

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 Место протокола RDP в структуре OSI.....	4
2 Принцип работы протокола RDP и анализ его поведения .....	9
3 Обнаружение сеанса удаленного управления с помощью разработан- ной программы .....	13
3.1 Описание работы программы «sniffer.py» .....	13
3.2 Описание работы программы «data-analysis.py» .....	15
4 Демонстрация работы программ .....	18
ЗАКЛЮЧЕНИЕ .....	25
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	26
Приложение А Код sniffer.py .....	28
Приложение Б Код data-analysis.py .....	32

## **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].

Стоит отметить, что RDP позволяет работать с удаленным компьютером, почти как с локальным. При успешном создании RDP-сессии пользователь может двигать мышкой, открывать файлы, диски, документы, программы, без каких-либо проблем использовать буфер обмена (Ctrl+C, Ctrl+V) не только для текста, но и для файлов. Также отлично работает передача сочетаний клавиш, переключения языков.

### 1.1 Место протокола RDP в структуре OSI

Эталонная модель OSI представляет собой 7-уровневую иерархическую сетевую иерархию, разработанную международной организацией по стандартам (International Standardization Organization — ISO). В рамках модели, любой протокол может взаимодействовать либо с протоколами своего уровня (горизонтальные взаимодействия), либо с протоколами уровня на единицу выше/ниже своего уровня (вертикальные взаимодействия). Каждый из семи уровней характеризуется типом данных, которым данный уровень оперирует и функционалом, который он предоставляет слою, находящемуся выше него [11]. Модель OSI включает в себя следующие уровни:

1. Физический уровень, который отвечает за передачу последовательности битов через канал связи;
2. Канальный уровень, где осуществляется разбиение данных на «кадры», размер которых обычно достигает от несколько сотен до нескольких тысяч байтов;
3. Сетевой уровень, на котором осуществляется структуризация и маршрутизация пакетов от отправителя к получателю;
4. Транспортный уровень, функцией которого является передача надежных последовательностей данных произвольной длины через коммуникацион-

- ную сеть от отправителя к получателю;
5. Сеансовый уровень, на котором происходит поддержка сессии связи, управление взаимодействием между приложениями;
  6. Уровень представления, который представляет данные в понятном для какой-либо конкретной машины виде;
  7. Прикладной уровень, предоставляющий набор интерфейсов для взаимодействия пользовательских процессов с сетью.

Вследствие этого, RDP является непосредственно протоколом прикладного уровня модели OSI, наряду с SMTP, HTTP, FTP и многими другими. Протоколы седьмого уровня используют TCP или UDP в качестве передачи информации. Поэтому данные протокола RDP хранятся в заголовках TCP и UDP.

Далее для понимания того, в каком виде информация передается от отправителя к получателю необходимо разобрать структуру пакета. Согласно вышесказанному с помощью протоколов TCP и UDP отправитель передает данные, принадлежащие RDP, получателю. Они хранятся в специальном поле данных. Его можно увидеть на рисунке 1, где изображена структура TCP-заголовка.

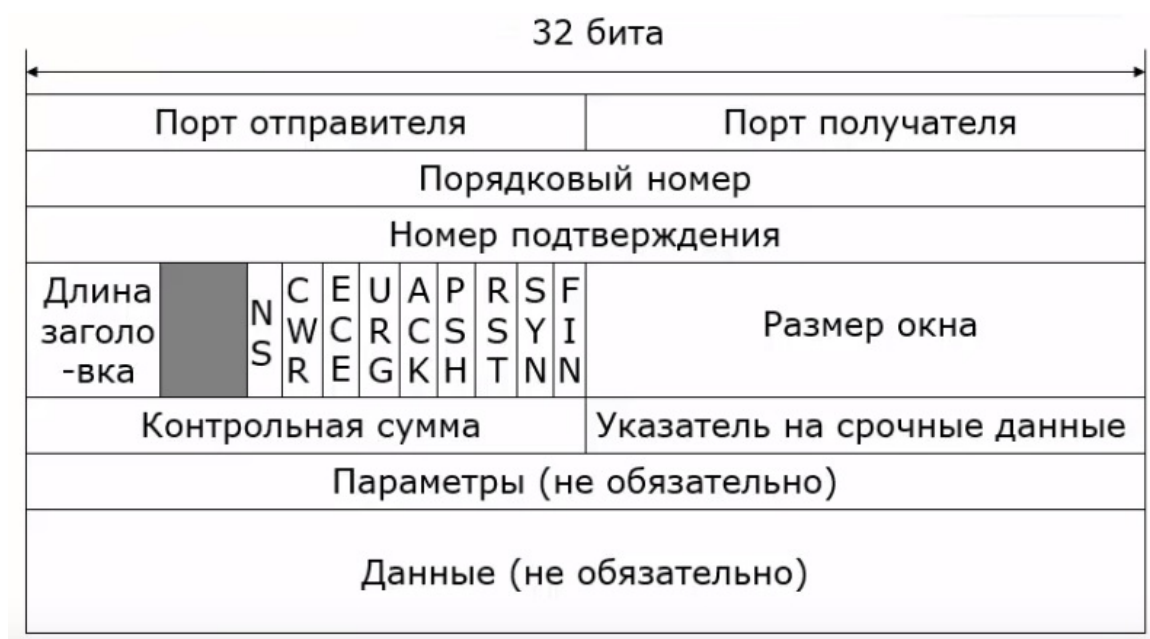


Рисунок 1 – Структура TCP-заголовка

Помимо поля данных в TCP-заголовке для дальнейшего анализа будут интересными поля, в которых хранится информация о портах отправителя и получателя. Стоит отметить, что при подключении к удаленному рабочему столу по умолчанию используется порт 3389. Поэтому для обнаружения RDP-сессии информация о портах будет достаточно полезной.

Также не менее интересной информацией являются установленные флаги, хранящиеся в поле флагов. В нем хранятся следующие управляющие биты:

1. NS — одноразовая сумма (Nonce Sum). По-прежнему является экспериментальным флагом, используемым для защиты от случайного злонамеренного сокрытия пакетов от отправителя [10]. Используется для улучшения работы механизма явного уведомления о перегрузке (Explicit Congestion Notification, ECN);
2. CWR — окно перегрузки уменьшено (Congestion Window Reduced). Данный флаг устанавливается (принимает значение равной единице) отправителем, чтобы показать, что TCP-фрагмент был получен с установленным полем ECE;
3. ECE — ECN-Эхо (ECN-Echo). Этот флаг показывает, поддерживает ли TCP-отправитель ECN;
4. URG — устанавливается, если необходимо передать ссылку на поле указателя срочности (Urgent pointer);
5. ACK — флаг подтверждения используется для подтверждения успешного получения пакета;
6. PSN — инструктирует получателя протолкнуть данные, накопившиеся в приемном буфере, в приложение пользователя;
7. RST — флаг сброса отправляется от получателя к отправителю, когда пакет отправляется на конкретный хост, который этого не ожидал;
8. SYN — начинает соединение и синхронизирует порядковые номера. Первый пакет, отправленный с каждой стороны, должен в обязательном порядке иметь установленным этот флаг;
9. FIN — означает, что данных от отправителя больше нет. Поэтому он используется в последнем пакете, отправленном отправителем.

Благодаря вышеописанным флагам можно узнать информацию о конкретном состоянии соединения.

IP пакет представляет собой отформатированную информацию в блоке, которая передается в сети. В настоящее время применяются две версии IP пакетов: IPv4 и IPv6. В данной работе будут рассматриваться пакеты IP версии 4, так как для анализа данных этого вполне достаточно. Поэтому далее необходимо рассмотреть IPv4-заголовок, который показан на рисунке 2.

Октет	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Версия			IHL			Тип обслуживания						Длина пакета																			
4	Идентификатор															Флаги		Смещение фрагмента														
8	Время жизни							Протокол							Контрольная сумма заголовка																	
12	IP-адрес отправителя																															
16	IP-адрес получателя																															
20	Параметры от 0-я до 10-и 32-х битовых слов																															
	Данные																															

Рисунок 2 – Структура IPv4-заголовка

IPv4 используется на сетевом уровне модели OSI для осуществления передачи пакетов между сетями.

В его заголовке достаточно важной информацией являются поля, в которых записаны IP-адреса отправителя и получателя. Ведь благодаря этой информации можно определить, между какими устройствами происходит обмен данными.

Также необходимо обратить внимание на 8-разрядное поле протокола. На рисунке оно обозначено как «Протокол». С помощью него можно идентифицировать протоколы следующего уровня (TCP, UDP, ICMP и другие) по его номеру. Например, если в IPv4-заголовке в поле «Протокол» будет задан номер 6, значит в поле данных хранится информация о протоколе TCP.

Согласно модели OSI, если перейти на один уровень ниже, а именно на канальный, то на нем осуществляется инкапсуляция пакета, полученного на сетевом уровне, где добавляется дополнительный заголовок, кадр Ethernet, к сегменту данных. Его структура показана на рисунке 3.



Рисунок 3 – Структура Ethernet кадра

Здесь достаточно важной информацией считаются поля адресов отправителя и получателя. Ведь в них содержатся MAC-адреса источника и назначения

соответственно. В поле «Тип», содержится номер, идентифицирующий тип сетевого протокола. Также в поле данных содержатся данные от более высокого уровня согласно модели OSI, а именно инкапсулированные данные о пакете.

Далее необходимо разобрать, опираясь на практическую часть, поведение протокола RDP при подключении к удаленному рабочему столу.



## 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 подключения к удаленному рабочему столу, во время которого осуществлялся захват трафика с помощью одной известной программы Wireshark. С помощью нее можно достаточно подробно рассмотреть структуру сообщений протоколов, поддерживающих RDP-сессию.

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 4, принадлежащие следующим протоколам:

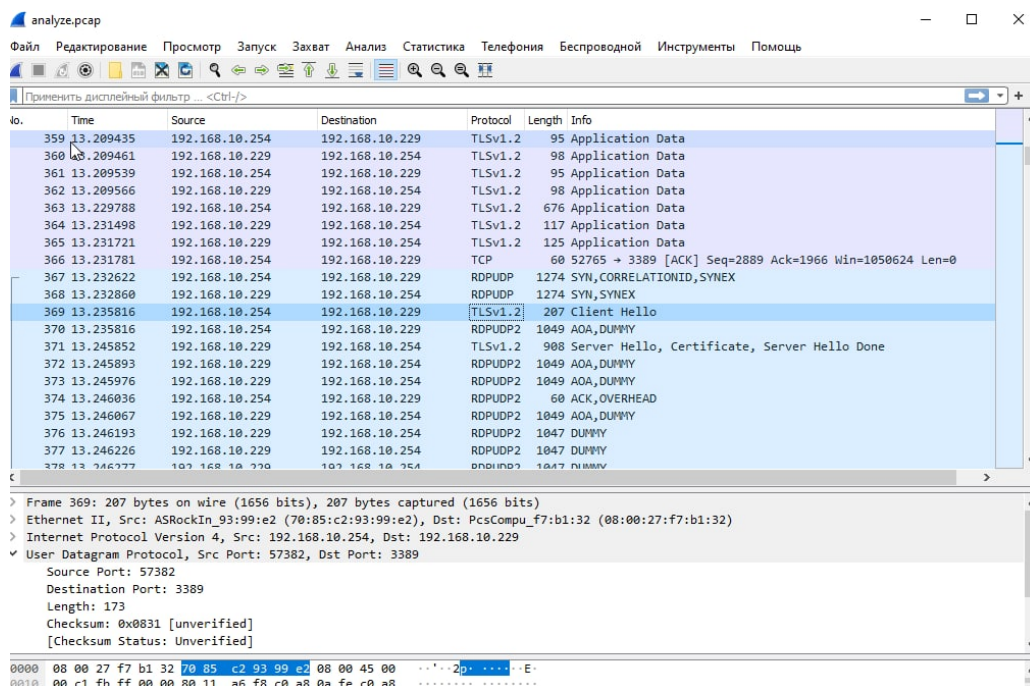


Рисунок 4 – Окно программы Wireshark после захвата трафика

- 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, осуществляемый в момент аутентификации. 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 5 можно увидеть запись random сервера.

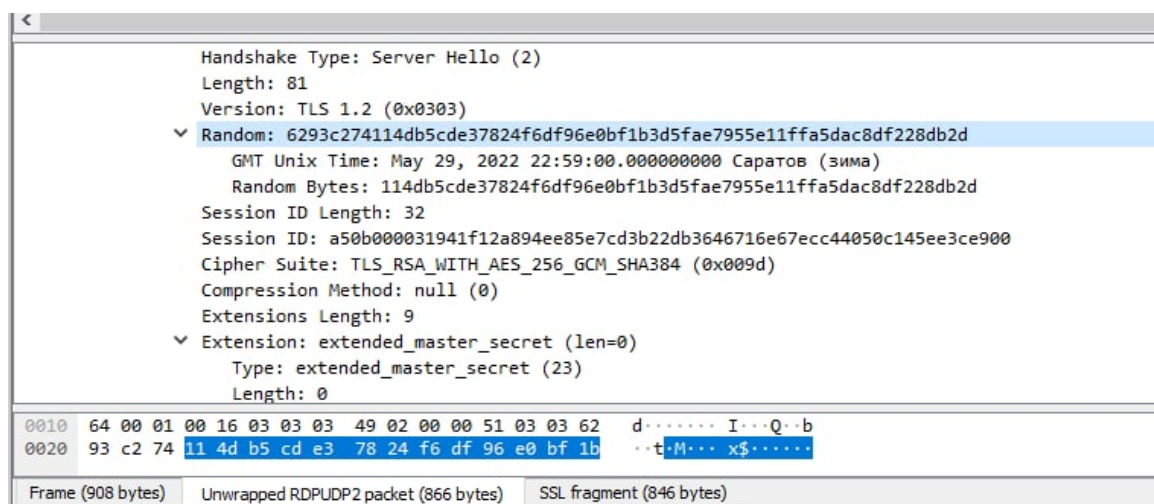


Рисунок 5 – Содержимое пакета, посылаемого от сервера клиенту (запись random сервера)

Совокупность открытого ключа и некоторая строка случайных байтов называется «сертификатом». Данная запись изображена на рисунке 6.

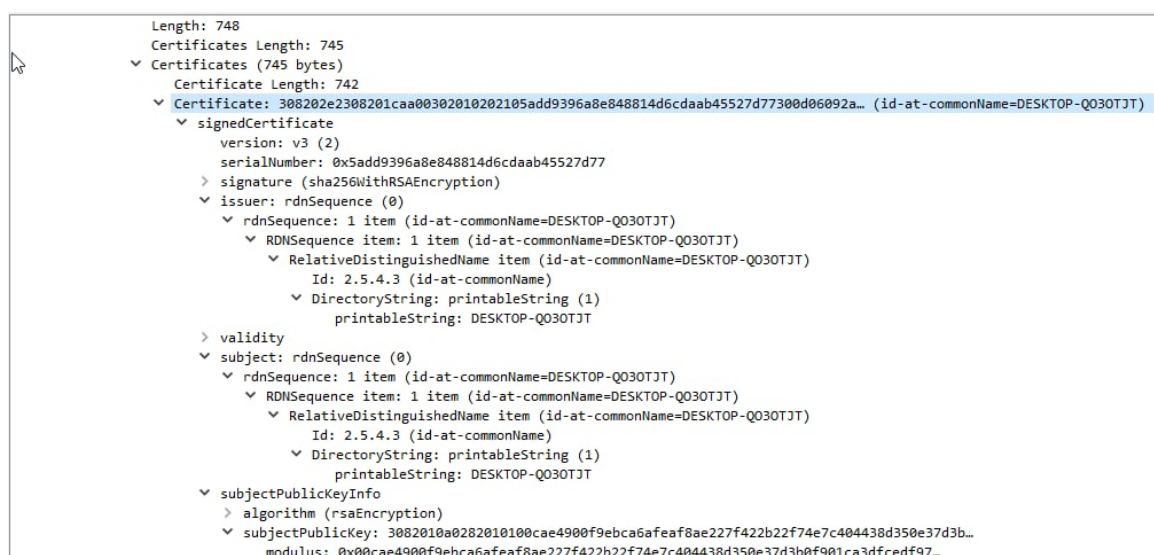


Рисунок 6 – Содержимое пакета, посылаемого от сервера клиенту (запись сертификата)

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 7.

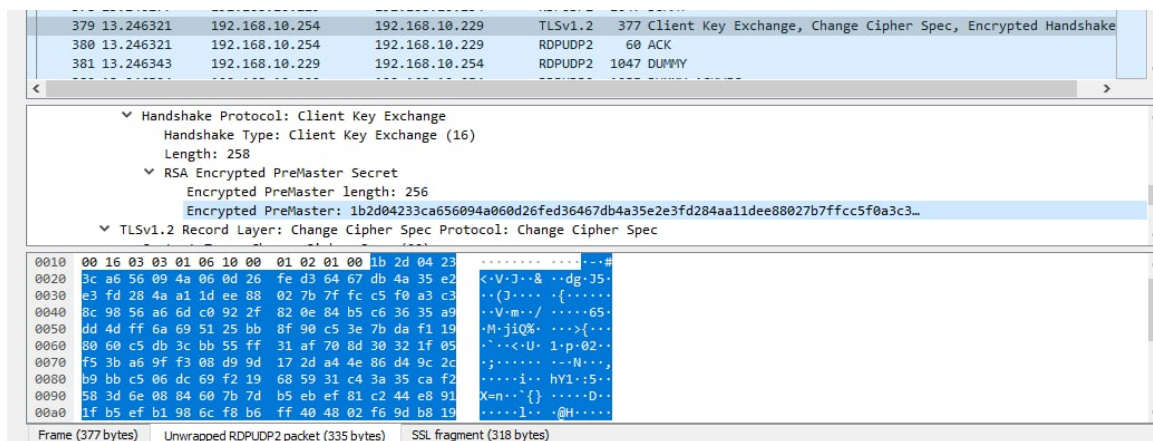


Рисунок 7 – Содержимое пакета, посылаемого от клиента серверу (запись premaster secret)

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.

После того, как всё настроено, сессия RDP готова к работе. На ПК клиента от сервера поступает графическое изображение (результат операций), которое происходит в результате отправки команд с клавиатуры или мыши. Стоит отметить, что для такого вида отправки сообщений подходит протокол UDP. Ведь область применения UDP заключается в обмене короткими сообщениями в режиме запрос-ответ, и этот протокол часто используется для передачи потоковых данных, например, аудио или видео. Поэтому неслучайно при RDP-сессии используется протокол UDP.

Осталось только понять, как обнаружить передачу информации, характерную для подключения к удаленному рабочему столу.

### **3 Обнаружение сеанса удаленного управления с помощью разработанной 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 «sniffer.py», 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. В этот момент информация о перехваченных пакетах будет отображаться в консоли, а также будет записываться необходимая информация каждого пакета для анализа в текстовый файл. В результате работы программы «sniffer.py» будет получен файл data.log. Для обработки содержимого данного файла была написана программа «data-analysis», реализованная также на языке Python. С помощью нее можно проанализировать весь перехваченный трафик относительно каждого IP-адреса, полученного из файла. Описание этих программ будет представлено ниже.

#### **3.1 Описание работы программы «sniffer.py»**

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

перехватывает сообщения, получаемые на канальном уровне модели OSI.

Опираясь на вышеизложенную информацию, чтобы раскодировать данную последовательность чисел, необходимо разобрать кадр Ethernet, представленный на рисунке 3. 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.

После получения информации об Ethernet кадре идет раскодирование данных, принадлежащих сетевому уровню модели OSI.

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* будет получена информация о времени жизни текущего пакета, о номере транспортного протокола, об IP-адресах отправителя и получателя.

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.

С помощью функции *get\_tcp\_segment* производится получение информации, содержащейся в TCP-протоколе. Таким образом, теперь программе известны порт получателя, порт отправителя, порядковый номер, номер подтверждения, флаги и данные, которые содержат в себе TCP-заголовок.

Аналогично получается раскодирование данных UDP-заголовка с помощью функции *get\_udp\_segment*.

Вся необходимая информация о пакетах для дальнейшего анализа записывается в файл *data.log* с помощью функции *write\_to\_file*.

Практически все выше перечисленные функции вызываются в *start\_to\_listen*, где осуществляется сбор информации о получаемых пакетах и вывод ее в консоль. Стоит отметить, что все выше описанные функции приведены в приложении А, где показан непосредственно сам код программы «sniffer.py». Соответственно код программы «data-analysis.py» представлен в приложении Б.

### 3.2 Описание работы программы «data-analysis.py»

При запуске программа попросит пользователя ввести название файла. Далее производится считывание файла с помощью функции *read\_from\_file*, где в это время сохраняется информация о каждом пакете, перехваченном во время работы программы «sniffer.py».

После считывания собирается некоторая общая информация о перехваченном трафике. Под общей информацией подразумевается: время начала и завершения перехвата трафика, количество пакетов, среднее количество пакетов в секунду, средний размер пакетов. Вместе с этим в консоль выводится список IP-адресов, участвующих в передаче пакетов по сети, в момент работы программы «sniffer.py». Пользователю предоставляется возможность выбрать интересующий его IP-адрес для дальнейшего анализа пакетов, связанных с ним. После выбора конкретного IP-адреса осуществляется вывод опять же общей информации, но уже только относительно выбранного IP-адреса. Т.е. выводится время первого и последнего перехваченных пакетов, где в качестве отправителя или получателя выступает данный IP-адрес.

Таким образом можно понять, в какой конкретно момент времени начался обмен информацией с тем или иным IP-адресом. Также рассчитываются количество пакетов, среднее количество пакетов в секунду и средний размер пакетов для определения общего объема информации, передаваемой в данный отрезок времени. Помимо этого, пользователю предоставляются некоторые опции, которые будут описаны ниже:

1. Пользователю предоставляется возможность вывести весь сетевой трафик, где в качестве отправителя или получателя выступает выбранный IP-адрес.
2. При выборе второй опции строится график отношения входящего и исходящего трафиков в единицу времени. Данное отношение рассчитывается по формуле

$$r_{ip} = \frac{V_{dest}}{V_{src}},$$

где  $V_{dest}$  и  $V_{src}$  — объемы соответственно входящего и исходящего трафика в единицу времени.

При большой величине входящего трафика, можно узнать в какой момент времени происходит обмен информацией с каким-либо другим IP-адресом.



3. При выборе третьей опции строится график отношения  $V_{udp}$  — объема входящего UDP-трафика и  $V_{tcp}$  объема входящего TCP-трафика. Отношение рассчитывается по формуле

$$r_{udp} = \frac{V_{udp}}{V_{tcp}}.$$

Так как во время RDP-сессии осуществляется передача пакетов по протоколу UDP и TCP, то анализируя полученную информацию исходя из этого графика, можно установить признаки RDP-сессии. Ведь превышение объема входящего UDP-трафика над объемом TCP-трафика может стать одним из признаков установки RDP-сессии, когда одному устройству нужно обменяться большим количеством сообщений с другим устройством в режиме запрос-ответ.

4. При выборе данной опции строится график разности количества исходящих и входящих TCP-пакетов, в которых флаг ACK имеет значение равное единице.

$$r_{ack} = V_{A_{out}} - V_{A_{in}},$$

где  $V_{A_{in}}$  и  $V_{A_{out}}$  — число входящих и исходящих ACK-флагов в TCP-трафике в единицу времени. При подключении к удаленному рабочему столу сервер отправляет клиенту TCP-пакеты с установленным флагом ACK, указывающим, что поле номера подтверждения задействовано. В таком случае с помощью графика, анализируя изменение величины  $r_{ack}$  в разные промежутки времени, можно идентифицировать устройство, совершающее подключение к удаленному рабочему столу.

5. При выборе пятой опции строятся два графика, показывающие частоту SYN-флагов и PSN-флагов в TCP-трафике. Частота SYN-флагов находится по формуле

$$r_{syn} = \frac{V_{S_{in}}}{V_{tcp}},$$

где  $V_{S_{in}}$  число входящих TCP-пакетов, в которых установлен флаг SYN = 1,  $V_{tcp}$  — число входящих TCP-пакетов в единицу времени. Пакеты с флагом SYN пересылаются между клиентом и сервером в ходе установления TCP-соединения, после чего начинается обмен данными с помощью пакетов без SYN-флага. Таким образом, число SYN-флагов, пришедших на сервер,



равно числу запросов на соединение, а частота SYN-флагов определяет долю служебных пакетов этого типа в ТСП-трафике.

Частота PSH-флагов вычисляется по формуле

$$r_{psh} = \frac{V_{P_{in}}}{V_{tcp}},$$

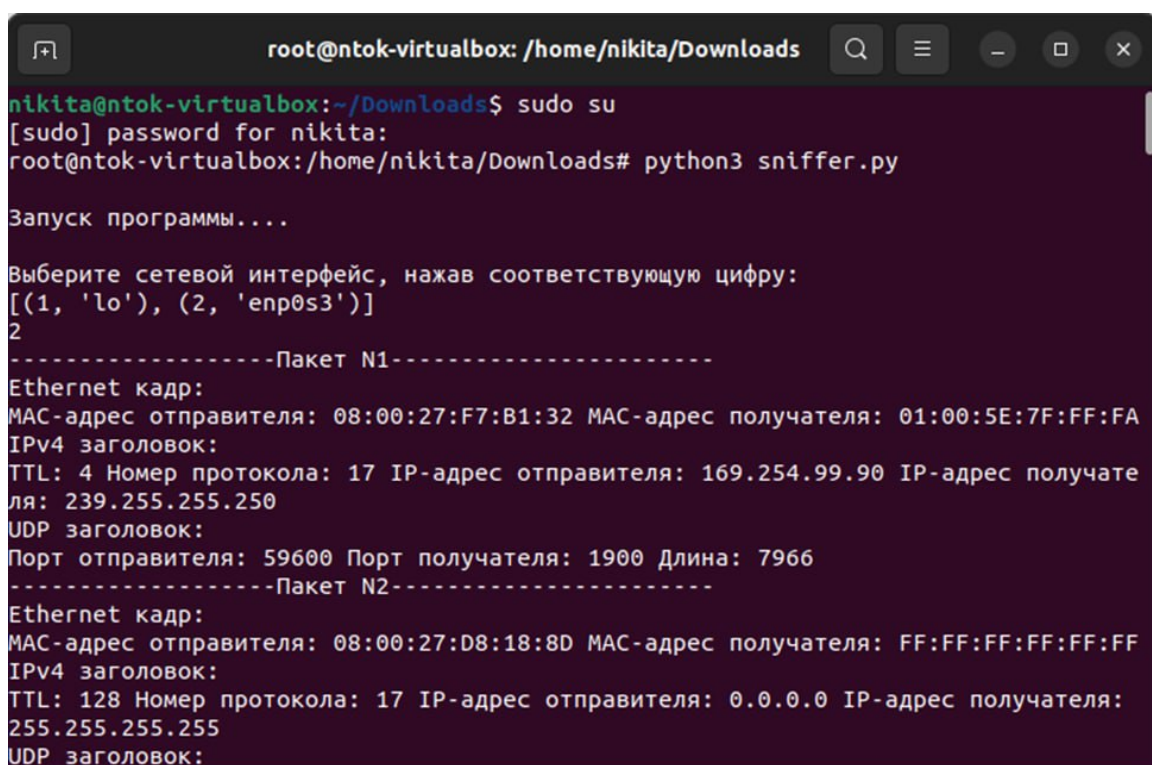
где  $V_{P_{in}}$  число входящих ТСП-пакетов, в которых установлен флаг PSH = 1,  $V_{tcp}$  — число входящих ТСП-пакетов в единицу времени. Опираясь на вышеописанную информацию, PSH-флаг инструктирует получателя, чтобы он отправил накопившиеся в приемном буфере данные в приложение отправителя. Таким образом, если значение величины  $r_{psh}$  резко возросло в некоторый промежуток времени, значит за это время одно устройство успело передать получателю большое количество пакетов.

После описания всех опций данной программы остается только показать, как это будет выглядеть на практике.

## 4 Демонстрация работы программ

Для тестирования данных программ было запущено три виртуальных машины. Две из них (ПК-1 и ПК-2) имеют операционную систему Windows 10 Professional версии 21H2. И у одной машины (ПК-3) поставлена операционная система Linux Ubuntu 22.04 LTS. Все устройства не имеют доступа к интернету, однако каждая виртуальная машина может взаимодействовать друг с другом.

Запустив программу «sniffer.py», был произведен захват трафика, как показано на рисунке 8.



```
root@ntok-virtualbox: /home/nikita/Downloads
nikita@ntok-virtualbox:~/Downloads$ sudo su
[sudo] password for nikita:
root@ntok-virtualbox:/home/nikita/Downloads# python3 sniffer.py

Запуск программы....

Выберите сетевой интерфейс, нажав соответствующую цифру:
[(1, 'lo'), (2, 'enp0s3')]
2
-----Пакет N1-----
Ethernet кадр:
MAC-адрес отправителя: 08:00:27:F7:B1:32 MAC-адрес получателя: 01:00:5E:7F:FF:FA
IPv4 заголовок:
TTL: 4 Номер протокола: 17 IP-адрес отправителя: 169.254.99.90 IP-адрес получателя: 239.255.255.250
UDP заголовок:
Порт отправителя: 59600 Порт получателя: 1900 Длина: 7966
-----Пакет N2-----
Ethernet кадр:
MAC-адрес отправителя: 08:00:27:D8:18:8D MAC-адрес получателя: FF:FF:FF:FF:FF:FF
IPv4 заголовок:
TTL: 128 Номер протокола: 17 IP-адрес отправителя: 0.0.0.0 IP-адрес получателя: 255.255.255.255
UDP заголовок:
```

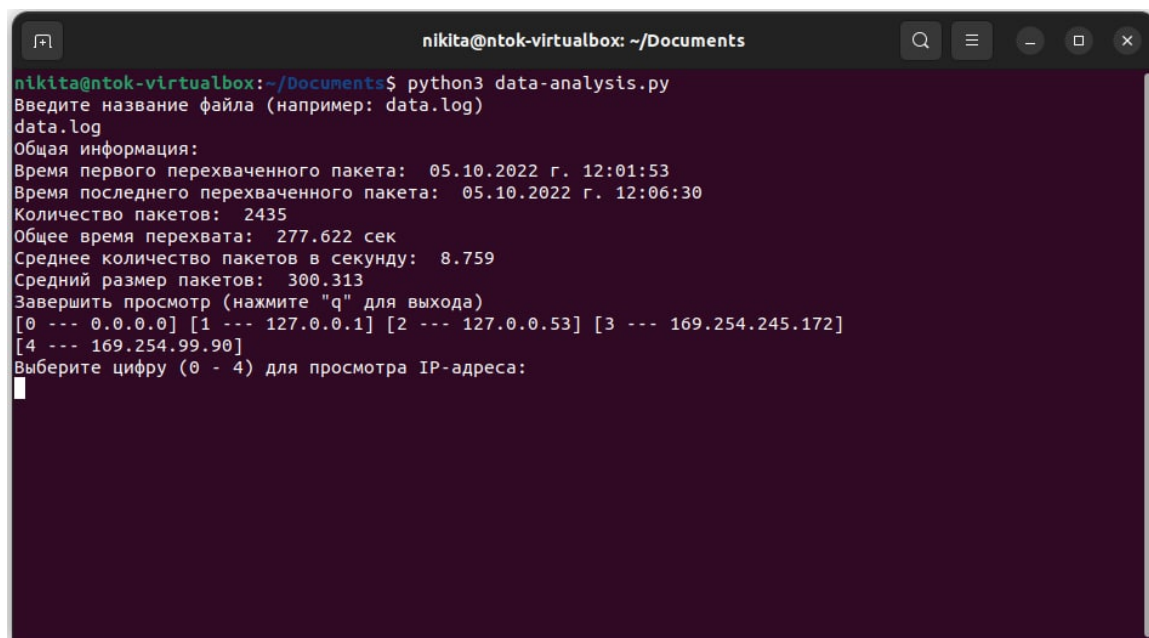
Рисунок 8 – Вид консоли при работе программы «sniffer.py»

Во время работы программы совершались следующие действия:

1. В 12:03 на ПК2 была запущена программа «Подключение к удаленному рабочему столу».
2. Примерно через 15 секунд было совершено подключение ПК2 к ПК1 по протоколу RDP.
3. Начиная с 12:03:50 производилось движение мышкой по рабочему столу, открытие текстовых файлов, лежащих на рабочем столе ПК1.
4. В 12:04:51 было завершено подключение к удаленному рабочему столу ПК1.

После завершения программы «sniffer.py» был получен файл data.log, в

котором хранится вся необходимая информация о каждом перехваченном пакете. Запустив программу «data-analysis.py» и введя название анализируемого файла, в консоли отображаются общие сведения о перехваченном трафике, как показано на рисунке 9.



```
nikita@ntok-virtualbox: ~/Documents
nikita@ntok-virtualbox:~/Documents$ python3 data-analysis.py
Введите название файла (например: data.log)
data.log
Общая информация:
Время первого перехваченного пакета: 05.10.2022 г. 12:01:53
Время последнего перехваченного пакета: 05.10.2022 г. 12:06:30
Количество пакетов: 2435
Общее время перехвата: 277.622 сек
Среднее количество пакетов в секунду: 8.759
Средний размер пакетов: 300.313
Завершить просмотр (нажмите "q" для выхода)
[0 --- 0.0.0.0] [1 --- 127.0.0.1] [2 --- 127.0.0.53] [3 --- 169.254.245.172]
[4 --- 169.254.99.90]
Выберите цифру (0 - 4) для просмотра IP-адреса:
█
```

Рисунок 9 – Вид консоли при работе программы «data-analysis.py»

Из рисунка 9 видно, что в данном промежутке времени в обмене данными принимали участие всего 5 IP-адресов, первые три из которых рассылает устройство ПК3. А последние два принадлежат ПК1 и ПК2. Осталось только определить какой IP-адрес принадлежит серверу, а какой — клиенту. Конечно, больше всего сейчас интересны последние два IP-адреса, поэтому далее пользователь введет значение 3, чтобы выбрать IP-адрес 169.254.245.172.

На рисунке 10 показана информация, связанная с IP-адресом 169.254.245.172.

```
nikita@ntok-virtualbox: ~/Documents
Общая информация:
Время первого перехваченного пакета: 05.10.2022 г. 12:01:53
Время последнего перехваченного пакета: 05.10.2022 г. 12:06:30
Количество пакетов: 2435
Общее время перехвата: 277.622 сек
Среднее количество пакетов в секунду: 8.759
Средний размер пакетов: 300.313
Завершить просмотр (нажмите "q" для выхода)
[0 --- 0.0.0.0] [1 --- 127.0.0.1] [2 --- 127.0.0.53] [3 --- 169.254.245.172]
[4 --- 169.254.99.90]
Выберите цифру (0 - 4) для просмотра IP-адреса:
3
Общая информация о трафике, связанном с 169.254.245.172
Время первого перехваченного пакета: 05.10.2022 г. 12:02:31
Время последнего перехваченного пакета: 05.10.2022 г. 12:04:50
Количество пакетов: 2278
Среднее количество пакетов в секунду: 16.296
Средний размер пакетов: 262.049
Выберите опцию:
1. Вывести весь трафик, связанный с 169.254.245.172
2. Построить график отношения входящего и исходящего трафиков
3. Построить график отношения объема входящего UDP-трафика и объема входящего TCP-трафика
4. Построить график разности числа исходящих и числа входящих ACK-флагов в единицу времени
5. Построить график частоты SYN и PSN флагов во входящих пакетах
6. Вернуться к выбору IP-адреса
```

Рисунок 10 – Вид консоли при выборе IP-адреса 169.254.245.172

При выборе второй опции будет построен график, изображенный на рисунке 11.



Рисунок 11 – График отношения входящего и исходящего трафиков относительно 169.254.245.172

Из рисунка 11 видно, что значение переменной  $V_{dest}$  начинает возрастать с 12:03:18, а на 12:03:20 эта величина достигает своего максимального значения.

Теперь для сравнения на следующем рисунке будет показан график отношения входящего и исходящего трафиков, но уже относительно IP-адреса 169.254.99.90.



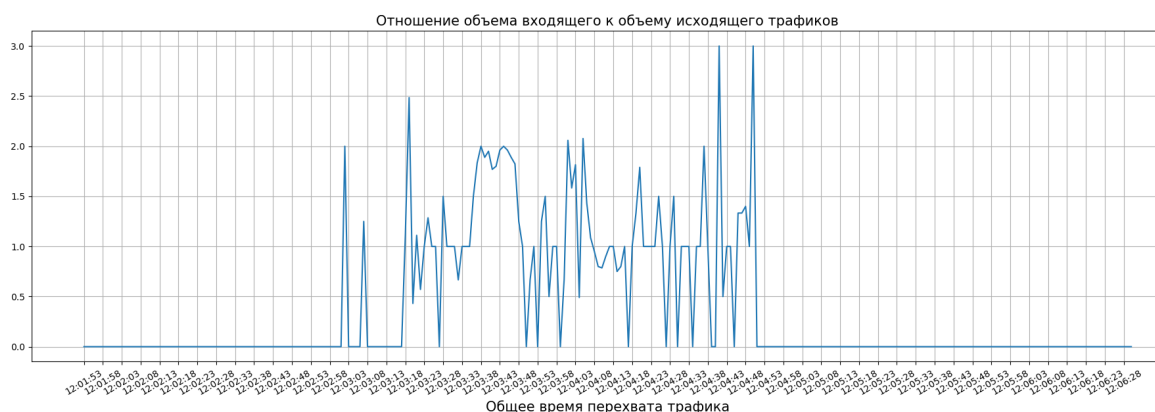


Рисунок 12 – График отношения входящего и исходящего трафиков относительно 169.254.99.90

Если посмотреть на следующий рисунок, то можно увидеть в данном трафике, что первый пакет был передан по порту 3389 в 12:03:02. Это также видно из графиков, показанных на рисунках 11 и 12. Тогда начиная с 12:03:18 произошла установка RDP-сессии.



Рисунок 13 – Просмотр трафика, связанного с 169.254.245.172

Исходя из рисунков 14 и 15, можно сделать вывод, что в промежутке 12:03:18 – 12:03:25 передается большой объем UDP-пакетов и скорее всего это произошло в момент подтверждения сертификата клиента сервером.



Рисунок 14 – График, построенный по формуле  $r_{udp} = \frac{V_{udp}}{V_{tcp}}$  относительно 169.254.245.172

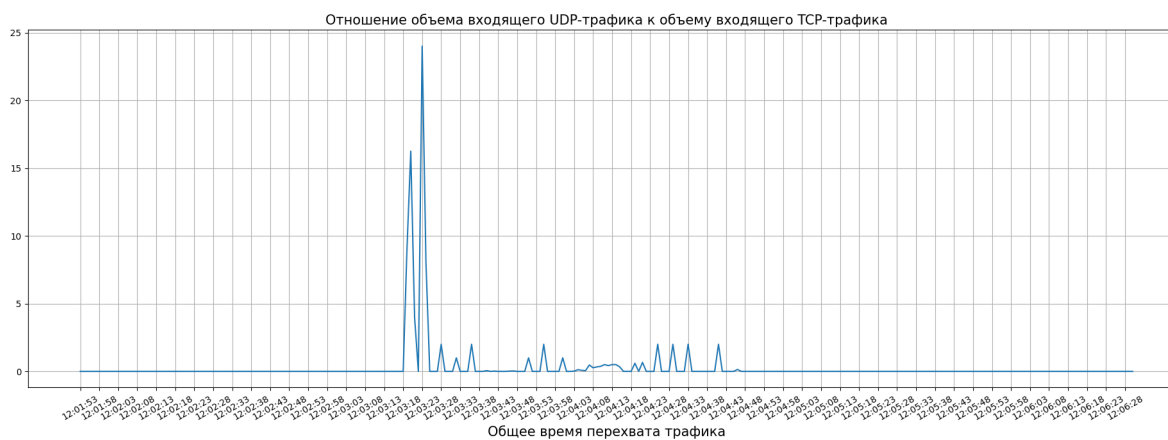


Рисунок 15 – График, построенный по формуле  $r_{udp} = \frac{V_{udp}}{V_{tcp}}$  относительно 169.254.99.90

На рисунках 16 и 17 изображены графики разности числа исходящих и входящих АСК-флагов в единицу времени.



Рисунок 16 – График, построенный по формуле  $r_{ack} = V_{ack_{out}} - V_{ack_{in}}$  относительно 169.254.245.172

Величина  $V_{ack_{in}}$  показывает, как часто сервер отказывает клиенту из-за перегрузки. На промежутке 12:03:36 – 12:03:49 видно, что осуществляется обмен

данными между клиентом и сервером. Как показано на рисунке 17 отрицательное значение величины  $r_{ack}$  показывает, что на данном промежутке времени сервер, получая от клиента ТСР-пакеты с установленным АСК-флагом, теряет возможность отвечать на запросы клиента.



Рисунок 17 – График, построенный по формуле  $r_{ack} = V_{Aout} - V_{Ain}$  относительно 169.254.99.90

Получается, что ПК1 (серверу) соответствует адрес 169.254.99.90, а ПК2 (клиенту) — 169.254.245.172.

На рисунках 18 и 19 изображены графики частот SYN- и PSH-флагов.

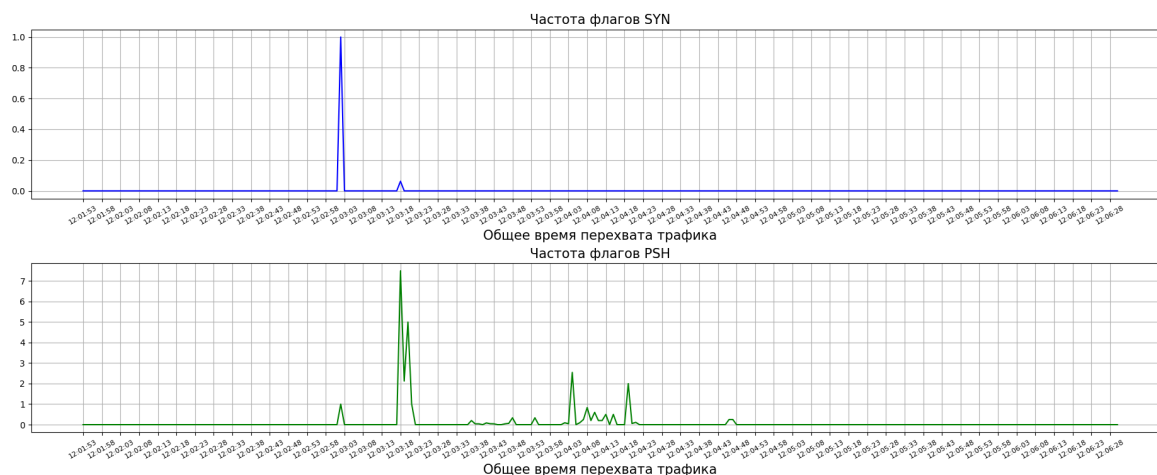


Рисунок 18 – График, построенный по формуле  $r_{syn} = \frac{V_{Sin}}{V_{tcp}}$  относительно 169.254.245.172



Рисунок 19 – График, построенный по формуле  $r_{psh} = \frac{V_{Pin}}{V_{tcp}}$  относительно 169.254.99.90

Исходя из количества PSN-флагов в момент времени 12:03:17 – 12:03:22, как показано на рисунке 18, на этом отрезке времени передаются также TCP-пакеты с установленным PSN-флагом, сигнализируя протоколу TCP отправить все данные, независимо от того, где и сколько их было уже передано.

По построенным графикам можно сделать несколько выводов. Например, из рисунка 14 отчетливо видно, что с 12:03:50 начинается рост значения  $V_{udp}$ , а значит увеличивается объем входящего UDP-трафика ПК2. Это можно аргументировать тем, что в этот момент времени происходят различные операции, сделанные на рабочем столе ПК1. В данном случае были произведены движения мышкой.

Также из рисунков 11 и 12 видно, что обмен данными между ПК1 и ПК2 прерывается в 12:04:51. Тогда можно предположить, что именно в это время закончилась RDP-сессия.



## **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 хоть RDP далеко не самый защищенный протокол, его обнаружение может стать затруднительным, особенно когда производится подключение к удаленному рабочему столу в реальных условиях с выходом в интернет. И для его анализа обычного перехвата трафика будет недостаточно. Однако, благодаря различным метрикам и построенным по ним графикам, по которым анализируют сетевые атаки, можно получить полезную информацию. Поэтому в дальнейшем, опираясь на проделанную работу, будут совершены попытки выявления протокола RDP от всех прочих протоколов сети интернет путем анализа данных по построенным графикам.

## 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.

- 10 Статья «TCP flags» [Электронный ресурс] / URL: <https://www.keycdn.com/support/tcp-flags#:~:text=ACK> (дата обращения 02.10.2022), Яз. англ.
- 11 Статья «Модель OSI» [Электронный ресурс] / URL: [http://neerc.ifmo.ru/wiki/index.php?title=OSI\\_Model](http://neerc.ifmo.ru/wiki/index.php?title=OSI_Model) (дата обращения 13.10.2022), Яз. рус.

## ПРИЛОЖЕНИЕ А

### Код sniffer.py

```
import socket, 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.adjciPacketList = 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:]
    try:
        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] ))
    except:
        print('Ошибка при считывании файла...')
        exit(0)

# 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
        if tmp == 0:
            tmp = 1
        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 объема входящего TCP-трафика
        4. Построить график разности числа исходящих и числа входящих ACK-флагов в единицу времени
        5. Построить график частоты SYN и PSN 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 трафика', 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')
else:
    if 0 <= k < len(IPList):
        choose_options(k, strt, fin, step)
    else:
        print(f'Введите число в пределах 0 - {len(IPList) - 1}')

```