Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа №3

**Информационные сети. Основы безопасности. Информационные сети. Основы безопасности.**

Тема:   
**«Атаки при установке TCP-соединения и протоколов прикладного уровня»**.

Выполнил студент гр. 753504

Осипик И.Ф.

Проверил

Протько М.И.

Минск, 2020

# 1. Введение

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

**ЗАДАНИЕ:**

1. Изучить теоретические сведения.
2. Создать приложение, реализующее атаки на протокол при установке TCP-соединения и в рамках заданного протокола прикладного уровня.

В интерфейсе приложения должны быть наглядно представлены:

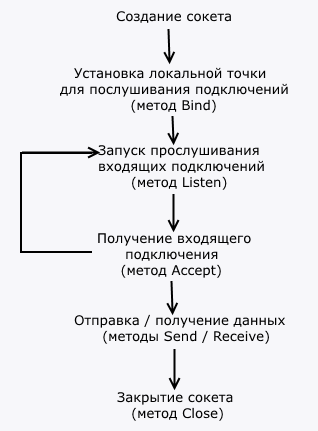
* Исходные данные протокола (модули, ключи, флаги, иные данные);
* Данные, передаваемые по сети каждой из сторон;
* Проверки, выполняемые каждым из участников.

Процесс взаимодействия между сторонами протокола может быть реализован при помощи буферных переменных.

Также необходимо выделить каждый из этапов атаки для того, чтобы его можно было отделить от остальных.

# 2**. Блок-схемы алгоритмов**

**Блок-схема сервера, работающего по протоколу ТСР:**



# 3. 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**

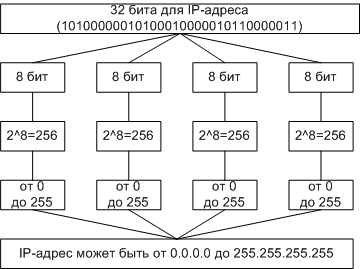
**IPv4** - адрес является уникальным 32-битным идентификатором IP-интерфейса в Интернет.

**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

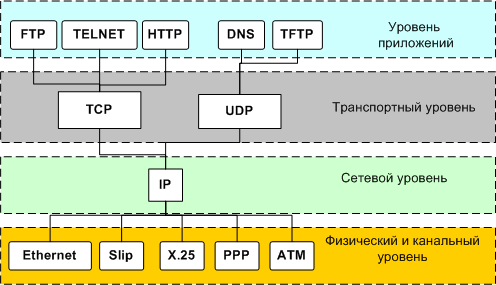
131 - номер хоста

**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.



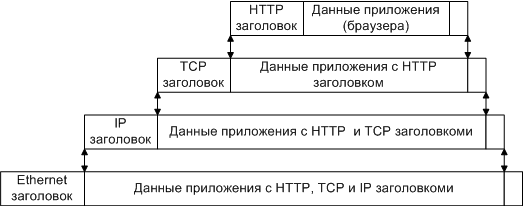
Стек протоколов TCP/IP

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.

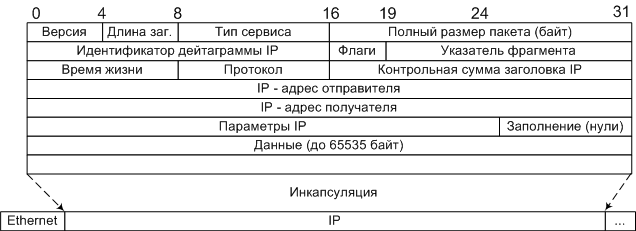
*Программа, реализующая функции того или иного протокола, часто называется модулем, например, “IP-модуль”, “модуль TCP”.*

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.

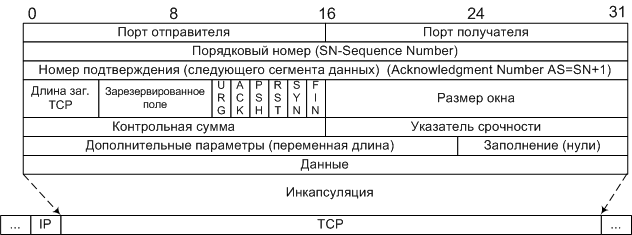
**Протокол надежной доставки сообщений TCP**

**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 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

**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.

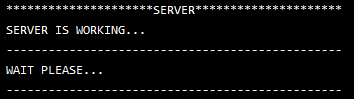
Работу порта, также можно проверить с помощью telnet.

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

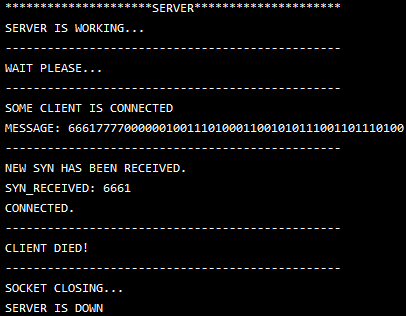
|  |  |  |
| --- | --- | --- |
| **Порт** | **Служба** | **Описание** |
| 0 | - | Зарезервировано |
| 13 | Daytime | Синхронизация времени |
| 20 | ftp-data | Канал передачи данных для FTP |
| 21 | ftp | Передача файлов |
| 23 | telnet | Сетевой терминал |
| 25 | SMTP | Передача почты |
| 37 | Time |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

# 4. Результаты выполнения программы

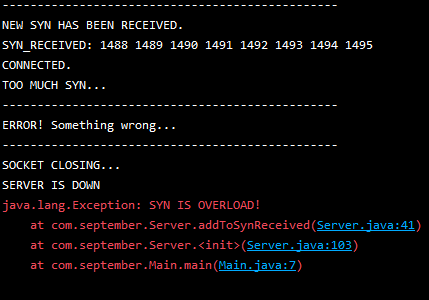
**Запуск сервера:**



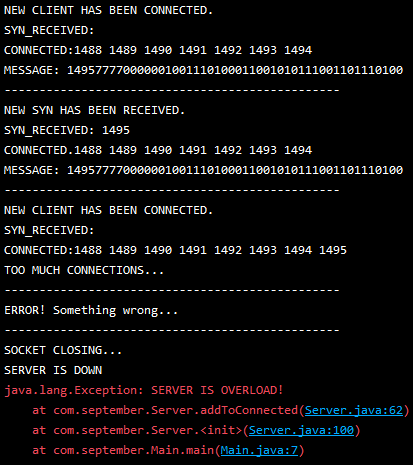
**Атака №1.** Подключение к определённому порту (сервер «засыпает» после проверки порта)**:**



**Атака №2.** Переполнение очереди на установление соединения (SYN)**:**



**Атака №3.** Перегрузка сервера (переполнение соединений)**:**



# 5. Выводы

В данной лабораторной работе изучил теоретические сведения об различных типах адресов (физический: МАС-адрес, сетевой: IP-адрес, символьный: DNS-имя) и их описания. Изучил теорию о базовых протоколах, таких как: IP, TCP и ICMP, об стеках протоколов TCP/IP, об их физическом, канальном, межсетевых и транспортных уровнях. Узнал информацию о назначении портов, по номерам которых транспортные протоколы определяют, какому приложению передать содержимое пакетов.

Научился разрабатывать приложение, реализующее атаки на протокол при установке TCP-соединения и в рамках заданного протокола прикладного уровня. В интерфейсе, которого должны быть наглядно представлены: исходные данные протокола (модули, ключи, флаги, иные данные); данные, передаваемые по сети каждой из сторон; проверки, выполняемые каждым из участников.

# 6. КОД ПРОГРАММЫ

package com.september**;**public class Main {  
  
 public static void main(String[] args) {  
 try {  
 Server s = new Server()**;** }catch (Exception e){  
 e.printStackTrace()**;** }  
 }  
}

package com.september**;**import java.io.\***;**import java.net.ServerSocket**;**import java.net.Socket**;**import java.net.SocketException**;**import java.util.ArrayList**;**class Server {  
 private int PORT = **7777;** private ArrayList<Integer> SYNReceivedList = new ArrayList<>()**;** // список клиентов которые отправили SYN  
 private ArrayList<Integer> CONNECTED = new ArrayList<>()**;** // Список подключённых клиентов  
  
 private final int SYN\_RECEIVED\_MAXSIZE = **7;** private final int CONNECTED\_MAXSIZE = **7;** //  
 private void addToSynReceived(TCP request**,** Socket clientSocket) throws Exception{  
 TCP response = new TCP().with\_source\_port(PORT)  
 .with\_destination\_port(request.destination\_port)  
 .with\_syn(**1**)**;** *out* = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()))**;** *out*.write(response.get\_String() + "\n")**;** *out*.flush()**;** SYNReceivedList.add(request.source\_port)**;** System.*out*.println("NEW SYN HAS BEEN RECEIVED.")**;** System.*out*.print("SYN\_RECEIVED: ")**;** for(Integer i : SYNReceivedList)  
 System.*out*.print(i + " ")**;** System.*out*.print("\nCONNECTED.")**;** for(Integer i : CONNECTED)  
 System.*out*.print(i + " ")**;** System.*out*.println("")**;** if (SYNReceivedList.size() > SYN\_RECEIVED\_MAXSIZE) {  
 System.*out*.println("TOO MUCH SYN...")**;** throw new Exception("SYN IS OVERLOAD!")**;** }  
 }  
  
 void addToConnected(TCP request) throws Exception{  
 SYNReceivedList.remove(SYNReceivedList.indexOf(request.source\_port))**;** CONNECTED.add(request.source\_port)**;** System.*out*.println("NEW CLIENT HAS BEEN CONNECTED.")**;** System.*out*.print("SYN\_RECEIVED:")**;** for(Integer i : SYNReceivedList)  
 System.*out*.print(i + " ")**;** System.*out*.print("\nCONNECTED:")**;** for(Integer i : CONNECTED)  
 System.*out*.print(i + " ")**;** System.*out*.println("")**;** if (CONNECTED.size() > CONNECTED\_MAXSIZE) {  
 System.*out*.println("TOO MUCH CONNECTIONS...")**;** throw new Exception("SERVER IS OVERLOAD!")**;** }  
 }  
  
 private static Socket *clientSocket***;** //сокет для общения  
 private static ServerSocket *server***;** // серверсокет  
 private static BufferedReader *in***;** // поток чтения из сокета  
 private static BufferedWriter *out***;** // поток записи в сокет  
  
 // ГЛАВНЫЙ КОНСТРУКТОР СЕРВЕРА  
 Server() throws Exception{  
 try {  
 try {  
 *server* = new ServerSocket(PORT)**;** // создаём сокет сервера  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SERVER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")**;** System.*out*.println("SERVER IS WORKING...")**;** System.*out*.println("------------------------------------------------")**;** System.*out*.println("WAIT PLEASE...")**;** System.*out*.println("------------------------------------------------")**;** *clientSocket* = *server*.accept()**;** // здесь сервер ожидает, когда кто-нибудь к нему подключится  
 System.*out*.println("SOME CLIENT IS CONNECTED")**;** try {  
  
 *in* = new BufferedReader(new InputStreamReader(*clientSocket*.getInputStream()))**;** *out* = new BufferedWriter(new OutputStreamWriter(*clientSocket*.getOutputStream()))**;** // бесконечный цикл работы сервера  
 while (true) {  
 String word = *in*.readLine()**;** // считываем сообщения от пользователя и выводим его  
 System.*out*.println("MESSAGE: " + word.replaceAll(" "**,**""))**;** System.*out*.println("------------------------------------------------")**;** TCP request = TCP.*from\_string*(word)**;** // восстанавливаем сообщение из строки  
 if (request.destination\_port != PORT) { // проверка на верное подключение (нужно чтобы совпали порты)  
 continue**;** } else if (CONNECTED.contains(request.source\_port)) { //проверяем, подключён ли клиент  
 continue**;** } else if (SYNReceivedList.contains(request.source\_port)) { // проверяем клиента в списке клиентов, которые отправили SYN  
 this.addToConnected(request)**;** // если его там нет, то добавляем его в список подключённых клиентов  
 continue**;** }  
 addToSynReceived(request**,** *clientSocket*)**;** // в противном случае, добавляем его в список клиентов которые отправили SYN  
 }  
 }  
 catch (SocketException e){  
 System.*out*.println("------------------------------------------------")**;** System.*out*.println("CLIENT DIED!")**;** }  
 catch (Exception e){  
 System.*out*.println("------------------------------------------------")**;** System.*out*.println("ERROR! Something wrong...")**;** e.printStackTrace()**;** } finally {  
 System.*out*.println("------------------------------------------------")**;** System.*out*.println("SOCKET CLOSING...")**;** *clientSocket*.close()**;** *in*.close()**;** *out*.close()**;** }  
 } finally {  
 System.*out*.println("SERVER IS DOWN")**;** *server*.close()**;** }  
 } catch (IOException e) {  
 System.*err*.println(e)**;** }  
 }  
  
}

package com.september**;**public class TCP {  
 // фрагметы TCP  
 int source\_port = **0;** // "сообщение откуда"  
 int destination\_port = **0;** // "Сообщение куда"  
 private int sequence\_number = **0;** private int acknowledgment\_number = **0;** // флаги  
 private int urg = **0;** private int ack = **0;** private int psh = **0;** private int rst = **0;** private int syn = **0;** private int fin = **0;** private String data = "test"**;** // передаваемое сообщение  
  
 // сеттеры фрагметов и флагов  
 TCP with\_source\_port(int val){  
 source\_port = val**;** return this**;** }  
 TCP with\_destination\_port(int val){  
 destination\_port = val**;** return this**;** }  
 TCP with\_sequence\_number(int val){  
 sequence\_number = val**;** return this**;** }  
 TCP with\_acknowledgment\_number(int val){  
 acknowledgment\_number = val**;** return this**;** }  
 TCP with\_urg(int val){  
 urg = val**;** return this**;** }  
 TCP with\_ack(int val){  
 ack = val**;** return this**;** }  
 TCP with\_psh(int val){  
 psh = val**;** return this**;** }  
 TCP with\_rst(int val){  
 rst = val**;** return this**;** }  
 TCP with\_syn(int val){  
 syn = val**;** return this**;** }  
 TCP with\_fin(int val){  
 fin = val**;** return this**;** }  
 TCP with\_data(String val){  
 data = val**;** return this**;** }  
  
 // метод записывающий все поля и порты в строку  
 String get\_String(){  
 String array = "" + source\_port //{0:016b}  
 + " " + destination\_port //{0:016b}  
 + " " +sequence\_number //{0:032b}  
 + " " + acknowledgment\_number //{0:032b}  
 + " " + urg  
 + " " + ack  
 + " " + psh  
 + " " + rst  
 + " " + syn  
 + " " + fin  
 + " " + data**;** return array**;** }  
  
 // метод преобразующий строку обратно в классы  
 static TCP from\_string(String s){  
 TCP obj = new TCP()**;** return obj.with\_source\_port(Integer.*parseInt*(s.split(" ")[**0**]**,10**))  
 .with\_destination\_port(Integer.*parseInt*(s.split(" ")[**1**]**,10**))  
 .with\_sequence\_number(Integer.*parseInt*(s.split(" ")[**2**]**,10**))  
 .with\_acknowledgment\_number(Integer.*parseInt*(s.split(" ")[**3**]**,10**))  
 .with\_urg((s.split(" ")[**4**].charAt(**0**) - '0'))  
 .with\_ack((s.split(" ")[**5**].charAt(**0**) - '0'))  
 .with\_psh((s.split(" ")[**6**].charAt(**0**) - '0'))  
 .with\_rst((s.split(" ")[**7**].charAt(**0**) - '0'))  
 .with\_syn((s.split(" ")[**8**].charAt(**0**) - '0'))  
 .with\_fin((s.split(" ")[**9**].charAt(**0**) - '0'))  
 .with\_data(s.split(" ")[**10**])**;** }  
}

package com.september**;**import java.io.BufferedReader**;**import java.io.BufferedWriter**;**import java.io.InputStreamReader**;**import java.io.OutputStreamWriter**;**import java.net.Socket**;**public class Hacker1 {  
 Hacker1(){  
 Socket clientSocket**;** BufferedReader reader**;** BufferedReader in**;** BufferedWriter out**;** try {  
 clientSocket = new Socket("localhost"**, 7777**)**;** reader = new BufferedReader(new InputStreamReader(System.*in*))**;** in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()))**;** out = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()))**;** String request = new TCP().with\_source\_port(**6661**).with\_destination\_port(**7777**).with\_syn(**1**).get\_String()**;** out.write(request + "\n")**;** out.flush()**;** String message = in.readLine()**;** System.*out*.println(message)**;** TCP responce = TCP.*from\_string*(message)**;** System.*out*.println(responce.ack)**;** }catch (Exception e){  
 e.printStackTrace()**;** }  
  
 }  
}

package com.september**;**import java.io.BufferedReader**;**import java.io.BufferedWriter**;**import java.io.InputStreamReader**;**import java.io.OutputStreamWriter**;**import java.net.Socket**;**public class Hacker2 {  
 Hacker2(){  
 Socket clientSocket**;** BufferedReader reader**;** BufferedReader in**;** BufferedWriter out**;** try {  
 clientSocket = new Socket("localhost"**, 7777**)**;** // этой строкой мы запрашиваем  
 reader = new BufferedReader(new InputStreamReader(System.*in*))**;** in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()))**;** out = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()))**;** for(int i = **0;** i < **10;** i++ ) {  
 String request = new TCP().with\_source\_port(**1488** + i).with\_destination\_port(**7777**).with\_syn(**1**).get\_String()**;** out.write(request + "\n")**;** out.flush()**;** }  
 }catch (Exception e){  
 e.printStackTrace()**;** }  
  
 }  
}

package com.september**;**import java.io.BufferedReader**;**import java.io.BufferedWriter**;**import java.io.InputStreamReader**;**import java.io.OutputStreamWriter**;**import java.net.Socket**;**public class Hacker3 {  
 Hacker3(){  
 Socket clientSocket**;** BufferedReader reader**;** BufferedReader in**;** BufferedWriter out**;** try {  
 clientSocket = new Socket("localhost"**, 7777**)**;** // этой строкой мы запрашиваем  
 reader = new BufferedReader(new InputStreamReader(System.*in*))**;** in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()))**;** out = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()))**;** for(int i = **0;** i < **10;** i++ ) {  
 String request = new TCP().with\_source\_port(**1488** + i).with\_destination\_port(**7777**).with\_syn(**1**).get\_String()**;** out.write(request + "\n")**;** out.flush()**;** String request2 = new TCP().with\_source\_port(**1488** + i).with\_destination\_port(**7777**).get\_String()**;** out.write(request + "\n")**;** out.flush()**;** }  
 }catch (Exception e){  
 e.printStackTrace()**;** }  
  
 }  
}

package com.september**;**public class Main {  
  
 public static void main(String[] args) {  
 Hacker1 a = new Hacker1()**;** }  
}

package com.september**;**public class TCP {  
 int source\_port = **0;** int destination\_port = **0;** int sequence\_number = **0;** int acknowledgment\_number = **0;** int urg = **0;** int ack = **0;** int psh = **0;** int rst = **0;** int syn = **0;** int fin = **0;** String data = "test"**;** TCP with\_source\_port(int val){  
 source\_port = val**;** return this**;** }  
 TCP with\_destination\_port(int val){  
 destination\_port = val**;** return this**;** }  
 TCP with\_sequence\_number(int val){  
 sequence\_number = val**;** return this**;** }  
 TCP with\_acknowledgment\_number(int val){  
 acknowledgment\_number = val**;** return this**;** }  
 TCP with\_urg(int val){  
 urg = val**;** return this**;** }  
 TCP with\_ack(int val){  
 ack = val**;** return this**;** }  
 TCP with\_psh(int val){  
 psh = val**;** return this**;** }  
 TCP with\_rst(int val){  
 rst = val**;** return this**;** }  
 TCP with\_syn(int val){  
 syn = val**;** return this**;** }  
 TCP with\_fin(int val){  
 fin = val**;** return this**;** }  
 TCP with\_data(String val){  
 data = val**;** return this**;** }  
  
 String get\_String(){  
 String array = "" + source\_port //{0:016b}  
 + " " + destination\_port //{0:016b}  
 + " " +sequence\_number //{0:032b}  
 + " " + acknowledgment\_number //{0:032b}  
 + " " + urg  
 + " " + ack  
 + " " + psh  
 + " " + rst  
 + " " + syn  
 + " " + fin  
 + " " + *text2bin*(data)**;** return array**;** }  
  
 static TCP from\_string(String s){  
 TCP obj = new TCP()**;** return obj.with\_source\_port(Integer.*parseInt*(s.split(" ")[**0**]**,10**))  
 .with\_destination\_port(Integer.*parseInt*(s.split(" ")[**1**]**,10**))  
 .with\_sequence\_number(Integer.*parseInt*(s.split(" ")[**2**]**,10**))  
 .with\_acknowledgment\_number(Integer.*parseInt*(s.split(" ")[**3**]**,10**))  
 .with\_urg((s.split(" ")[**4**].charAt(**0**) - '0'))  
 .with\_ack((s.split(" ")[**5**].charAt(**0**) - '0'))  
 .with\_psh((s.split(" ")[**6**].charAt(**0**) - '0'))  
 .with\_rst((s.split(" ")[**7**].charAt(**0**) - '0'))  
 .with\_syn((s.split(" ")[**8**].charAt(**0**) - '0'))  
 .with\_fin((s.split(" ")[**9**].charAt(**0**) - '0'))  
 .with\_data(*bin2text*(s.split(" ")[**10**]))**;** }  
  
 static String text2bin(String s){  
 byte[] bytes = s.getBytes()**;** StringBuilder binary = new StringBuilder()**;** for(byte b : bytes){  
 int val = b**;** for(int i = **0;** i < **8;** i++){  
 binary.append((val & **128**) == **0** ? **0** : **1**)**;** val <<= **1;** }  
 }  
 return binary.toString()**;** }  
 static String bin2text(String s){  
 StringBuilder text = new StringBuilder()**;** for(int i = **0;** i <= s.length() - **8;** i += **8**){  
 text.append((char)Integer.*parseInt*(s.substring(i**,**i+**8**)**,2**))**;** }  
 return text.toString()**;** }  
  
}