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 К. А. Тимофеев

Проверил Е. А. Лещенко

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данной лабораторной работы заключается в изучении теоретических основ работы сети Internet, а также физического, канального и транспортного уровней. Кроме того, задачей является разработка программы, которая будет имитировать атаки на протокол при установке TCP-соединения. В ходе работы необходимо создать интерфейс приложения, который наглядно демонстрирует передаваемые данные, исходные данные протокола и проверки каждой из сторон.

1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Протоколом называется набор правил, задающих форматы сообщений и процедуры, которые позволяют компьютерам и прикладным программам обмениваться информацией. Эти правила соблюдаются каждым компьютером в сети, в результате чего любой хост-получатель может понять отправленное ему сообщение. Набор протоколов TCP/IP можно рассматривать как многоуровневую структуру.

На этом рисунке показан стек протоколов TCP/IP. Он делится на следующие уровни (начиная с верхнего): прикладной, транспортный, сетевой, интерфейсный и аппаратный (см. рисунок 1.1).

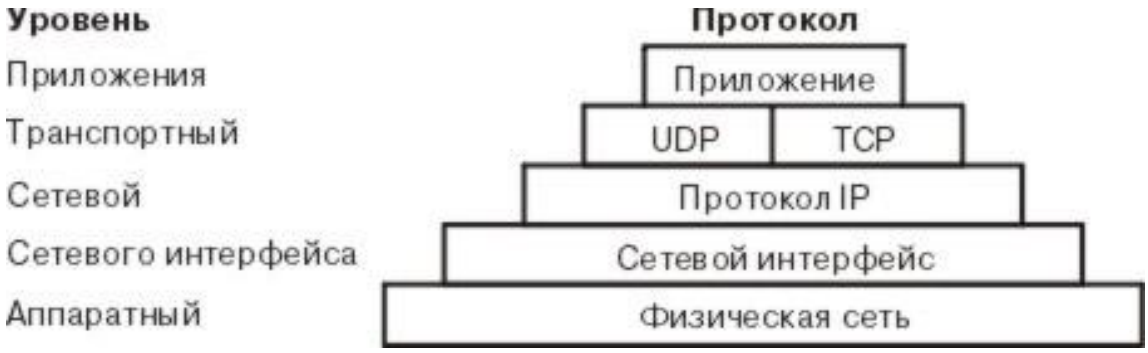

Рисунок 1.1 – Набор протоколов TCP/IP

В протоколе TCP/IP строго зафиксированы правила передачи информации от отправителя к получателю. Сообщение или поток данных приложения отправляется протоколу Internet транспортного уровня, то есть протоколу пользовательских дейтаграмм (UDP) или протоколу управления передачей (TCP). Получив данные от приложения, эти протоколы разделяют всю информацию на небольшие блоки, которые называются пакетами. К каждому пакету добавляется адрес назначения, а затем пакет передается на следующий уровень протоколов Internet, то есть сетевой уровень.

На сетевом уровне пакет помещается в дейтаграмму протокола Internet (IP), к которой добавляется заголовок и концевик. Протокол сетевого уровня определяет адрес следующего пункта назначения IP-дейтаграммы (она может быть передана сразу получателю или на промежуточный шлюз) и отправляют ее на уровень сетевого интерфейса.

Уровень сетевого интерфейса принимает IP-дейтаграммы и передает их в виде *кадров* с помощью аппаратного обеспечения, такого как адаптер Ethernet или Token-Ring (см. рисунок 1.2).

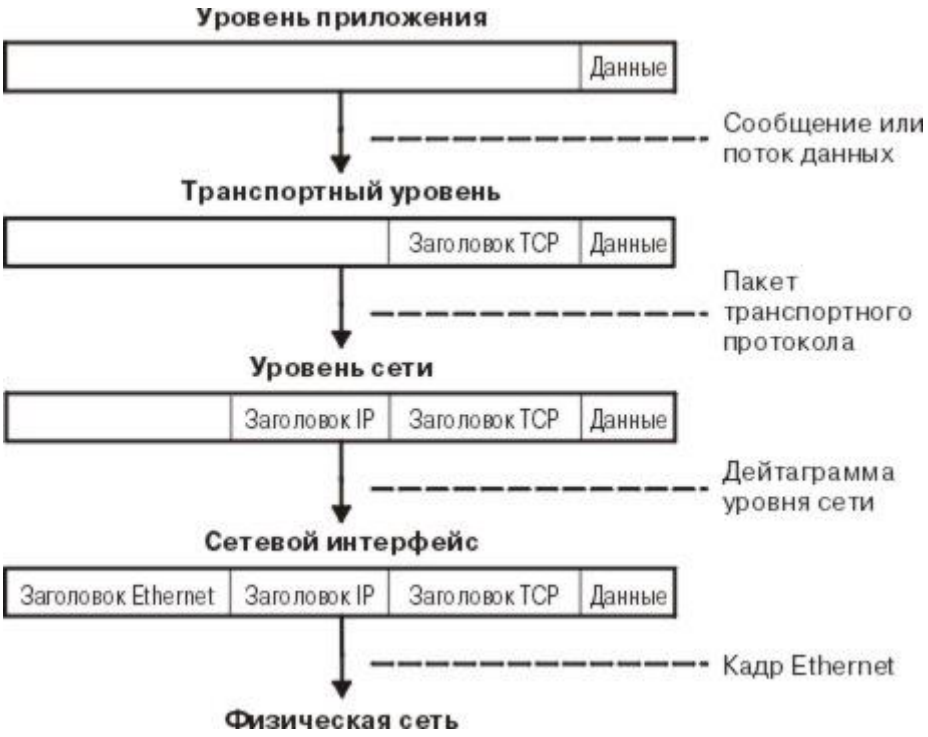

Рисунок 1.2 –Передача информации от приложения целевому хосту

Syn-flood атака - это одна из разновидностей DoS-атак. Принцип следующий: злоумышленник посылает огромное количество запросов установки соединения на атакуемый сервер. Сервер, видя сегменты с флагом SYN, выделяет необходимые ресурсы для поддержания соединения и отправляет в ответ сегменты с флагами SYN и ACK, переходя в состояние syn-received (такое состояние еще называют полуоткрытым соединением). Злоумышленник, не шлет ответные ACK сегменты, а продолжает бомбардировать сервер SYN-запросами, тем самым вынуждая сервер создавать все больше и больше полуоткрытых соединений. Сервер, понятное дело, располагает ограниченными ресурсами, и потому имеет лимит на количество полуоткрытых соединений. Ввиду этой ограниченности, при достижении предельного числа полуоткрытых соединений сервер начинает отклонять новые попытки соединения; таким образом и достигается отказ в обслуживании (Denial of Service).

2 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

В результате разработки программы было создано консольное приложение, реализующее протокол TCP а также симуляцию атаки SynFlood.

На рисунке 2.1 представлен результат работы программы в обычном режиме.

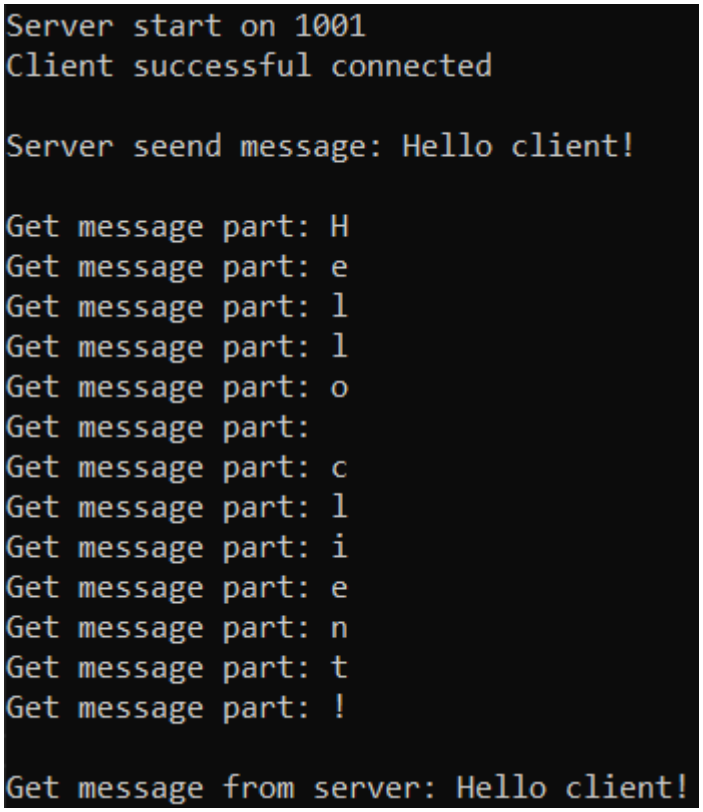

Рисунок 2.1 – Результат работы программы в обычном режиме

На рисунке 2.2 представлен результат работы программы при атаке SynFlood.

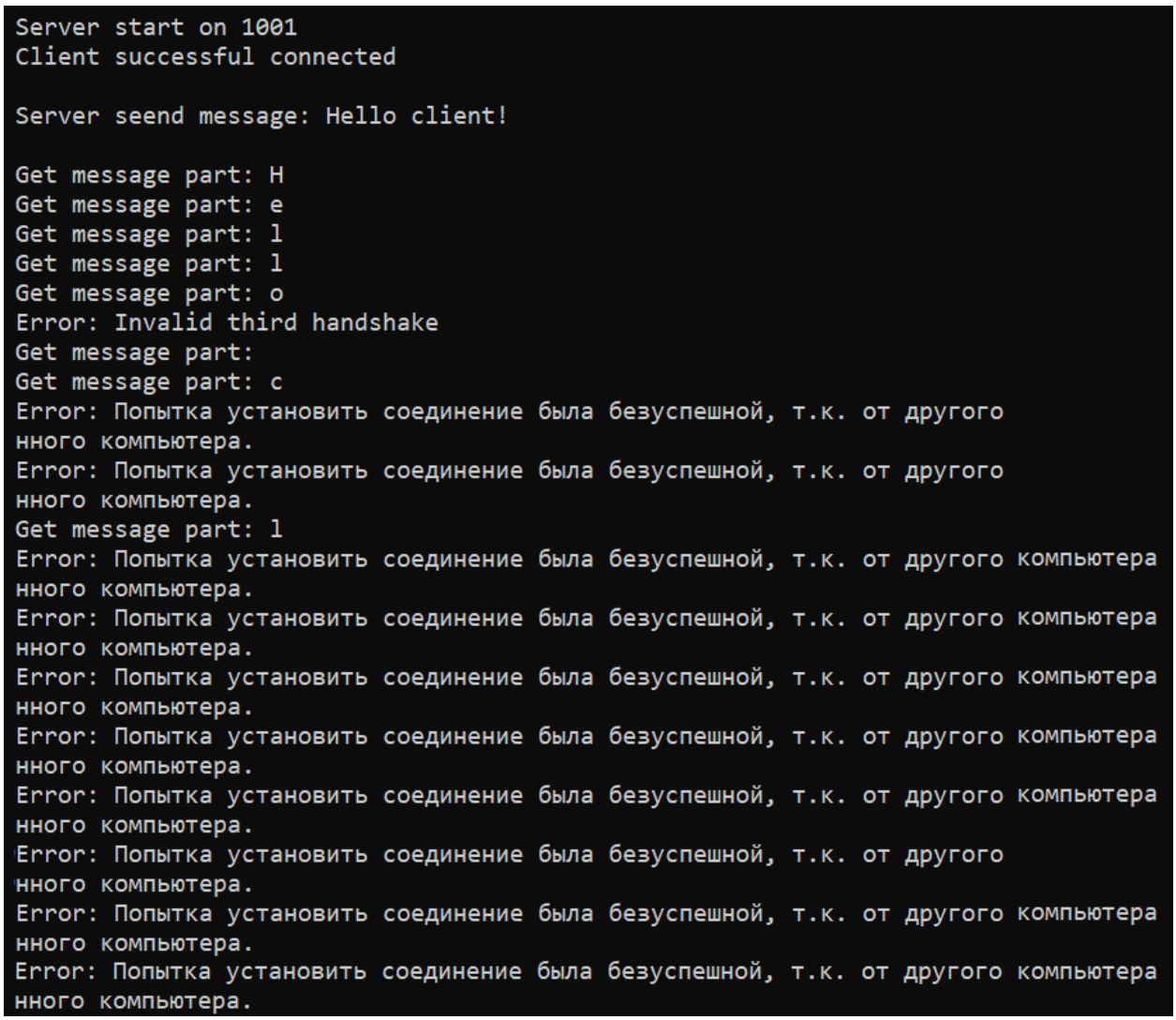

Рисунок 2.2 – Результат работы программы при атаке SynFlood.

ПРИЛОЖЕНИЕ А
(обязательное)

Исходный код программы

```
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Text.Json;


namespace Tcp;


public class Client
{
    private static readonly Lazy<Client> _lazy = new(() => new
Client(new(IPAddress.Parse("127.0.0.1"), 1001)));
    public static Client Instance => _lazy.Value;

    private Client(IPEndPoint iPEndPoint)
    {
        Ip = iPEndPoint;
    }

    public IPEndPoint Ip { get; }

    public Task ConnectToServer(IPEndPoint serverIP, CancellationTokenSource cancelTokenSource)
    {
        Socket clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
        clientSocket.Bind(Ip);

        try
        {
            clientSocket.Connect(serverIP);

            uint r1 = Convert.ToUInt32(Random.Shared.Next());

            // first handshake
            Send(clientSocket, new TcpPacket()
            {
                SourcePort = (ushort)Ip.Port,
                DestinationPort = (ushort)serverIP.Port,
                SequenceNumber = r1,
                AcknowledgmentNumber = 0,
                Syn = true,
                WindowSize = 4,
                Data = []
            });

            // second handshake
            TcpPacket packet = Read(clientSocket);

            if (packet is not { Syn: true, Ack: true })
                throw new Exception("Invalid second handshake: SYN or ACK is not set.");

            if (packet.AcknowledgmentNumber != r1 + 1)
                throw new Exception("Invalid second handshake: packet AcknowledgmentNumber not
equal to r1 + 1.");

            uint r2 = packet.SequenceNumber;

            // third handshake
            Send(clientSocket, new TcpPacket()
            {
                SourcePort = (ushort)Ip.Port,
                DestinationPort = (ushort)serverIP.Port,
                SequenceNumber = 1,
                AcknowledgmentNumber = r2 + 1,
                Ack = true,
                WindowSize = 4,
                Data = []
            });
```

```
            StringBuilder messageFromServer = new();

            for (packet = Read(clientSocket); !packet.Fin; packet = Read(clientSocket))
            {
                var part = packet.Data.AsString();
                Console.WriteLine($"Get message part: {part}");

                messageFromServer.Append(part);

                Thread.Sleep(200);

                Send(clientSocket, new TcpPacket()
                {
                    SourcePort = (ushort)Ip.Port,
                    DestinationPort = (ushort)serverIP.Port,
                    SequenceNumber = 1,
                    AcknowledgmentNumber = packet.SequenceNumber + (uint)packet.Data.Length,
                    Ack = true,
                    WindowSize = 4,
                    Data = []
                });
            }

            Console.WriteLine($"\nGet message from server: {messageFromServer}\n");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }
        finally
        {
            if (clientSocket.Connected)
                clientSocket.Close();
        }

        cancelTokenSource.Cancel();
        return Task.CompletedTask;
    }

    private void Send(Socket socket, TcpPacket packet)
    {
        byte[] responseBuffer = packet.AsBytes();
        socket.Send(responseBuffer);
    }

    public static TcpPacket Read(Socket socket)
    {
        byte[] messageBuffer = new byte[4096];
        List<byte> res = new();

        int bytesRead = socket.Receive(messageBuffer);
        res.AddRange(messageBuffer[0..bytesRead]);

        return AsTcpPacket([.. res]);
    }

    public static TcpPacket AsTcpPacket(byte[] data)
    {
        return JsonSerializer.Deserialize<TcpPacket>(data.AsString())!;
    }
}
```

```
public static class Extensions
{
    public static byte[] AsBytes(this TcpPacket packet)
    {
        return JsonSerializer.Serialize(packet).AsBytes();
    }
```

```
    public static byte[] AsBytes(this string str)
    {
        return Encoding.UTF32.GetBytes(str);
    }

    public static string AsString(this byte[] bytes, int offset)
    {
        return Encoding.UTF32.GetString(bytes, 0, offset);
    }

    public static string AsString(this byte[] bytes)
    {
        return Encoding.UTF32.GetString(bytes);
    }

    public static TcpPacket AsTcpPacket(this byte[] data)
    {
        return JsonSerializer.Deserialize<TcpPacket>(data.AsString())!;
    }
}


using System.Net;
using System.Net.Http.Headers;
using System.Net.Sockets;
using System.Text;

namespace Tcp;

public class Server
{
    private static Lazy<Server> _lazy = new(() => new Server(new(IPAddress.Parse("127.0.0.1"),
1000)));
    public static Server Instance => _lazy.Value;


    private Server(IPEndPoint ip)
    {
        Ip = ip;
    }

    public IPEndPoint Ip { get; }

    private readonly Dictionary<int, Socket> _clients = [];


    public async Task ConnectClient(CancellationToken token)
    {
        Socket listenerSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp) { ReceiveTimeout = 300 };

        try
        {
            listenerSocket.Bind(Ip);

            listenerSocket.Listen(15);
            Console.WriteLine($"Server start on {Ip.Port}");

            while (true)
            {
                if (token.IsCancellationRequested)
                    break;

                Socket clientSocket = await listenerSocket.AcceptAsync(token);
                // Console.WriteLine($"Client from {clientSocket.RemoteEndPoint} try to
connect");

                _clients.Add(((IPEndPoint)clientSocket.RemoteEndPoint!).Port, clientSocket);

                Thread clientThread = new Thread(ProcessClient);
                clientThread.Start(((IPEndPoint)clientSocket.RemoteEndPoint).Port);
            }
        }
        catch (OperationCanceledException)
        { }
        catch (Exception ex)
        {
            Console.WriteLine($"Ошибка: {ex.Message}");
```

```
        }
    }

    private void ProcessClient(object obj)
    {
        try
        {
            int port = (int)obj;

            TcpPacket packet = Read(port);

            // first handshake
            if (packet is not { Syn: true, Data.Length: 0 })
            {
                throw new Exception($"""
                    Invalid first handshake
                    {packet}
                    """);
            }

            uint sequenceNumber = Convert.ToUInt32(Random.Shared.Next());
            uint acknowledgmentNumber = packet.SequenceNumber + 1;
            ushort windowSize = packet.WindowSize;

            // second handshake
            Send(port, new TcpPacket()
            {
                SourcePort = (ushort)Ip.Port,
                DestinationPort = (ushort)port,
                SequenceNumber = sequenceNumber,
                AcknowledgmentNumber = acknowledgmentNumber,
                Syn = true,
                Ack = true,
                WindowSize = windowSize,
                Data = [],
            });

            packet = Read(port);

            // Invalid third handshake
            if (packet is not { Ack: true, Data.Length: 0 })
                throw new Exception("Invalid third handshake");

            if (packet.AcknowledgmentNumber != sequenceNumber + 1)
                throw new Exception("Invalid third handshake");

            Console.WriteLine("Client successful connected\n");


            string message = "Hello client!";


            Console.WriteLine($"Server seend message: {message}\n");

            List<TcpPacket> packets = ToPackets(message.AsBytes(), windowSize, (ushort)Ip.Port,
(ushort)port, sequenceNumber, acknowledgmentNumber).ToList();


            foreach (var packetToClient in packets)
            {
                Send(port, packetToClient);

                packet = Read(port);

                if (packet is not { Ack: true, Data.Length: 0 })
                    throw new Exception("Incorrect confirmation packet");

                if (packet.Rst)
                {
                    Console.WriteLine($"Emergency connection termination{packet.SourcePort}");
                    _clients[packet.SourcePort].Close();
                    return;
                }

                if (packetToClient.SequenceNumber + windowSize != packet.AcknowledgmentNumber)
                    throw new Exception("Invalid sequence number in client packet");
            }
```

```
            Send(port, new TcpPacket()
            {
                SourcePort = (ushort)Ip.Port,
                DestinationPort = (ushort)port,
                SequenceNumber = sequenceNumber,
                AcknowledgmentNumber = acknowledgmentNumber,
                Fin = true,
                WindowSize = windowSize,
                Data = []
            });
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }
        finally
        {
            int port = (int)obj;

            if (_clients[port].Connected)
                _clients[port].Close();
        }
    }

    private void Send(int port, TcpPacket packet)
    {
        byte[] responseBuffer = packet.AsBytes();
        _clients[port].Send(responseBuffer);
    }

    public TcpPacket Read(int port)
    {
        byte[] messageBuffer = new byte[4096];
        List<byte> res = [];

        int bytesRead = _clients[port].Receive(messageBuffer);
        res.AddRange(messageBuffer[0..bytesRead]);

        return res.ToArray().AsTcpPacket();
    }

    private static IEnumerable<TcpPacket> ToPackets(
        byte[] data,
        ushort windowsSize,
        ushort sourcePort,
        ushort destinationPort,
        uint sequenceNumber,
        uint acknowledgmentNumber)
    {
        for (int id = 0; id < data.Length; id += windowsSize, sequenceNumber += windowsSize)
        {
            var dataSlice = data[id..Math.Min(id + windowsSize, data.Length)];

            yield return new TcpPacket()
            {
                SourcePort = sourcePort,
                DestinationPort = destinationPort,
                SequenceNumber = acknowledgmentNumber,
                WindowSize = windowsSize,
                Data = dataSlice
            };
        }
    }
}
```