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. 853504

Шевченя И.В.

Проверил

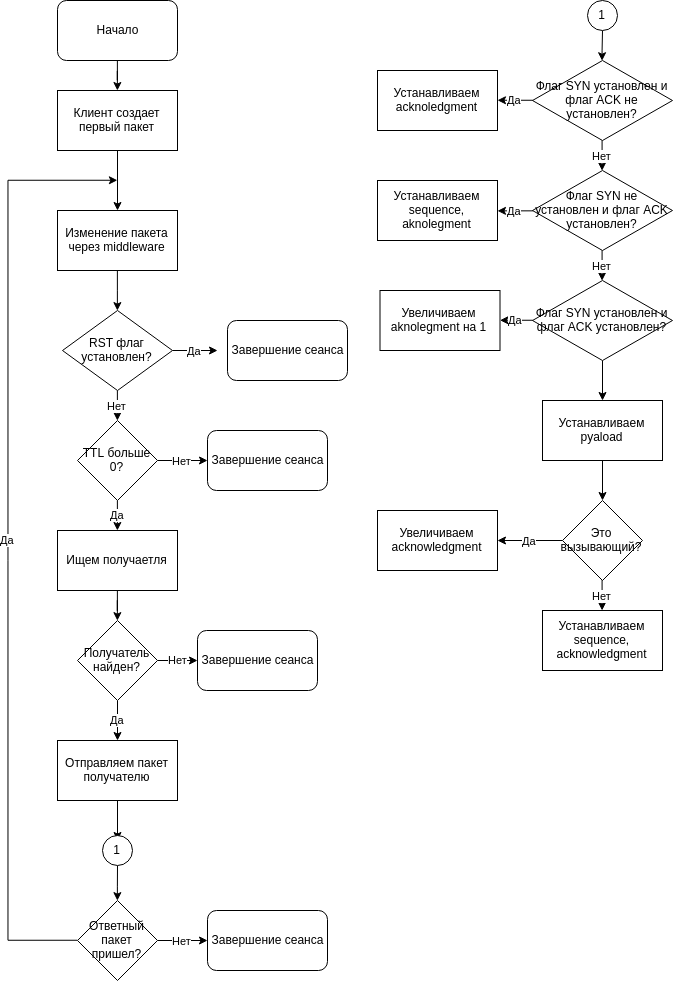
Протько М.И.

Минск, 2021

# 1. Введение

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

2. Блок-схемы алгоритмов



**3. Теоретические сведения**

**Адресация в сети Internet.**

**Типы адресов.**

Типы адресов:

* **Физический (MAC-адрес)**
* **Сетевой (IP-адрес)**
* **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**

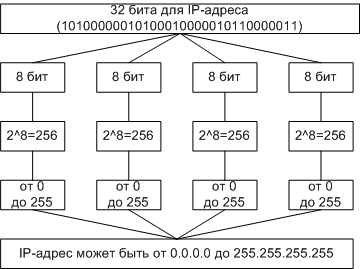
**IPv4** - адрес является уникальным 32-битным идентификатором IP-интерфейса в Интернет.

**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

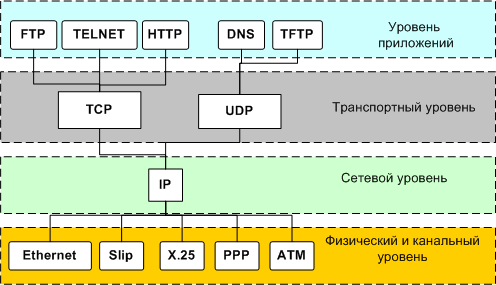
131 - номер хоста

**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.



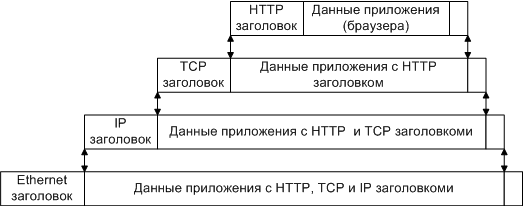
Стек протоколов TCP/IP

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.

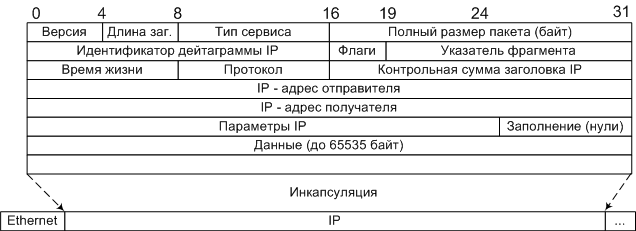
*Программа, реализующая функции того или иного протокола, часто называется модулем, например, “IP-модуль”, “модуль TCP”.*

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.

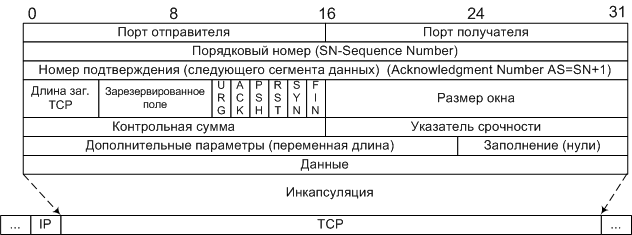
**Протокол надежной доставки сообщений TCP**

**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.

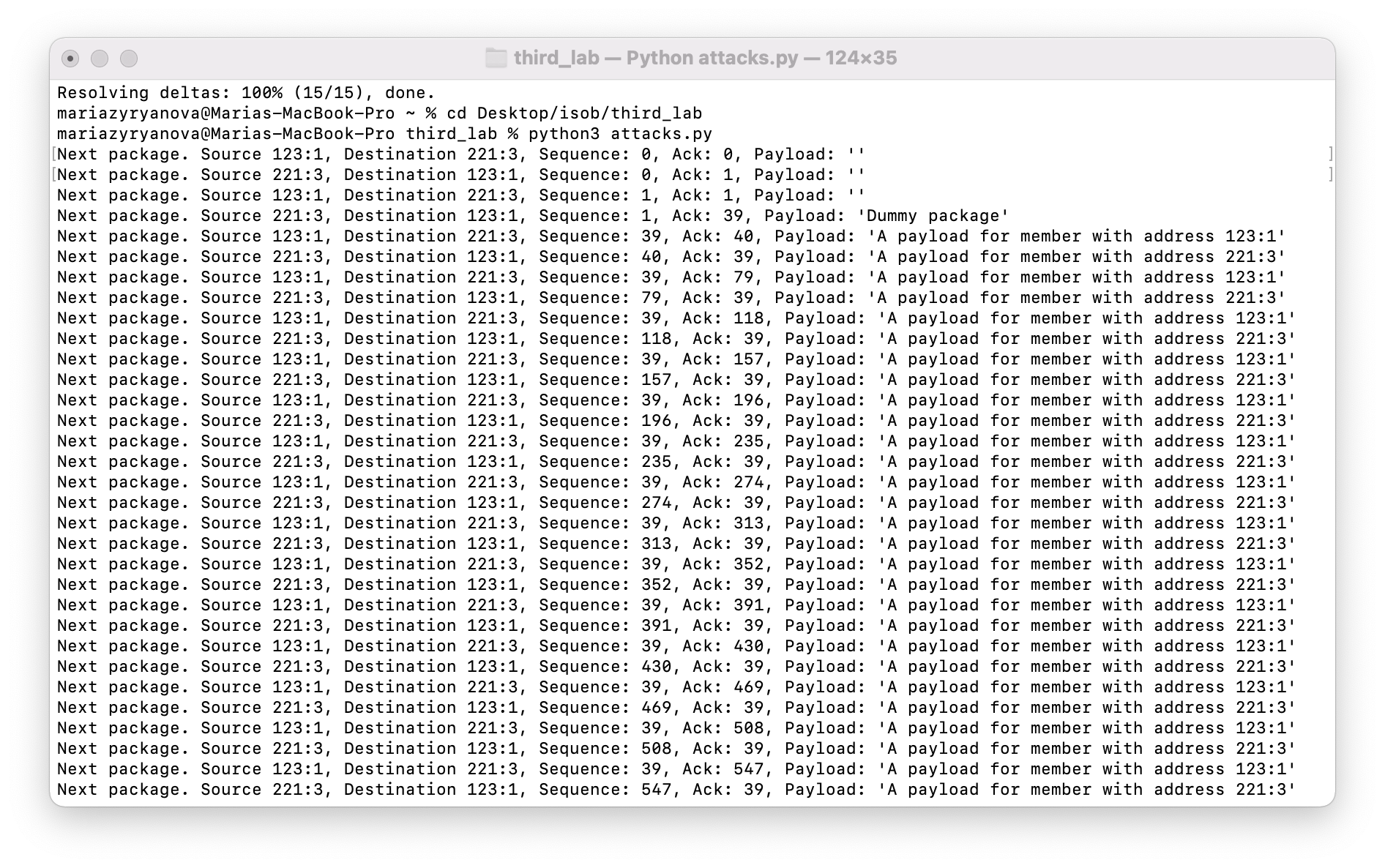
Работу порта, также можно проверить с помощью telnet.

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

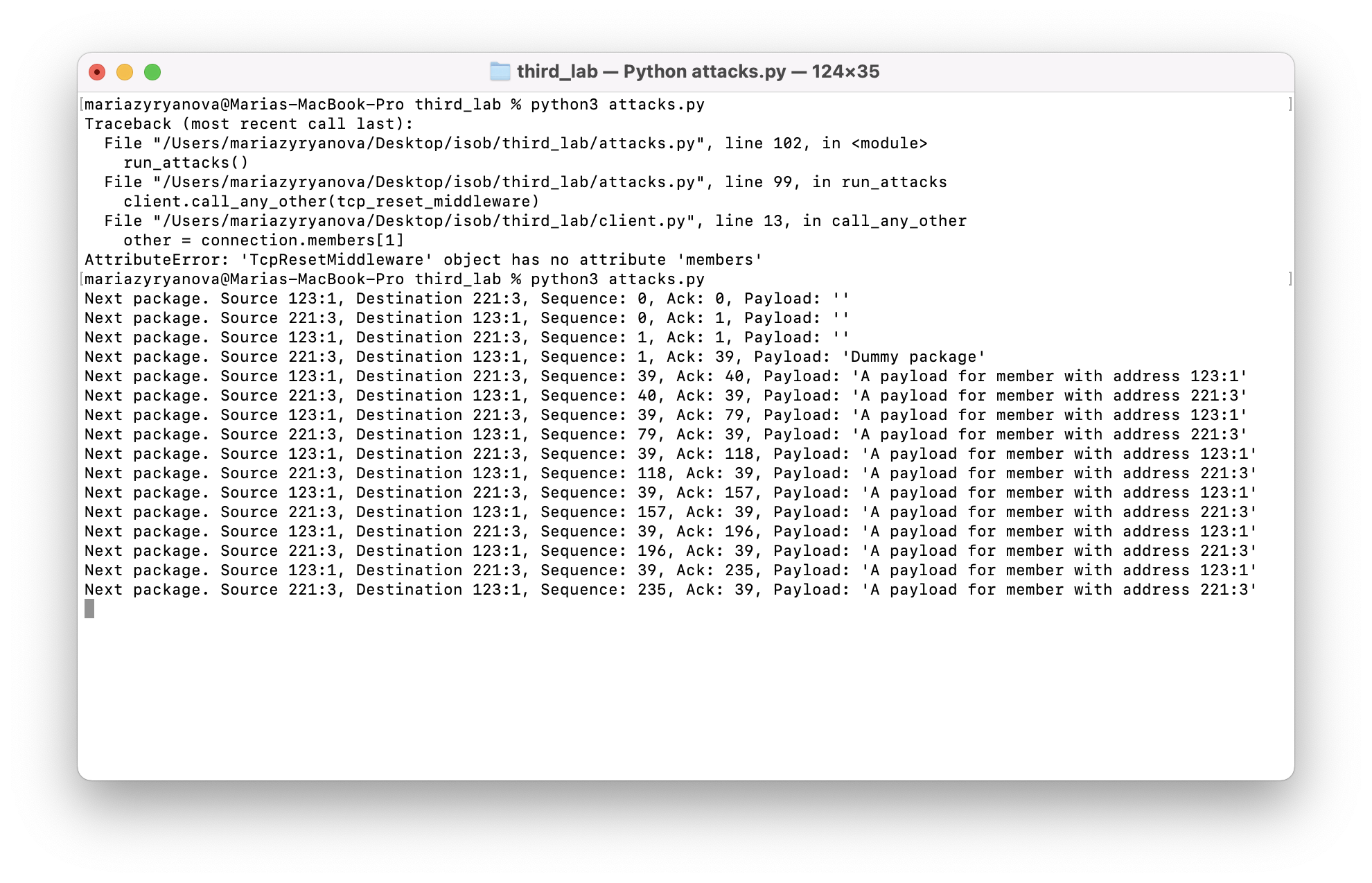
|  |  |  |
| --- | --- | --- |
| **Порт** | **Служба** | **Описание** |
| 0 | - | Зарезервировано |
| 13 | Daytime | Синхронизация времени |
| 20 | ftp-data | Канал передачи данных для FTP |
| 21 | ftp | Передача файлов |
| 23 | telnet | Сетевой терминал |
| 25 | SMTP | Передача почты |
| 37 | time |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

# 4. Результаты выполнения программы

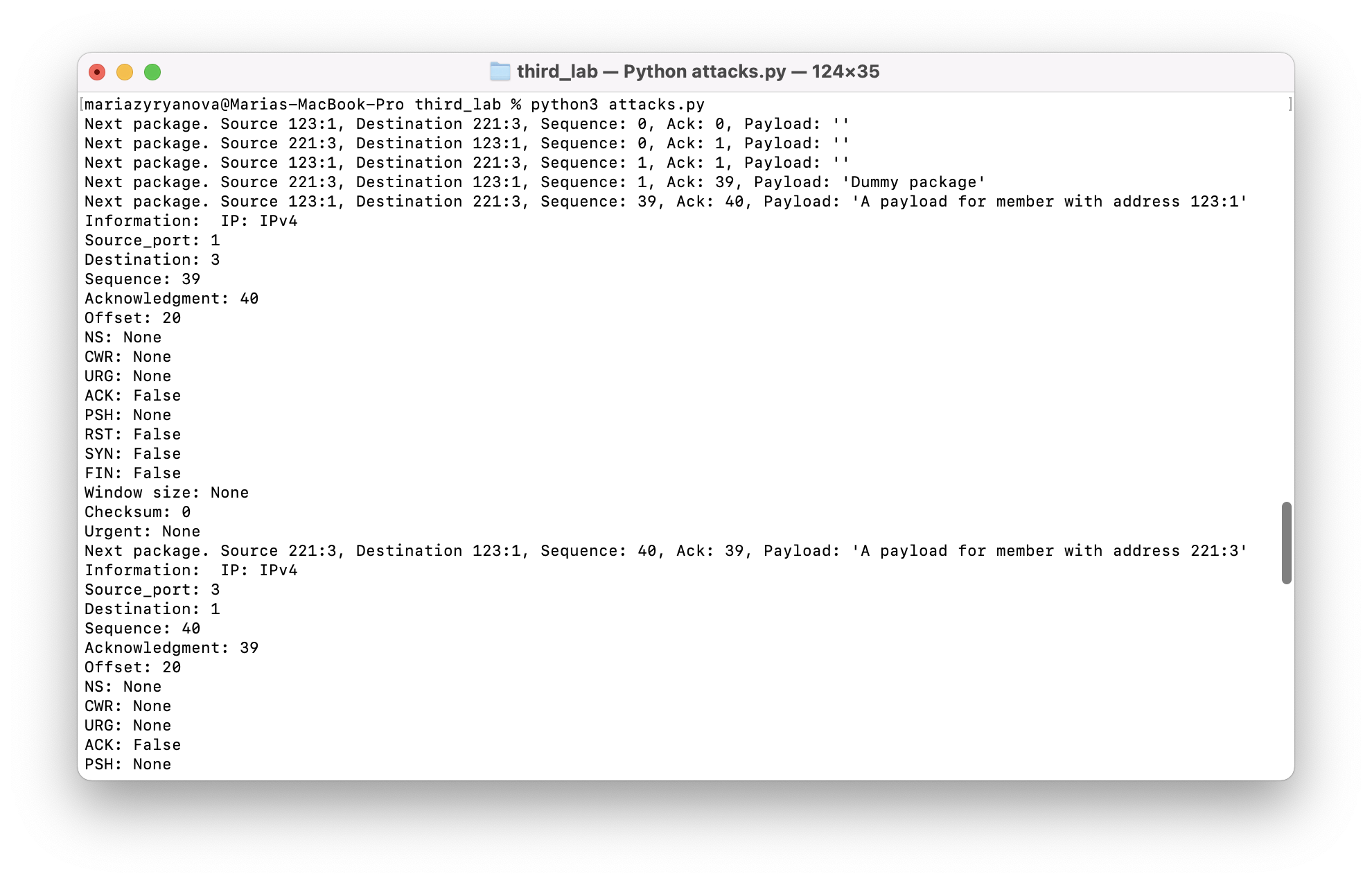
**Обычное взаимодействие:**

****

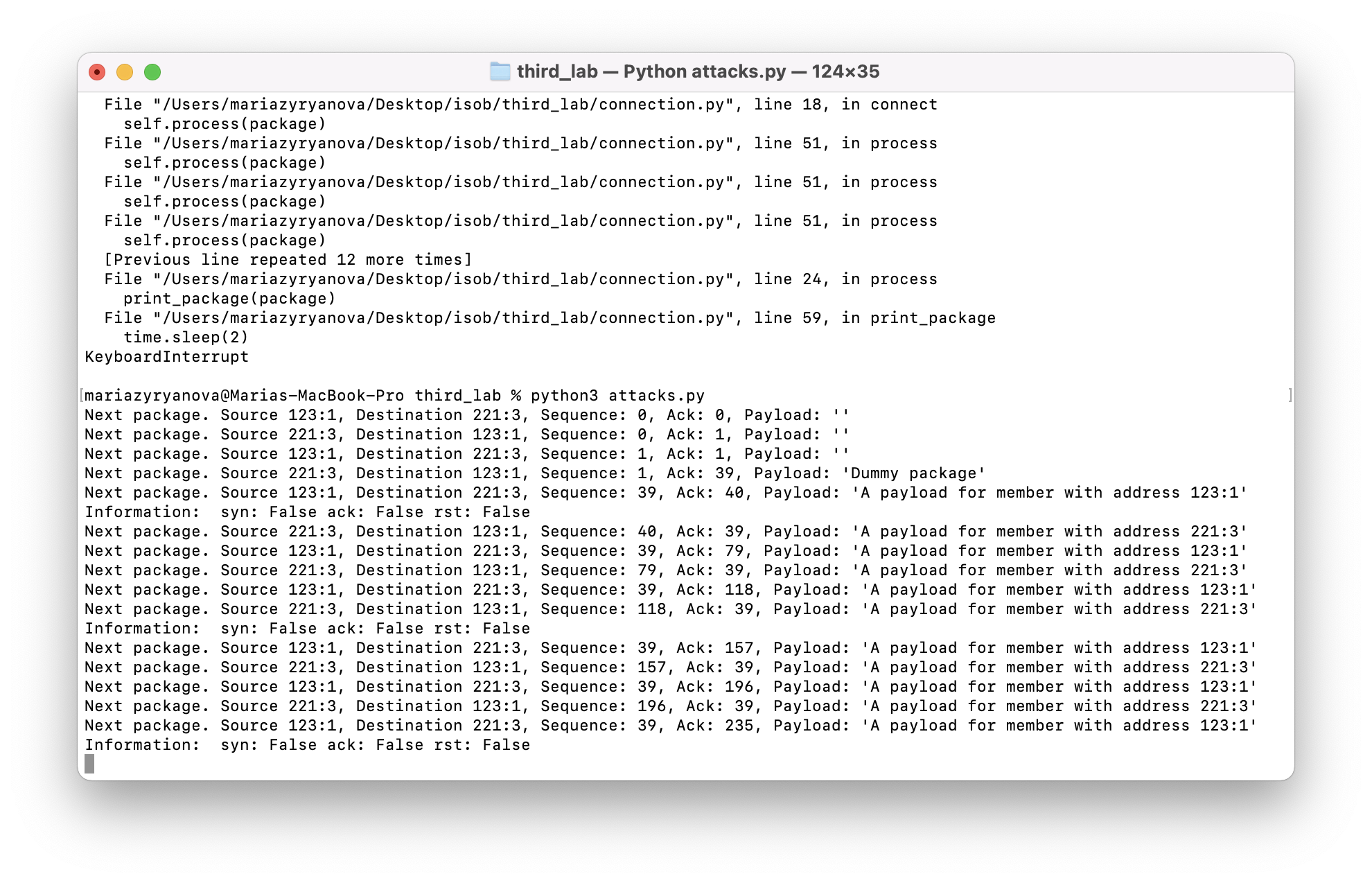
**TCP reset:**

****

**Sniffing:**

****

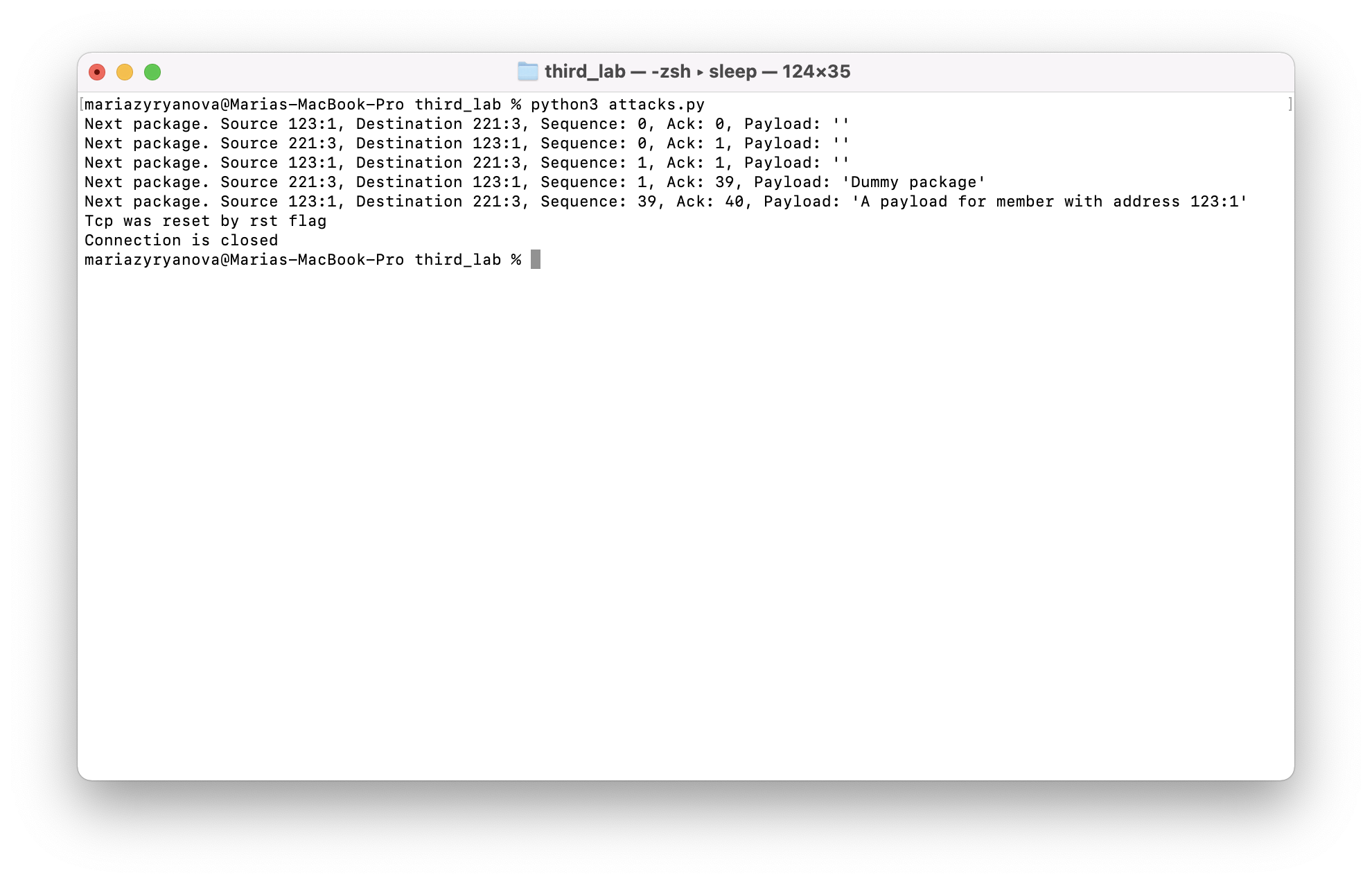
**Passive scan:**

****

**Fake ip address:**

****

**RST:**

****

**Connection Hijack:**

****

# 5. Выводы

В ходе выполнения лабораторной работы были cоздано приложение, реализующее атаки на протокол при установке TCP-соединения и в рамках заданного протокола прикладного уровня.

# 6. КОД ПРОГРАММЫ

**IP:**

class IP:  
 def \_\_init\_\_(self, source\_ip, destination\_ip, payload):  
 self.version = 4  
 self.ihl = 5  
 self.dscp = None  
 self.ecn = None  
 self.total\_length = 576  
 self.id = None  
 self.flags = None  
 self.fragment\_offset = None  
 self.ttl = 15  
 self.protocol = 6  
 self.checksum = None  
 self.source\_ip = source\_ip  
 self.destination\_ip = destination\_ip  
 self.payload = payload  
  
 def \_\_str\_\_(self):  
 return f"IPv{self.version}"  
  
 def \_\_repr\_\_(self):  
 return "\n".join(  
 [  
 f"Version: {self.version}",  
 f"IHL: {self.ihl}",  
 f"DSCP: {self.dscp}",  
 f"ECN: {self.ecn}",  
 f"Total length: {self.total\_length}",  
 f"ID: {self.id}",  
 f"Flags: {self.flags}",  
 f"Fragment offset: {self.fragment\_offset}",  
 f"TTL: {self.ttl}",  
 f"Protocol: {self.protocol}",  
 f"Checksum: {self.checksum}",  
 f"Source IP: {self.source\_ip}",  
 f"Destination IP: {self.destination\_ip}",  
 f"Parameters: {self.payload}",  
 ]  
 )

**TCP:**

class TCP:  
 def \_\_init\_\_(self, source\_port, destination\_port, ip):  
 self.ip = ip  
 self.source\_port = source\_port  
 self.destination\_port = destination\_port  
 self.sequence = 0  
 self.acknowledgment = 0  
 self.offset = 20  
 self.ns = None  
  
 # Reserved  
 self.cwr = None  
 self.ece = None  
  
 # Flags  
 self.urg = None  
 self.ack = False  
 self.psh = None  
 self.rst = False  
 self.syn = False  
 self.fin = False  
 self.window\_size = None  
 self.checksum = 0  
 self.urgent = None  
  
 def \_\_str\_\_(self):  
 return f"Source {self.ip.source\_ip}:{self.source\_port}, " \  
 f"Destination {self.ip.destination\_ip}:{self.destination\_port}, " \  
 f"Sequence: {self.sequence}, " \  
 f"Ack: {self.acknowledgment}, " \  
 f"Payload: '{self.ip.payload}'"  
  
 def \_\_repr\_\_(self):  
 return "\n".join(  
 [  
 f"IP: {self.ip}",  
 f"Source\_port: {self.source\_port}",  
 f"Destination: {self.destination\_port}",  
 f"Sequence: {self.sequence}",  
 f"Acknowledgment: {self.acknowledgment}",  
 f"Offset: {self.offset}",  
 f"NS: {self.ns}",  
 f"CWR: {self.cwr}",  
 f"URG: {self.urg}",  
 f"ACK: {self.ack}",  
 f"PSH: {self.psh}",  
 f"RST: {self.rst}",  
 f"SYN: {self.syn}",  
 f"FIN: {self.fin}",  
 f"Window size: {self.window\_size}",  
 f"Checksum: {self.checksum}",  
 f"Urgent: {self.urgent}",  
 ]  
 )

**Connection:**

import time  
  
  
class Connection:  
 def \_\_init\_\_(self, members, middlewares):  
 self.members = members  
 self.middlewares = middlewares  
 self.closed = False  
 self.connected = False  
  
 def \_\_find\_receiver(self, package):  
 for member in self.members:  
 if member.ip\_address == package.ip.destination\_ip and member.tcp\_port == package.destination\_port:  
 return member  
  
 def connect(self, package):  
 self.connected = True  
 self.process(package)  
  
 def process(self, package):  
 if not self.connected or self.closed:  
 return  
  
 print\_package(package)  
  
 for middleware in self.middlewares:  
 package = middleware.change(package)  
  
 if package.rst:  
 print('Tcp was reset by rst flag')  
 self.close()  
 return  
  
 package.ip.ttl -= 1  
 if package.ip.ttl <= 0:  
 print('Package ttl is expired')  
 self.close()  
 return  
  
 receiver = self.\_\_find\_receiver(package)  
 if receiver is None:  
 print('Unknown destination {}:{}'.format(package.ip.destination\_ip, package.destination\_port))  
 self.close()  
 return  
  
 package = receiver.receive(package)  
 if package is None:  
 print('One of members stop sending requests')  
 self.close()  
 else:  
 self.process(package)  
  
 def close(self):  
 self.closed = True  
 print('Connection is closed')  
  
  
def print\_package(package):  
 time.sleep(2)  
 print("Next package. {}".format(package))

**Client:**

from third\_lab.ip import IP  
from third\_lab.tcp import TCP  
  
  
class Client:  
 def \_\_init\_\_(self, ip\_address, tcp\_port):  
 self.ip\_address = ip\_address  
 self.tcp\_port = tcp\_port  
 self.caller = False  
  
 def call\_any\_other(self, connection):  
 self.caller = True  
 other = connection.members[1]  
 package = self.\_\_build\_package(other, self.\_\_generate\_payload())  
 connection.connect(package)  
  
 def \_\_build\_package(self, receiver, payload):  
 ip = IP(self.ip\_address, receiver.ip\_address, "")  
 tcp = TCP(self.tcp\_port, receiver.tcp\_port, ip)  
 tcp.sequence = 0  
 tcp.syn = True  
 return tcp  
  
 def \_build\_answer(self, package, payload):  
 ip = IP(package.ip.destination\_ip, package.ip.source\_ip, "")  
 tcp = TCP(package.destination\_port, package.source\_port, ip)  
  
 if package.syn and package.ack:  
 tcp.sequence = package.acknowledgment  
 tcp.acknowledgment = package.sequence + 1  
 tcp.ack = True  
 return tcp  
  
 if package.syn:  
 tcp.sequence = 0  
 tcp.acknowledgment = package.sequence + 1  
 tcp.syn = True  
 tcp.ack = True  
 return tcp  
  
 if package.ack:  
 tcp.sequence = package.acknowledgment  
 tcp.acknowledgment = len(payload)  
 tcp.ip.payload = "Dummy package"  
 return tcp  
  
 tcp.ip.payload = payload  
  
 if self.caller:  
 tcp.sequence = package.acknowledgment  
 tcp.acknowledgment = package.sequence + len(payload)  
 else:  
 tcp.sequence = package.acknowledgment  
 tcp.acknowledgment = package.sequence  
 return tcp  
  
 def \_\_generate\_payload(self):  
 return "A payload for member with address {}:{}".format(self.ip\_address, self.tcp\_port)  
  
 def receive(self, package):  
 answer = self.\_build\_answer(package, self.\_\_generate\_payload())  
 return answer

**Hacker:**

from third\_lab.client import Client  
  
  
class Hacker(Client):  
 def \_\_init\_\_(self, ip\_address, tcp\_port):  
 super(Hacker, self).\_\_init\_\_(ip\_address, tcp\_port)  
  
 def receive(self, package):  
 return self.\_build\_answer(package, "Hacker server payload")

**Attacks:**

import time  
  
from third\_lab.connection import Connection, print\_package  
from third\_lab.client import Client  
from third\_lab.hacker import Hacker  
  
  
class RSTMiddleware:  
 def \_\_init\_\_(self):  
 self.call\_number = 0  
  
 def change(self, package):  
 self.call\_number += 1  
 if self.call\_number == 5:  
 package.rst = True  
 return package  
  
  
class TcpResetMiddleware:  
 def \_\_init\_\_(self):  
 self.call\_number = 0  
  
 def change(self, package):  
 self.call\_number += 1  
 if self.call\_number == 5:  
 package.rst = False  
 return package  
  
  
class PassiveScan:  
 def \_\_init\_\_(self):  
 self.call\_number = 0  
  
 def change(self, package):  
 self.call\_number += 1  
 if self.call\_number >= 5 and self.call\_number % 5 == 0:  
 package.ip.payload = "Passive scan"  
 print("Information: ", "syn:", package.syn, "ack:", package.ack, "rst:", package.rst)  
 return package  
  
  
class Sniffing:  
 def \_\_init\_\_(self):  
 self.call\_number = 0  
  
 def change(self, package):  
 self.call\_number += 1  
 if self.call\_number >= 5:  
 package.ip.payload = "Sniffing"  
 print("Information: ", repr(package))  
 return package  
  
  
class FakeIpAddressMiddleware:  
 def \_\_init\_\_(self, ip\_address, tcp\_port):  
 self.ip\_address = ip\_address  
 self.tcp\_port = tcp\_port  
 self.call\_number = 0  
  
 def change(self, package):  
 self.call\_number += 1  
 if self.call\_number == 5:  
 package.ip.destination\_ip = self.ip\_address  
 package.destination\_port = self.tcp\_port  
 return package  
  
  
class ConnectionHijack:  
 def \_\_init\_\_(self):  
 self.call\_number = 0  
  
 def change(self, package):  
 self.call\_number += 1  
 if self.call\_number >= 5:  
 package.ip.payload = "Connection hijacked"  
 t = package.sequence  
 package.sequence = package.acknowledgment  
 package.acknowledgment = t + len(package.ip.payload)  
 package.ip.destination\_ip = package.ip.source\_ip  
 package.destination\_port = package.source\_port  
 print\_package(package)  
 return package  
  
  
def run\_attacks():  
 client = Client(123, 1)  
 server1 = Client(221, 3)  
 server2 = Hacker(231, 2)  
  
 tcp\_reset\_middleware = TcpResetMiddleware()  
 sniffing = Sniffing()  
 passive\_scan = PassiveScan()  
 fake\_ip\_address\_middleware = FakeIpAddressMiddleware(231, 2)  
 rst\_middleware = RSTMiddleware()  
 connection\_hijack = ConnectionHijack()  
 connection = Connection([client, server1, server2], [passive\_scan])  
  
 client.call\_any\_other(connection)  
  
  
run\_attacks()