Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа № 3

**Атаки при установке TCP-соединения и протоколов прикладного уровня**

Выполнил студент гр. 853505

Лазарева Е.В.

Проверил

Протько М.И.

Минск, 2021

1. **Введение**

Целью данной лабораторной работы было изучить атаки при установке TCP-соединения и протоколов прикладного уровня и создать приложение, реализующее атаки на протокол при установке TCP-соединения и в рамках заданного протокола прикладного уровня.

1. **Теоретические сведения**

**Адресация в сети Internet.**

**Типы адресов.**

Типы адресов:

1. **Физический (MAC-адрес)**
2. **Сетевой (IP-адрес)**
3. **Символьный (DNS-имя)**

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

* Локальный адрес компьютера. Для узлов, входящих в локальные сети - это МАС-адрес сетевого адаптера. Эти адреса назначаются производителями оборудования и являются уникальными адресами.
* 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.
* Символьный идентификатор-имя (DNS), например, [www.kstu.ru](http://www.kstu.ru/).

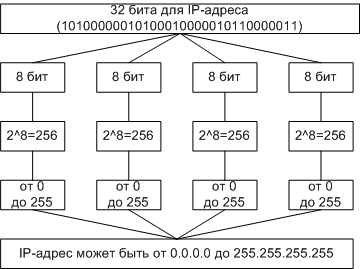
**IP-адреса**

**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. Например, адрес

10100000010100010000010110000011   
записывается как  
10100000.01010001.00000101.10000011 = 160.81.5.131



Перевод адреса из двоичной системы в десятичную

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

160.81.5.131 - IP-адрес

160.81.5. - номер сети

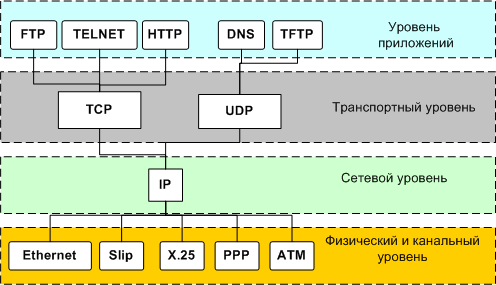
131 - номер хоста

**Базовые протоколы (IP, TCP)**

**Стек протоколов TCP/IP**

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

* Открытые стандарты протоколов, разрабатываемые независимо от программного и аппаратного обеспечения;
* Независимость от физической среды передачи;
* Система уникальной адресации;
* Стандартизованные протоколы высокого уровня для распространенных пользовательских сервисов.



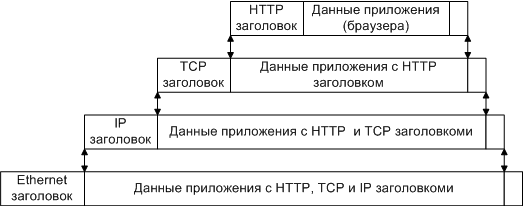
Стек протоколов TCP/IP

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

**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](http://ipm.kstu.ru/internet/doc/rfc/rfc2460.txt) (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.

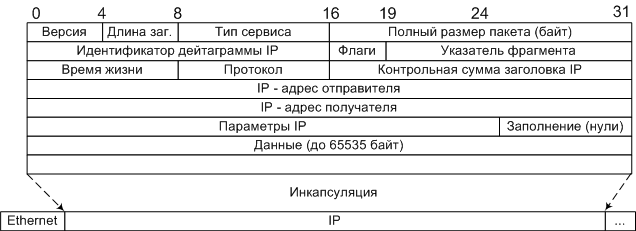
*Программа, реализующая функции того или иного протокола, часто называется модулем, например, “IP-модуль”, “модуль TCP”.*

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

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

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

**Длина заг.** - длина заголовка IP-пакета.

**Тип сервиса** (TOS - type of service) - Тип сервиса (подробнее рассмотрен в лекции 8).

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, что бы пакеты не блуждали вечно.

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

Коды некоторые протоколов [RFC-1700](http://ipm.kstu.ru/internet/doc/rfc/rfc1700.txt) (1994)

|  |  |  |
| --- | --- | --- |
| **Код** | **Протокол** | **Описание** |
| 0 | - | Зарезервировано |
| 1 | ICMP | Протокол контрольных сообщений |
| 2 | IGMP | Групповой протокол управления |
| 4 | IP | IP-поверх-IP (туннели) |
| 6 | TCP | Протокол управления передачей |
| 8 | EGP | Протокол внешней маршрутизации |
| 9 | IGP | Протокол внутренней маршрутизации |
| 17 | UDP | Протокол дейтограмм пользователя |
| 35 | IDRP | Междоменный протокол маршрутизации |
| 36 | XTP | Xpress транспортный протокол |
| 46 | RSVP | Протокол резервирования ресурсов канала |
| 88 | IGRP | внутренний протокол маршрутизации |
| 89 | OSPFIGP | внутренний протокол маршрутизации |
| 97 | ETHERIP | Ethernet-поверх-IP |
| 101-254 | - | не определены |
| 255 | - | зарезервировано |

Маршрутизация.

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

Маршрутная информация, может быть:

* Статической (маршрутные таблицы прописываются вручную)
* Динамической (маршрутную информацию распространяют специальные протоколы)

**Транспортный уровень**

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**

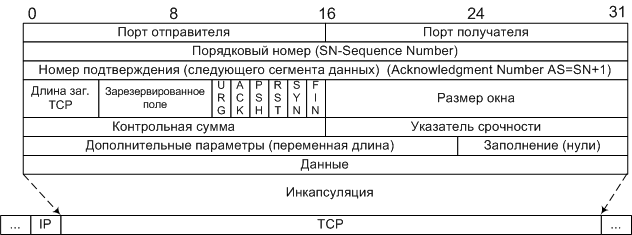
**TCP (Transfer Control Protocol)**– протокол контроля передачи, протокол TCP применяется в тех случаях, когда требуется гарантированная доставка сообщений.

Первая и последняя версия TCP - [RFC-793](http://ipm.kstu.ru/internet/doc/rfc/rfc793.txt) (Transmission Control Protocol J. Postel Sep-01-1981).

Основные особенности:

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

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



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.

Программа Ping

Программа для проверки соединения и работы с удаленным хостом.

Программа TraceRoute - позволяет проверить маршрут до удаленного хоста.

Программа nmap - позволяет сканировать порты.

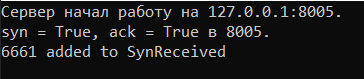
Работу порта, также можно проверить с помощью telnet.

Некоторые заданные порты [RFC-1700](http://ipm.kstu.ru/internet/doc/rfc/rfc1700.txt) (1994) 43%

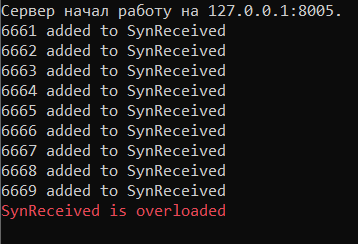
|  |  |  |
| --- | --- | --- |
| **Порт** | **Служба** | **Описание** |
| 0 | - | Зарезервировано |
| 13 | Daytime | Синхронизация времени |
| 20 | ftp-data | Канал передачи данных для FTP |
| 21 | ftp | Передача файлов |
| 23 | telnet | Сетевой терминал |
| 25 | SMTP | Передача почты |
| 37 | time | Синхронизация времени |
| 43 | Whois | Служба Whois |
| 53 | DNS | Доменные имена |
| 67 | bootps | BOOTP и DHCP - сервер |
| 68 | bootps | BOOTP и DHCP - клиент |
| 69 | tftp | Упрощенная передача почты |
| 80 | HTTP | Передача гипертекста |
| 109 | POP2 | Получение почты |
| 110 | POP3 | Получение почты |
| 119 | NNTP | Конференции |
| 123 | NTP | Синхронизация времени |
| 137 | netbios-ns | NETBIOS - имена |
| 138 | netbios-dgm | NETBIOS Datagram Service |
| 139 | netbios-ssn | NETBIOS Session Service |
| 143 | imap2 | Получение почты |
| 161 | SNMP | Протокол управления |
| 210 | z39.50 | Библиотечный протокол |
| 213 | IPX | IPX - протокол |
| 220 | imap3 | Получение почты |
| 443 | HTTPs | HTTP с шифрованием |
| 520 | RIP | Динамическая маршрутизация |
| **Диапазон 1024-65535** | | |
| 1024 | - | Зарезервировано |
| 6000-6063 | X11 | Графический сетевой терминал |

**3. Результат работы программы**

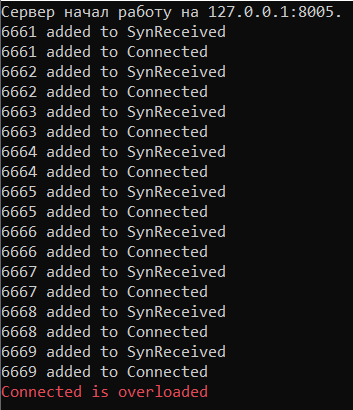
Passive scan



Syn flooding



Full connection flooding



1. **Код программы**

Config.cs

namespace Lab3

{

public static class Config

{

public static readonly string host = "127.0.0.1";

public static readonly int ServerListenPort = 8005;

public static readonly int ServerSendPort = 1245;

}

}

Hack1.cs

using System;

using System.Net;

using System.Net.Sockets;

using System.Threading.Tasks;

namespace Lab3

{

class Hack1

{

public static Task Hack = new Task(Run);

public static void Run()

{

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

try

{

TcpSegment tcp = new TcpSegment(6661, Config.ServerListenPort, syn: true);

socket.Bind(new IPEndPoint(IPAddress.Parse(Config.host), 6661));

for (int i = Config.ServerListenPort - 10; i <= Config.ServerListenPort; i++)

{

var point = new IPEndPoint(IPAddress.Parse(Config.host), Config.ServerListenPort);

tcp.destinationPort = i;

point.Port = i;

socket.SendTo(tcp.ToByteArray(), point);

}

socket.Close();

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

socket.Bind(new IPEndPoint(IPAddress.Parse(Config.host), 6661));

byte[] data = new byte[1024];

int len = socket.Receive(data);

var message = TcpSegment.FromByteArray(TcpSegment.GetBytes(data, len));

Console.WriteLine($"syn = {message.syn}, ack = {message.ack} в {message.sourcePort}.");

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

socket.Close();

}

}

}

}

Hack2.cs

using System.Net;

using System.Net.Sockets;

using System.Threading.Tasks;

namespace Lab3

{

class Hack2

{

public static Task Hack = new Task(Run);

public static int count = 10;

private static void Run()

{

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

try

{

socket.Bind(new IPEndPoint(IPAddress.Parse(Config.host), 6661));

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

{

TcpSegment tcp = new TcpSegment(6661 + i, Config.ServerListenPort, syn: true);

socket.SendTo(tcp.ToByteArray(), new IPEndPoint(IPAddress.Parse(Config.host), Config.ServerListenPort));

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

socket.Close();

}

}

}

}

Hack3.cs

using System;

using System.Net;

using System.Net.Sockets;

using System.Threading.Tasks;

namespace Lab3

{

class Hack3

{

public static Task Hack = new Task(Run);

public static int count = 10;

private static EndPoint server = new IPEndPoint(IPAddress.Parse(Config.host), Config.ServerListenPort);

private static void Run()

{

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

try

{

socket.Bind(new IPEndPoint(IPAddress.Parse(Config.host), 6660));

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

{

TcpSegment tcp = new TcpSegment(6661 + i, Config.ServerListenPort, syn: true);

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

byte[] data = new byte[1024];

Socket socket1 = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);

try

{

socket1.Bind(new IPEndPoint(IPAddress.Parse(Config.host), 6661 + i));

var count = socket1.Receive(data);

var msg = TcpSegment.FromByteArray(TcpSegment.GetBytes(data, count));

var re = new TcpSegment(msg.destinationPort, msg.sourcePort, ack: true);

socket1.SendTo(re.ToByteArray(), server);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

socket1.Close();

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

socket.Close();

}

}

}

}

Server.cs

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

using System.Threading.Tasks;

namespace Lab3

{

static class Server

{

static IPEndPoint listenPoint = new IPEndPoint(IPAddress.Parse(Config.host), Config.ServerListenPort);

public static Task Listen = new Task(Listener);

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

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

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

static int MaxSynRecived = 8;

static int MaxEstablished = 8;

public static void Listener()

{

Console.WriteLine($"Сервер начал работу на {listenPoint.Address}:{listenPoint.Port}.");

try

{

listenSocket.Bind(listenPoint);

while (true)

{

byte[] data = new byte[1024];

int len = listenSocket.Receive(data);

TcpSegment message = TcpSegment.FromByteArray(TcpSegment.GetBytes(data, len));

if (message.destinationPort != Config.ServerListenPort)

continue;

else if (Established.Contains(message.sourcePort))

continue;

else if (SynRecived.Contains(message.sourcePort) && message.ack)

AddToConnected(message);

else if (!SynRecived.Contains(message.sourcePort) && message.syn)

AddToSynReceived(message);

}

}

catch (Exception ex)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(ex.Message);

Console.ResetColor();

}

finally

{

listenSocket.Close();

}

}

public static void AddToSynReceived(TcpSegment request)

{

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

sender.Bind(new IPEndPoint(IPAddress.Parse("127.0.0.1"), 1111));

try

{

var response = new TcpSegment(Config.ServerListenPort, request.sourcePort, syn: true, ack: true);

sender.SendTo(response.ToByteArray(),

new IPEndPoint(IPAddress.Parse(Config.host), request.sourcePort));

SynRecived.Add(request.sourcePort);

Console.WriteLine($"{request.sourcePort} added to SynReceived");

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

sender.Close();

}

if (SynRecived.Count > MaxSynRecived)

throw new Exception("SynReceived is overloaded");

}

public static void AddToConnected(TcpSegment request)

{

Established.Add(request.sourcePort);

SynRecived.Remove(request.sourcePort);

Console.WriteLine($"{request.sourcePort} added to Connected");

if (Established.Count > MaxEstablished)

throw new Exception("Connected is overloaded");

}

}

}

TcpSegment.cs

using System.Collections;

using System.Text;

namespace Lab3

{

class TcpSegment

{

public int sourcePort;

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 data;

public TcpSegment(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 = "")

{

this.sourcePort = sourcePort;

this.destinationPort = destinationPort;

this.sequenceNumber = sequenceNumber;

this.acknowledgmentNumber = acknowledgmentNumber;

this.urg = urg;

this.ack = ack;

this.psh = psh;

this.rst = rst;

this.syn = syn;

this.fin = fin;

this.data = data;

}

private BitArray ToBitArray()

{

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

CopyTo(BitArray16FromInt(sourcePort), res, 0);

CopyTo(BitArray16FromInt(destinationPort), res, 16);

CopyTo(BitArray32FromInt(sequenceNumber), res, 32);

CopyTo(BitArray32FromInt(acknowledgmentNumber), res, 64);

res[106] = urg;

res[107] = ack;

res[108] = psh;

res[109] = rst;

res[110] = syn;

res[111] = fin;

CopyTo(BitArrayFromStr(data), res, 160);

return res;

}

public byte[] ToByteArray()

{

BitArray bitArray = ToBitArray();

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

bitArray.CopyTo(data, 0);

return data;

}

public static TcpSegment FromByteArray(byte[] source)

{

return FromBitArray(new BitArray(source));

}

private static TcpSegment FromBitArray(BitArray source)

{

return new TcpSegment(sourcePort: IntFromBitArray(Slice(source, 0, 16)),

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

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

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

urg: source[106],

ack: source[107],

psh: source[108],

rst: source[109],

syn: source[110],

fin: source[111],

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

}

private static BitArray BitArray16FromInt(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 BitArray BitArray32FromInt(int value)

{

int[] i = { value };

return new BitArray(i);

}

private static BitArray BitArrayFromStr(string str)

{

byte[] data = Encoding.UTF8.GetBytes(str);

return new BitArray(data);

}

private static int IntFromBitArray(BitArray source)

{

int[] data = new int[1];

source.CopyTo(data, 0);

return data[0];

}

private static string StrFromBitArray(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 CopyTo(BitArray source, BitArray dest, int start)

{

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

{

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

}

}

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

{

byte[] res = new byte[len];

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

{

res[i] = source[i];

}

return res;

}

}

}

Program.cs

using System;

namespace Lab3

{

class Program

{

static void Main(string[] args)

{

Server.Listen.Start();

Hack1.Hack.Start();

Console.ReadLine();

}

}

}

1. **Вывод**

В данной лабораторной работе были изучены атаки при установке TCP-соединения и протоколов прикладного уровня и создано приложение, реализующее атаки на протокол при установке TCP-соединения и в рамках заданного протокола прикладного уровня.