Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Информационные сети. Основы безопасности»

|  |  |
| --- | --- |
|  |  |

**ОТЧЕТ**

к лабораторной работе № 3

на тему «Атаки при установке *TCP*-соединения и протоколов прикладного уровня»

Выполнил             Н. В. Климкович

Проверил                           Е. А. Лещенко

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_heading=h.30j0zll)

[2 Краткие теоретические сведения 4](#_heading=h.1fob9te)

[3 Результаты выполнения лабораторной работы 12](#_heading=h.tyjcwt)

[Выводы](#_heading=h.3dy6vkm) 13

[Список использованных источников 14](#_heading=h.1t3h5sf)

[Приложение А (обязательное) Листинг исходного кода 15](#_heading=h.4d34og8)

# **1 ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения данной лабораторной работы является создание приложения, реализующее атаки на протокол при установке *TCP*-соединения и в рамках заданного протокола прикладного уровня. В интерфейсе приложения должны быть наглядно представлены исходные данные, данные, передаваемые по сети каждой из сторон, проверки, выполняемые каждый из участников.

# **2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Типы адресов: физический (*MAC*-адрес), сетевой (*IP*-адрес), символьный (*DNS*-имя).

Компьютер в сети TCP/IP может иметь адреса трех уровней (но не менее двух):

1 Локальный адрес компьютера. Для узлов, входящих в локальные сети - это *МАС*-адрес сетевого адаптера. Эти адреса назначаются производителями оборудования и являются уникальными адресами.

2 *IP*-адрес, состоящий из 4 байт, например, 109.26.17.100. Этот адрес используется на сетевом уровне. Он назначается администратором во время конфигурирования компьютеров и маршрутизаторов.

3 Символьный идентификатор-имя (*DNS*), например, www.kstu.ru.

*IPv4* - адрес является уникальным 32-битным идентификатором   
*IP*-интерфейса в Интернет.

*IPv6* - адрес является уникальным 128-битным идентификатором   
*IP*-интерфейса в Интернет, иногда называют *Internet*-2, адресного пространства *IPv4* уже стало не хватать, поэтому постепенно вводят новый стандарт.

*IP*-адреса принято записывать разбивкой всего адреса по октетам (8), каждый октет записывается в виде десятичного числа, числа разделяются точками. Пример переводы адреса из двоичной системы в десятичную представлен на рисунке 2.1.

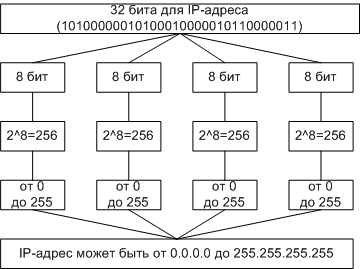


Рисунок 2.1 – Перевод адреса из двоичной системы в десятичную

*IPv6* - адрес является уникальным 128-битным идентификатором   
*IP*-интерфейса в Интернет, иногда называют *Internet*-2, адресного пространства *IPv4* уже стало не хватать, поэтому постепенно вводят новый стандарт.

IP-адрес хоста состоит из номера IP-сети, который занимает старшую область адреса, и номера хоста в этой сети, который занимает младшую часть:

1 160.81.5.131 - IP-адрес.

2 160.81.5. - номер сети.

3 131 - номер хоста.

TCP/IP - собирательное название для набора (стека) сетевых протоколов разных уровней, используемых в Интернет. Особенности TCP/IP:

1 Открытые стандарты протоколов, разрабатываемые независимо от программного и аппаратного обеспечения.

2 Независимость от физической среды передачи.

3 Система уникальной адресации.

4 Стандартизованные протоколы высокого уровня для распространенных пользовательских сервисов.

Стек протоколов TCP/IP представлен на рисунке 2.2.

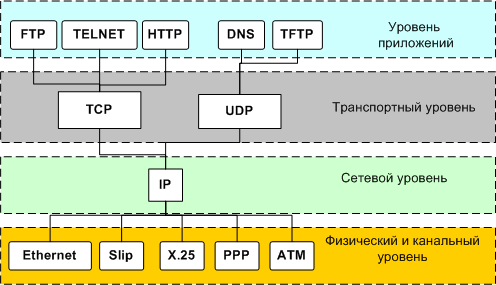


Рисунок 2.2 – Стек протоколов TCP/IP

Стек протоколов TCP/IP делится на 4 уровня: прикладной, транспортный, межсетевой, физический и канальный.

Позже была принята 7-ми уровневая модель ISO.

Данные передаются в пакетах. Пакеты имеют заголовок и окончание, которые содержат служебную информацию. Данные, более верхних уровней вставляются, в пакеты нижних уровней. Пример инкапсуляции пакетов в стеке TCP/IP представлен на рисунке 2.3.

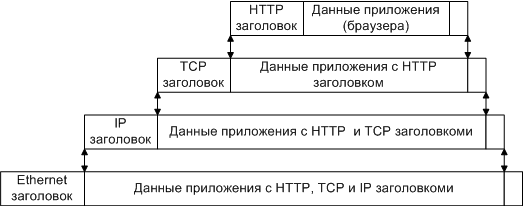


Рисунок 2.3 – Пример инкапсуляции пакетов в стеке TCP/IP

Физический и канальный уровень. Стек TCP/IP не подразумевает использования каких-либо определенных протоколов уровня доступа к среде передачи и физических сред передачи данных. От уровня доступа к среде передачи требуется наличие интерфейса с модулем IP, обеспечивающего передачу IP-пакетов. Также требуется обеспечить преобразование IP-адреса узла сети, на который передается IP-пакет, в MAC-адрес. Часто в качестве уровня доступа к среде передачи могут выступать целые протокольные стеки, тогда говорят об IP поверх ATM, IP поверх IPX, IP поверх X.25 и т.п.

Межсетевой уровень и протокол IP. Основу этого уровня составляет IP-протокол.

IP (Internet Protocol) – интернет-протокол. Первый стандарт IPv4 определен в RFC-760 (DoD standard Internet Protocol J. Postel Jan-01-1980). Последняя версия IPv6 - RFC-2460 (Internet Protocol, Version 6 (IPv6) Specification S. Deering, R. Hinden December 1998).

Основные задачи: адресация, маршрутизация, фрагментация датаграмм, передача данных.

Протокол IP доставляет блоки данных от одного IP-адреса к другому.

Программа, реализующая функции того или иного протокола, часто называется модулем, например, “IP-модуль”, “модуль TCP”.

Когда модуль IP получает IP-пакет с нижнего уровня, он проверяет IP-адрес назначения:

1 Если IP-пакет адресован данному компьютеру, то данные из него передаются на обработку модулю вышестоящего уровня (какому конкретно - указано в заголовке IP-пакета).

2 Если же адрес назначения IP-пакета - чужой, то модуль IP может принять два решения: первое - уничтожить IP-пакет, второе - отправить его дальше к месту назначения, определив маршрут следования - так поступают маршрутизаторы.

Также может потребоваться, на границе сетей с различными характеристиками, разбить IP-пакет на фрагменты (фрагментация), а потом собрать в единое целое на компьютере-получателе.

Если модуль IP по какой-либо причине не может доставить IP-пакет, он уничтожается. При этом модуль IP может отправить компьютеру-источнику этого IP-пакета уведомление об ошибке; такие уведомления отправляются с помощью протокола ICMP, являющегося неотъемлемой частью модуля IP. Более никаких средств контроля корректности данных, подтверждения их доставки, обеспечения правильного порядка следования IP-пакетов, предварительного установления соединения между компьютерами протокол IP не имеет. Эта задача возложена на транспортный уровень. Структура дейтограммы IP. Слова по 32 бита представлена на рисунке 2.4.

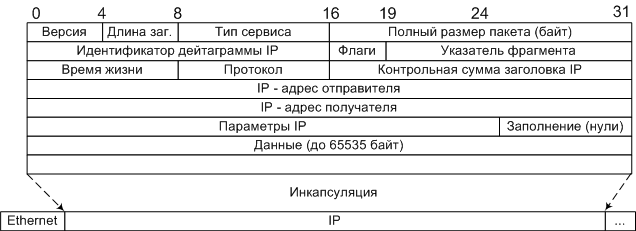


Рисунок 2.4 – Структура дейтограммы IP. Слова по 32 бита

Версия - версия протокола IP (например, 4 или 6).

Длина заголовка - длина заголовка IP-пакета.

Тип сервиса (TOS - type of service) - Тип сервиса.

TOS играет важную роль в маршрутизации пакетов. Интернет не гарантирует запрашиваемый TOS, но многие маршрутизаторы учитывают эти запросы при выборе маршрута (протоколы OSPF и IGRP).

Идентификатор дейтаграммы, флаги (3 бита) и указатель фрагмента - используются для распознавания пакетов, образовавшихся путем фрагментации исходного пакета.

Время жизни (TTL - time to live) - каждый маршрутизатор уменьшает его на 1, чтобы пакеты не блуждали вечно.

Протокол - Идентификатор протокола верхнего уровня указывает, какому протоколу верхнего уровня принадлежит пакет (например: TCP, UDP).

Коды некоторые протоколов RFC-1700 (1994) представлены на рисунке 2.5.



Рисунок 2.5 – Коды некоторые протоколов RFC-1700 (1994)

Протокол IP является маршрутизируемый, для его маршрутизации нужна маршрутная информация.

Маршрутная информация, может быть:

1 Статической (маршрутные таблицы прописываются вручную).

2 Динамической (маршрутную информацию распространяют специальные протоколы).

Протоколы транспортного уровня обеспечивают прозрачную доставку данных между двумя прикладными процессами. Процесс, получающий или отправляющий данные с помощью транспортного уровня, идентифицируется на этом уровне номером, который называется номером порта. Таким образом, роль адреса отправителя и получателя на транспортном уровне выполняет номер порта (или проще - порт).

Анализируя заголовок своего пакета, полученного от межсетевого уровня, транспортный модуль определяет по номеру порта получателя, какому из прикладных процессов направлены данные, и передает эти данные соответствующему прикладному процессу. Номера портов получателя и отправителя записываются в заголовок транспортным модулем, отправляющим данные; заголовок транспортного уровня содержит также и другую служебную информацию; формат заголовка зависит от используемого транспортного протокола.

На транспортном уровне работают два основных протокола: UDP и TCP.

Протокол надежной доставки сообщений TCP (Transfer Control  
Protocol) – протокол контроля передачи, протокол TCP применяется в тех случаях, когда требуется гарантированная доставка сообщений.

Первая и последняя версия TCP - RFC-793 (Transmission Control Protocol J. Postel Sep-01-1981).

Основные особенности:

1 Устанавливается соединение.

2 Данные передаются сегментами. Модуль TCP нарезает большие сообщения (файлы) на пакеты, каждый из которых передается отдельно, на приемнике наоборот файлы собираются. Для этого нужен порядковый номер (Sequence Number - SN) пакета:

3 Посылает запрос на следующий пакет, указывая его номер в поле "Номер подтверждения" (AS). Тем самым, подтверждая получение предыдущего пакета.

4 Делает проверку целостности данных, если пакет битый посылает повторный запрос.

Структура дейтограммы TCP, слова по 32 бита представлены на  
рисунке 2.6.

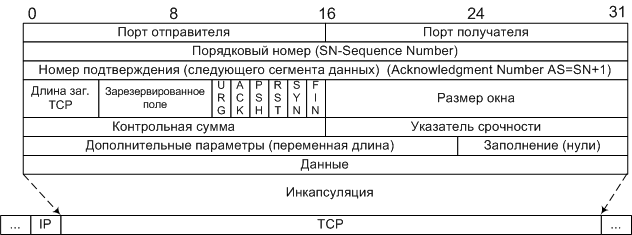


Рисунок 2.6 – Структура дейтограммы TCP. Слова по 32 бита

Длина заголовка - задается словами по 32бита.

Размер окна - количество байт, которые готов принять получатель без подтверждения.

Контрольная сумма - включает псевдо заголовок, заголовок и данные.

Указатель срочности - указывает последний байт срочных данных, на которые надо немедленно реагировать.

URG - флаг срочности, включает поле "Указатель срочности", если =0 то поле игнорируется.

ACK - флаг подтверждение, включает поле "Номер подтверждения, если = 0 то поле игнорируется.

PSH - флаг требует выполнения операции push, модуль TCP должен срочно передать пакет программе.

RST - флаг прерывания соединения, используется для отказа в соединении

SYN - флаг синхронизация порядковых номеров, используется при установлении соединения.

FIN - флаг окончание передачи со стороны отправителя

Назначение портов

По номеру порта транспортные протоколы определяют, какому приложению передать содержимое пакетов.

Порты могут принимать значение от 0-65535 (два байта 2^16).

Номера портам присваиваются таким образом: имеются стандартные номера (например, номер 21 закреплен за сервисом FTP, 23 - за telnet, 80 - за HTTP), а менее известные приложения пользуются произвольно выбранными локальными номерами (как правило, больше>1024), некоторые из них также зарезервированы:

1 Программа Ping.

2 Программа для проверки соединения и работы с удаленным хостом.

3 Программа TraceRoute - позволяет проверить маршрут до удаленного хоста.

4 Программа nmap - позволяет сканировать порты.

5 Работу порта, также можно проверить с помощью telnet.

Некоторые заданные порты RFC-1700 (1994) 43% представлены на рисунке 2.7.

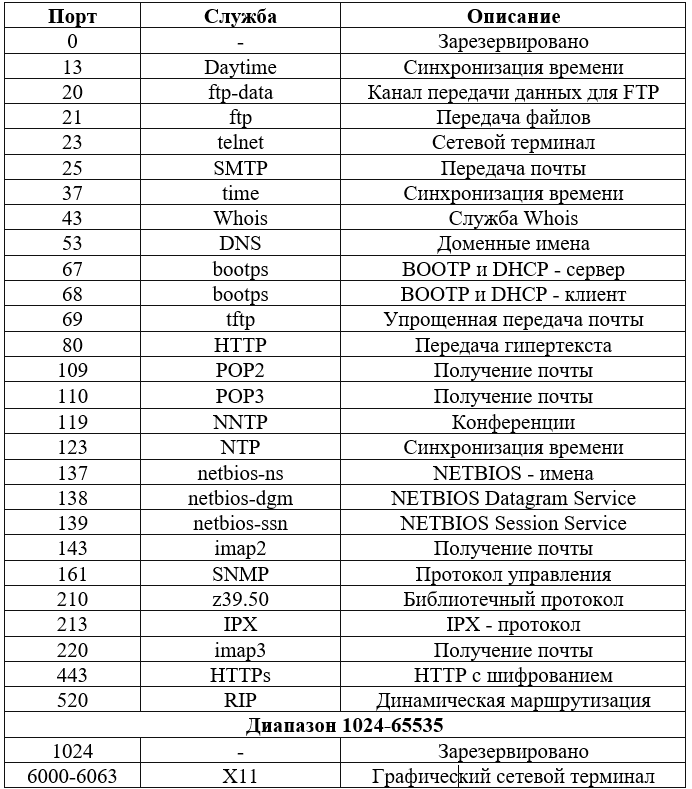


Рисунок 2.7 – Некоторые заданные порты RFC-1700 (1994) 43%

# **3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ**

В ходе выполнения лабораторной было создано приложение, реализующее атаки на протокол при установке *TCP*-соединения и в рамках заданного протокола прикладного уровня. В ходе работы программа производит атаку на сервер путем подключения большого количества пользователей. У сервера установлено ограничение на количество одновременных подключений флага SYN. Результат атаки на флаг SYN представлен на рисунке 3.1.

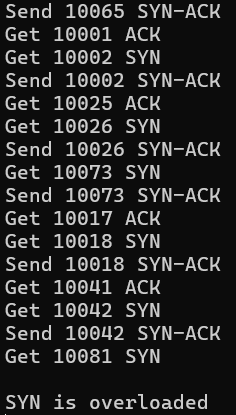


Рисунок 3.1 – Результат атаки на флаг SYN

Также у сервера установлено ограничение на количество подключенных пользователей. Результат атаки на количество подключенных пользователей представлен на рисунке 3.2.

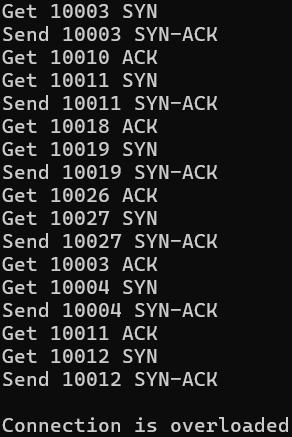


Рисунок 3.2 – Результат атаки на количество подключенных пользователей

# **ВЫВОДЫ**

В ходе лабораторной работы было создано приложение, реализующее атаки на протокол при установке *TCP*-соединения и в рамках заданного протокола прикладного уровня. В интерфейсе приложения наглядно представлены исходные данные, данные, передаваемые по сети каждой из сторон, и проверки, выполняемые каждым из участников.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] What is a SYN flood attack? [Электронный ресурс]. – Режим доступа: https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/. – Дата доступа: 27.02.2024.

[2] What is Transmission Control Protocol TCP/IP? [Электронный  
ресурс]. – Режим доступа: https://www.fortinet.com/. – Дата доступа: 28.02.2024.

## **ПРИЛОЖЕНИЕ А**

## **(обязательное)**

## **Листинг исходного кода**

Листинг 1 – Программный код класса *Attack*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace ISOB3

{

class Attack

{

private static EndPoint server = new IPEndPoint(IPAddress.Parse(Settings.host), Settings.ServerPort);

public static void Run()

{

Socket serverSocket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);

try

{

serverSocket.Bind(new IPEndPoint(IPAddress.Parse(Settings.host), 10000));

Parallel.For(1, 100, i =>

{

TCP tcp = new TCP(10000 + i, Settings.ServerPort, syn: true);

serverSocket.SendTo(tcp.ToByteArray(), server);

Socket ClientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);

try

{

ClientSocket.Bind(new IPEndPoint(IPAddress.Parse(Settings.host), 10000 + i));

byte[] data = new byte[100];

var countByte = ClientSocket.Receive(data);

var message = TCP.GetByte(TCP.BitSection(data, countByte));

var response = new TCP(message.DestinationPort, message.SenderPort, ack: true);

ClientSocket.SendTo(response.ToByteArray(), server);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

ClientSocket.Close();

}

});

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

serverSocket.Close();

}

}

}

}

Листинг 2 – Программный код класса *Program*

using System;

using System.Threading.Tasks;

namespace ISOB3

{

class Program

{

static void Main(string[] args)

{

Task.Run(() => Server.Listener());

Task.Run(() => Attack.Run());

Console.ReadLine();

}

}

}

Листинг 3 – Программный код класса *Server*

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

namespace ISOB3

{

static class Server

{

static Socket ListenSocket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);

static IPEndPoint ListenPoint = new IPEndPoint(IPAddress.Parse(Settings.host), Settings.ServerPort);

static List<int> Syn = new List<int>();

static List<int> Connections = new List<int>();

static int MaxSyn = 10;

static int MaxConnection = 10;

public static void Listener()

{

try

{

ListenSocket.Bind(ListenPoint);

while (true)

{

byte[] data = new byte[1024];

int len = ListenSocket.Receive(data);

TCP message = TCP.GetByte(TCP.BitSection(data, len));

if (message.DestinationPort != Settings.ServerPort) continue;

else if (Connections.Contains(message.SenderPort)) continue;

else if (Syn.Contains(message.SenderPort) && message.ACK)

{

Connections.Add(message.SenderPort);

Syn.Remove(message.SenderPort);

if (Connections.Count > MaxConnection)

throw new Exception("Connection is overloaded");

Console.WriteLine($"Get {message.SenderPort} ACK");

}

else if (!Syn.Contains(message.SenderPort) && message.SYN)

{

Console.WriteLine($"Get {message.SenderPort} SYN");

Socket sender = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);

sender.Bind(new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9999));

try

{

var response = new TCP(Settings.ServerPort, message.SenderPort, syn: true, ack: true);

sender.SendTo(response.ToByteArray(), new IPEndPoint(IPAddress.Parse(Settings.host), message.SenderPort));

Syn.Add(message.SenderPort);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

sender.Close();

}

if (Syn.Count > MaxSyn)

throw new Exception("SYN is overloaded");

Console.WriteLine($"Send {message.SenderPort} SYN-ACK");

}

}

}

catch (Exception ex)

{

Console.WriteLine($"\n{ex.Message}");

}

finally

{

ListenSocket.Close();

}

}

}

}

Листинг 4 – Программный код класса *Settings*

namespace ISOB3

{

public static class Settings

{

public static readonly string host = "127.0.0.1";

public static readonly int ServerPort = 8888;

}

}

Листинг 5 – Программный код класса *TCP*

using System.Collections;

using System.Text;

namespace ISOB3

{

class TCP

{

public int SenderPort;

public int DestinationPort;

public int SequenceNumber;

public int AcknowledgmentNumber;

public bool URG;

public bool ACK;

public bool PSH;

public bool RST;

public bool SYN;

public bool FIN;

public string OtherData;

public TCP(int sourcePort = 0, int destinationPort = 0, int sequenceNumber = 0, int acknowledgmentNumber = 0,

bool urg = false, bool ack = false, bool psh = false, bool rst = false,

bool syn = false, bool fin = false, string data = "")

{

SenderPort = sourcePort;

DestinationPort = destinationPort;

SequenceNumber = sequenceNumber;

AcknowledgmentNumber = acknowledgmentNumber;

URG = urg;

ACK = ack;

PSH = psh;

RST = rst;

SYN = syn;

FIN = fin;

OtherData = data;

}

public static TCP GetByte(byte[] src)

{

var source = new BitArray(src);

return new TCP(sourcePort: BitToInt(Slice(source, 0, 16)),

destinationPort: BitToInt(Slice(source, 16, 16)),

sequenceNumber: BitToInt(Slice(source, 32, 32)),

acknowledgmentNumber: BitToInt(Slice(source, 64, 32)),

urg: source[106],

ack: source[107],

psh: source[108],

rst: source[109],

syn: source[110],

fin: source[111],

data: BitToStr(Slice(source, 160, source.Length - 160)));

}

private BitArray TCPToBitArray()

{

BitArray res = new BitArray(160 + OtherData.Length \* 8);

CopyBit(Convert16Bit(SenderPort), res, 0);

CopyBit(Convert16Bit(DestinationPort), res, 16);

CopyBit(new BitArray(new int[] { SequenceNumber }), res, 32);

CopyBit(new BitArray(new int[] { AcknowledgmentNumber }), res, 64);

res[106] = URG;

res[107] = ACK;

res[108] = PSH;

res[109] = RST;

res[110] = SYN;

res[111] = FIN;

CopyBit(new BitArray(Encoding.UTF8.GetBytes(OtherData)), res, 160);

return res;

}

public byte[] ToByteArray()

{

BitArray bitArray = TCPToBitArray();

byte[] data = new byte[bitArray.Length / 8];

bitArray.CopyTo(data, 0);

return data;

}

private static BitArray Convert16Bit(int value)

{

BitArray res = new BitArray(16);

for (int i = 0; value > 0 && i < res.Length; i++)

{

res[i] = value % 2 == 0 ? false : true;

value /= 2;

}

return res;

}

private static int BitToInt(BitArray source)

{

int[] data = new int[1];

source.CopyTo(data, 0);

return data[0];

}

private static string BitToStr(BitArray source)

{

byte[] data = new byte[source.Length / 8];

source.CopyTo(data, 0);

return Encoding.UTF8.GetString(data);

}

private static BitArray Slice(BitArray source, int start, int count)

{

BitArray slice = new BitArray(count);

for (int i = 0; i < count; i++)

{

slice[i] = source[i + start];

}

return slice;

}

private void CopyBit(BitArray source, BitArray dest, int start)

{

for (int i = 0; i < source.Length; i++)

{

dest[start + i] = source[i];

}

}

public static byte[] BitSection(byte[] source, int len)

{

byte[] res = new byte[len];

for (int i = 0; i < len; i++)

{

res[i] = source[i];

}

return res;

}

}

}