БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Факультет КСИС

Специальность ИиТП

Контрольная работа №2

по дисциплине «Информационные сети. Основы безопасности»

методы и средства защиты программного, аппаратного обеспечения от

несанкционированного использования

Выполнил студент: Драгун О.В.

группа 893551

Зачетная книжка № 2520050

Минск 2022

Оглавление

[Введение 3](#_Toc97397267)

[Краткие теоретические сведения 4](#_Toc97397268)

[Блок-схема алгоритма 8](#_Toc97397269)

[Демонстрация работы программы 9](#_Toc97397270)

[Вывод 13](#_Toc97397271)

[Исходный код программы 14](#_Toc97397272)

# Введение

1). Реализовать на выбор 3 метода обфуксации программного кода приложения, разработанного в рамках лабораторных работ 4,5, позволяющие защитить ПО от несанкционированного использования в следующих комбинациях:

1.1) По одному.

1.2) Любые 2 на выбор из трех одновременно.

1.3) Все три одновременно.

2). Протестировать работоспособность приложения с запутанным программным кодом.

3). Проверить и пояснить следующие свойства, которым должна удовлетворять запутанная программа:

3.1) Запутывание должно быть замаскированным. То, что к программе были применены запутывающие преобразования, не должно бросаться в глаза.

3.2) Запутывание не должно быть регулярным. Регулярная структура запутанной программы или её фрагмента позволяет человеку отделить запутанные части и даже идентифицировать алгоритм запутывания.

# Краткие теоретические сведения

Обфускаация или запутывание кода — приведение исходного текста или исполняемого кода программы к виду, сохраняющему её функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию при декомпиляции.

«Запутывание» кода может осуществляться на уровне алгоритма, исходного текста и/или ассемблерного текста. Для создания запутанного ассемблерного текста могут использоваться специализированные компиляторы, использующие неочевидные или недокументированные возможностисреды исполнения программы. Существуют также специальные программы, производящие обфускацию, называемые обфускаторами.

**2.1 Минификация**

Минификация. Этот метод позволяет заметно сократить исходный код программы, заменяя длинные человекочитаемые имена переменных на короткие, удаляет константы, сжимает код программы, удаляя пробелы, отступы и прочие спец символы которые упрощают чтение программ но не влияют на результат исполнения программы.

Данный тип обфускации на самом деле же редко используется чтобы запутать злоумышленника, зачастую так делают с целью уменьшения расходуемой памяти занимаемой самим скриптом. Самый наглядный пример - js скрипты. При использовании современного веб приложения будут в любом случае задействованы js скрипты. Для того чтобы исполнить данный скрипт на стороне клиента, его надо передать веб сервером клиенту, и соответственно получается существенное сокращение расхода трафика, если вместо читаемых кодов передавать минифицированные скрипты.

При использовании такого алгоритма обфускации невозможно восстановить исходный код программы, даже зная алгоритм, но возможно восстановить формат кода (например, известные код форматеры для c++ - uncrustify и clang-format). В лабораторной работе используется минификатор PyMinifier, который может работать в двух режимах - минификация побочных символов и минификация имён переменных. Также на примере были разобраны минифицированные js скрипты для библиотеки bootstrap.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.1 Исходный вариант фрагмента js библиотеки

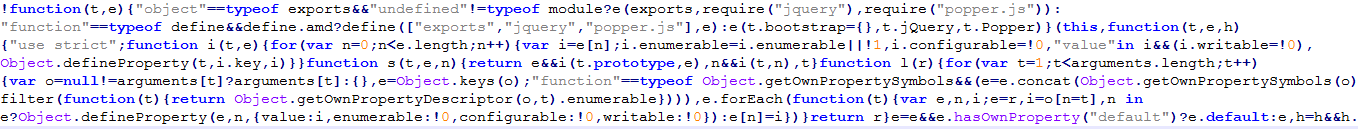


Рисунок 1.2 Минифицированный вариант фрагмента той же библиотеки

**2.2 Обфускация на уровне алгоритма**

Обфускация на уровне алгоритма - также один из популярных методов обфускации. Данный метод уже может активно применяться для запутывания злоумышленника, так как даже при восстановлении формата кода (если был использован предыдущий метод обфускации), код остаётся по прежнему практически не читаемым в некоторых местах. Один из самых простых примеров.

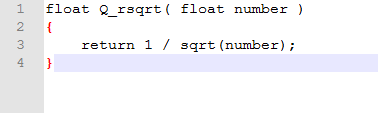


Рисунок 2.1 Обратный корень

Изображение выглядит как текст

Автоматически созданное описание  
Рисунок 2.2 Быстрый обратный квадратный корень

Примечание: данный пример достоверно точно нельзя назвать обфускацией, хотя и оба метода выполняют одну и ту же функцию, но учитывая время появления первого алгоритма (1999 год, вероятно разработан для игры Quake 3: Arena для подсчёта нормалей), он работал гораздо быстрее чем второй вариант, так как процессорная функция sqrt стоит сильно дороже чем вышепредставленный аналог.

Данный метод обфускации будет реализован вручную, для наглядной демонстрации данного типа на конкретном примере.

**2.3 Обфускация на уровне байт-кода**

Обфускация на уровне байт-кода. Также один из популярных методов обфускации, который позволяет менять программу уже на самом низком уровне. В случае С# и Java можно менять промежуточный код на соответствующих платформах, в случае с компилируемыми языками можно использовать специальные компиляторы которые будут вставлять лишние и запутывающие инструкции прямо в наш скомпилированный код. В случае питона можно скомпилированные в рантайме объекты (co\_objects) обфусцировать, запутывая инструкции и подставляя также лишние или более тяжело читаемые инструкции вместо старых.

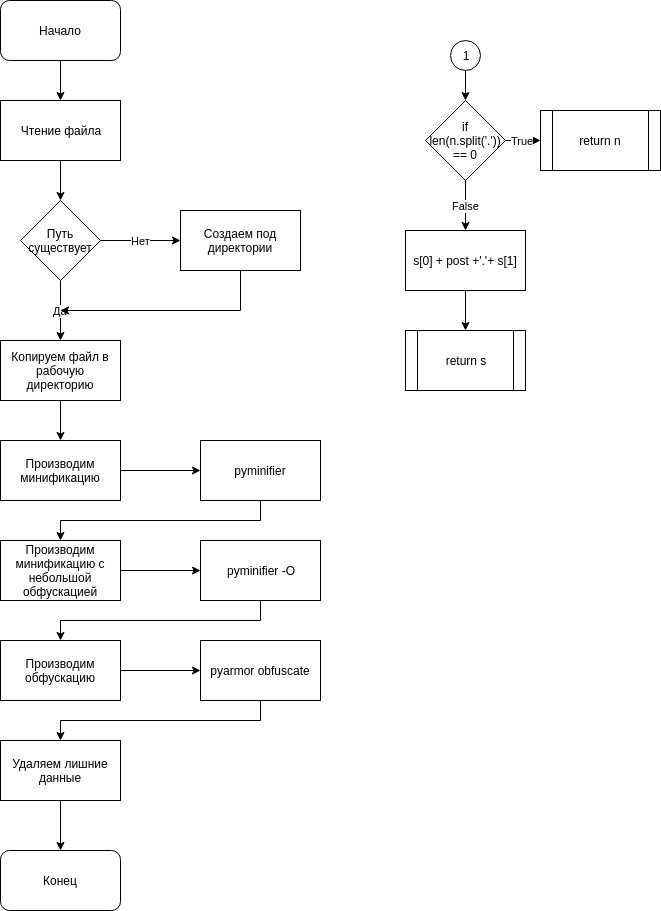
Для такого метода используется библиотека Pyarmor, которая умеет:

- обфусцировать скомпилированные объекты

- подчищать локальные переменные на каждом кадре (frame) виртуальной машины

- обфусцировать код всей программы, шифруя байт-код и проверяя его на рантайме. Таким образом, исходный код python скрипта полностью шифруется и не доступен злоумышленнику без ключа.

# Блок-схема алгоритма

Рисунок 3.1. Блок-схема алгоритма программы

# Демонстрация работы программы

Обфусцировал первую контрольную работу, полные файлы будут в приложении к ответу

import time  
  
  
def print\_package(package):  
 try:  
 time.sleep(1.5)  
 except KeyboardInterrupt:  
 raise BaseException("Interrupted by user")  
  
 print("Next package. {}".format(package))  
  
  
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 # let it be 576 to not fragment package  
 self.id = None  
 self.flags = None  
 self.fragment\_offset = None  
 self.ttl = 15 # let it be max  
 self.protocol = 6 # tcp code  
 self.checksum = None # ignore in ip  
 self.source\_ip = source\_ip  
 self.destination\_ip = destination\_ip  
 self.payload = payload  
  
  
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 # there is no additional options  
 self.ns = None  
 self.cwr = None  
 self.ece = None  
 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 'Source {}:{}, Destination {}:{}, Seq: {}, Ack: {} Payload: "{}"'.format(  
 self.ip.source\_ip, self.source\_port, self.ip.destination\_ip, self.destination\_port,  
 self.sequence, self.acknowledgment, self.ip.payload)

Рисунок 4.1. Часть неизмененного кода.

import time  
def print\_package(package):  
 try:  
 time.sleep(1.5)  
 except KeyboardInterrupt:  
 raise BaseException("Interrupted by user")  
 print("Next package. {}".format(package))  
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  
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  
 self.cwr=None  
 self.ece=None  
 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

Рисунок 4.2. Часть минимизированного кода.

import time  
D=KeyboardInterrupt  
v=BaseException  
V=print  
o=None  
N=False  
U=True  
E=len  
a=time.sleep  
def x(u):  
 try:  
 a(1.5)  
 except D:  
 raise v("Interrupted by user")  
 V("Next package. {}".format(u))  
class G:  
 def \_\_init\_\_(n,L,z,s):  
 n.version=4  
 n.ihl=5  
 n.dscp=o  
 n.ecn=o  
 n.total\_length=576   
 n.id=o  
 n.flags=o  
 n.fragment\_offset=o  
 n.ttl=15   
 n.protocol=6   
 n.checksum=o   
 n.source\_ip=L  
 n.destination\_ip=z  
 n.payload=s  
class j:  
 def \_\_init\_\_(n,J,g,ip):  
 n.ip=ip  
 n.source\_port=J  
 n.destination\_port=g  
 n.sequence=0  
 n.acknowledgment=0  
 n.offset=20   
 n.ns=o  
 n.cwr=o  
 n.ece=o  
 n.urg=o  
 n.ack=N  
 n.psh=o  
 n.rst=N  
 n.syn=N  
 n.fin=N  
 n.window\_size=o  
 n.checksum=0  
 n.urgent=o

Рисунок 4.3. Часть Минимизированного кода со слабой обфускацией

from pytransform import pyarmor\_runtime  
pyarmor\_runtime()  
\_\_pyarmor\_\_(\_\_name\_\_, \_\_file\_\_, b'\x50\x59\x41\x52\x4d\x4f\x52\x00\x00\x03\x09\x00\x61\x0d\x0d\x0a\x09\x30\xe0\x02\x00\x00\x00\x00\x01\x00\x00\x00\x40\x00\x00\x00\xdb\x2b\x00\x00\x00\x00\x00\x18\x71\x25\xf7\x06\xd8\xbb\xe5\x9d\x31\x60\xc4\x6c\x9e\xad\x7f\xff\x00\x00\x00\x00\x00\x00\x00\x00\xf4\xf5\x1a\x24\xe5\xef\xff\x61\x9a\x78\x59\xd7\xfa\xf6\x78\xb3\xfe\x12\x8e\x46\x1e\xb4\xb3\xb3\x27\xa2\x20\xd1\xb7\x0b\xca\xce\x95\x91\xe7\xd4\xc7\xa0\x4e\xe4\x3b\x73\x46\x0c\xca\x04\x44\xe0\xfb\xf4\xd3\x1d\xe7\xc7\xb1\x4f\x22\x2a\x27\xd6\xe9\x17\x3a\x71\x34\x1d\x46\x48\x34\xa5\x03\x13\xea\x78\x9c\x7d\x76\x96\xa9\x3c\x72\x05\x92\x0e\xb0\xcd\x66\xcf\xce\x40\x6f\x12\xfd\x03\xfa\x2f\x6f\xd0\x58\x1c\xef\xe9\x7f\xb4\x77\xd6\x16\x61\xed\xa0\x13\xea\xc2\xae\xaf\x59\xca\x34\x37\x35\x9f\x30\x03\x29\x17\x5e\x95\xfb\x73\x2c\x22\x78\x16\xb5\x35\xb3\xec\xcf\x0c\x5c\x1c\x01\x85\x31\x7e\xcf\x73\x99\xd9\x0e\x7e\x23\x48\x3d\xf1\x47\xc6\x75\x14\x26\xa3\x65\x08\x4b\x12\x9f\x38\x6c\x07\xe2\x7c\x92\x61\x5d\xe2\xd9\x82\x9c\x64\xe4\xe7\x58\x73\x96\x1a\x6b\x99\x2e\x14\x6c\xf1\x99\xe4\xea\xd2\  
(\_\_\_\_\_\_\_\_\_\_\_\_Тут пропущено много подобных символов\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) \x81\x98\x12\x6c\xb5\x5c\x4d\x0e\x16\x12\xc5\xc2\x68\xe7\xbc\x66\x13\xe1\x93\x7c\xbc\x11\x5b\xd4\x6c\xb7\xfd\x06\x26\xb0\x4b\x78\x6e\x77\xb8\x9f\x56\x8c\x4a\x6e\x50\xaa\x90\xf9\x05\x6f\x72\x1e\x0e\x15\x74\x90\xe1\xfe\x04\xf7\x22\x81\x47\x06\xf2\x57\x87\x9e\xa9\x34\xcc\xb5\x08\x9c\x47\xb7\xb7\x08\xb9\x75\x86\x1e\xf5\x08\x83\xb2\xa2\x88\x3e\x05\xf6\xc6\x0f\x80\x35\xf8\xf0\x90\x9a\x0f\x0d\x61\xd5\x43\xa1\x0d\xe1\x19\x81\x5a\x9d\x0d\x0e\xff\xa7\x50\x44\xbf\x9b\xa7\x5a\xcb\xf8\xa1\xbf\x4f\xa9\x60\xf3\xac\xb4\x3f\x37\xe1\xe6\x4b\x13\x1a\xf8\xfc\x18\x3c\x63\x7b\x0c\x0f\xee\x64\xf6\xae\x53\x12\x9c\x65\xd9\xf6\x4f\x02\x4c\xf9\xb1\x0d\x65\x8a\xbb\x97\x7a\x14\x74\x07\x7f\xea\x31\xd2\x47\x01\x7e\x57\x26\xd0\xe4\xd5\x73\xfd\x76\xd9\xdb\x0a\x16\x7a\x90\xdc\x69\x91\xae\x53\xf7\x0b\x80\x68\x34\x8f\xa1\xd4\xfb\x89\x14\x2f\xc4\x05\x87\x57\x20\xa4\x7a\xc3\x9b\xac\x1e\x32\x96\x95\xd6\x37\x40\x2e\xac\x82\x30\x48\x21\xc1\xf6\x27\x10\x1a\x5d\xfe\xd9\x36\x86\x7d\x2b\x0b\x37\xf9\xcf\x51\xa2\x67\x10\x31\xea\xfb\xd2\xe5\x5a\xf4\x5a\x64\x19\x7d\x53\x5e\x5f\x4a\xd4\x70\xff\x2f\xc2\xbb\xc9\x85\x9b\x76\xba\x5f\xdf\xda\x9e\x8d\x59\x04\x1a\x01\x5a\xaf\x3f\xee\x88\x16\x57\xcc\x63\x2e\x17\x3a\x5f\x12\x92\x1b\xdc\x8f\x89\x1f\xfd\xa7\xcd\xb8\x8d\xbf\xd9\x65\x51\x8e\x70\x03\x5c\x83\x3c\xc4\x0d\xed\x6a\x87\x52\xa6\x44\x8d\xe0\x56\xbc\x52\xf3\xfc\xf3\x4e\x80\x1c\xa1\xdc\xfd\x47\x9a\x84\xa6\x31\x24\x72\x72\x65\x60\xec\xb8\x9e\x7f\x26\x26\xa9\x96\xbd\x28\xd4\x07\x66\x75\xfd\xe1\x2e\x92\x8e\xd0\x4d\x9a\xb1\xc2\x06\x24\xb8\xce\x21\x29\x9b\xc0\xa7\xc8\xca\x96\x60\x00\xae\xab\xb6\x49\x7e\xbc\xb8\x73\x2a\xd2\xaf\xe4\x89\xbb\x9d\x8f\xfc\xcb\x67\x21\xa7\x0a\x0e\xb5\x72\x92\xe3\xde\x19\xb1\x3e\x81\x5f\x60\x4b\x8c\x6f\x47\xe0\x42\x5d\x33\xed\x21', 2)

Рисунок 4.4. Сильно обфусцированный код

# Вывод

В ходе написания лабораторной работы были изучены методы обфускации кода а также применены на практике. Были протестированы три метода обфускации в различных комбинациях, которые могут защитить продукт в серьёзной промышленной разработке.

# Исходный код программы

import time  
  
  
def print\_package(package):  
 try:  
 time.sleep(1.5)  
 except KeyboardInterrupt:  
 raise BaseException("Interrupted by user")  
  
 print("Next package. {}".format(package))  
  
  
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 # let it be 576 to not fragment package  
 self.id = None  
 self.flags = None  
 self.fragment\_offset = None  
 self.ttl = 15 # let it be max  
 self.protocol = 6 # tcp code  
 self.checksum = None # ignore in ip  
 self.source\_ip = source\_ip  
 self.destination\_ip = destination\_ip  
 self.payload = payload  
  
  
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 # there is no additional options  
 self.ns = None  
 self.cwr = None  
 self.ece = None  
 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 'Source {}:{}, Destination {}:{}, Seq: {}, Ack: {} Payload: "{}"'.format(  
 self.ip.source\_ip, self.source\_port, self.ip.destination\_ip, self.destination\_port,  
 self.sequence, self.acknowledgment, self.ip.payload)  
  
  
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')  
  
  
class Member:  
  
 def \_\_init\_\_(self, ip\_address, tcp\_port):  
 self.ip\_address = ip\_address  
 self.tcp\_port = tcp\_port  
 self.caller = False  
  
 def callAnyOther(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  
  
  
class HackerMember(Member):  
  
 def \_\_init\_\_(self, ip\_address, tcp\_port):  
 self.ip\_address = ip\_address  
 self.tcp\_port = tcp\_port  
 self.caller = False  
  
 def receive(self, package):  
 answer = self.\_build\_answer(package, "Hacker server payload")  
 return answer  
  
  
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 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 = Member(123, 1)  
 server1 = Member(321, 3)  
 server2 = HackerMember(231, 2)  
  
 tcpResetMiddleware = TcpResetMiddleware()  
 fakeIpAddressMiddleware = FakeIpAddressMiddleware(231, 2)  
 rstMiddleware = RSTMiddleware()  
 connectionHijack = ConnectionHijack()  
 connection = Connection([client, server1, server2], [connectionHijack])  
  
 client.callAnyOther(connection)  
  
  
run\_attacks()