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. 953505

Яцына А. М.

Проверил:

Олисейчик В.В.

Минск, 2022

**Введение**

Цель данной лабораторной работы создать приложение, которое совершает атаки на протокол при установке TCP-соединения и в рамках данного протокола прикладного уровня.

**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**

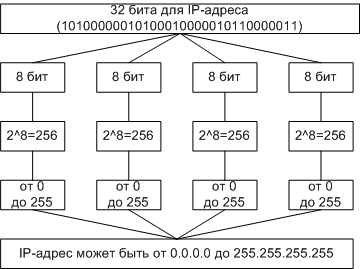
**IPv4** - адрес является уникальным 32-битным идентификатором IP-интерфейса в Интернет.

**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

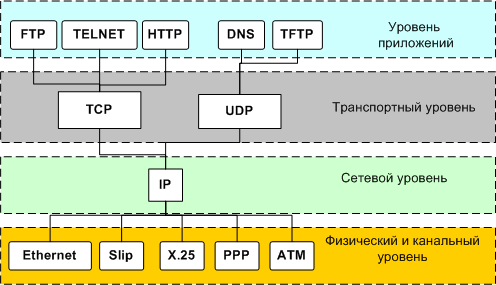
131 - номер хоста

**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.



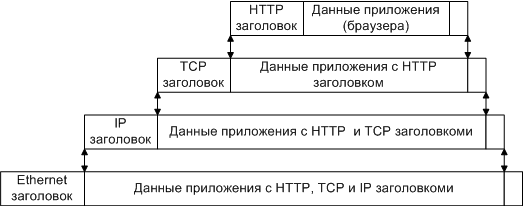
Стек протоколов TCP/IP

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.

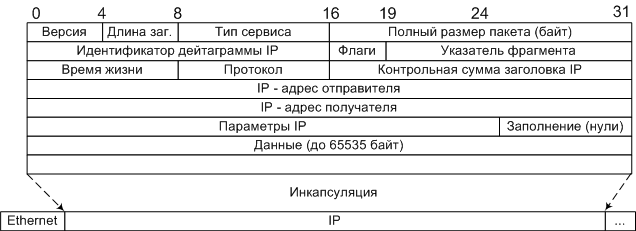
*Программа, реализующая функции того или иного протокола, часто называется модулем, например, “IP-модуль”, “модуль TCP”.*

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.

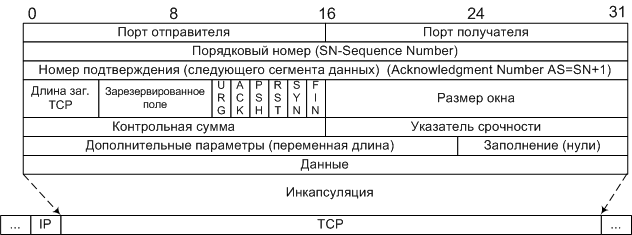
**Протокол надежной доставки сообщений TCP**

**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.

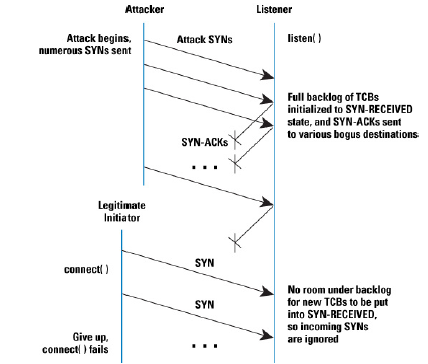
Программа nmap - позволяет сканировать порты.

Работу порта, также можно проверить с помощью telnet.

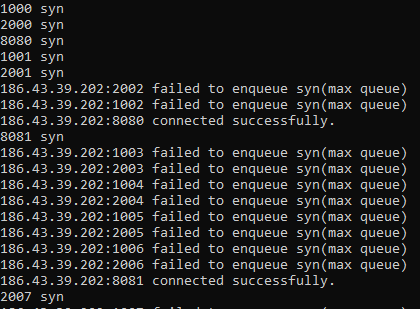
Некоторые заданные порты [RFC-1700](http://ipm.kstu.ru/internet/doc/rfc/rfc1700.txt) (1994) 43%

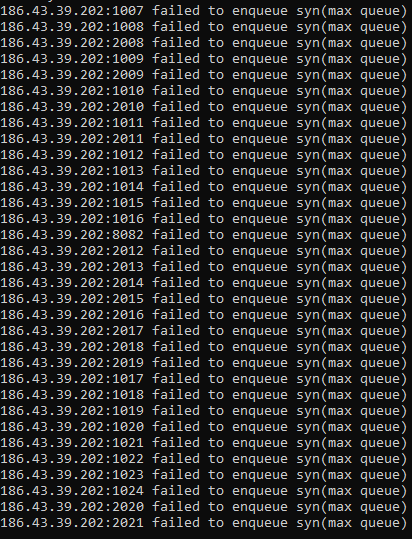
|  |  |  |
| --- | --- | --- |
| **Порт** | **Служба** | **Описание** |
| 0 | - | Зарезервировано |
| 13 | Daytime | Синхронизация времени |
| 20 | ftp-data | Канал передачи данных для FTP |
| 21 | ftp | Передача файлов |
| 23 | telnet | Сетевой терминал |
| 25 | SMTP | Передача почты |
| 37 | time | 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 |

**Блок схемы алгоритма**



**Результат работы**





**Выводы**

Выше мы рассмотрели теоретические сведения об различных типах адресов. Изучил теорию о базовых протоколах, об стеках протоколов TCP/IP, об их физическом, канальном, межсетевых и транспортных уровнях. Разработал приложение.

**Код программы**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading.Tasks;

namespace lab3

{

public class attack

{

public void Listen(int fakePort)

{

try

{

for (int i = 0; i < 30;)

{

var endPoint = new IPEndPoint(IPAddress.Parse(Config.serverAddress), Config.serverPort);

var socketTCP = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

socketTCP.Connect(endPoint);

var message = new CMessage()

{

SourceAddr = Config.fakeAddress,

sourcePort = fakePort,

Message = "fake",

Time = DateTime.Now,

DestinationAddr = Config.serverAddress,

destinationPort = Config.serverPort,

syn = true,

ack = false,

};

socketTCP.Send(CustomConverter.Serialize(message));

fakePort++;

i++;

socketTCP.Shutdown(SocketShutdown.Both);

socketTCP.Close();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

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 lab3

{

public class Client

{

public void Listen()

{

try

{

var asourcePort = 8080;

var answer = new CMessage();

for (int i = 0; i < 4;)

{

var endPoint = new IPEndPoint(IPAddress.Parse(Config.serverAddress), Config.serverPort);

var socketTCP = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

socketTCP.Connect(endPoint);

var message = new CMessage()

{

SourceAddr = Config.fakeAddress,

sourcePort = asourcePort,

Message = $"Data from client {i}",

Time = DateTime.Now,

DestinationAddr = Config.serverAddress,

destinationPort = Config.serverPort,

syn = true,

ack = false,

};

if (answer.ack)

{

message.sourcePort = answer.destinationPort;

message.Message = answer.Message;

message.syn = false;

message.ack = true;

asourcePort--;

i--;

}

socketTCP.Send(CustomConverter.Serialize(message));

var buffer = new byte[1024];

var size = 0;

do

{

size = socketTCP.Receive(buffer);

var obj = CustomConverter.Deserialize(buffer);

if (obj is CMessage)

answer = (CMessage)obj;

}

while (socketTCP.Available > 0);

i++;

asourcePort++;

socketTCP.Shutdown(SocketShutdown.Both);

socketTCP.Close();

if(i > 1)

Thread.Sleep(500);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab3

{

[SerializableAttribute]

public class CMessage

{

public DateTime Time { get; set; }

public string SourceAddr { get; set; }

public string DestinationAddr { get; set; }

public string Message { get; set; }

public int sourcePort { get; set; }

public int destinationPort { get; set; }

public bool syn { get; set; }

public bool ack { get; set; }

public CMessage()

{

}

public CMessage(string msg)

{

Time = DateTime.Now;

Message = msg;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab3

{

public class Config

{

public static readonly string serverAddress = "127.0.0.1";

public static readonly string fakeAddress = "186.43.39.202";

public static readonly int serverPort = 7999;

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

using System.Threading.Tasks;

namespace lab3

{

public static class CustomConverter

{

public static byte[] Serialize(object anySerializableObject)

{

using (var memoryStream = new MemoryStream())

{

(new BinaryFormatter()).Serialize(memoryStream, anySerializableObject);

return memoryStream.ToArray();

}

}

public static object Deserialize(byte[] data)

{

using (var memoryStream = new MemoryStream(data))

{

return (new BinaryFormatter()).Deserialize(memoryStream);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace lab3

{

internal class Program

{

static void Main(string[] args)

{

var server = new Server();

var client = new Client();

var attack = new attack();

try

{

Task.Run(() => server.Listen());

Thread.Sleep(5000);

Task.Run(() => attack.Listen(1000));

Task.Run(() => client.Listen());

Task.Run(() => attack.Listen(2000));

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

Console.ReadKey();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading.Tasks;

namespace lab3

{

internal class Server

{

private List<int> synCon = new List<int>();

private int maxQueue = 5;

public void Listen()

{

var endPoint = new IPEndPoint(IPAddress.Parse(Config.serverAddress), Config.serverPort);

var currentSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

try

{

currentSocket.Bind(endPoint);

currentSocket.Listen(5);

while (true)

{

var handler = currentSocket.Accept();

var buffer = new byte[1024];

var size = 0;

var data = new CMessage();

do

{

size = handler.Receive(buffer);

var obj = CustomConverter.Deserialize(buffer);

if (obj is CMessage)

data = (CMessage)obj;

}

while (handler.Available > 0);

var msg = new CMessage()

{

SourceAddr = data.DestinationAddr,

sourcePort = data.destinationPort,

Message = "",

DestinationAddr = data.SourceAddr,

destinationPort = data.sourcePort,

syn = false,

ack = false

};

if (data.syn && !synCon.Contains(data.sourcePort))

{

msg.SourceAddr = data.DestinationAddr;

msg.sourcePort = data.destinationPort;

msg.Message = data.Message;

msg.Time = data.Time;

msg.DestinationAddr = data.SourceAddr;

msg.destinationPort = data.sourcePort;

msg.syn = false;

msg.ack = true;

if (synCon.Count < maxQueue)

{

synCon.Add(data.sourcePort);

Console.WriteLine($"{data.sourcePort} syn");

}

else

{

Console.WriteLine($"{data.SourceAddr}:{data.sourcePort} failed to enqueue syn(max queue)");

}

}

else if (data.ack && synCon.Contains(data.sourcePort))

{

synCon.Remove(data.sourcePort);

Console.WriteLine($"{data.SourceAddr}:{data.sourcePort} connected successfully.");

}

handler.Send(CustomConverter.Serialize(msg));

handler.Shutdown(SocketShutdown.Both);

handler.Close();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

Console.ReadKey();

}

finally

{

currentSocket.Shutdown(SocketShutdown.Both);

currentSocket.Close();

}

}

}

}