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

Выполнил:

студент гр. 153503

Бобко И.В.

Проверил:

Лещенко Е.А.

Минск 2024

СОДЕРЖАНИЕ

[Введенеие 3](#_Toc157436533)

[1 Краткие теоретические сведения 4](#_Toc157436534)

[2 Результат выполнения программы 11](#_Toc157436535)

[Приложение А 12](#_Toc157436537)

# ВВЕДЕНИЕ

Целью выполнения данной лабораторной работы является создание приложения, реализующего атаки на протокол при установке TCP-соединения и в рамках заданного протокола прикладного уровня.

# 1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

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: состоит 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](http://www.kstu.ru/).

**2.1 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. Пример перевода IP-адреса из двоичной системы в десятичную представлен на рисунке 2.1.

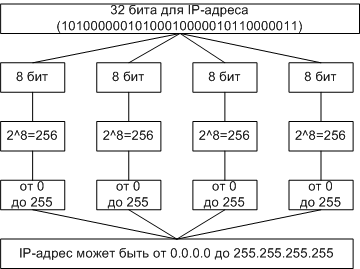


Рисунок 2.1 – Перевод адреса из двоичной системы в десятичную

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

**2.2 Базовые протоколы (IP, TCP)**

TCP/IP – собирательное название для набора (стека) сетевых протоколов разных уровней, используемых в Интернет. К особенностям TCP/IP относятся:

– открытые стандарты протоколов, разрабатываемые независимо от программного и аппаратного обеспеченичя;

– независимость от физической среды передачи;

– система уникальной адресации;

– стандартизованные протоколы высокого уровня для распространенных пользовательских сервисов.

Уровни стека протоколов TCP/IP представлены на рисунке 2.2.

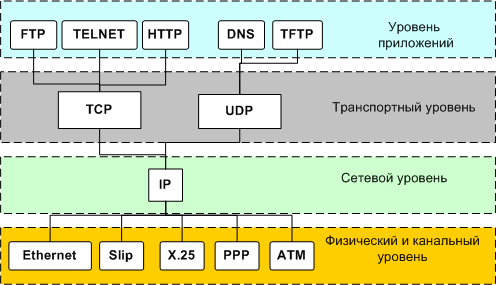


Рисунок 2.2 – Стек протоколов TCP/IP

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.

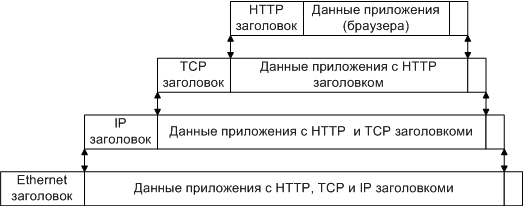


Рисунок 2.3 – Пример инкапсуляции пакетов в стеке TCP/IP

**2.3 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.

**2.4 Межсетевой уровень и протокол IP**

Основу этого уровня составляет IP-протокол. IP (Internet Protocol) – интернет-протокол. Первый стандарт IPv4 определен в RFC-760. Последняя версия IPv6 – [RFC-2460](http://ipm.kstu.ru/internet/doc/rfc/rfc2460.txt).

Основные задачи протокола IP:

– 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.

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: уничтожить IP-пакет или 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, что называется фрагментацией, 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. Пример структуры дейтограммы IP представлен на рисунке 2.4.

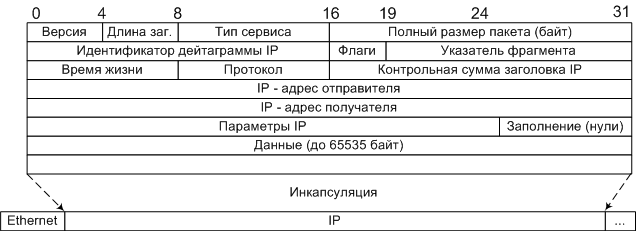


Рисунок 2.4 – Структура дейтограммы IP. Слова по 32 бита.

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

Длина заголовка – длина заголовка IP-пакета.

Тип сервиса играет важную роль в маршрутизации пакетов. Интернет не гарантирует запрашиваемый тип сервиса, но многие маршрутизаторы учитывают эти запросы при выборе маршрута.

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

Время жизни 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).

**2.5 Маршрутизация**

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

Маршрутная информация, может быть:

1 Статической: маршрутные таблицы прописываются вручную.

2 Динамической: маршрутную информацию распространяют специальные протоколы.

**2.6 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.

**2.7 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).

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 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 представлена на рисунке 2.5.

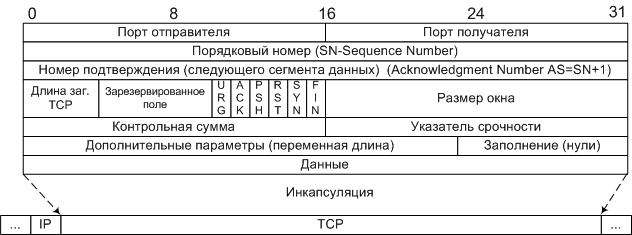


Рисунок 2.5 – Структура дейтограммы TCP

Длина заголовка – задается словами по 32бита.

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

Контрольная сумма – включает псевдозаголовок, 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

**2.8 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.

# 2 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

На рисунках 2.1, 2.2 изображен вывод работы в различных ситуациях:

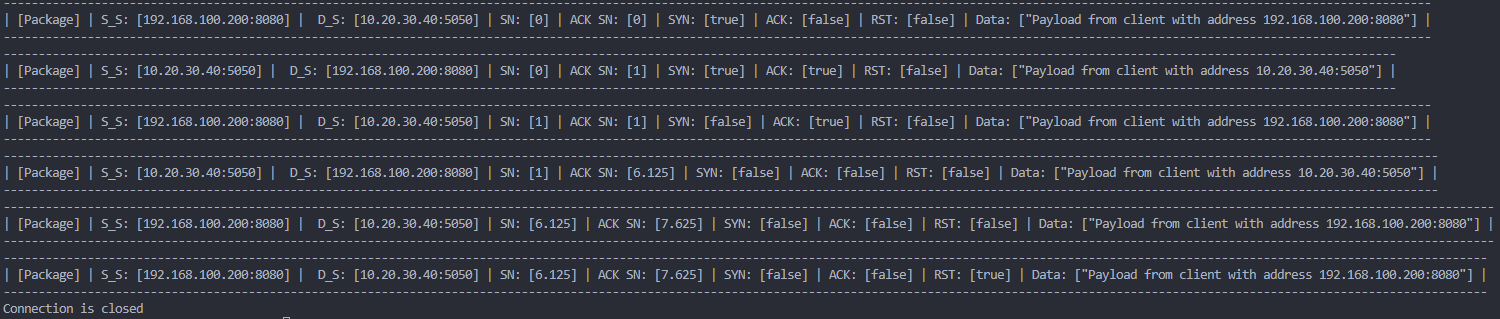


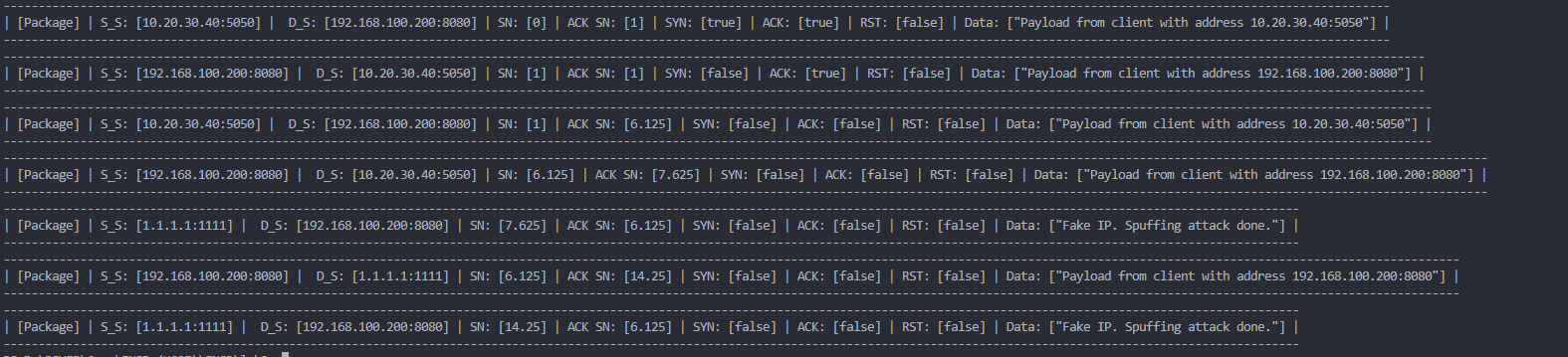
Рисунок 2.1 – RST атака  
  


Рисунок 2.2 – Атака на изменение IP-адреса и порта назначения

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Исходный код программы**

Листинг 1 – Файл client.js:

import {IP} from "./ip.js"

import {TCP} from "./tcp.js"

export class Client {

constructor(ip\_address, tcp\_port) {

this.ip\_address = ip\_address;

this.tcp\_port = tcp\_port;

this.caller = false;

}

call(connection, index) {

this.caller = true;

const other = connection.endpoints[index];

const packageParam = this.\_build\_package(other, this.\_generate\_payload());

connection.send(packageParam);

}

\_build\_package(receiver, payload) {

const ip = new IP(this.ip\_address, receiver.ip\_address, payload);

const tcp = new TCP(this.tcp\_port, receiver.tcp\_port, ip);

tcp.sn = 0;

tcp.syn = true;

return tcp;

}

\_build\_answer(packageParam, payload) {

const ip = new IP(packageParam.ip.destination\_ip, packageParam.ip.source\_ip, payload);

const response = new TCP(packageParam.destination\_port, packageParam.source\_port, ip);

if (packageParam.syn && packageParam.ack) {

response.sn = packageParam.ack\_sn;

response.ack\_sn = packageParam.sn + 1;

response.ack = true;

return response;

}

if (packageParam.syn && !packageParam.ack) {

response.sn = 0;

response.ack\_sn = packageParam.sn + 1;

response.syn = true;

response.ack = true;

return response;

}

if (packageParam.ack) {

response.sn = packageParam.ack\_sn;

response.ack\_sn += payload.length / 8;

response.ip.payload = "Connection acknowledged";

return response;

}

response.ip.payload = payload;

if (this.caller) {

response.sn = packageParam.ack\_sn;

response.ack\_sn = packageParam.sn + payload.length / 8;

} else {

response.sn = packageParam.ack\_sn;

response.ack\_sn = packageParam.sn;

}

return response;

}

\_generate\_payload() {

return `Payload from client with address ${this.ip\_address}:${this.tcp\_port}`;

}

receive(packageParam) {

return this.\_build\_answer(packageParam, this.\_generate\_payload());

}

}

export class HackerClient extends Client {

constructor(ipAddress, tcpPort) {

super(ipAddress, tcpPort)

this.caller = false;

}

receive(packageParam) {

return this.\_build\_answer(packageParam, "Fake IP. Spuffing attack done.")

}

}

Листинг 2 – Файл connection.js:

function sleepFor(sleepDuration) {

let now = new Date().getTime();

while (new Date().getTime() < now + sleepDuration) {

}

}

function printPackage(packageParam) {

try {

sleepFor(1500);

const msg = `| [Package] ${packageParam} |`;

const frameLen = msg.length;

console.log('-'.repeat(frameLen));

console.log(msg);

console.log('-'.repeat(frameLen));

} catch (error) {

throw new Error("Interrupted by user");

}

}

export class Connection {

constructor(endpoints, middlewares) {

this.endpoints = endpoints;

this.middlewares = middlewares;

this.closed = false;

this.connected = false;

}

\_findReceiver(packageParam) {

for (const endpoint of this.endpoints) {

if (endpoint.ip\_address === packageParam.ip.destination\_ip && endpoint.tcp\_port === packageParam.destination\_port) {

return endpoint;

}

}

return null;

}

send(packageParam) {

this.connected = true;

this.process(packageParam);

}

process(packageParam) {

if (!this.connected || this.closed) {

return;

}

printPackage(packageParam);

for (const middleware of this.middlewares) {

packageParam = middleware.change(packageParam);

}

if (packageParam.rst) {

printPackage(packageParam);

this.close();

return;

}

packageParam.ip.ttl -= 1;

if (packageParam.ip.ttl <= 0) {

console.log('Package TTL is expired');

this.close();

return;

}

const receiver = this.\_findReceiver(packageParam);

if (receiver === null) {

console.log(`Unknown destination ${packageParam.ip.destination\_ip}:${packageParam.destination\_port}`);

this.close();

return;

}

packageParam = receiver.receive(packageParam);

if (packageParam === null) {

console.log('One of members stop sending requests');

this.close();

} else {

this.process(packageParam);

}

}

close() {

this.closed = true;

console.log('Connection is closed');

}

}

Листинг 3 – Файл middleware.js:

export class RSTMiddleware {

constructor() {

this.call\_number = 0;

}

change(packageParam) {

this.call\_number++;

if (this.call\_number === 5) {

packageParam.rst = true;

}

return packageParam;

}

}

export class FakeIpAddressMiddleware {

constructor(ip\_address, tcp\_port) {

this.ip\_address = ip\_address;

this.tcp\_port = tcp\_port;

this.call\_number = 0;

}

change(packageParam) {

this.call\_number++;

if (this.call\_number === 5) {

packageParam.ip.destination\_ip = this.ip\_address;

packageParam.destination\_port = this.tcp\_port;

}

return packageParam;

}

}

Листинг 4 – Файл app.js:

import {Client, HackerClient} from './network/client.js'

import {Connection} from './network/connection.js';

import {FakeIpAddressMiddleware, RSTMiddleware} from './network/middleware.js';

function runAttacks() {

const client = new Client("192.168.100.200", 8080);

const server = new Client("10.20.30.40", 5050);

const hackerServer = new HackerClient("1.1.1.1", 1111);

const fakeIpAddressAttack = new FakeIpAddressMiddleware("1.1.1.1", 1111);

const rstAttack = new RSTMiddleware();

const connection = new Connection([client, server, hackerServer], [fakeIpAddressAttack]);

client.call(connection, 1);

}

runAttacks();

Листинг 5 – Файл ip.js:

export class IP {

constructor(source\_ip, destination\_ip, data) {

this.version = 4;

this.ihl = 5;

this.dscp = null;

this.ecn = null;

this.total\_length = 576;

this.id = null;

this.flags = null;

this.fragment\_offset = null;

this.ttl = 20;

this.protocol = 6;

this.checksum = null;

this.source\_ip = source\_ip;

this.destination\_ip = destination\_ip;

this.data = data;

}

}

Листинг 6 – Файл tsp.js:

export class TCP {

constructor(source\_port, destination\_port, ip) {

this.ip = ip;

this.source\_port = source\_port;

this.destination\_port = destination\_port;

this.sn = 0;

this.ack\_sn = 0;

this.offset = 20;

this.ns = null;

this.cwr = null;

this.ece = null;

this.urg = null;

this.ack = false;

this.psh = null;

this.rst = false;

this.syn = false;

this.fin = false;

this.window\_size = null;

this.checksum = 0;

this.urgent = null;

}

toString() {

return `| S\_S: [${this.ip.source\_ip}:${this.source\_port}] | D\_S: [${this.ip.destination\_ip}:${this.destination\_port}] | SN: [${this.sn}] | ACK SN: [${this.ack\_sn}] | SYN: [${this.syn}] | ACK: [${this.ack}] | RST: [${this.rst}] | Payload: ["${this.ip.data}"]`;

}

}