

МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

**ОБНАРУЖЕНИЕ СЕТЕВОГО RDP ТРАФИКА МЕТОДОМ АНАЛИЗА  
ЕГО ПОВЕДЕНИЯ**

**КУРСОВАЯ РАБОТА**

студента 3 курса 331 группы  
направления 10.05.01 — Компьютерная безопасность  
факультета КНиИТ  
Токарева Никиты Сергеевича

Научный руководитель  
доцент

\_\_\_\_\_

Гортинский А. В.

Заведующий кафедрой

\_\_\_\_\_

Абросимов М. Б.

Саратов 2022

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	3
1 Определение RDP .....	4
1.1 Безопасность протокола RDP.....	4
2 Принцип работы протокола RDP и анализ его поведения .....	5
3 Обнаружение сеанса удаленного управления с помощью программы.....	9
3.1 Описание функций программы .....	9
4 Демонстрация работы программы .....	16
ЗАКЛЮЧЕНИЕ .....	21
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	22
Приложение А Код sniffer.py .....	24
Приложение Б Код data-analysis.py .....	28

## **ВВЕДЕНИЕ**

Информация – это сведения об окружающем мире и протекающих в нём процессах, которые зафиксированы на каком-либо носителе. Благодаря протоколам удаленного доступа можно распоряжаться базами данных, информацией, которая хранится на другом устройстве. В недавнем прошлом большинство схем удаленного доступа характеризовалось высокой стоимостью, низкой производительностью, небольшой скоростью передачи данных, недостаточным уровнем защищенности передаваемой информации [1].

Сейчас, когда практически все предприятия перешли на дистанционный формат работы, компании выбирают протокол RDP, так как он прост в настройке и в использовании. Но далеко не все уделяют особое внимание безопасности собственных рабочих мест. Поэтому предприятия могут быть атакованы злоумышленниками.

В данной работе будут разобраны принцип работы RDP, анализ его поведения, а также методы обнаружения данного протокола.

## **1 Определение RDP**

Протокол RDP (от англ. Remote Desktop Protocol — протокол удалённого рабочего стола) — патентованный протокол прикладного уровня компании Microsoft и приобретен ею у другой компании Polycom, который предоставляет пользователю графический интерфейс для подключения к другому компьютеру через сетевое соединение. Для этого пользователь запускает клиентское программное обеспечение RDP, а на другом компьютере должно быть запущено программное обеспечение сервера RDP [2].

Клиенты для подключения по RDP существуют для большинства версий Microsoft Windows, Linux, Unix, macOS, iOS, Android и других операционных систем. Стоит отметить, что RDP-серверы встроены в операционные системы Windows. По умолчанию подключения, созданные с помощью RDP, используют UDP-порт 3389 и TCP порт 3389, по которым осуществляется передача данных.

### **1.1 Безопасность протокола RDP**

Как уже известно, что для операционной системы Windows постоянно выходят различные обновления, включая обновлений RDS (от англ. Remote Desktop Services — службы удаленных рабочих столов). В связи с этим возникают различные уязвимости при инициализации RDP-сессии. В основном они не связаны непосредственно с протоколом RDP, но касаются службы удаленных рабочих столов RDS и позволяют при успешной эксплуатации путем отправления специального запроса через RDP получить возможность выполнения произвольного кода на уязвимой системе, даже не проходя при этом процедуру проверки подлинности. Достаточно лишь иметь доступ к хосту или серверу с уязвимой системой Windows. Таким образом, любая система, доступная из сети Интернет, является уязвимой при отсутствии установленных последних обновлений безопасности Windows.

Если стоит задача защитить удаленный доступ, то, конечно, необходимо использовать надежный пароль, обновить свое программное обеспечение до последней версии, также можно использовать VPN подключение, чтобы получить IP-адрес виртуальной сети и добавить его в правило исключения брандмауэра RDP. Стоит отметить, что существует много разных способов, чтобы защитить подключение с помощью протокола RDP и более подробно это описано в документации Microsoft.

## 2 Принцип работы протокола RDP и анализ его поведения

Принцип работы RDP базируется на протоколе TCP. Соединение клиент-сервер происходит на транспортном уровне. После инициализации пользователь проходит аутентификацию. В случае успешного подтверждения сервер передает клиенту управление. Стоит отметить, что под понятием слова «клиент» подразумевается любое устройство (персональный компьютер, планшет или смартфон), а «сервер» — удаленный компьютер, к которому оно подключается.

Протокол RDP внутри себя поддерживает виртуальные каналы, через которые пользователю передаются дополнительные функции операционной системы, например, можно распечатать документ, воспроизвести видео или скопировать файл в буфер обмена.

Далее в работе будет описан процесс установки RDP-сессии, во время которой осуществляется захват трафика с помощью одной известной программы Wireshark. С помощью нее можно достаточно подробно рассмотреть структуру сообщений протоколов.

Для начала будет произведено подключение с помощью «Удаленного рабочего стола». Это средство представляет собой встроенную в Windows программу, предназначенную для удалённого доступа. При его использовании предполагается, что пользователь будет подключаться к одному компьютеру с другого устройства, находящегося в той же локальной сети. В качестве клиента и сервера будут выступать компьютеры с операционной системой Windows 10 Professional версии 21H2.

Для подключения к удаленному рабочему столу были заданы статические IP-адреса. Клиенту был присвоен статический IP-адрес 192.168.10.254, а серверу — 192.168.10.229, соответственно маска сети 255.255.255.0. После того, как были заданы IP-адреса, необходимо зайти в настройки Windows, чтобы включить возможность подключения к удаленному рабочему столу. Об этом более подробно описано в статьях [3] и [4]. Далее на сервере был произведен запуск анализа трафика с помощью приложения Wireshark. После подключения к удаленному компьютеру программа-анализатор трафика начала «захватывать» пакеты, как показано на рисунке 1, принадлежащие следующим протоколам:

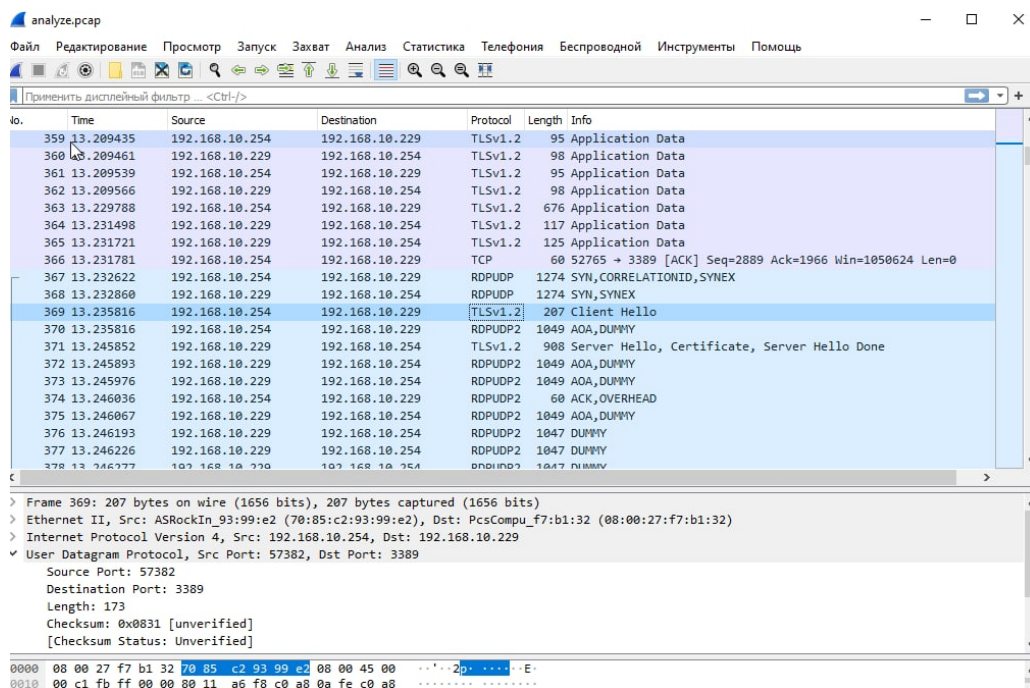


Рисунок 1 – Окно программы Wireshark после захвата трафика

- RDPUDP — протокол RDP, использующий для передачи данных UDP-протокол.
- RDPUDP2 также относится к протоколу RDP. Он был разработан для повышения производительности сетевого соединения по сравнению с соответствующим соединением RDP-UDP [8].
- TLSv1.2 — протокол защиты транспортного уровня, обеспечивающий защищенную передачу между узлами в сети интернет. В данном случае обеспечивает безопасность RDP-сессии.

Во время работы программы Wireshark было найдено достаточное количество пакетов, принадлежащих RDP, которые содержат в себе достаточно интересную информацию. Поэтому стоит рассказать о том, как происходит стандартный способ защиты RDP. Это можно представить в несколько этапов:

1. Клиент объявляет серверу о своем намерении использовать стандартный протокол RDP.
2. Сервер соглашается с этим и отправляет клиенту свой собственный открытый ключ, полученный при шифровании алгоритмом RSA, а также некоторую строку случайных байтов (обычно её называют «random сервером»), генерируемую сервером. На рисунке 2 можно увидеть запись random сервера.

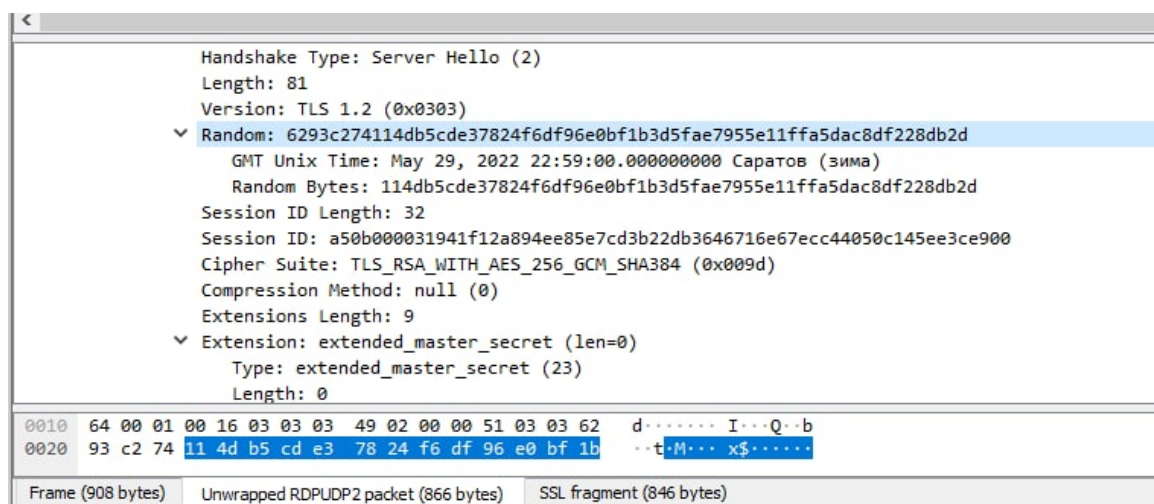


Рисунок 2 – Содержимое пакета, посылаемого от сервера клиенту (запись random сервера)

Совокупность открытого ключа и некоторая строка случайных байтов называется «сертификатом». Данная запись изображена на рисунке 3.

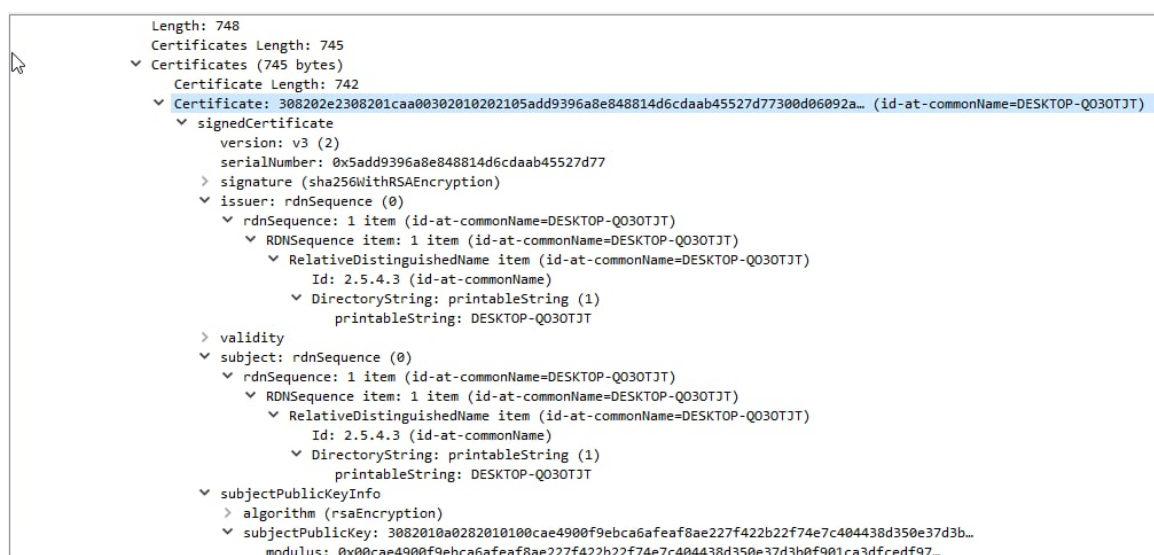


Рисунок 3 – Содержимое пакета, посылаемого от сервера клиенту (запись сертификата)

Сертификат подписывается службой терминалов, например, RDS, с использованием закрытого ключа для обеспечения подлинности.

3. Теперь клиент посылает некоторую строку случайных байтов, которая называется «premaster secret», показанная на рисунке 4.

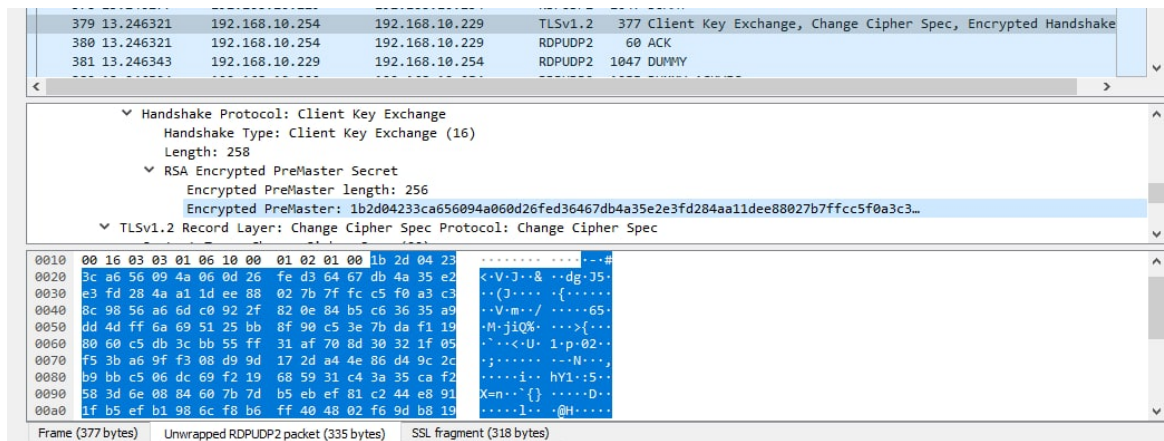


Рисунок 4 – Содержание пакета, посылаемого от клиента серверу (запись premaster secret)

Данная запись шифруется открытым ключом, которая может быть расшифрована сервером только с помощью закрытого ключа службы терминалов.

4. Сервер расшифровывает premaster secret с помощью собственного закрытого ключа.
5. В случае успеха клиент и сервер получают свои сеансовые ключи из random сервера и premaster secret. Далее они используются для симметричного шифрования остальной части сеанса.

Осталось только понять, как обнаружить подключение к удаленному рабочему столу.



### **3 Обнаружение сеанса удаленного управления с помощью программы**

Одним из методов выявления сообщений, передаваемых по сети, является сниффер — это программное обеспечение, которое анализирует входящий и исходящий трафик с компьютера, подключенного к интернету. Для данной работы была написана на языке Python программа, перехватывающая трафик сети.

Данный сниффер принимает пакеты четвертой версии интернет-протокола, пакеты IPv4, содержащие в поле данных сообщения протоколов других уровней. В этом случае здесь будут рассматриваться сообщения протоколов транспортного уровня, а именно TCP- и UDP-протоколы.

Для успешного перехвата трафика, необходимо установить неразборчивый режим на сетевой интерфейс, чтобы сетевая плата принимала все пакеты независимо от того, кому они адресованы. Данный выбор зависит от способа подключения устройства к сети. Например, в Linux есть виртуальный интерфейс («lo»), который ваш компьютер использует для связи с самим собой, также существуют интерфейсы относящиеся к проводному соединению («enp0s3») и беспроводному («wlp2s0»). В данном случае все устройства будут подключены к сети через Ethernet.

После выбора сетевого интерфейса сниффер начнет перехватывать трафик, как показано. В этот момент информация о перехваченных пакетах будет отображаться в консоли, а также будет записываться необходимая информация каждого пакета для анализа в файл data.log.

#### **3.1 Описание функций программы**

Для анализа трафика в сети был создан сокет — программный интерфейс для обеспечения обмена данными между процессами. В силу заданных параметров он получал пакеты, представленные в виде некоторой последовательности чисел, записанных в шестнадцатеричной системе счисления.

```
server = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
```

Чтобы раскодировать данную последовательность чисел, необходимо обратиться к структуре пакета. Для начала нужно рассмотреть кадр Ethernet, представленный на рисунке 5.



Рисунок 5 – Структура Ethernet кадра

Стоит отметить, что в данном заголовке нужно раскодировать 14 байт, 12 из которых MAC-адреса получателя и отправителя и 2 байта, идентифицирующие протокол сетевого уровня. К примеру 0x0800 – Ipv4, 0x86DD – IPv6 и т.д. С помощью функций `get_ethernet_frame()` и `get_mac_addr()` производится получение всех необходимых данных заголовка Ethernet.

```
# Получение ethernet-кадра
def get_ethernet_frame(data):
    dest_mac, src_mac, proto = struct.unpack('!6s6sH', data[:14])
    return get_mac_addr(dest_mac), get_mac_addr(src_mac), socket.htons(proto)

# Получение MAC-адреса
def get_mac_addr(mac_bytes):
    mac_str = ''
    for el in mac_bytes:
        mac_str += format(el, '02x').upper() + ':'
    return mac_str[:len(mac_str) - 1]
```

После получения информации об Ethernet кадре идет раскодирование интернет-протокола. В данной работе будут рассматриваться пакеты протокола IPv4, так как для обнаружения RDP-сессии этого вполне достаточно. Поэтому необходимо рассмотреть IPv4-заголовок, который показан на рисунке 6.

4 бита Номер версии	4 бита Длина заголовка	8 бит Тип сервиса				16 бит Общая длина																
		PR	D	T	R																	
16 бит Идентификатор пакета								3 бита Флаги		13 бит Смещение фрагмента												
									D													
8 бит Время жизни		8 бит Протокол верхнего уровня				16 бит Контрольная сумма																
32 бита IP-адрес источника																						
32 бита IP-адрес назначения																						
Параметры и выравнивание																						

Рисунок 6 – Структура IPv4-заголовка

Обычно длина заголовка IP равна 20 байт, т.е. пять 32-битных слов, однако при увеличении объема служебной информации эта длина может быть увеличена за счет использования дополнительных байт в поле параметров и выравниваний. Благодаря полю, где содержится длина заголовка, можно правильно раскодировать оставшуюся последовательность байт. С помощью функций `get_ipv4_data()` и `ipv4_dec()` будут получена информация о времени жизни текущего пакета, о номере транспортного протокола, об IP-адресах отправителя и получателя.

```
# Получение IPv4-заголовка
def get_ipv4_data(data):
    version_header_length = data[0]
    header_length = (version_header_length & 15) * 4
    ttl, proto, src, dest = struct.unpack('!8xBB2x4s4s', data[:20])
    return ttl, proto, ipv4_dec(src), ipv4_dec(dest), data[header_length:]

# Получение IP-адреса формата X.X.X.X
def ipv4_dec(ip_bytes):
    ip_str = ''
    for el in ip_bytes:
        ip_str += str(el) + '.'
    return ip_str[:-1]
```

После того, как был получен номер транспортного протокола, можно раскодировать их данные. Как уже упоминалось ранее, в качестве транспортных

протоколов будут рассматриваться TCP и UDP протоколы. На рисунке 7 изображена структура tcp-заголовка.



Рисунок 7 – Структура TCP-заголовка

С помощью функции `get_tcp_segment()` производится получение информации, содержащейся в TCP-протоколе. Получается, теперь программе известны порт получателя, порт отправителя, порядковый номер, номер подтверждения и флаги. Однако самая ценная информация для данной работы — это порты и данные, которые содержатся в TCP- и UDP-заголовках.

```
# Получение TCP-сегмента данных
def get_tcp_segment(data):
    src_port, dest_port, sequence, ack, some_block = struct.unpack('!HHLLH', data[:14])
    return src_port, dest_port, sequence, ack, data[(some_block >> 12) * 4:]
```

Аналогично получается раскодирование данных UDP-заголовка с помощью функции `get_udp_segment()`.

```
# Получение UDP-сегмента данных
def get_udp_segment(data):
    src_port, dest_port, size = struct.unpack('!HH2xH', data[:8])
    return src_port, dest_port, size, data[8:]
```

Далее необходимо рассмотреть данные, полученные после обработки TCP- и UDP-заголовков. В функции `scan_port()` проверяется порт очередного пакета.

Если порт совпадает с портом, заданным в начале программы (по умолчанию он равен 3389), то значит осуществляется попытка подключения к удаленному рабочему столу устройства, находящегося в текущей локальной сети. Также здесь осуществляется проверка известных IP-адресов, взятых из файла white-list.log.

```
# Проверка порта по-умолчанию
def scan_port(src_ipv4, dest_ipv4, src_mac, src_port, dest_port):
    if dest_port == def_port:
        fl = False
        for key in white_list.keys():
            if key[1] == src_ipv4:
                fl = True
                break
        if not fl:
            for key in white_list.keys():
                if key[1] == dest_ipv4:
                    tup = (src_ipv4, src_mac)
                    if tup not in black_list:
                        black_list.append(tup)
                        write_to_file((src_ipv4, src_mac, key[0], src_port, dest_port), False)
```

В функции scan\_inf() производится анализ данных сегмента TCP, а format\_data() — функция для корректного представления полученных данных.

```
# Проверка данных TCP-сегмента
def scan_inf(r_data, src_ipv4, dest_ipv4, src_mac, dest_mac, dest_port, src_port):
    global Current_object
    global black_list
    global Packet_cnt
    data = format_data(r_data)
    flag = False
    for key in white_list.keys():
        if key[1] == src_ipv4:
            flag = True
            break
    if not flag:
        for key, value in white_list.items():
            if value[1] in data and key[1] == dest_ipv4:
                Current_object = (key[0], key[1], value[0])
                tup = (src_ipv4, src_mac)
                if tup not in black_list:
                    black_list.append(tup)
                    write_to_file(( src_ipv4, src_mac, Current_object[0]
                                , src_port, dest_port ), False)
    if Current_object:
        if Current_object[2] in data:
            for key in white_list.keys():
```

```

        if key[1] == src_ipv4:
            write_to_file(( dest_ipv4, dest_mac, Current_object[0]
                           , src_port, dest_port ), True)

            break

    Current_object = ''
else:
    Packet_cnt += 1
    if Packet_cnt > 100:
        Packet_cnt = 0
        Current_object = ''

```

После вызова функции `get_tcp_segment()` могут быть получены данные, которые уже относятся к прикладному уровню.

Практически все выше перечисленные функции содержатся в `start_to_listen()`, где производится вывод в консоль информации о получаемых пакетах, делаются вызовы функций для декодирования последовательности чисел.

*# Перехват трафика и вывод информации в консоль*

```
def start_to_listen(interface):
```

```
    global Current_object
```

```
    global Cur_number
```

```
    os.system(f'ip link set {socket.if_indextoname(interface)} promisc on')
```

```
    server = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
```

```
    # server.bind((socket.if_indextoname(interface), 0))
```

```
    while True:
```

```
        # Получение пакетов в виде набора hex-чисел
```

```
        raw_data, _ = server.recvfrom(65565)
```

```
        dest_mac, src_mac, protocol = get_ethernet_frame(raw_data)
```

```
        # Если это интернет-протокол четвертой версии
```

```
        if protocol == 8:
```

```
            print(f'-----Пакет N{Cur_number}-----')
```

```
            Cur_number += 1
```

```
            print('Ethernet кадр: ')
```

```
            print('MAC-адрес отправителя: ' + str(src_mac), 'MAC-адрес получателя: ' + str(dest_mac))
```

```
            ttl, proto, src_ipv4, dest_ipv4, data_ipv4 = get_ipv4_data(raw_data[14:])
```

```
            print('IPv4 заголовок:')
```

```
            print( 'TTL: ' + str(ttl)
```

```
                , 'Номер протокола: ' + str(proto)
```

```
                , 'IP-адрес отправителя: ' + str(src_ipv4)
```

```
                , 'IP-адрес получателя: ' + str(dest_ipv4))
```

```
        # Если это UDP-протокол
```

```
        if proto == 17:
```

```
            src_port_udp, dest_port_udp, size, data_udp = get_udp_segment(data_ipv4)
```

```
            print('UDP заголовок:')
```

```
            print( 'Порт отправителя: ' + str(src_port_udp), 'Порт получателя: ' +
```

```

        str(dest_port_udp), 'Размер: ' + str(size) )

    scan_port(src_ipv4, dest_ipv4, src_mac, src_port_udp, dest_port_udp)
    # Если это TCP-протокол
    if proto == 6:
        src_port_tcp, dest_port_tcp, sequence, ack, data_tcp = get_tcp_segment(data_ipv4)
        print('TCP заголовок:')
        print( 'Порт отправителя: ' + str(src_port_tcp)
              , 'Порт получателя: ' + str(dest_port_tcp)
              , 'Порядковый номер: ' + str(sequence)
              , 'Номер подтверждения: ' + str(ack) )

    scan_port(src_ipv4, dest_ipv4, src_mac, src_port_tcp, dest_port_tcp)

    th_inf = threading.Thread(target=scan_inf, args=[ data_tcp
                                                    , src_ipv4
                                                    , dest_ipv4
                                                    , src_mac
                                                    , dest_mac
                                                    , dest_port_tcp
                                                    , src_port_tcp ])

    th_inf.start()
    keyboard.add_hotkey('Space', wait_key)

```

После описания всех главных функций можно перейти к проверке корректности работы программы.

#### 4 Демонстрация работы программы

Допустим к локальной сети подключено несколько устройств. Для тестирования данной программы было запущено четыре виртуальных машины:

- компьютер №1 с операционной системой Windows 10 Professional версии 21H2 IP-адресом 192.168.10.229
- компьютер №2 с операционной системой Windows 10 Professional версии 21H2 IP-адресом 192.168.10.254
- компьютер №3 с операционной системой Windows 10 Professional версии 21H2 IP-адресом 192.168.10.21
- компьютер №4 с операционной системой Linux Ubuntu версии 22.04 LTS IP-адресом 192.168.10.107

В ходе работы был создан файл white-list.log в который записаны имена компьютеров и их IP-адреса. Содержимое файла показано на рисунке 8.

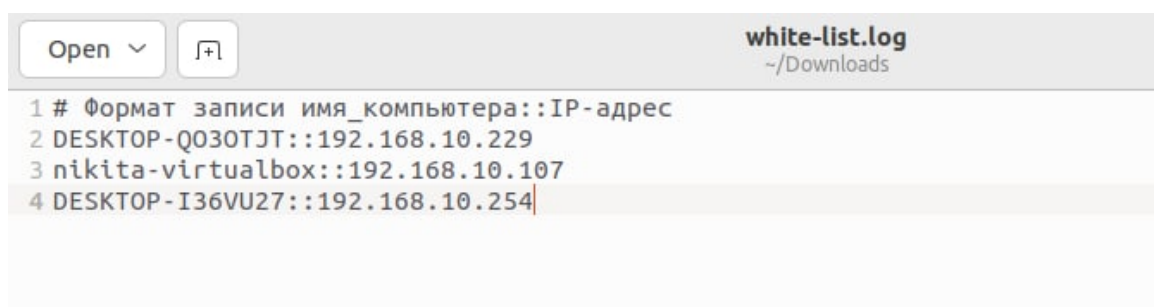


Рисунок 8 – Содержимое файла white-list.log

Стоит отметить, что информация о компьютере №3 не добавлена специально, так как его будут считать неизвестным устройством, остальные, информация о которых записана в файл, являются известными соответственно.

Теперь необходимо провести несколько тестирований данной программы:

1. Пусть компьютер №3 выполнит подключение к компьютеру №1, где в это время на компьютере №4 будет уже работать программа rdp-sniffer.py. Из рисунка 9 видно, как программа запрашивает порт по умолчанию, а затем предоставляет выбор сетевого интерфейса.



```
nikita@nikita-VirtualBox: ~/Downloads
nikita@nikita-VirtualBox:~$ cd Downloads/
nikita@nikita-VirtualBox:~/Downloads$ sudo python3 rdp-sniffer.py
[sudo] password for nikita:

Запуск программы...

Хотите поменять RDP порт для анализа трафика? (по умолчанию 3389)
Если да, то нажmite 1, иначе - 0

Выберите сетевой интерфейс, нажав соответствующую цифру:
[(1, 'lo'), (2, 'enp0s3')]
2
```

Рисунок 9 – Вид консоли при запуске программы

После осуществления подключения к удаленному рабочему столу в консоли начали появляться записи о пакетах, которые относятся к протоколу RDP, как показано на рисунке 10.

```
nikita@nikita-VirtualBox: ~/Downloads

-----Пакет N482-----
Ethernet кадр:
MAC-адрес отправителя: 08:00:27:D8:18:8D MAC-адрес получателя: 08:00:27:F7:B1:32
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.21 IP-адрес получателя: 192.168.10.229
TCP заголовок:
Порт отправителя: 61088 Порт получателя: 3389 Порядковый номер: 3471883107 Номер подтверждения: 0
-----Пакет N483-----
Ethernet кадр:
MAC-адрес отправителя: 08:00:27:F7:B1:32 MAC-адрес получателя: 08:00:27:D8:18:8D
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.229 IP-адрес получателя: 192.168.10.21
TCP заголовок:
Порт отправителя: 3389 Порт получателя: 61088 Порядковый номер: 460425162 Номер подтверждения: 3471883108
-----Пакет N484-----
Ethernet кадр:
MAC-адрес отправителя: 08:00:27:D8:18:8D MAC-адрес получателя: 08:00:27:F7:B1:32
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.21 IP-адрес получателя: 192.168.10.229
TCP заголовок:
Порт отправителя: 61088 Порт получателя: 3389 Порядковый номер: 3471883108 Номер подтверждения: 460425163
-----Пакет N485-----
Ethernet кадр:
MAC-адрес отправителя: 08:00:27:D8:18:8D MAC-адрес получателя: 08:00:27:F7:B1:32
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.21 IP-адрес получателя: 192.168.10.229
TCP заголовок:
Порт отправителя: 61088 Порт получателя: 3389 Порядковый номер: 3471883108 Номер подтверждения: 460425163
```

Рисунок 10 – Вид консоли при работе программы

Видно, что пакеты доставляются через TCP-порт 3389 протокола RDP.

Теперь осталось проверить появились ли какие-нибудь записи в файле information.log.

На рисунке 11 изображены 3 записи, одна из которых была сделана из-за обнаружения 3389-го порта, а две записи — при раскодировании данных протокола TLS. При анализе трафика с помощью программы Wireshark также наблюдались повторяющиеся пакеты протокола TLS, в которых содержалась практически одна и та же информация. Скорее всего это связано с тем, что устройства в процессе обмена данными «договорились» между собой, например, об изменении шифра или порта. Поэтому такого рода пакеты приходится отправлять повторно. Так как программа отслеживает все такие пакеты, то в файл было сделано две записи в разные моменты времени.

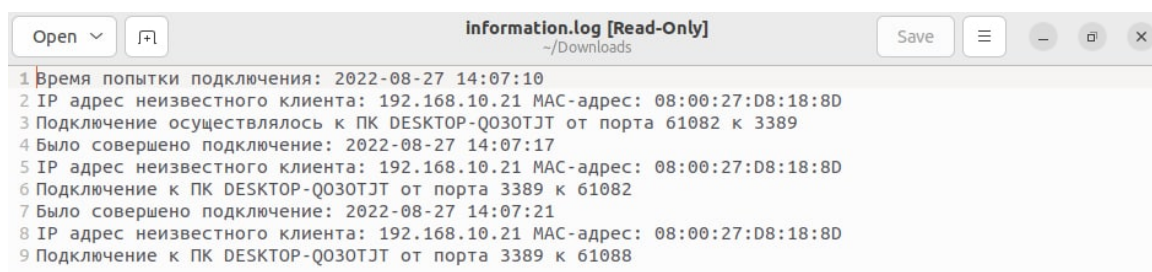


Рисунок 11 – Содержимое файла information.log после подключения по порту 3389

Таким образом благодаря файлу information.log можно узнать время установки RDP-сессии, IP-адрес и MAC-адрес неизвестного клиента

2. Допустим будет совершено подключение по другому RDP порту или в программе ввести совсем другой порт, по которому будет проводится анализ пакетов. Пусть Компьютер №3 будет подключаться к компьютеру №1 по порту, например 13389, как показано на рисунке 12.

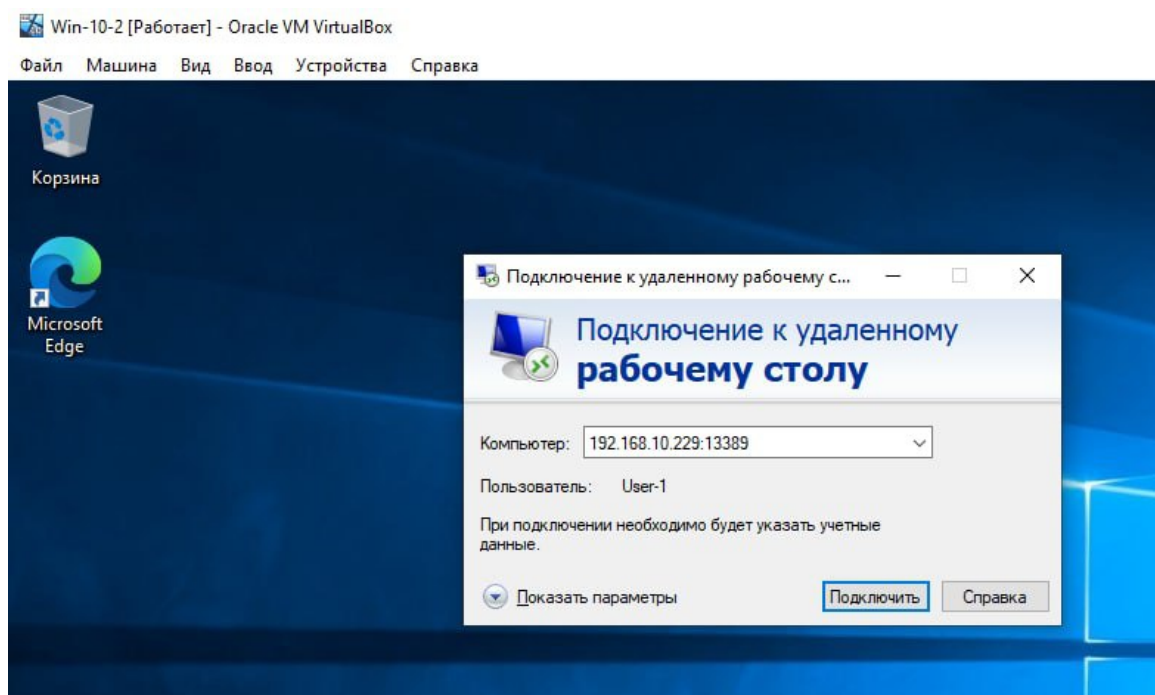


Рисунок 12 – Подключение к компьютеру №1 по другому порту

Тогда при подключении к удаленному рабочему столу в файле information.log появятся следующие записи, которые изображены на рисунке 13.

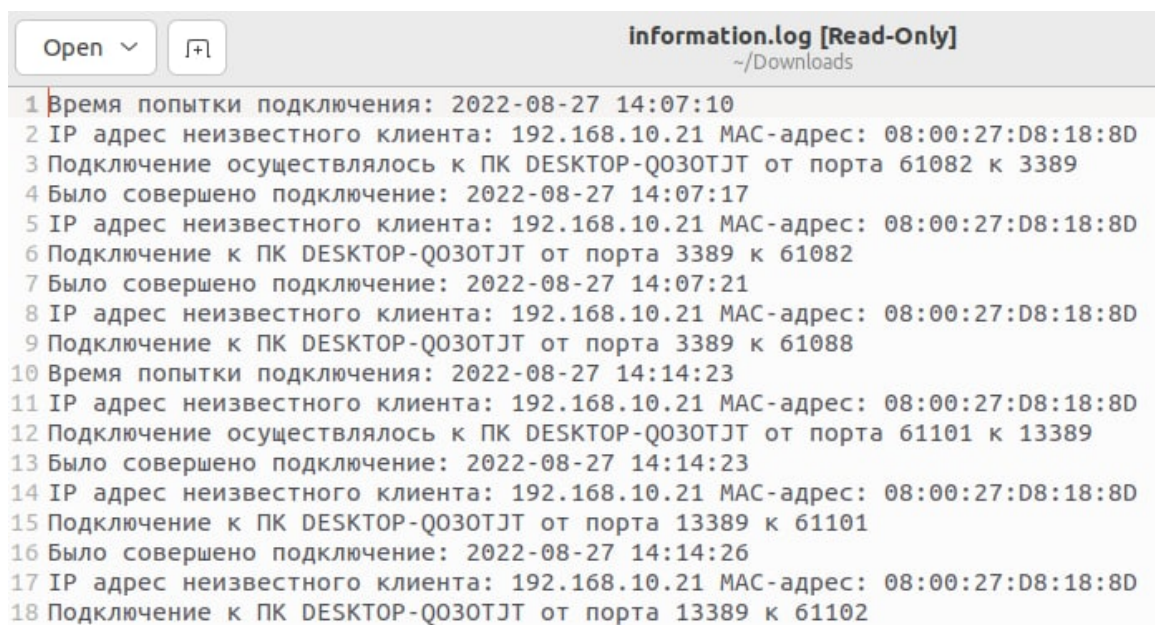


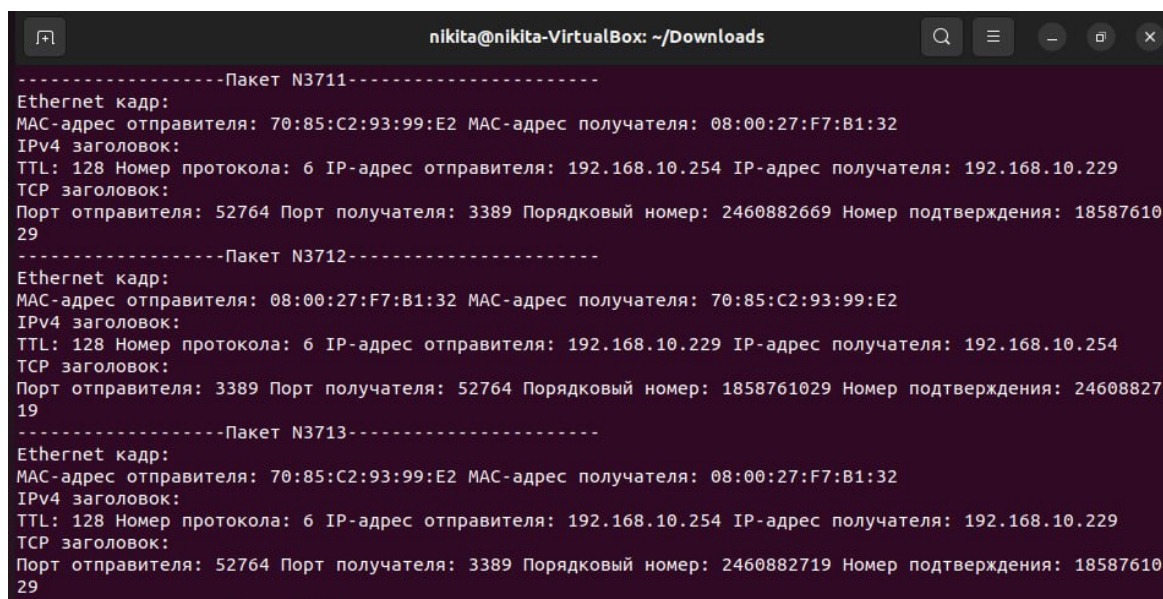
Рисунок 13 – Содержимое файла information.log после подключения по порту 13389

Получается программа установила то, что RDP-сессия произошла, опираясь на данные, которые содержатся в сообщениях протокола TLS.

3. Теперь осталось проверить подключение по протоколу RDP компьютера



№2 к компьютеру №1. При установке RDP-сессии программа отслеживает пакеты, как показано на рисунке 14.



```
nikita@nikita-VirtualBox: ~/Downloads
-----Пакет N3711-----
Ethernet кадр:
MAC-адрес отправителя: 70:85:C2:93:99:E2 MAC-адрес получателя: 08:00:27:F7:B1:32
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.254 IP-адрес получателя: 192.168.10.229
TCP заголовок:
Порт отправителя: 52764 Порт получателя: 3389 Порядковый номер: 2460882669 Номер подтверждения: 1858761029
-----Пакет N3712-----
Ethernet кадр:
MAC-адрес отправителя: 08:00:27:F7:B1:32 MAC-адрес получателя: 70:85:C2:93:99:E2
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.229 IP-адрес получателя: 192.168.10.254
TCP заголовок:
Порт отправителя: 3389 Порт получателя: 52764 Порядковый номер: 1858761029 Номер подтверждения: 2460882719
-----Пакет N3713-----
Ethernet кадр:
MAC-адрес отправителя: 70:85:C2:93:99:E2 MAC-адрес получателя: 08:00:27:F7:B1:32
IPv4 заголовок:
TTL: 128 Номер протокола: 6 IP-адрес отправителя: 192.168.10.254 IP-адрес получателя: 192.168.10.229
TCP заголовок:
Порт отправителя: 52764 Порт получателя: 3389 Порядковый номер: 2460882719 Номер подтверждения: 1858761029
```

Рисунок 14 – Вид консоли при подключении к удаленному рабочему столу

Стоит отметить, что в файл `information.log` записи не были сделаны. Компьютер №2 считается верифицированным устройством, так как информация о нем содержится в файле `white-list.log`. Это сделано для того чтобы можно было точно определять несанкционированные подключения к удаленному рабочему столу.

## **ЗАКЛЮЧЕНИЕ**

В результате проделанной работы были разобраны методы обнаружения подключения к удаленному рабочему столу по протоколу RDP, где с помощью различных программ удалось рассмотреть принцип работы RDP-протокола. Стоит отметить, что RDP далеко не самый защищенный протокол. Хотя корпорация Microsoft регулярно выпускает обновления для своего программного обеспечения. Однако RDP-сессия становится уязвимой из-за упущений в безопасности, например из-за некорректной конфигурации сервисов или установки устаревших обновлений системы. В таком случае злоумышленник может использовать такие просчеты в своих целях. А людям, ответственным за безопасность компьютерной сети, остается только придумывать новые методы обнаружения и предотвращения несанкционированных подключений к удаленному рабочему столу.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Книга Ибе О.С. «Компьютерные сети и службы удаленного доступа» / пер. с англ. - Москва, издательство: «ДМК Пресс», Яз. рус.
- 2 Удалённый рабочий стол RDP: как включить и как подключиться по RDP [Электронный ресурс] / URL: <https://hackware.ru/?p=11835> (дата обращения 03.05.2022), Яз. рус.
- 3 How to use remote desktop [Электронный ресурс] / URL: <https://support.microsoft.com/en-us/windows/how-to-use-remote-desktop-5fe128d5-8fb1-7a23-3b8a-41e636865e8c> (дата обращения 27.05.2022), Яз. англ.
- 4 Статья «Как исправить ошибку удаленного рабочего стола не удастся подключиться к удаленному компьютеру» [Электронный ресурс] / URL: <https://okdk.ru/kak-ispravit-oshibku-udalennogo-rabochego-stola-ne-udaetsya-podkljuchitsya-k-udalennomu-kompjuteru/> (дата обращения 27.05.2022), Яз. рус.
- 5 Документация Remote Utilities «RDP» [Электронный ресурс] / URL: <https://www.remoteutilities.com/support/docs/rdp/> (дата обращения 27.05.2022), Яз. англ.
- 6 Документация по стандартным библиотекам языка Python [Электронный ресурс] / URL: <https://docs.python.org/3/library/socket.html> (дата обращения 25.06.2022), Яз. англ.
- 7 Статья «Интерактивная система просмотра системных руководств (man-ов)» [Электронный ресурс] / URL: <https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=socket&category=2> (дата обращения 25.06.2022), Яз. англ.
- 8 Документация Microsoft «Протоколы» [Электронный ресурс] / URL: [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-rdpeudp2/d8bf9a56-90f3-4608-8f98-9600ed69876b](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-rdpeudp2/d8bf9a56-90f3-4608-8f98-9600ed69876b) (дата обращения 28.05.2022), Яз. рус.
- 9 Статья «Wireshark Tutorial: Decrypting RDP Traffic» [Электронный ресурс] / URL: <https://unit42-paloaltonetworks-com.translate.goog/wireshark->

tutorial-decrypting-rdp-traffic/?\_x\_tr\_sl=en&\_x\_tr\_tl=ru&\_x\_tr\_hl=ru&\_x\_tr\_pto=op,wapp (дата обращения 28.05.2022), Яз. англ.

## ПРИЛОЖЕНИЕ А

### Код sniffer.py

```
import socket, datetime, struct
import os, time
import keyboard

# Получение ethernet-кадра
def get_ethernet_frame(data):
    dest_mac, src_mac, proto = struct.unpack('!6s6sH', data[:14])
    return get_mac_addr(dest_mac), get_mac_addr(src_mac), socket.htons(proto)

# Получение MAC-адреса
def get_mac_addr(mac_bytes):
    mac_str = ''
    for el in mac_bytes:
        mac_str += format(el, '02x').upper() + ':'
    return mac_str[:len(mac_str) - 1]

# Получение IPv4-заголовка
def get_ipv4_data(data):
    version_header_length = data[0]
    header_length = (version_header_length & 15) * 4
    ttl, proto, src, dest = struct.unpack('!8xBB2x4s4s', data[:20])
    return str(ttl), proto, ipv4_dec(src), ipv4_dec(dest), data[header_length:]

# Получение IP-адреса формата X.X.X.X
def ipv4_dec(ip_bytes):
    ip_str = ''
    for el in ip_bytes:
        ip_str += str(el) + '.'
    return ip_str[:-1]

# Получение UDP-сегмента данных
def get_udp_segment(data):
    src_port, dest_port, size = struct.unpack('!HH2xH', data[:8])
    return str(src_port), str(dest_port), str(size), data[8:]

# Получение TCP-сегмента данных
def get_tcp_segment(data):
    src_port, dest_port, sequence, ack, some_block = struct.unpack('!HLLH', data[:14])
    return str(src_port), str(dest_port), str(sequence), str(ack), \
        some_block, data[(some_block >> 12) * 4:]
```



```

# Форматирование данных для корректного представления
def format_data(data):
    if isinstance(data, bytes):
        data = ''.join(r'\x{:02x}'.format(el) for el in data)
    return data

# Перехват трафика и вывод информации в консоль
def start_to_listen(s_listen):
    NumPacket = 1
    while True:
        # Получение пакетов в виде набора hex-чисел
        raw_data, _ = s_listen.recvfrom(65565)
        arr_data = [''] * 17
        arr_data[0], arr_data[1] = str(NumPacket), str(time.time())
        arr_data[2] = str(len(raw_data))
        # Если это интернет-протокол четвертой версии
        arr_data[4], arr_data[3], protocol = get_ethernet_frame(raw_data)
        if protocol == 8:
            print(f'-----Пакет N{NumPacket}-----')
            NumPacket += 1
            print('Ethernet кадр: ')
            print( 'MAC-адрес отправителя: ' + arr_data[3]
                  , 'MAC-адрес получателя: ' + arr_data[4] )
            ttl, proto, arr_data[6], arr_data[7], data_ipv4 = get_ipv4_data(raw_data[14:])
            print('IPv4 заголовок:')
            print( 'TTL: ' + ttl
                  , 'Номер протокола: ' + str(proto)
                  , 'IP-адрес отправителя: ' + arr_data[6]
                  , 'IP-адрес получателя: ' + arr_data[7])
            # Если это UDP-протокол
            if proto == 17:
                arr_data[5] = 'UDP'
                arr_data[8], arr_data[9], length, data_udp = get_udp_segment(data_ipv4)
                print('UDP заголовок:')
                print( 'Порт отправителя: ' + arr_data[8], 'Порт получателя: ' +
                      arr_data[9], 'Длина: ' + length )
                arr_data[10], arr_data[11] = str(len(data_udp)), format_data(data_udp)
                write_to_file(arr_data)
            # Если это TCP-протокол
            if proto == 6:
                arr_data[5] = 'TCP'
                arr_data[8], arr_data[9], arr_data[10], \
                arr_data[11], flags, data_tcp = get_tcp_segment(data_ipv4)
                fl_urg = str((flags & 32) >> 5)
                fl_ack = str((flags & 16) >> 4)

```

```

fl_psh = str((flags & 8) >> 3)
fl_rst = str((flags & 4) >> 2)
fl_syn = str((flags & 2) >> 1)
fl_fin = str(flags & 1)
print('TCP заголовок:')
print( 'Порт отправителя: ' + arr_data[8]
      , 'Порт получателя: ' + arr_data[9]
      , 'Порядковый номер: ' + arr_data[10]
      , 'Номер подтверждения: ' + arr_data[11] )
print('Флаги:')
print( 'URG: ' + fl_urg, 'ACK: ' + fl_ack, 'PSH: ' + fl_psh
      , 'RST: ' + fl_rst, 'SYN: ' + fl_syn, 'FIN: ' + fl_fin )
arr_data[12], arr_data[13], arr_data[14] = fl_ack, fl_psh, fl_syn
arr_data[15], arr_data[16] = str(len(data_tcp)), format_data(data_tcp)
write_to_file(arr_data)
if keyboard.is_pressed('space'):
    s_listen.close()
    print('Завершение программы...')
    break

# Запись в файл
def write_to_file(a):
    try:
        with open('data.log', 'a') as f:
            if a[5] == 'TCP':
                f.write( 'No:' + a[0] + ';' + 'Time:' + a[1] + ';' +
                        'Pac-size:' + a[2] + ';' + 'MAC-src:' + a[3] + ';' +
                        'MAC-dest:' + a[4] + ';' + 'Type:' + a[5] + ';' +
                        'IP-src:' + a[6] + ';' + 'IP-dest:' + a[7] + ';' +
                        'Port-src:' + a[8] + ';' + 'Port-dest:' + a[9] + ';' +
                        'Seq:' + a[10] + ';' + 'Ack:' + a[11] + ';' +
                        'Fl-ack:' + a[12] + ';' + 'Fl-psh:' + a[13] + ';' +
                        'Fl-syn:' + a[14] + ';' + 'Len-data:' + a[15] + ';' +
                        'Data:' + a[16] + ';\n')
            else:
                f.write( 'No:' + a[0] + ';' + 'Time:' + a[1] + ';' +
                        'Pac-size:' + a[2] + ';' + 'MAC-src:' + a[3] + ';' +
                        'MAC-dest:' + a[4] + ';' + 'Type:' + a[5] + ';' +
                        'IP-src:' + a[6] + ';' + 'IP-dest:' + a[7] + ';' +
                        'Port-src:' + a[8] + ';' + 'Port-dest:' + a[9] + ';' +
                        'Len-data:' + a[10] + ';' + 'Data:' + a[11] + ';\n')
        f.close()
    except:
        print('Ошибка записи в файл...')
        pass

```

```
# Осуществление запуска программы
if __name__ == '__main__':
    print('\nЗапуск программы....\n')

    print('Выберите сетевой интерфейс, нажав соответствующую цифру:')
    print(socket.if_nameindex())
    interface = int(input())
    os.system(f'ip link set {socket.if_indextoname(interface)} promisc on')
    s_listen = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
    start_to_listen(s_listen)
```

## ПРИЛОЖЕНИЕ Б

### Код data-analysis.py

```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import time
from colorama import init, Back, Fore

init(autoreset=True)
FileName = ''
Packet_list = []
Object_list = []
Labels_list = []
x_axisLabels = []

# Класс, содержащий информацию о каком-либо пакете
class PacketInf:

    def __init__( self, numPacket, timePacket, packetSize, mac_src, mac_dest, protoType
                  , ip_src, ip_dest, port_src, port_dest, len_data, data
                  , seq=None, ack=None, fl_ack=None, fl_psh=None, fl_syn=None):
        self.numPacket = int(numPacket)
        self.timePacket = float(timePacket)
        self.packetSize = int(packetSize)
        self.mac_src = mac_src
        self.mac_dest = mac_dest
        self.ip_src = ip_src
        self.ip_dest = ip_dest
        self.port_src = port_src
        self.port_dest = port_dest
        self.len_data = int(len_data)
        self.data = data
        self.protoType = protoType
        self.seq = seq
        self.ack = ack
        self.fl_ack = fl_ack
        self.fl_psh = fl_psh
        self.fl_syn = fl_syn

# Класс, содержащий информацию относительно какого-либо IP-адреса
class ExploreObject:

    def __init__(self, ip):
        self.ip = ip
        self.strt_time = None
        self.fin_time = None
        self.amnt_packet = None
        self.avg_packet_num = None
```

```

self.avg_packet_size = None

self.in_out_rel_data = None
self.ack_flags_diff_data = None
self.udp_tcp_rel_data = None
self.syn_flags_freq_data = None
self.psh_flags_freq_data = None
self.adjcIPList = None
self.adjcPacketList = None

# Считывание с файла и заполнение массива
# Packet_list объектами класса PacketInf
def read_from_file(inf):
    a = []
    while True:
        beg = inf.find(':')
        end = inf.find(';')
        if beg == -1 and end == -1:
            break
        else:
            a.append(inf[beg + 1: end])
            inf = inf[end + 1:]
    if a[5] == 'TCP':
        Packet_list.append(PacketInf( a[0], a[1], a[2], a[3], a[4], a[5]
                                     , a[6], a[7], a[8], a[9], a[15], a[16]
                                     , a[10], a[11], a[12], a[13], a[14] ))
    elif a[5] == 'UDP':
        Packet_list.append(PacketInf( a[0], a[1], a[2], a[3], a[4], a[5]
                                     , a[6], a[7], a[8], a[9], a[10], a[11] ))

# Получение общей информации о текущей
# попытке перехвата трафика
def get_common_data():
    IPList = []
    numPacketsPerSec = []
    curTime = Packet_list[0].timePacket + 1
    fin = Packet_list[-1].timePacket + 1
    Labels_list.append(time.strftime('%H:%M:%S', time.localtime(Packet_list[0].timePacket)))
    cntPacket = 0
    i = 0
    while curTime < fin:
        for k in range(i, len(Packet_list)):
            if Packet_list[k].timePacket > curTime:
                numPacketsPerSec.append(cntPacket)
                Labels_list.append(time.strftime('%H:%M:%S', time.localtime(curTime)))
                cntPacket = 0

```

```

        i = k
        break
    cntPacket += 1
    curTime += 1
numPacketsPerSec.append(cntPacket)
for p in Packet_list:
    CurIP = p.ip_src
    if CurIP not in IPList:
        IPList.append(CurIP)
return IPList, numPacketsPerSec

# Получение данных об отношении входящего
# трафика к исходящему в единицу времени
def get_in_out_rel(exploreIP, strt, fin):
    cntInput = 0
    cntOutput = 0
    rel_list = []
    curTime = strt + 1
    fin += 1
    pos = 0
    while curTime < fin:
        for k in range(pos, len(Packet_list)):
            if Packet_list[k].timePacket > curTime:
                if cntOutput != 0:
                    rel_list.append(cntInput / cntOutput)
                else:
                    rel_list.append(0.0)
                cntInput = 0
                cntOutput = 0
                pos = k
                break
            if Packet_list[k].ip_src == exploreIP:
                cntOutput += 1
            if Packet_list[k].ip_dest == exploreIP:
                cntInput += 1
        curTime += 1
    if cntOutput != 0:
        rel_list.append(cntInput / cntOutput)
    else:
        rel_list.append(0.0)
    return rel_list

# Получение данных о разности количества
# исходящих ACK-флагов и количества входящих
# ACK-флагов
def get_ack_flags_diff(exploreIP, strt, fin):

```

```

cntInput = 0
cntOutput = 0
diff_list = []
curTime = strt + 1
fin += 1
pos = 0
while curTime < fin:
    for k in range(pos, len(Packet_list)):
        if Packet_list[k].timePacket > curTime:
            diff_list.append(cntOutput - cntInput)
            cntInput = 0
            cntOutput = 0
            pos = k
            break
        if Packet_list[k].protoType == 'TCP' and Packet_list[k].fl_ack == '1':
            if Packet_list[k].ip_src == exploreIP:
                cntOutput += 1
            if Packet_list[k].ip_dest == exploreIP:
                cntInput += 1
        curTime += 1
diff_list.append(cntOutput - cntInput)
return diff_list

# Получение данных об отношении количества
# входящего UDP-трафика на количество
# исходящего TCP-трафика в единицу времени
def get_udp_tcp_rel(exploreIP, strt, fin):
    cntUDP = 0
    cntTCP = 0
    curTime = strt + 1
    fin += 1
    pos = 0
    rel_list = []
    while curTime < fin:
        for k in range(pos, len(Packet_list)):
            if Packet_list[k].timePacket > curTime:
                if cntTCP != 0:
                    rel_list.append(cntUDP / cntTCP)
                else:
                    rel_list.append(0.0)
                cntTCP = 0
                cntUDP = 0
                pos = k
                break
            if Packet_list[k].ip_dest == exploreIP:
                if Packet_list[k].protoType == 'TCP':
                    cntTCP += 1

```

```

        if Packet_list[k].protoType == 'UDP':
            cntUDP += 1
        curTime += 1
    if cntTCP != 0:
        rel_list.append(cntUDP / cntTCP)
    else:
        rel_list.append(0.0)
    return rel_list

# Получение данных о частоте SYN-флагов
def get_syn_flags_freq(exploreIP, strt, fin):
    cntSynTCP = 0
    cntTCP = 0
    rel_list = []
    curTime = strt + 1
    fin += 1
    pos = 0
    while curTime < fin:
        for k in range(pos, len(Packet_list)):
            if Packet_list[k].timePacket > curTime:
                if cntTCP != 0:
                    rel_list.append(cntSynTCP / cntTCP)
                else:
                    rel_list.append(0.0)
                cntSynTCP = 0
                cntTCP = 0
                pos = k
                break
            if Packet_list[k].ip_dest == exploreIP and Packet_list[k].protoType == 'TCP':
                if Packet_list[k].fl_syn == '1':
                    cntSynTCP += 1
                else:
                    cntTCP += 1
            curTime += 1
    if cntTCP != 0:
        rel_list.append(cntSynTCP / cntTCP)
    else:
        rel_list.append(0.0)
    return rel_list

# Получение данных о частоте PSH-флагов
def get_psh_flags_freq(exploreIP, strt, fin):
    cntPshTCP = 0
    cntTCP = 0
    rel_list = []
    curTime = strt + 1

```



```

fin += 1
pos = 0
while curTime < fin:
    for k in range(pos, len(Packet_list)):
        if Packet_list[k].timePacket > curTime:
            if cntTCP != 0:
                rel_list.append(cntPshTCP / cntTCP)
            else:
                rel_list.append(0.0)
            cntPshTCP = 0
            cntTCP = 0
            pos = k
            break
        if Packet_list[k].ip_dest == exploreIP and Packet_list[k].protoType == 'TCP':
            if Packet_list[k].fl_psh == '1':
                cntPshTCP += 1
            else:
                cntTCP += 1
        curTime += 1
    if cntTCP != 0:
        rel_list.append(cntPshTCP / cntTCP)
    else:
        rel_list.append(0.0)
return rel_list

# Получение общей информации о трафике,
# связанном с выбранным IP-адресом
def get_inf_about_IP(exploreIP):
    adjcPacketList = []
    adjcIPList = []
    for p in Packet_list:
        if p.ip_src == exploreIP:
            adjcPacketList.append(p)
            adjcIPList.append(p.ip_dest)
        if p.ip_dest == exploreIP:
            adjcPacketList.append(p)
            adjcIPList.append(p.ip_src)
    return adjcPacketList, adjcIPList

# Вывод пакетов, связанных с выбранным IP-адресом
def print_adjacent_packets(adjcPacketList):
    cnt = 0
    for p in adjcPacketList:
        t = time.strftime('%H:%M:%S', time.localtime(p.timePacket))
        if cnt % 2 == 1:
            print( f'Номер пакета: {p.numPacket};', f'Время: {t};'

```

```

        , f'Размер: {p.packetSize};', f'MAC-адрес отправителя: {p.mac_src};'
        , f'MAC-адрес получателя: {p.mac_dest};'
        , f'IP-адрес отправителя: {p.ip_src};', f'IP-адрес получателя: {p.ip_dest};'
        , f'Протокол: {p.protoType};', f'Порт отправителя: {p.port_src};'
        , f'Порт получателя: {p.port_dest};', f'Количество байт: {p.len_data};' )
    else:
        print( Back.CYAN + Fore.BLACK + f'Номер пакета: {p.numPacket};' + f' Время: {t};' +
              f' Размер: {p.packetSize};' + f' MAC-адрес отправителя: {p.mac_src};' +
              f' MAC-адрес получателя: {p.mac_dest};' +
              f' IP-адрес отправителя: {p.ip_src};' + f' IP-адрес получателя: {p.ip_dest};' +
              f' Протокол: {p.protoType};' + f' Порт отправителя: {p.port_src};' +
              f' Порт получателя: {p.port_dest};' + f' Количество байт: {p.len_data};' )

    cnt += 1

# Вывод пар (число, IP-адрес) для
# предоставления выбора IP-адреса
# пользователю
def print_IP_list(IPList):
    num = 0
    cnt = 1
    for el in IPList:
        if cnt > 3:
            cnt = 0
            print ( '[' + str(num), '---', el, end=']\n' )
        else:
            print ( '[' + str(num), '---', el, end='] ' )
        cnt += 1
        num += 1

# Получение меток и "шага" для оси абсцисс
def get_x_labels(total_time):
    step = 0
    if total_time > 500:
        step = 8
    elif total_time > 100:
        step = 5
    elif total_time > 50:
        step = 2
    for i in range(0, len(Labels_list), step):
        x_axisLabels.append(Labels_list[i])
    return step

# Выбор опций для выбранного IP-адреса
def choose_options(k, strt, fin, step):
    curIP = Object_list[k].ip

```

```

if Object_list[k].adjcPacketList == None:
    Object_list[k].adjcPacketList, Object_list[k].adjcIPList = get_inf_about_IP(curIP)
if Object_list[k].strt_time == None:
    Object_list[k].strt_time = time.localtime(Object_list[k].adjcPacketList[0].timePacket)
if Object_list[k].fin_time == None:
    Object_list[k].fin_time = time.localtime(Object_list[k].adjcPacketList[-1].timePacket)
if Object_list[k].amnt_packet == None:
    Object_list[k].amnt_packet = len(Object_list[k].adjcPacketList)
if Object_list[k].avg_packet_num == None:
    tmp = Object_list[k].adjcPacketList[-1].timePacket - \
        Object_list[k].adjcPacketList[0].timePacket
    Object_list[k].avg_packet_num = round(Object_list[k].amnt_packet / tmp, 3)
if Object_list[k].avg_packet_size == None:
    avgSize = 0
    for p in Object_list[k].adjcPacketList:
        avgSize += p.len_data
    Object_list[k].avg_packet_size = round(avgSize / Object_list[k].amnt_packet, 3)
while True:
    print(f'Общая информация о трафике, связанном с {curIP}')
    print('Время первого перехваченного пакета: ',
          time.strftime('%d.%m.%Y г. %H:%M:%S', Object_list[k].strt_time) )
    print('Время последнего перехваченного пакета: ',
          time.strftime('%d.%m.%Y г. %H:%M:%S', Object_list[k].fin_time) )
    print('Количество пакетов: ', Object_list[k].amnt_packet)
    print('Среднее количество пакетов в секунду: ', Object_list[k].avg_packet_num)
    print('Средний размер пакетов: ', Object_list[k].avg_packet_size)
    print(f'Выберите опцию:
    1. Вывести весь трафик, связанный с {curIP}
    2. Построить график отношения входящего и исходящего трафиков
    3. Построить график отношения объема входящего UDP-трафика и объёма входящего TCP-трафика
    4. Построить график разности числа исходящих и числа входящих АСК-флагов в единицу времени
    5. Построить график частоты SYN и PSH флагов во входящих пакетах
    6. Вернуться к выбору IP-адреса ""')
    b1 = input()
    if b1 == '1':
        print_adjacent_packets(Object_list[k].adjcPacketList)
    elif b1 == '2':
        if Object_list[k].in_out_rel_data == None:
            data = get_in_out_rel(curIP, strt, fin)
            Object_list[k].in_out_rel_data = data
        x = [i for i in range(0, len(Object_list[k].in_out_rel_data))]
        x_labels = [i for i in range(0, len(x), step)]
        fig = plt.figure(figsize=(16, 6), constrained_layout=True)
        f = fig.add_subplot()
        f.grid()
        f.set_title('Отношение объема входящего к объему исходящего трафиков', fontsize=15)
        f.set_xlabel('Общее время перехвата трафика', fontsize=15)
        plt.plot(x, Object_list[k].in_out_rel_data)

```

```

plt.xticks(x_labels, x_axisLabels, rotation=30)
plt.show()
elif bl == '3':
    if Object_list[k].udp_tcp_rel_data == None:
        data = get_udp_tcp_rel(curIP, strt, fin)
        Object_list[k].udp_tcp_rel_data = data
    x = [i for i in range(0, len(Object_list[k].udp_tcp_rel_data))]
    x_labels = [i for i in range(0, len(x), step)]
    fig = plt.figure(figsize=(16, 6), constrained_layout=True)
    f = fig.add_subplot()
    f.grid()
    f.set_title( 'Отношение объема входящего UDP-трафика к объему входящего TCP-трафика'
                , fontsize=15 )
    f.set_xlabel('Общее время перехвата трафика', fontsize=15)
    plt.plot(x, Object_list[k].udp_tcp_rel_data)
    plt.xticks(x_labels, x_axisLabels, rotation=30)
    plt.show()
elif bl == '4':
    if Object_list[k].ack_flags_diff_data == None:
        data = get_ack_flags_diff(curIP, strt, fin)
        Object_list[k].ack_flags_diff_data = data
    x = [i for i in range(0, len(Object_list[k].ack_flags_diff_data))]
    x_labels = [i for i in range(0, len(x), step)]
    fig = plt.figure(figsize=(16, 6), constrained_layout=True)
    f = fig.add_subplot()
    plt.plot(x, Object_list[k].ack_flags_diff_data)
    f.grid()
    f.set_title('Разность числа исходящих и числа входящих АСК-флагов', fontsize=15)
    f.set_xlabel('Общее время перехвата трафика', fontsize=15)
    plt.xticks(x_labels, x_axisLabels, rotation=30)
    plt.show()
elif bl == '5':
    if Object_list[k].syn_flags_freq_data == None:
        data = get_syn_flags_freq(curIP, strt, fin)
        Object_list[k].syn_flags_freq_data = data
    if Object_list[k].psh_flags_freq_data == None:
        data = get_psh_flags_freq(curIP, strt, fin)
        Object_list[k].psh_flags_freq_data = data
    x = [i for i in range(0, len(Object_list[k].syn_flags_freq_data))]
    x_labels = [i for i in range(0, len(x), step)]
    fig = plt.figure(figsize=(16, 6), constrained_layout=True)
    gs = gridspec.GridSpec(ncols=1, nrows=2, figure=fig)
    fig_1 = fig.add_subplot(gs[0, 0])
    fig_1.grid()
    plt.plot(x, Object_list[k].syn_flags_freq_data, 'b')
    plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=8)
    fig_2 = fig.add_subplot(gs[1, 0])
    fig_2.grid()

```

```

plt.plot(x, Object_list[k].psh_flags_freq_data, 'g')
plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=8)
fig_1.set_title('Частота флагов SYN', fontsize=15)
fig_1.set_xlabel('Общее время перехвата трафика', fontsize=15)
fig_2.set_title('Частота флагов PSH', fontsize=15)
fig_2.set_xlabel('Общее время перехвата трафика', fontsize=15)
plt.show()
elif bl == '6':
    break

if __name__ == '__main__':
    print('Введите название файла (например: data.log)')
    FileName = input()
    while True:
        if not Packet_list:
            try:
                f = open(FileName, 'r')
            except:
                print('Некорректное название файла!')
                exit(0)
            while True:
                inf = f.readline()
                if not inf:
                    break
                read_from_file(inf)
            f.close()
            IPList, numPacketsPerSec = get_common_data()
            strt = Packet_list[0].timePacket
            fin = Packet_list[-1].timePacket
            strt_time = time.localtime(strt)
            fin_time = time.localtime(fin)
            avgNumPacket = 0
            for el in numPacketsPerSec:
                avgNumPacket += el
            avgNumPacket /= len(numPacketsPerSec)
            avgSizePacket = 0
            for p in Packet_list:
                avgSizePacket += p.packetSize
            avgSizePacket /= len(Packet_list)
            step = get_x_labels(int(fin - strt))

    print('Общая информация:')
    print('Время первого перехваченного пакета: ',
          time.strftime('%d.%m.%Y г. %H:%M:%S', strt_time) )
    print('Время последнего перехваченного пакета: ',
          time.strftime('%d.%m.%Y г. %H:%M:%S', fin_time) )
    print('Количество пакетов: ', len(Packet_list))

```

```

print('Общее время перехвата: ', round(fin - strt, 3), 'сек')
print('Среднее количество пакетов в секунду: ', round(avgNumPacket, 3))
print('Средний размер пакетов: ', round(avgSizePacket, 3))
print('Завершить просмотр (нажмите \"q\" для выхода)')
for k in range(0, len(IPList)):
    Object_list.append(ExploreObject(IPList[k]))
print_IP_list(IPList)
print(f'\nВыберите цифру (0 - {len(IPList) - 1}) для просмотра IP-адреса:')
k = input()
if k == 'q':
    break
try:
    k = int(k)
except:
    print('Некорректный ввод. Завершение программы...')
    break
else:
    if 0 <= k < len(IPList):
        choose_options(k, strt, fin, step)
    else:
        print(f'Введите число в пределах 0 - {len(IPList) - 1}')

```