum);

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

ipNode.Nodes.Add("Destination: " +
ipHeader.DestinationAddress.ToString());
```

```
return ipNode;

}

//fukcja która zwraca dane z pakietu TCP w formie drzewka

private TreeNode MakeTCPTreeNode(TCPHeader tcpHeader)

{

TreeNode tcpNode = new TreeNode();


tcpNode.Text = "TCP";


tcpNode.Nodes.Add("Source Port: " + tcpHeader.SourcePort);

tcpNode.Nodes.Add("Destination Port: " + tcpHeader.DestinationPort);

tcpNode.Nodes.Add("Sequence Number: " + tcpHeader.SequenceNumber);


if (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 (tcpHeader.UrgentPointer != "")

tcpNode.Nodes.Add("Urgent Pointer: " + tcpHeader.UrgentPointer);


return tcpNode;

}
```

```
//funkcja która zwraca dane z pakietu UDP w formie drzewka

private TreeNode MakeUDPTreeNode(UDPHeader udpHeader)

{

TreeNode udpNode = new TreeNode();


udpNode.Text = "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;

}


//funkcja która zwraca dane z pakietu DNS w formie drzewka

private TreeNode MakeDNSTreeNode(byte[] byteData, int nLength)

{

DNSHeader dnsHeader = new DNSHeader(byteData, nLength);


TreeNode dnsNode = new TreeNode();


dnsNode.Text = "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 HosyEntry = Dns.GetHostEntry((Dns.GetHostName()));

if (HosyEntry.AddressList.Length > 0)

{

foreach (IPAddress ip in HosyEntry.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)

{


}

}

}
```