

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

**СТАТИСТИЧЕСКИЙ АНАЛИЗ СЕТЕВОГО ТРАФИКА ДЛЯ
ОБНАРУЖЕНИЯ АКТИВНОЙ RDP-СЕССИИ**

КУРСОВАЯ РАБОТА

студента 4 курса 431 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Токарева Никиты Сергеевича

Научный руководитель
доцент

Гортинский А. В.

Заведующий кафедрой

Абросимов М. Б.

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Определение RDP	4
1.1 Место протокола RDP в структуре OSI	4
2 Обзор существующих методов обнаружения RDP-трафика	9
3 Программная реализация метода обнаружения RDP-трафика	12
3.1 Определение активных сессий путем анализа TCP-соединения	17
3.2 Обработка данных и построение графиков для анализа поведения RDP-трафика	21
4 Анализ распределения размера пакетов	23
4.1 Вычисление среднего значения и стандартного отклонения размеров пакетов	23
4.2 Определение верхней и нижней границ диапазона значений размеров пакетов для каждого интервала времени	24
4.3 Анализ полученных данных на наличие признаков RDP-сессии	24
5 Анализ распределения временных интервалов между пакетами	28
5.1 Вычисление среднего значения и стандартного отклонения интервалов	28
5.2 Определение пороговых значений для интервалов	29
5.3 Анализ полученных данных на наличие признаков RDP-сессии	29
6 Анализ частоты флагов PSN	30
6.1 Расчет частоты флагов PSN для каждого интервала времени	30
6.2 Анализ полученных данных на наличие признаков RDP-сессии	30
7 Некоторые модификации для улучшения обнаружения RDP-сессии	33
8 Тестирование программы на определение наличия или отсутствия RDP-сессии	37
ЗАКЛЮЧЕНИЕ	45
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	46
Приложение А Код traffic-detection.py	48

ВВЕДЕНИЕ

Сегодня удаленный доступ к компьютерам является важным элементом современного мира. Сотрудники компаний могут работать на расстоянии, а IT-специалисты могут удаленно управлять компьютерами, находящимися в другой стране. Однако, в то же время, удаленный доступ может стать уязвимостью компьютерной системы. Один из наиболее распространенных протоколов для удаленного доступа является RDP (Remote Desktop Protocol).

Цель данной курсовой работы — разработка метода статистического анализа сетевого трафика для обнаружения активной RDP-сессии. Будут использованы статистические методы анализа, такие как распределение временных интервалов между пакетами, нахождение стандартного отклонения и среднего значения, для выявления характеристик, свойственных протоколу RDP.

В работе будет представлено описание алгоритма, позволяющего производить статистический анализ сетевого трафика для обнаружения активной RDP-сессии, а также оценка эффективности методов на реальных сетевых данных. Результаты данной работы могут быть использованы в качестве инструмента для мониторинга сетевого трафика.

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 [1].

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 [4]. 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.

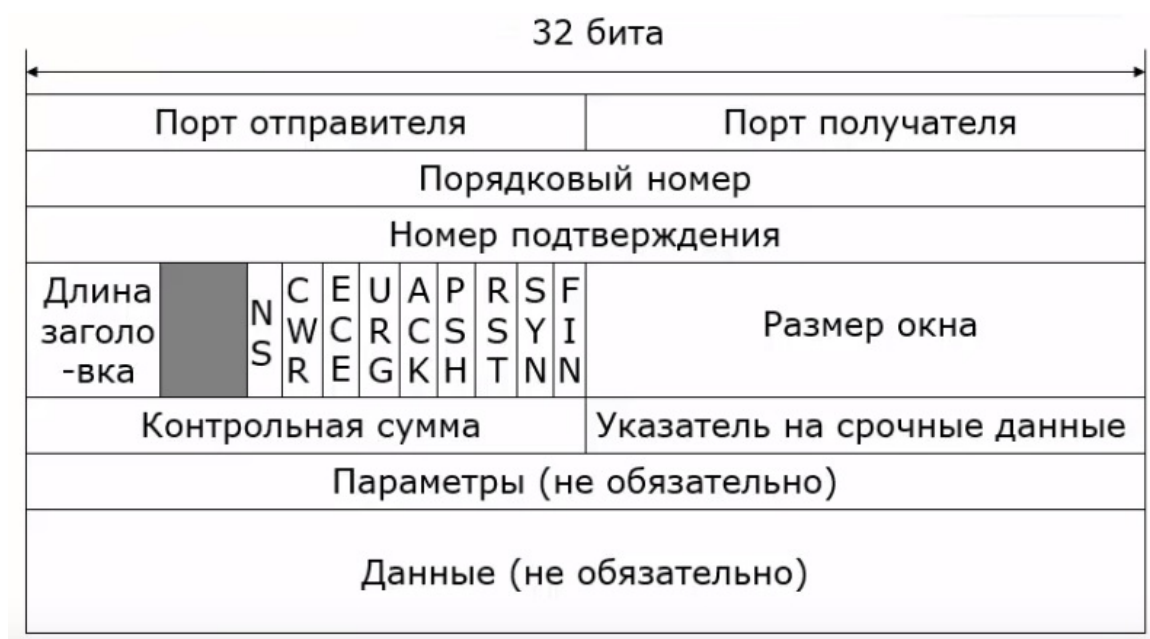


Рисунок 1 – Структура TCP-заголовка

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 [5]. 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. PSH — 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 рассмотреть методы обнаружения RDP-трафика, которые позволят эффективно выявить наличие использования RDP-протокола.

2 Обзор существующих методов обнаружения RDP-трафика

Обзор существующих методов обнаружения RDP-трафика представляет собой анализ различных подходов и техник, применяемых для выявления использования RDP-протокола в сетевом трафике.

Существует несколько методов обнаружения RDP-трафика, которые могут использоваться для мониторинга сети и выявления потенциальных угроз:

1. Анализ портов: RDP-протокол обычно использует TCP-порт 3389, поэтому можно использовать анализ портов для обнаружения трафика, проходящего через этот порт.
2. Поиск заголовков пакетов: RDP-протокол имеет уникальную сигнатуру в заголовке пакетов, которые могут быть использованы для обнаружения его наличия в сети.
3. Машинное обучение: Машинное обучение может быть использовано для создания моделей, которые могут обнаруживать RDP-трафик на основе статистических данных и образцов поведения сети.
4. Анализ временных интервалов: Временные интервалы между пакетами RDP-трафика обычно меньше, чем между другими типами трафика, что можно использовать для обнаружения RDP-сессий.
5. Анализ размеров пакетов: Размеры пакетов RDP-трафика обычно больше, чем у других типов трафика, что также может помочь в обнаружении RDP-сессий.
6. Анализ флагов пакетов: определенные флаги пакетов могут указывать на использование RDP-протокола. Например, флаг PSN может указывать на передачу данных в реальном времени в рамках RDP-сессии.

Хотя все вышеперечисленные методы обнаружения RDP-трафика могут быть полезными инструментами для обнаружения RDP-сессии, но ни один из них не является идеальным.

Если брать в рассмотрение анализ портов, то этот метод неэффективен по нескольким причинам. Во-первых, злоумышленники могут изменить порт, используемый для RDP-соединения, чтобы избежать обнаружения. Во-вторых, если на одном компьютере работает несколько RDP-сессий, они могут использовать разные порты, что затрудняет обнаружение RDP-трафика на основе порта. В-третьих, RDP-трафик может быть запакован в другой протокол, который использует другой порт, что также затрудняет обнаружение по порту.

Поиск заголовков пакетов также может быть ненадежным методом обнаружения RDP-трафика, потому что некоторые приложения могут использовать измененные заголовки, чтобы скрыть свой трафик. Кроме того, если RDP-трафик зашифрован, то заголовки пакетов могут быть недоступны для анализа. Также возможно наличие поддельных заголовков, созданных злоумышленниками для обхода системы обнаружения RDP-трафика. Все это делает поиск заголовков пакетов не надежным методом для обнаружения RDP-трафика в некоторых случаях.

При использовании машинного обучения для обнаружения RDP-трафика может возникнуть ряд проблем:

1. Необходимость большого объема данных: Для того чтобы создать надежную модель машинного обучения для обнаружения RDP-трафика, требуется большой объем данных для обучения. Данные должны включать в себя как положительные, так и отрицательные примеры RDP-трафика, что может быть сложно собрать.
2. Низкая точность: Машинное обучение может иметь низкую точность при обнаружении RDP-трафика из-за возможных ошибок классификации. Например, некоторые другие протоколы могут иметь схожие характеристики с RDP-трафиком, что может привести к неверной классификации.
3. Низкая скорость: Машинное обучение может быть времязатратным процессом. Обучение модели может занять много времени и требовать больших вычислительных ресурсов.
4. Адаптация к новым типам RDP-трафика: Машинное обучение может не справиться с обнаружением новых типов RDP-трафика, которые отличаются от тех, которые были использованы при обучении модели.

Все эти факторы могут привести к тому, что машинное обучение не будет надежным методом обнаружения RDP-трафика. Однако, если используется достаточно объемный и репрезентативный набор данных для обучения, а также проводится тщательное тестирование модели, то машинное обучение может быть эффективным методом обнаружения RDP-трафика.

Стоит отметить, что каждый из методов анализа временных интервалов, размеров пакетов и флагов пакетов имеет свои собственные недостатки. Тем не менее, все три метода могут быть реализованы совместно.

Основная часть работы состояла из двух этапов. На первом этапе был

разработан сниффер, предназначенный для перехвата сетевого трафика. В рамках этого этапа были созданы функции записи и чтения сетевого трафика в файл, а также возможность построения графиков, которые позволяли просмотреть сетевой трафик относительно заданного IP-адреса и порта, и проведения статистической обработки этих параметров.

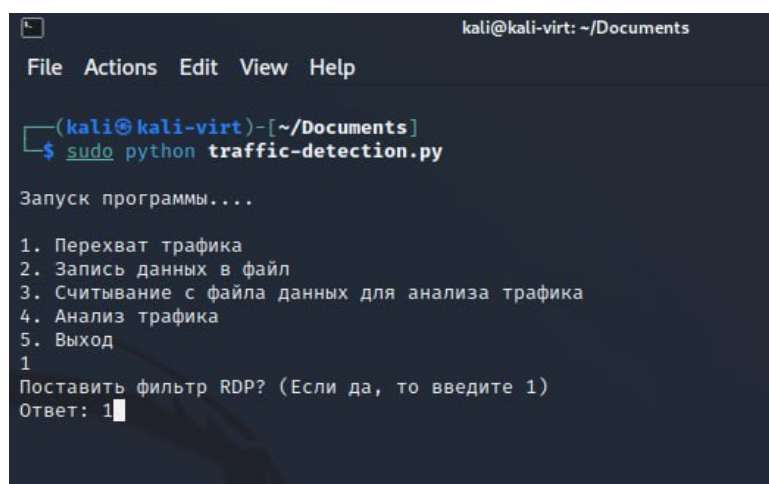
На втором этапе были применены эффективные преобразования признаков, которые внедрены в программу как средства оценки вероятности наличия протокола RDP. Кроме того, проводилась калибровка граничных значений.

Далее в курсовой работе будет представлен обзор программы «traffic-detection.py», в котором будут описаны ее основные возможности. Затем будет подробно рассмотрен каждый из статистических методов анализа сетевого трафика, которые были внедрены в данную программу. В конце работы будут представлены результаты нескольких тестов, проведенных с использованием программы «traffic-detection.py».

3 Программная реализация метода обнаружения RDP-трафика

При запуске программы «traffic-detection.py» пользователю предоставляется выбрать одну из следующих опций:

1. Перехват трафика: при выборе данной опции происходит перехват трафика с помощью sniffера, программного обеспечения, которое анализирует входящий и исходящий трафик с компьютера. Далее пользователю предлагают установить RDP-фильтр при осуществлении перехвата трафика, как показано на рисунке 4.



```
kali@kali-virt: ~/Documents
File Actions Edit View Help
(kali@kali-virt)-[~/Documents]
$ sudo python traffic-detection.py
Запуск программы....
1. Перехват трафика
2. Запись данных в файл
3. Считывание с файла данных для анализа трафика
4. Анализ трафика
5. Выход
1
Поставить фильтр RDP? (Если да, то введите 1)
Ответ: 1
```

Рисунок 4 – Вид консоли при выборе опции «Перехват трафика»

Если ввести в консоль цифру «1», то программа будет выводить информацию только о тех перехваченных пакетах, которые содержат признаки протокола RDP. Если пользователь не вводит никаких цифр и оставляет поле ввода пустым, то в консоли будут отображаться все пакеты, которые перехватывает sniffer. Также пользователю нужно выбрать сетевой интерфейс, по которому производится перехват трафика, как показано на следующем рисунке.

```
kali@kali-virt: ~/Documents
File Actions Edit View Help

(kali@kali-virt)-[~/Documents]
$ sudo python traffic-detection.py

Запуск программы....

1. Перехват трафика
2. Запись данных в файл
3. Считывание с файла данных для анализа трафика
4. Анализ трафика
5. Выход
1
Поставить фильтр RDP? (Если да, то введите 1)
Ответ: 1

Выберите сетевой интерфейс, нажав соответствующую цифру:
[(1, 'lo'), (2, 'eth0'), (3, 'eth1')]
2

Начался процесс захвата трафика ...

█
```

Рисунок 5 – Выбор интерфейса и начало перехвата трафика

Чтобы остановить перехват сетевого трафика, необходимо нажать клавишу «пробел». После завершения перехвата трафика пользователю предлагают ввести название файла, чтобы записать информацию о всех перехваченных пакетах в файл, как показано на рисунке 6.

```
Пакет No1609
Время перехвата: 05:10:2023 21:40:53
Протокол: UDP
MAC-адрес отправителя: C8:02:10:3B:68:0E
MAC-адрес получателя: FF:FF:FF:FF:FF:FF
Отправитель: 192.168.1.112:9956
Получатель: 192.168.1.255:9956
Признаки: Нет;
Вероятность RDP-сессии 0%

Завершение программы ...

Данные собраны. Перехвачено: 1609 пакетов(-а)

Хотите записать перехваченный трафик в файл? (да - нажмите 1)
Ответ: █
```

Рисунок 6 – Завершение перехвата трафика после нажатия клавиши «пробел»

2. Запись данных в файл: если в результате перехвата трафика было захвачено несколько пакетов, то можно записать всю перехваченную информацию в файл, введя имя файла. Добавление этой опции было целью расширения возможностей пользователя по сохранению данных в файл.
3. Считывание с файла для анализа данных: для анализа данных можно использовать опцию считывания информации из файла. Она позволяет извлекать только ту информацию о пакетах, которая была предварительно

записана с помощью программы «traffic-detection.py».

4. Анализ трафика: когда пользователь выбирает данную опцию, программа выводит в консоль информацию о всех возможных сессиях, которые продолжились более 10 секунд в момент перехвата трафика. Обнаружение этих сессий будет описано позже. Выводится также некоторая общая информация о перехваченном трафике, такая как время начала и завершения перехвата трафика, количество пакетов, среднее количество пакетов в секунду и средний размер пакетов. Кроме того, выводится список IP-адресов, участвующих в передаче пакетов по сети, как показано на рисунке 7.

```
Общая информация:
Время первого перехваченного пакета: 25.04.2023 г. 21:11:05
Время последнего перехваченного пакета: 25.04.2023 г. 21:13:09
Количество пакетов: 6595
Общее время перехвата: 124.606 сек
Среднее количество пакетов в секунду: 52.76
Средний размер пакетов: 190.365
Завершить просмотр (нажмите "q" для выхода)
[0 — 149.154.167.41] [1 — 108.177.14.188] [2 — 23.61.216.238] [3 — 192.168.1.202]
[4 — 192.168.1.133] [5 — 192.168.1.112] [6 — 173.194.222.94] [7 — 52.182.141.63]
[8 — 192.168.1.90] [9 — 93.186.225.198] [10 — 20.8.16.139] [11 — 64.233.164.100]
[12 — 192.168.1.156] [13 — 239.255.255.250] [14 — 213.180.193.90] [15 — 192.168.1.1]
[16 — 20.54.37.64] [17 — 87.240.129.186] [18 — 8.8.8.8] [19 — 20.231.121.79]
[20 — 192.168.56.1] [21 — 8.8.4.4] [22 — 192.168.1.255] [23 — 192.168.1.187]
[24 — 192.168.56.255] [25 — 224.0.0.251] [26 — 104.66.124.233] [27 — 224.0.0.113]

Выберите цифру (0 - 27) для просмотра IP-адреса:
4
```

Рисунок 7 – Вывод общей информации о перехваченном трафике

Пользователь может выбрать интересующий его IP-адрес для дальнейшего анализа пакетов, связанных с ним. После выбора IP-адреса пользователю предоставляется выбор конкретного порта, по которому выбранный IP-адрес осуществлял передачу сообщений, как показано на рисунке 8.

```
Завершить просмотр (нажмите "q" для выхода)
[0 — 149.154.167.41] [1 — 108.177.14.188] [2 — 23.61.216.238] [3 — 192.168.1.202]
[4 — 192.168.1.133] [5 — 192.168.1.112] [6 — 173.194.222.94] [7 — 52.182.141.63]
[8 — 192.168.1.90] [9 — 93.186.225.198] [10 — 20.8.16.139] [11 — 64.233.164.100]
[12 — 192.168.1.156] [13 — 239.255.255.250] [14 — 213.180.193.90] [15 — 192.168.1.1]
[16 — 20.54.37.64] [17 — 87.240.129.186] [18 — 8.8.8.8] [19 — 20.231.121.79]
[20 — 192.168.56.1] [21 — 8.8.4.4] [22 — 192.168.1.255] [23 — 192.168.1.187]
[24 — 192.168.56.255] [25 — 224.0.0.251] [26 — 104.66.124.233] [27 — 224.0.0.113]

Выберите цифру (0 - 27) для просмотра IP-адреса:
4
Список портов которые участвовали в соединении с данным IP-адресом
[0 — None] [1 — 57645] [2 — 57610] [3 — 54039]
[4 — 64562] [5 — 65105] [6 — 50305] [7 — 53]
[8 — 50302] [9 — 80] [10 — 50304] [11 — 3389]
[12 — 443] [13 — 50303]

Выберите цифру (0 - 13) для выбора порта:
11
```

Рисунок 8 – Вывод информации о портах относительно конкретного IP-адреса

Затем выводится общая информация только относительно выбранного IP-адреса и порта, такая как время первого и последнего перехваченных

пакетов, где данный IP-адрес выступает в качестве отправителя или получателя. Таким образом можно понять, в какой конкретно момент времени начался обмен информацией с тем или иным IP-адресом.

После вывода общей информации пользователю предоставляется следующий функционал:

- а) Вывод сетевого трафика, где в качестве отправителя или получателя выступает выбранный IP-адрес.
- б) Построение графика отношения объема входящего трафика и исходящего трафика 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.

- в) Построение графика отношения V_{udp} — объема входящего UDP-трафика и V_{tcp} объема входящего TCP-трафика. Отношение рассчитывается по формуле

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

Стоит отметить, что во время RDP-сессии передача пакетов может осуществляться по протоколам UDP и TCP. Хотя в большинстве программ удаленного рабочего стола передача сообщений происходит только по протоколу TCP. Однако существуют до сих пор приложения, которые используют и протокол UDP, и протокол TCP. Например, приложение ОС Windows «Подключение к удаленному рабочему столу» (Remote Desktop Connection, RDC) использует для передачи пакетов по умолчанию оба транспортных протокола. Это сделано для того чтобы оптимизировать передачу данных, обеспечивая надежную доставку управляющих сообщений и минимизируя задержки при передаче потоковых данных.

- г) Построение графика 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 рабо-

чему столу сервер отправляет клиенту ТСП-пакеты с установленным флагом АСК, указывающим, что поле номера подтверждения задействовано. Изменяясь во времени, значение r_{ack} может использоваться для определения активной сессии в определенные моменты времени с помощью графика.

- д) Построение двух графиков, показывающих частоту SYN-флагов и PSH-флагов в ТСП-трафике. Частота SYN-флагов находится по формуле

$$r_{syn} = \frac{V_{Sin}}{V_{tcp}},$$

где V_{Sin} число входящих ТСП-пакетов, в которых установлен флаг SYN = 1, V_{tcp} — число входящих ТСП-пакетов в единицу времени. В процессе установления ТСП-соединения между клиентом и сервером передаются пакеты с флагом SYN, а обмен данными начинается с использованием пакетов без этого флага. Таким образом, количество SYN-флагов, полученных сервером, соответствует числу запросов на соединение, а частота их появления определяет долю служебных пакетов этого типа в ТСП-трафике.

Частота PSH-флагов вычисляется по формуле

$$r_{psh} = \frac{V_{Pin}}{V_{tcp}},$$

где V_{Pin} число входящих ТСП-пакетов, в которых установлен флаг PSH = 1, V_{tcp} — число входящих ТСП-пакетов в единицу времени. Флаг PSH (Push) в ТСП-заголовке используется для указания конечной точке передачи данных о том, что все буферизованные данные должны быть немедленно отправлены получателю, а не ждать буферизации следующих данных. Когда отправитель устанавливает флаг PSH в заголовке ТСП-сегмента, он указывает получателю, что данные в этом сегменте должны быть переданы верхнему уровню протокола немедленно, без буферизации на приемной стороне. Таким образом, если значение величины r_{psh} резко возросло в некоторый промежуток времени, значит за это время одно устройство успело передать другому устройству большое количество пакетов.

- е) Для получения представления о количестве передачи пакетов в сети были построены два графика: один показывает количество входящих

пакетов в единицу времени, а другой — исходящих. Таким образом, эти графики позволяют оценить количество передаваемых пакетов в сети в единицу времени.

- ж) Построение двух графиков, показывающих максимальные размеры входящих и исходящих пакетов в единицу времени. Эти графики показывают, какие максимальные размеры пакетов передаются по сети в каждую секунду.
- з) Последняя опция позволяет пользователю вернуться к выбору другого IP-адреса.

Перед построением каждого графика пользователю предоставляется возможность добавить второй IP-адрес, с которым выбранный IP-адрес взаимодействовал в момент перехвата трафика, как показано на рисунке 9.

```
Выберите цифру (0 - 13) для выбора порта:
11
Общая информация о трафике, связанном с 192.168.1.133
Время первого перехваченного пакета: 25.04.2023 г. 21:11:17
Время последнего перехваченного пакета: 25.04.2023 г. 21:13:01
Общее время: 104 сек.
Количество пакетов: 6021
Среднее количество пакетов в секунду: 57.894
Средний размер пакетов: 127.37
Выберите опцию:
1. Вывести весь трафик, связанный с 192.168.1.133
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. Построить график отображения количества пакетов в единицу времени
7. Построить график отображения максимумов среди пакетов в единицу времени
8. Вернуться к выбору IP-адреса
2
Изобразить на графике еще один объект. Выберите IP-адрес для добавления (введите цифру)
[0 — None] [1 — 192.168.1.156]
□
```

Рисунок 9 – Предоставление пользователю возможности выбрать второй IP-адрес

Если пользователь выбирает второй IP-адрес, появляется новое окно, в котором отображаются данные о двух графиках. В противном случае появляется окно, где изображены данные только об одном ранее выбранном IP-адресе.

5. Выход: при выборе данной опции программа «traffic-detection.py» завершает свою работу.

3.1 Определение активных сессий путем анализа TCP-соединения

Как уже упоминалось ранее при выборе опции «Анализ трафика» появляется информация об активных сессиях, как показано на следующем рисунке.

```
3. Считывание с файла данных для анализа трафика
4. Анализ трафика
5. Выход
4

Было перехвачено 3 сессии(-й)

Информация о сессии #1:
Инициатор подключения: 192.168.1.156
Целевое устройство: 192.168.1.133
Порт подключения: 3389
Время установки соединения: 25.04.2023 г. 21:11:17
Время завершения соединения: 25.04.2023 г. 21:13:01
Общее время соединения: 104.1 сек
Найдена RDP-сессия с вероятностью 100%!!!

Информация о сессии #2:
Инициатор подключения: 192.168.1.133
Целевое устройство: 104.66.124.233
Порт подключения: 80
Время установки соединения: 25.04.2023 г. 21:11:44
Время завершения соединения: 25.04.2023 г. 21:13:02
Общее время соединения: 78.55 сек

Информация о сессии #3:
Инициатор подключения: 192.168.1.133
Целевое устройство: 20.231.121.79
Порт подключения: 80
Время установки соединения: 25.04.2023 г. 21:11:45
Время завершения соединения: 25.04.2023 г. 21:13:00
Общее время соединения: 75.34 сек
```

Рисунок 10 – Вывод информации об активных сессиях

Под активной сессией будем понимать связь между двумя устройствами, в которой происходит обмен данными. В сетевом трафике, активная сессия обычно определяется как установленное соединение между двумя устройствами, которое использует определенный протокол для передачи данных. Активная сессия образуется при установке TCP-соединения. Такой процесс также называют «трехсторонним рукопожатием» (Three-way Handshake). Он состоит из следующих этапов:

1. Клиент отправляет серверу пакет с установленным флагом SYN (Synchronize Sequence Number), который указывает на начало соединения. В этом пакете клиент выбирает начальное значение порядкового номера (sequence number), которое будет использоваться в дальнейшем.
2. Сервер получает пакет с флагом SYN и отвечает на него пакетом с установленными флагами SYN и ACK (Acknowledgment), подтверждая получение запроса на установку соединения и передавая свой sequence number.
3. Клиент получает пакет с флагами SYN и ACK, проверяет подтверждение ACK и отправляет пакет с установленным флагом ACK, подтверждая свою

готовность к соединению и передавая серверу свой *sequence number*.

В момент перехвата трафика программа «*traffic-detection.py*» проверяет каждый пакет TCP на наличие флага SYN. Если пакет содержит флаг SYN, то это значит, что некоторое устройство (инициатор подключения) пытается установить соединение с другим устройством (целевым устройством).

В этот момент программа добавляет в список *Session_list* новый элемент класса «*Session*», в котором хранится следующая информация:

- IP-адреса инициатора подключения и целевого устройствами;
- время перехвата данного пакета;
- порт получателя, на который осуществляется попытка TCP-соединения;
- начальное значение *sequence number*.

Далее программа проверяет каждый элемент списка *Session_list* на наличие последующих пакетов TCP с флагом ACK (*Acknowledgment*). Если пакет содержит флаг ACK и флаг SYN, а также если IP-адрес получателя равен IP-адресу инициатора подключения, IP-адрес отправителя равен IP-адресу целевого устройства, порт отправителя равен порту, сохраненному в текущем элементе *Session_list*, и значение номера подтверждения (*acknowledgment number*) равно значению *sequence number*, увеличенному на единицу, тогда целевое устройство пытается подтвердить запрос на установку TCP-соединения. В случае успешного подтверждения, информация о значении *sequence number* в текущей сессии обновляется, а также добавляется информация о значении *acknowledgment number* перехваченного пакета.

На следующем рисунке показано, что программе удалось перехватить два последовательно идущих пакета, где одно инициатор подключения делает запрос на подключение к целевому устройству.

```
-----Пакет No8-----
Время перехвата: 05:10:2023 21:26:03
Протокол: TCP
MAC-адрес отправителя: 08:00:27:60:30:4A
MAC-адрес получателя: 08:00:27:7C:A4:D5
Отправитель: 192.168.56.107:49679
Получатель: 192.168.56.109:3389
Порядковый номер: 3215948962; Номер подтверждения: 0
SYN:1; ACK:0; PSH:0; RST:0; FIN:0

Признаки: Установка соединения (SYN);
Вероятность RDP-сессии 0%

-----Пакет No9-----
Время перехвата: 05:10:2023 21:26:03
Протокол: TCP
MAC-адрес отправителя: 08:00:27:7C:A4:D5
MAC-адрес получателя: 08:00:27:60:30:4A
Отправитель: 192.168.56.109:3389
Получатель: 192.168.56.107:49679
Порядковый номер: 489028548; Номер подтверждения: 3215948963
SYN:1; ACK:1; PSH:0; RST:0; FIN:0

Признаки: Подтверждение установки соединения (SYN-ACK);
Вероятность RDP-сессии 0%

-----Пакет No10-----
```

Рисунок 11 – Сообщение о перехвате пакета с установлением нового TCP-соединения

Когда запрос на установку соединения получен и подтвержден, программа ищет TCP-пакет с sequence number, равным acknowledgment number, сохраненному на предыдущем этапе, и acknowledgment number текущего пакета, равным sequence number + 1, сохраненному также на предыдущем этапе. Это действие означает, что инициатор подключения готов к соединению, и можно считать, что соединение установлено.

Когда обе стороны передали все необходимые данные и произошел обмен подтверждениями о получении последних пакетов данных, TCP-соединение считается завершенным. В таких пакетах обычно устанавливается флаг завершения FIN и флаг подтверждения ACK. Если в пакетах установлен флаг сброса RST и флаг подтверждения ACK, то TCP-соединение также может быть прервано. При прохождении по всем незавершенным сессиям, если программа «traffic-detection.py» находит такой пакет, в котором помимо установленного флага подтверждения ACK установлен либо флаг FIN, либо флаг RST, то она считает текущую сессию завершенной и рассчитывает общее время данной сессии. Если сессия продлилась менее 10 секунд, она удаляется из списка *Session_list*. В противном случае она остается в списке для дальнейшего анализа трафика.

На рисунках 12-13 изображены перехваченной программой пакеты, уведомляющие целевое устройство о завершении или прерывании активной сессии.

```

Пакет No8213
Время перехвата: 05:10:2023 22:01:02
Протокол: TCP
MAC-адрес отправителя: 08:00:27:BA:DE:D8
MAC-адрес получателя: 08:00:27:60:30:4A
Отправитель: 192.168.1.147:57256
Получатель: 192.168.1.133:3389
Порядковый номер: 869329320; Номер подтверждения: 785973941
SYN:0; ACK:1; PSH:1; RST:0; FIN:0

Признаки: Ведется сессия; Подозрение на RDP-сессию!; Передача клавиатурных и мышинных событий;
Вероятность RDP-сессии 100%

Пакет No8214
Время перехвата: 05:10:2023 22:01:02
Протокол: TCP
MAC-адрес отправителя: 08:00:27:BA:DE:D8
MAC-адрес получателя: 08:00:27:60:30:4A
Отправитель: 192.168.1.147:57256
Получатель: 192.168.1.133:3389
Порядковый номер: 869329351; Номер подтверждения: 785973941
SYN:0; ACK:1; PSH:0; RST:0; FIN:1

Признаки: Подозрение на RDP-сессию!; Сессия закончена; Передача клавиатурных и мышинных событий;
Вероятность RDP-сессии 100%

```

Рисунок 12 – Перехват пакета с установленным FIN-флагом

```

Пакет No1336
Время перехвата: 05:10:2023 21:27:23
Протокол: TCP
MAC-адрес отправителя: 08:00:27:60:30:4A
MAC-адрес получателя: 08:00:27:7C:A4:D5
Отправитель: 192.168.56.107:49682
Получатель: 192.168.56.109:3389
Порядковый номер: 10912734; Номер подтверждения: 2789913270
SYN:0; ACK:1; PSH:1; RST:0; FIN:0

Признаки: Ведется сессия; Подозрение на RDP-сессию!; Передача клавиатурных и мышинных событий;
Вероятность RDP-сессии 100%

Пакет No1337
Время перехвата: 05:10:2023 21:27:23
Протокол: TCP
MAC-адрес отправителя: 08:00:27:60:30:4A
MAC-адрес получателя: 08:00:27:7C:A4:D5
Отправитель: 192.168.56.107:49682
Получатель: 192.168.56.109:3389
Порядковый номер: 10912743; Номер подтверждения: 2789913270
SYN:0; ACK:1; PSH:0; RST:1; FIN:0

Признаки: Подозрение на RDP-сессию!; Сессия прервана; Передача клавиатурных и мышинных событий;
Вероятность RDP-сессии 100%

```

Рисунок 13 – Перехват пакета с установленным RST-флагом

На рисунках помимо признаков TCP-соединения также отображаются некоторые сообщения, связанные с RDP-сессией. Подробнее об этих сообщениях будет рассказано в следующих разделах. Однако перед этим необходимо изучить проблему выявления признаков протокола RDP.

3.2 Обработка данных и построение графиков для анализа поведения RDP-трафика

Тестирование программы происходило на нескольких виртуальных машинах (ВМ), имеющие разные операционные системы. Использовались операционные системы Windows 10 Professional версии 21H2 и Kali Linux 2022.4 Release. В дальнейшем данные операционные системы будем обозначать как Win и Kali соответственно. В эксперименте всегда принимали участие три виртуальные машины. Программа «traffic-detection.py» запускалась на третьей ВМ, а между первыми двумя ВМ устанавливалось соединение по протоколу RDP.

Рассматривались следующие соединения:

- Соединение Win - Win: устанавливалось соединение между двумя VM Windows 10 с помощью приложения «Подключение к удаленному рабочему столу»;
- Соединение Win - Kali: производилось подключение к Kali Linux с помощью приложения «Подключение к удаленному рабочему столу». Для осуществления такого подключения на Kali Linux запускался сервис XRDP, бесплатный протокол удаленного доступа, основанный на протоколе RDP (Microsoft Remote Desktop);
- Соединение Kali - Win: для подключения к Windows 10 был использован клиент удаленного рабочего стола Remmina.;
- Соединение Kali - Kali: подключение к Kali Linux совершалось с помощью клиента удаленного рабочего стола Remmina.

Это было сделано для того, чтобы проанализировать процесс подключения по протоколу RDP между различными операционными системами. Ведь при реальной атаке вероятность того, что операционные системы будут одинаковыми, крайне мала. Далее будут рассмотрены статистические методы анализа сетевого трафика, которые были выявлены в результате анализа данных различных типов соединений.

4 Анализ распределения размера пакетов

Размеры пакетов RDP-трафика обычно больше, чем у других типов трафика, что может быть использовано для обнаружения RDP-сессий. Однако, необходимо учитывать, что размеры пакетов могут варьироваться в зависимости от многих факторов, таких как тип передаваемой информации, настройки сети и протокола передачи, а также особенности конфигурации клиента и сервера RDP.

Тем не менее, можно предположить, что большинство пакетов RDP будут иметь относительно постоянный размер в течение сессии, особенно для передачи графических данных. Это может быть использовано для определения наличия активной RDP-сессии.

Например, можно рассчитать средний размер пакета для определенного временного интервала и определить, отличается ли этот размер от среднего значения для других протоколов. Также можно рассчитать стандартное отклонение размеров пакетов и определить, есть ли значительные отклонения от этого значения для определенного интервала времени, что может указывать на активную RDP-сессию.

Однако, стоит отметить, что использование только распределения размеров пакетов не может дать полной уверенности в том, что происходит передача RDP-трафика, так как размеры пакетов могут быть изменены в разных версиях протокола, и могут использоваться другими протоколами с похожими размерами пакетов. Поэтому, рекомендуется использовать этот метод в сочетании с другими методами обнаружения RDP-трафика.

4.1 Вычисление среднего значения и стандартного отклонения размеров пакетов

Программа «traffic-detection.py» анализирует все активные сессии каждые 5 секунд. В каждом таком интервале времени вычисляется среднее значение размера пакетов по формуле:

$$\mu = \frac{1}{n} \sum_{i=1}^n p_{s_i},$$

где n — количество пакетов, перехваченных за интервал времени в 5 секунд, p_{s_i} ($1 \leq i \leq n$) — размер каждого пакета.

Для расчета стандартного отклонения размеров пакетов была использована следующая формула:

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (p_{s_i} - \mu)^2},$$

где n — количество пакетов, p_{s_i} ($1 \leq i \leq n$) — размер каждого пакета, μ — среднее значение размеров пакетов.

4.2 Определение верхней и нижней границ диапазона значений размеров пакетов для каждого интервала времени

Рассчитав среднее значение и стандартное отклонение в пятисекундный интервал времени, программа делает следующие операции:

1. Производится определение верхней (ВГ) и нижней (НГ) границы диапазона значений размеров пакетов, в котором должно находиться большинство пакетов для этого интервала времени. Эти границы могут быть определены путем добавления или вычитания отклонения от среднего значения размеров пакетов, к верхней или нижней границе. В данном случае $НГ = (\mu - 4\sigma)$ и $ВГ = (\mu + 4\sigma)$.
2. Проверяется каждый размер пакета в интервале времени на соответствие этим границам. Если размер пакета выходит за пределы этого диапазона значений ($p_{s_i} < (\mu - 4\sigma)$ или $p_{s_i} > (\mu + 4\sigma)$ ($1 \leq i \leq n$)), то это может указывать на наличие активной RDP-сессии.
3. Программа определяет наличие пакетов с аномальными размерами, характерными для признаков RDP-сессии, на основе того, удовлетворяют ли более 60% перехваченных пакетов вышеописанным условиям в определенном интервале времени.

4.3 Анализ полученных данных на наличие признаков RDP-сессии

Стоит отметить, что выбор коэффициента, множителя для стандартного отклонения, выбирался, на основе соединений, в каждом из которых производилась установка RDP-сессии.

График на рисунке 14 отображает максимальное значение пакетов, рассчитанных за единицу времени, при использовании соединения Win-Win. На этом графике представлены только максимальные значения, которые были рассчитаны на основе пакетов, относящихся к протоколу RDP. По этому графику можно сделать несколько выводов:

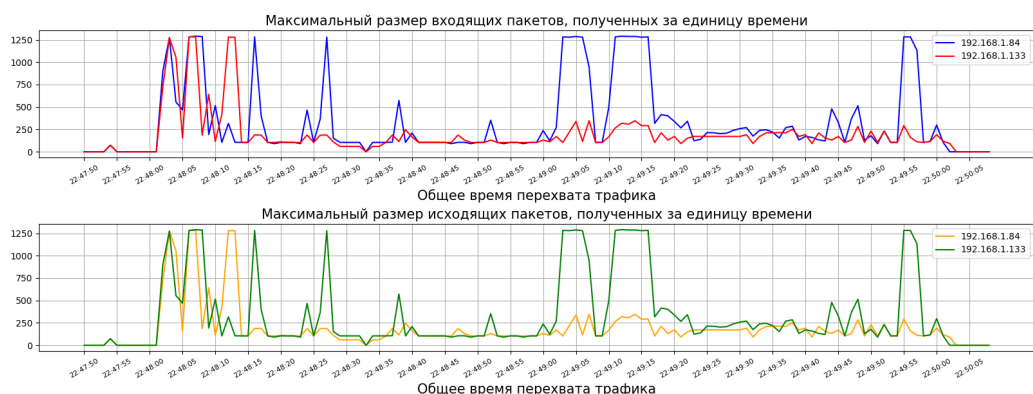


Рисунок 14 – График отображения максимумов среди пакетов, рассчитанных в единицу времени (соединение Win - Win)

- максимальный размер таких пакетов не превышает 1300 байт;
- Обычно, по количеству пакетов, переданных между устройствами, можно определить инициатора подключения и целевое устройство. В данном случае инициатором подключения является устройство с IP-адресом 192.168.1.84, а целевым устройством - устройство с IP-адресом 192.168.1.133, так как первое получило большее количество пакетов с максимальными размерами байт;
- В промежуток времени между 22:48:00 и 22:48:15 размеры пакетов инициатора подключения и целевого устройства достигают максимального размера байт в тот момент, когда происходит этап процесса аутентификации и защиты передаваемых данных (обмен сертификатами). Этот этап происходит в начале установления соединения и позволяет клиенту и серверу проверить подлинность друг друга и договориться о параметрах безопасности соединения. Такой обмен, когда размеры пакетов целевого устройства достигают максимума, замечен только при подключении между двумя VM Windows 10.

Также можно сделать аналогичные выводы по остальным соединениям из рисунков 15 - 17. Графики показывают, что максимальные размеры пакетов в других соединениях значительно отличаются от соединения Win - Win. Кроме того, в момент, когда происходит этап процесса аутентификации и защиты передаваемых данных, не наблюдается такого явного обмена пакетами.

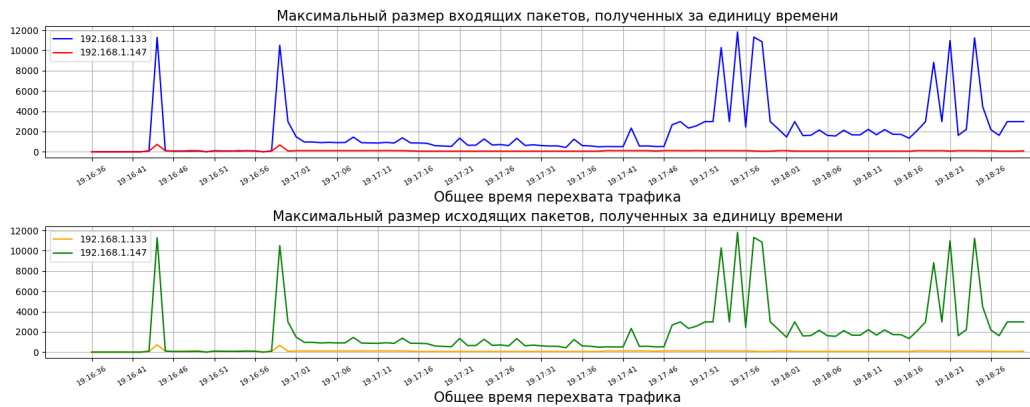


Рисунок 15 – График отображения максимумов среди пакетов в единицу времени (соединение Win - Kali)



Рисунок 16 – График отображения максимумов среди пакетов в единицу времени (соединение Kali - Win)

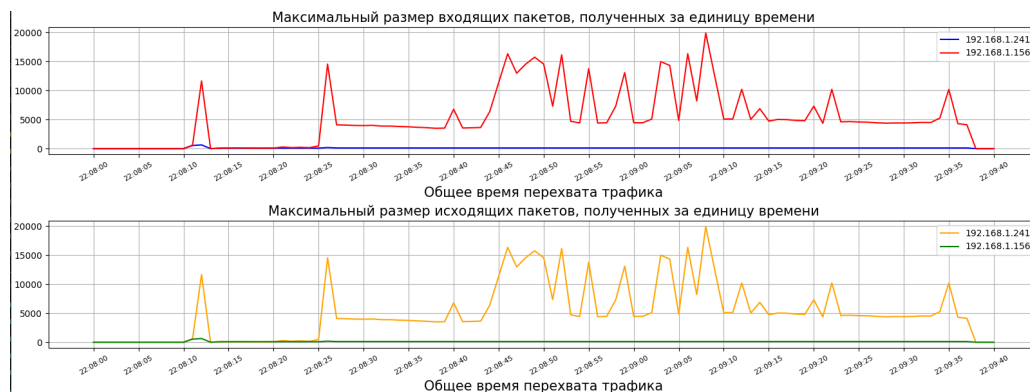


Рисунок 17 – График отображения максимумов среди пакетов в единицу времени (соединение Kali - Kali)

На следующем рисунке показано одно из подключений по SSH, в котором можно заметить аномальные размеры пакетов только в самом начале подключения. В последующих интервалах времени размеры пакетов не выходят за пределы НГ и ВГ, поэтому программе в данном случае удастся различить протоколы RDP и SSH.

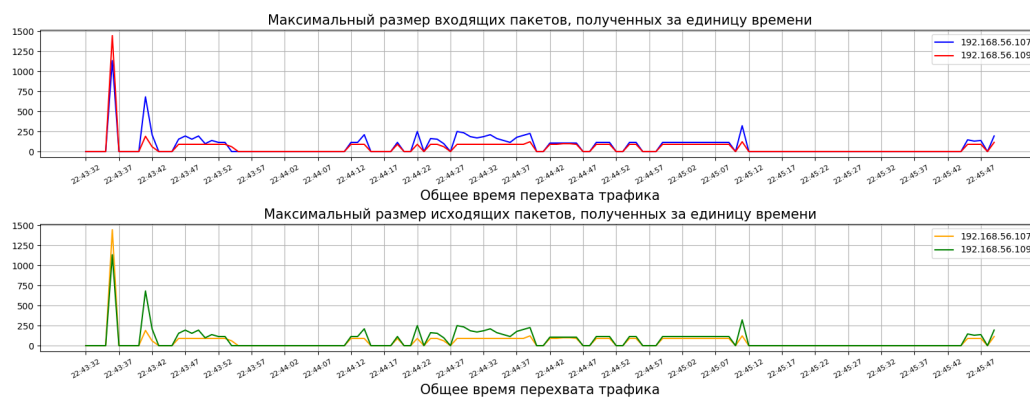


Рисунок 18 – График отображения максимумов среди пакетов в единицу времени (подключение по SSH)

Исходя из вышеописанных рассуждений, именно таким образом программа «traffic-detection.py» проводит анализ распределения пакетов.

5 Анализ распределения временных интервалов между пакетами

Анализ распределения временных интервалов между пакетами может быть полезен для обнаружения RDP-сессий. Обычно временные интервалы между пакетами RDP-трафика меньше, чем между пакетами других типов трафика. Это связано с тем, что RDP-протокол предназначен для передачи данных в режиме реального времени и требует высокой скорости передачи данных для обеспечения плавной работы удаленного рабочего стола. Поэтому, если на сети обнаруживается высокая частота пакетов с маленькими временными интервалами, это может быть признаком активной RDP-сессии. Однако следует учитывать, что также могут быть и другие типы трафика, которые также используют высокую скорость передачи данных и могут иметь маленькие временные интервалы между пакетами, поэтому этот метод должен использоваться вместе с другими методами обнаружения RDP-трафика.

5.1 Вычисление среднего значения и стандартного отклонения интервалов

Расчет временных интервалов программа «traffic-detection.py» делает для каждой активной сессии. Она запоминает время предыдущего пакета t_{prev} и находит разность текущего (t_{cur}) перехваченного пакета и предыдущего ($t_{cur} - t_{prev}$). Каждые пять секунд программа вычисляет среднее значение интервалов времени по формуле:

$$\mu = \frac{1}{n} \sum_{i=1}^{n-1} t_i,$$

где n — количество пакетов, перехваченных за интервал времени в 5 секунд, t_i ($1 \leq i \leq n - 1$) — интервал времени между двумя последовательно идущими пакетами.

Для расчета стандартного отклонения размеров пакетов была использована следующая формула:

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n-1} (t_i - \mu)^2},$$

где n — количество пакетов, перехваченных за интервал времени в 5 секунд, t_i ($1 \leq i \leq n - 1$) — интервал времени между двумя последовательно идущими пакетами, μ — среднее значение интервалов времени.

5.2 Определение пороговых значений для интервалов

После того как были рассчитаны среднее значение и стандартное отклонение в пятисекундный интервал времени, программа производит следующие операции:

1. Для определения верхней (ВГ) и нижней (НГ) границ диапазона значений временных интервалов пакетов используется метод добавления или вычитания отклонения от среднего значения интервалов времени к верхней или нижней границе. В данном случае, НГ и ВГ определяются как $НГ = (\mu - \frac{5}{9}\sigma)$ и $ВГ = (\mu + \frac{5}{9}\sigma)$.
2. В пятисекундном интервале времени проверяется каждый $t_i (1 \leq i \leq n - 1)$ на соответствие этим границам. Если некоторый интервал времени выходит за пределы этого диапазона значений ($t_i < (\mu - \frac{5}{9}\sigma)$ или $t_i > (\mu + \frac{5}{9}\sigma)$ ($1 \leq i \leq n - 1$)), то это может указывать на наличие активной RDP-сессии.
3. Программа определяет наличие пакетов с аномальными временными интервалами, характерными для признаков RDP-сессии, на основе того, удовлетворяют ли более 50% перехваченных пакетов вышеописанным условиям в определенном интервале времени.

5.3 Анализ полученных данных на наличие признаков RDP-сессии

Важно отметить, что выбор коэффициента $\frac{5}{9}$, множителя для стандартного отклонения, был сделан на основе соединений, в каждом из которых была установлена RDP-сессия. В большинстве случаев стандартное отклонение оказывалось больше среднего значения интервалов времени. Это означает, что значения разбросаны вокруг среднего значения более широко, чем при более низком стандартном отклонении. Почти все интервалы времени выходили за пределы значений НГ и ВГ. После небольшого подбора был найден коэффициент, равный $\frac{5}{9}$, который позволил программе во всех типах соединения обнаруживать маленькие временные интервалы, которые могут быть признаками RDP-сессии.

Однако нельзя полностью полагаться на данный метод, так как высокая частота пакетов может также являться признаком каких-либо других протоколов, например, HTTP или HTTPS. Поэтому рассматривать его отдельно не имеет смысла.

6 Анализ частоты флагов PSN

Флаг PSN (Push) используется в протоколах удаленного рабочего стола, включая RDP и VNC. Этот флаг устанавливается в TCP-заголовке и сообщает получающей стороне, что передаваемые данные должны быть немедленно переданы приложению-получателю без буферизации на стороне получателя. Флаг PSN часто используется в протоколах, которые используют потоковую передачу данных, таких как терминальные протоколы или удаленный рабочий стол, чтобы уменьшить задержки в передаче данных и улучшить отзывчивость приложения.

В протоколе RDP флаг PSN может использоваться для передачи клавиатурных и мышиных событий с клиента на сервер, а также для отправки команд и получения ответов на них. Он также может использоваться для передачи буферизованных изображений и звуковых данных.

Таким образом, рассчитывая частоту флагов PSN в каждом интервале времени можно обнаружить RDP-сессии.

6.1 Расчет частоты флагов PSN для каждого интервала времени

Каждые 5 секунд, программа считает TCP-пакеты и TCP-пакеты с установленным флагом PSN. По завершении 5 секунд были получены две величины: V_{Pin} — объем входящего трафика с установленным флагом PSN и V_{tcp} — число входящих TCP-пакетов в пятисекундный интервал времени.

Таким образом, частота PSN-флагов равна:

$$r_{psh} = \frac{V_{Pin}}{V_{tcp}}$$

Вместе с этим программа смотрит на значения частоты PSN-флагов, рассчитанные в предыдущие интервалы времени. Посчитав их среднее значение μ_{psh} , программа проверяет следующие условия: если $cur_{psh} > 0$ и $|\mu_{psh} - cur_{psh}| < 0.3$, где cur_{psh} — текущий интервал времени, то в данный момент совершаются клавиатурные или мышиные события.

6.2 Анализ полученных данных на наличие признаков RDP-сессии

Просматривая каждый TCP-пакет, программа проверяет наличие установленного флага PSN в тех пакетах, где IP-адрес получателя является целевым устройством. Т.е. программа пытается анализировать тот момент, когда инициатор подключения отправляет TCP-пакеты с установленным PSN-флагом.

На рисунках 19 - 22 показаны графики, на которых изображена частота PSH-флагов при установке RDP-сессии в различных типах соединения. В данном случае инициаторами подключения являются устройства, показанные желтым цветом, а целевые устройства — зеленым цветом. Главной задачей программы являлось нахождение таких интервалов времени в которых среднее значение частоты флагов будет примерно одинаково и больше нуля.



Рисунок 19 – График частоты PSH флагов (соединение Win - Win)

На примере рисунка 20 можно легко увидеть, когда пользователь взаимодействовал с мышью или клавиатурой, а когда оставался бездействующим. Например, на промежутке между 19:17:16 и 19:17:37 не было зафиксировано движений мыши или нажатий клавиш на клавиатуре, тогда как на остальных временных отрезках пользователь совершал какие-либо действия. Таким образом, если в определенный момент времени частота PSH-флагов равна нулю, можно сделать вывод, что в это время никаких действий не происходило.



Рисунок 20 – График частоты PSH флагов (соединение Win - Kali)



Рисунок 21 – График частоты PSH флагов (соединение Kali - Win)

Однако, при соединении двух ВМ Kali программа будет выдавать ложные срабатывания в момент бездействия пользователя. На промежутке времени между 22:08:25 и 22:08:40 пользователь не совершал никаких действий при

активной RDP-сессии, и из рисунка 22 видно, что частота флагов PSH не равна нулю. В этом временном интервале передается фиксированное количество PSH-флагов.

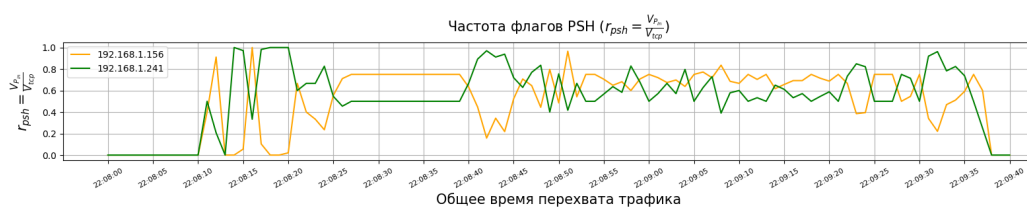


Рисунок 22 – График частоты PSH флагов (соединение Kali - Kali)

В данном случае из-за такой особенности типа соединения Kali - Kali программа «traffic-detection.py» не сможет точно определить взаимодействия с мышью и клавиатурой.

7 Некоторые модификации для улучшения обнаружения RDP-сессии

Стоит отметить, что из вышеописанных статистических методов анализа сетевого трафика самым ненадежным оказался анализ временных интервалов между пакетами. Данный метод постоянно выдавал ложные срабатывания, так как программа находила маленькие интервалы времени в других протоколах, не похожих на RDP. После целого ряда различных тестирований была придумана небольшая модификация, которая должна работать в совокупности с этим ненадежным методом. Она заключается в нахождении отношения объема входящего трафика и исходящего трафика в единицу времени.

На рисунках 23 - 26 показаны графики отношения входящего и исходящего трафика, рассчитанные при активной RDP-сессии в четырех типах соединения. Инициаторами подключения являются устройства, показанные синим цветом, а целевые устройства — оранжевым цветом.



Рисунок 23 – График отношения объема входящего и исходящего трафика (соединение Win - Win)

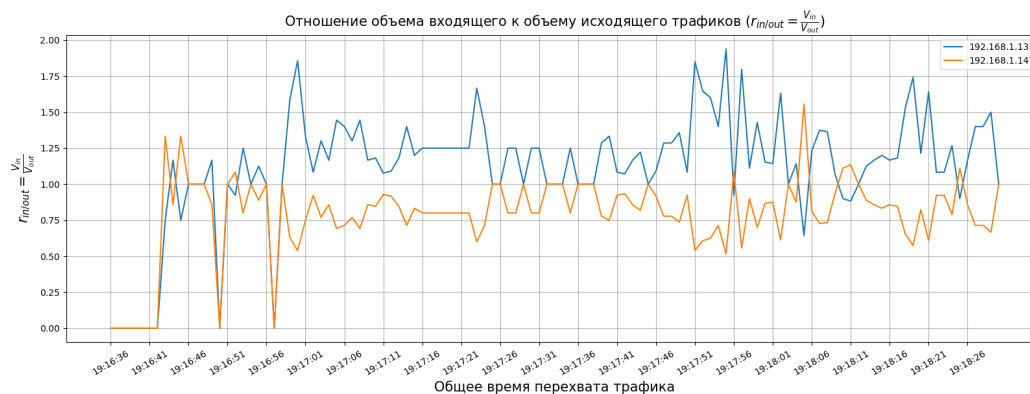


Рисунок 24 – График отношения объема входящего и исходящего трафика (соединение Win - Kali)

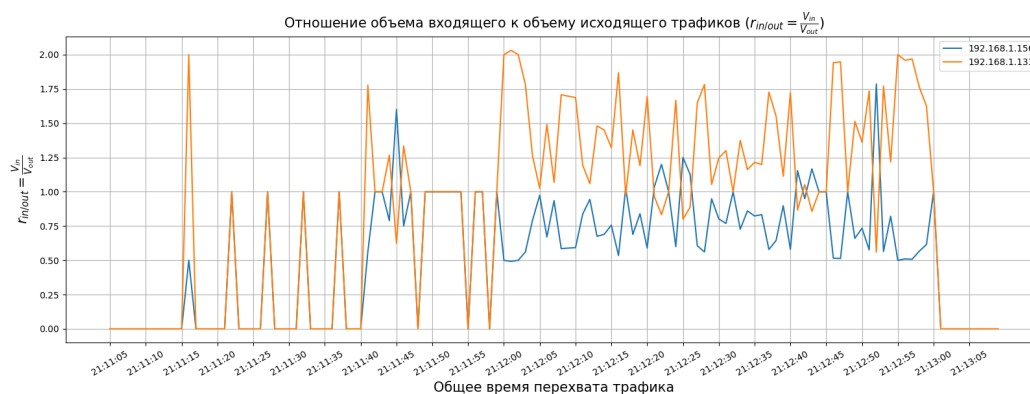


Рисунок 25 – График отношения объема входящего и исходящего трафика (соединение Kali - Win)

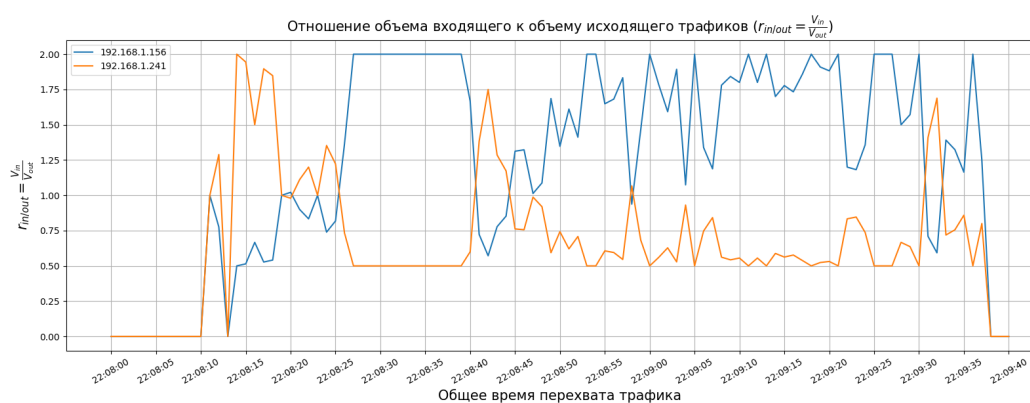


Рисунок 26 – График отношения объема входящего и исходящего трафика (соединение Kali - Kali)

Как можно заметить из рисунков 23 - 26, то значения отношения входящего и исходящего трафика при активной RDP-сессии находятся в основном между 0.5 и 2.0. Конечно, нельзя однозначно утверждать, что такой диапазон значений характерен только для протокола RDP, например, на следующем рисунке показан график отношения объема входящего и исходящего трафиков при активной SSH-сессии. Из рисунка 27 видно, что в некоторые промежутки времени значения попадают в промежуток [0.5, 2.0]. Однако, метод анализа временных интервалов между пакетами в этом случае не будет реагировать на данные значения так как при подключении по SSH наблюдается низкая скорость передачи пакетов, и программа «traffic-detection.py» посчитает, что здесь нет никаких признаков RDP-сессии.

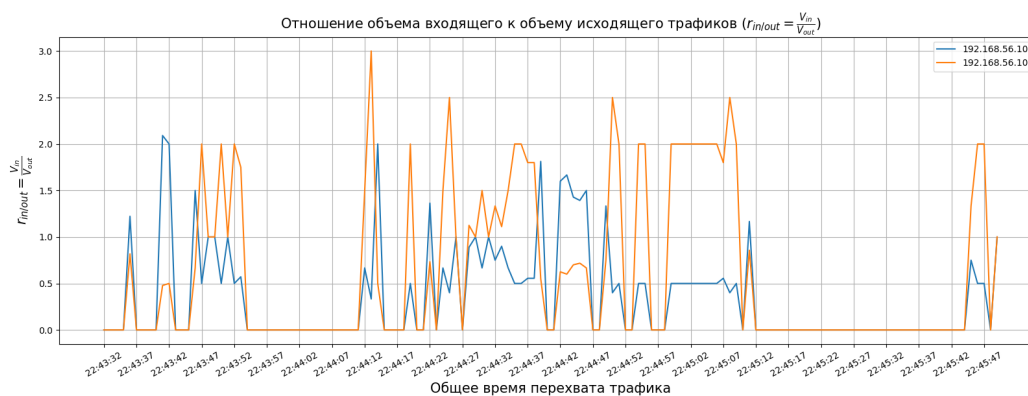


Рисунок 27 – График отношения объема входящего и исходящего трафика (подключение по SSH)

На рисунке изображен график соотношения объема входящего и исходящего HTTP-трафика. При использовании метода анализа временных интервалов между пакетами программа «traffic-detection.py» в некоторых случаях может неправильно интерпретировать временные интервалы и считать их значением, превышающим пороговые значения, что может быть ошибочно расценено как признак RDP-сессии, хотя на самом деле происходит перехват HTTP-трафика. Однако в таких ситуациях нахождение соотношения входящего и исходящего трафика может помочь различить протоколы RDP, HTTP и HTTPS.

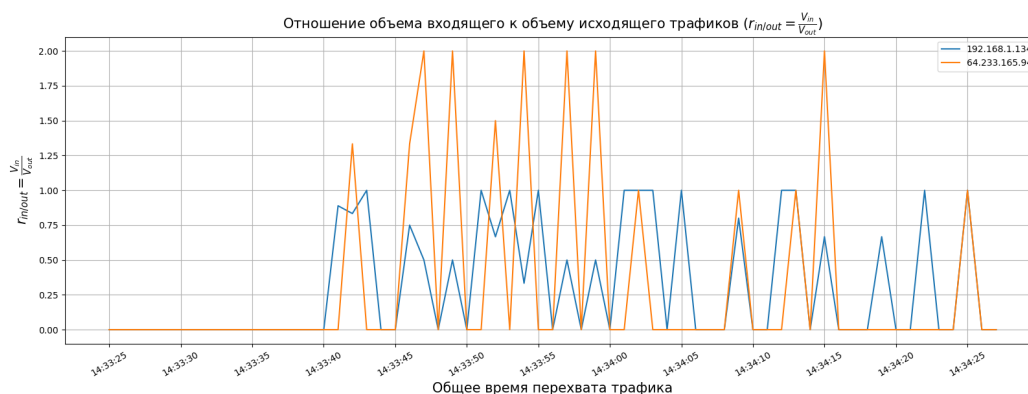


Рисунок 28 – График отношения объема входящего и исходящего трафика (подключение по HTTP)

Таким образом отношение входящего к исходящему трафикам дополняет метод анализа временных интервалов между пакетами.

На протяжении всей активной сессии программа «traffic-detection.py» считает количество пакетов входящего и исходящего трафика инициатора подключения и целевого устройства. Когда проходит одна секунда с момента подсчета пакетов, программа находит отношения по следующим формулам:

$$r_{init} = \frac{V_{i_{dest}}}{V_{i_{src}}},$$

где $V_{i_{dest}}$ и $V_{i_{src}}$ — объемы входящего и исходящего трафика инициатора, рассчитанные в единицу времени.

$$r_{targ} = \frac{V_{t_{dest}}}{V_{t_{src}}},$$

где $V_{t_{dest}}$ и $V_{t_{src}}$ — объемы входящего и исходящего трафика целевого устройства, рассчитанные в единицу времени.

Каждые пять секунд производится анализ состояния сети, где вычисляется средние значения величин r_{init_k} и r_{targ_k} ($1 \leq k \leq 5$)

$$\mu_{init} = \frac{1}{5} \sum_{k=1}^5 r_{init_k},$$

где r_{init_k} ($1 \leq k \leq 5$) — отношения входящего и исходящего трафика инициатора подключения.

$$\mu_{targ} = \frac{1}{5} \sum_{k=1}^5 r_{targ_k},$$

где r_{init_k} ($1 \leq k \leq 5$) — отношения входящего и исходящего трафика целевого устройства.

Далее программа проверяет следующее: если значения $\mu_{init} \in (1, 2)$ и $\mu_{targ} \in [0.5, 1)$ или $\mu_{targ} \in (1, 2)$ и $\mu_{init} \in [0.5, 1)$, а также $|\mu_{init} - \mu_{targ}| \in (0.2, 1.8)$, значит на данном временном интервале возможно наличие признаков RDP-сессии.

И если на этом же пятисекундном интервале анализ временных интервалов между пакетами показал положительный результат, то здесь действительно наблюдается RDP-сессия.

8 Тестирование программы на определение наличия или отсутствия RDP-сессии

Тестирование заключалось в том, что запускались три виртуальные машины, на первых двух создавалась активная RDP-сессия, а на третьей ВМ запускалась программа «traffic-detection.py» в режиме перехвата трафика.

На следующем рисунке показано подключение по протоколу RDP при соединении Kali - Win. На ВМ Kali (192.168.1.147) был установлен клиент удаленного рабочего стола Remmina, с помощью которого было совершено соединение с ВМ Windows 10 (192.168.1.133).

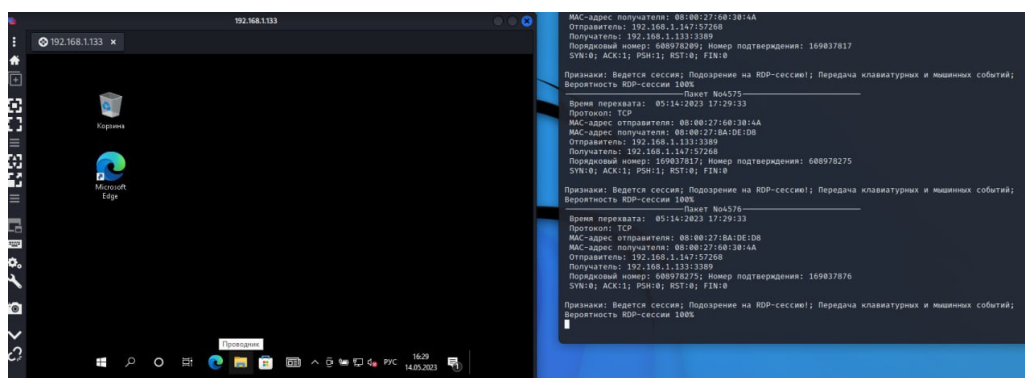


Рисунок 29 – Работа программы при установке RDP-сессии (соединение Kali - Win)

Из рисунка 29 видно, что подключение совершалось по порту 3389, который используется протоколом RDP по умолчанию, и программа смогла однозначно определить наличие в сети активной RDP-сессии. Можно заметить в «признаках» сообщение «Передача клавиатурных и мышиных событий». Это результат работы метода анализа частоты PSH-флагов. В программе вероятность считается исходя из результатов состояния сети, сделанных на предыдущих временных интервалах относительно данной сессии. Однако здесь очень простой случай, так как программа обнаружила стандартный RDP-порт.

Рассмотрим следующий пример, где установка RDP-сессии происходит на другой порт. На рисунке 30 изображено изменение значения стандартного RDP-порта на 13389. Это операция делается в редакторе реестра Windows. Информацию о том, как это можно сделать описана в документации Microsoft [9].

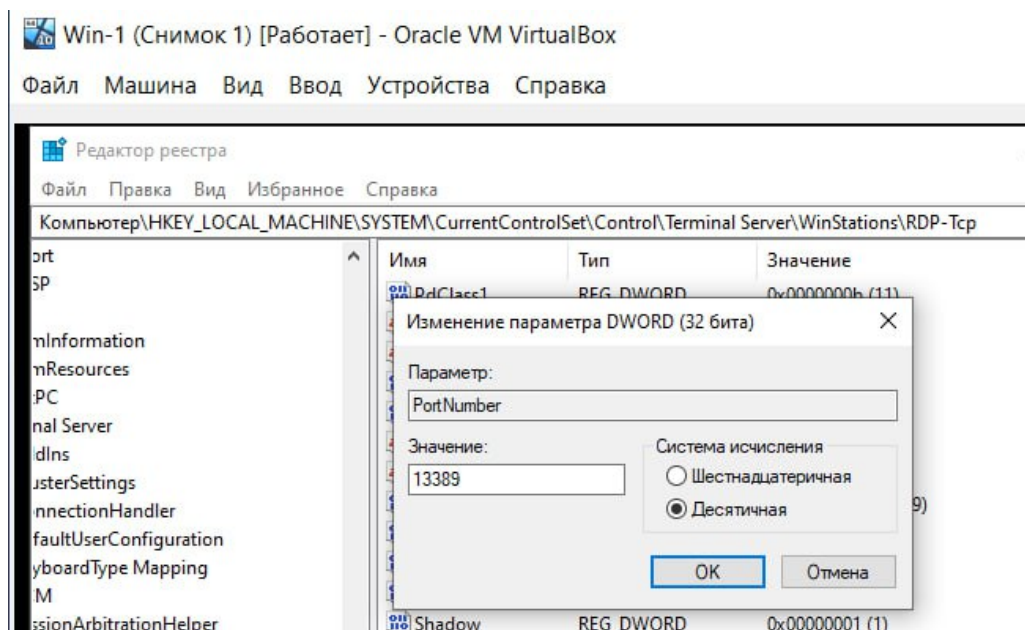


Рисунок 30 – Изменение номера порта по умолчанию на порт 13389

На следующем рисунке показано, что на третьей ВМ был начат перехват трафика с использованием фильтра RDP. Данное условие подразумевает то, что в консоль будет выводиться информация только о тех пакетах которые несут в себе признаки RDP.

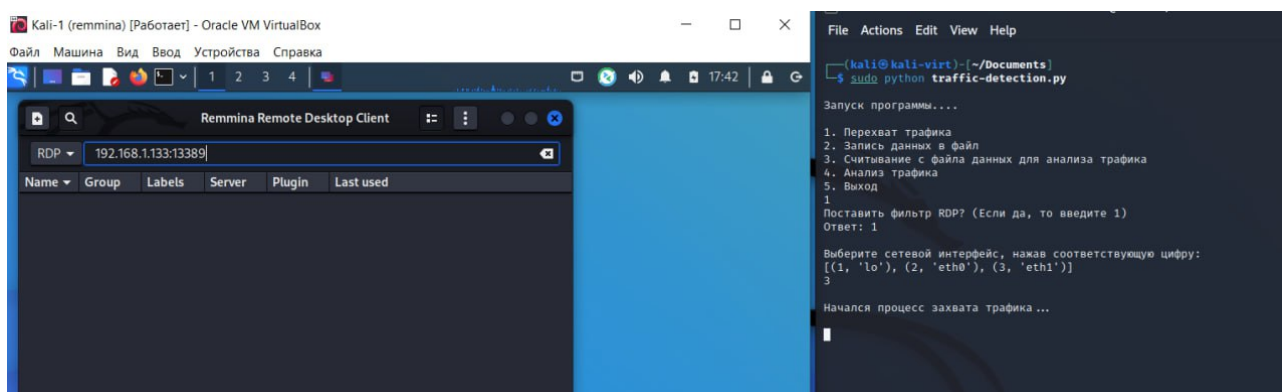


Рисунок 31 – Установка RDP-сессии по порту 13389

Спустя 25 секунд после установки RDP-сессии программа посчитает количество таких временных интервалов, в которых были найдены признаки RDP-сессии. Если таких временных интервалов окажется больше 50%, то программа будет выводить информацию о текущих пакетах.

На рисунке 32 показана вероятность RDP-сессии на текущий момент времени.

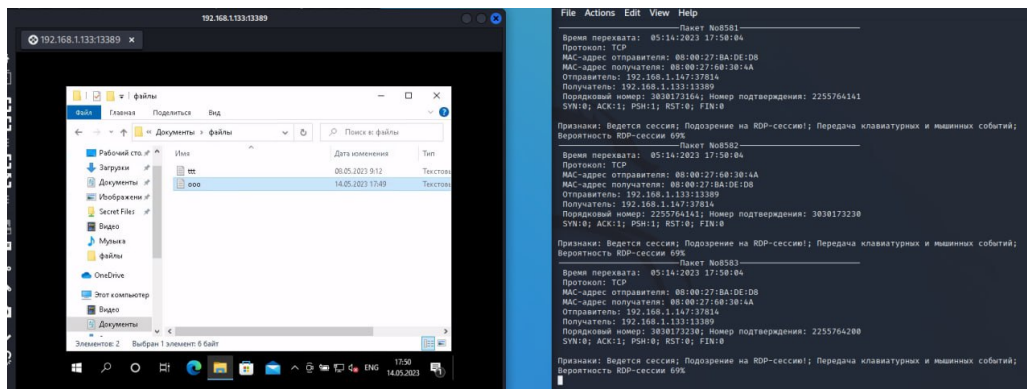


Рисунок 32 – Демонстрация работы программы при активной RDP-сессии

А из следующего рисунка видно, что спустя несколько секунд после установки соединения, вероятность RDP-сессии увеличилась. Это связано с тем, что за некоторый интервал времени производились взаимодействия с мышкой и клавиатурой.

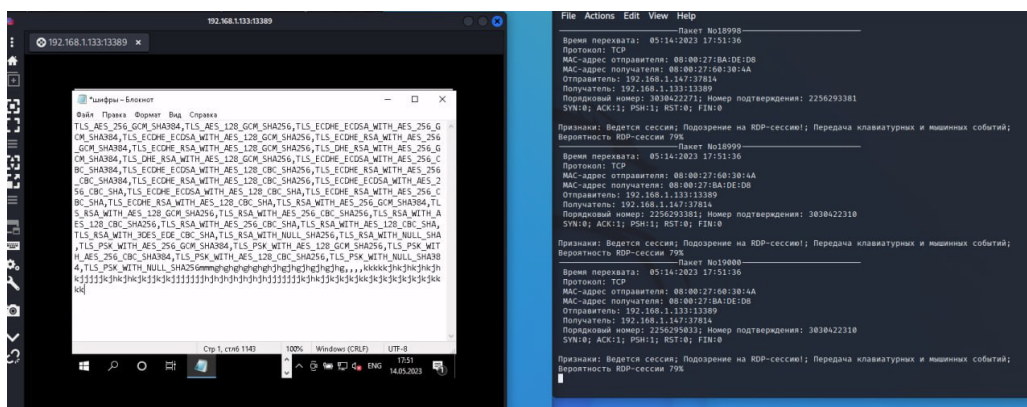


Рисунок 33 – Увеличение вероятности RDP-сессии

Стоит отметить, что если процентное соотношение временных интервалов достигает больше 70%, то каждый следующий интервал программа относит к признакам RDP-сессии до самого ее завершения или прерывания.

Далее будет рассмотрено пара примеров работы программы, в которых отсутствует RDP-сессия.

В качестве эксперимента была запущена дополнительная VM Windows 10 (192.168.1.84), в которой была папка. К ней был предоставлен общий доступ и в нее же был добавлен файл размером около 6 МБ, как показано на рисунке 34.

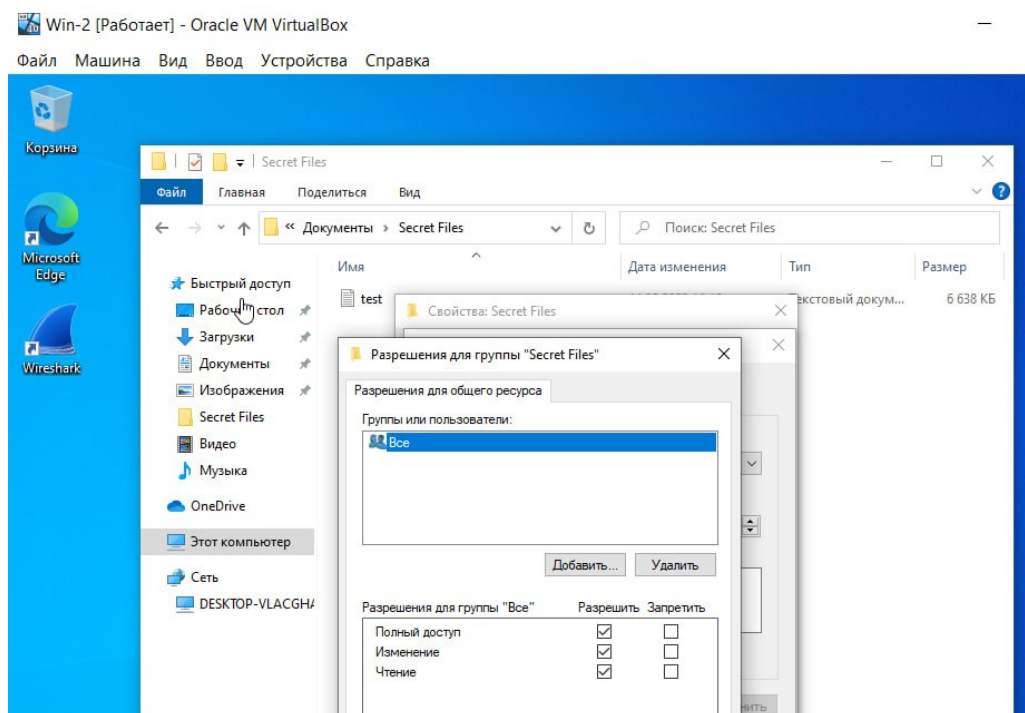


Рисунок 34 – Предоставление папке общего доступа

На следующем рисунке видно, что VM Windows 10 (192.168.1.133) имеет доступ к этой папке.

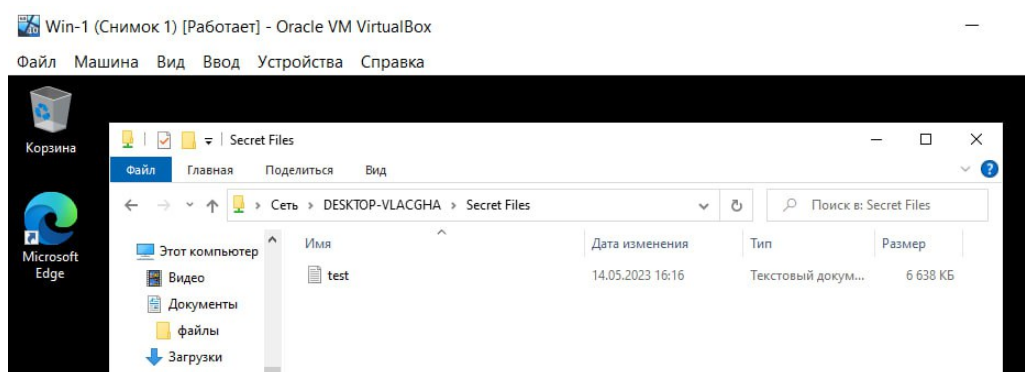


Рисунок 35 – Рабочий стол VM Windows 10 (192.168.1.133)

Эксперимент заключается в том, что VM Windows 10 (192.168.1.133) должна сохранить к себе файл, находящейся в общей папке. А в этот момент должна быть запущена программа «traffic-detection.py» в режиме перехвата трафика. И необходимо проверить найдет ли она что-нибудь в данной ситуации. Для начала опыт проводился с выключенным фильтром RDP. Т.е. программа выводила в консоль абсолютно все пакеты, которые ей удалось перехватить.

В момент перехвата сетевого трафика была установлена сессия, в которой применялся порт 445, как показано на рисунке 36. Данный порт принадлежит протоколу SMB (Server Message Block). Это протокол сетевого уровня,

который используется для обмена файлами, печати и других ресурсов между компьютерами в сети. Он является одним из стандартных протоколов Windows и используется для обмена данными между компьютерами под управлением Windows.

```
File Actions Edit View Help
-----Пакет No25-----
Время перехвата: 05:14:2023 17:02:20
Протокол: TCP
MAC-адрес отправителя: 08:00:27:60:30:4A
MAC-адрес получателя: 08:00:27:AD:64:87
Отправитель: 192.168.1.133:49931
Получатель: 192.168.1.84:445
Порядковый номер: 112334627; Номер подтверждения: 0
SYN:1; ACK:0; PSH:0; RST:0; FIN:0

Признаки: Установка соединения (SYN);
Вероятность RDP-сессии 0%
-----Пакет No26-----
Время перехвата: 05:14:2023 17:02:20
Протокол: TCP
MAC-адрес отправителя: 08:00:27:AD:64:87
MAC-адрес получателя: 08:00:27:60:30:4A
Отправитель: 192.168.1.84:445
Получатель: 192.168.1.133:49931
Порядковый номер: 3981065270; Номер подтверждения: 112334628
SYN:1; ACK:1; PSH:0; RST:0; FIN:0

Признаки: Подтверждение установки соединения (SYN-ACK);
Вероятность RDP-сессии 0%
-----Пакет No27-----
Время перехвата: 05:14:2023 17:02:20
Протокол: TCP
MAC-адрес отправителя: 08:00:27:60:30:4A
MAC-адрес получателя: 08:00:27:AD:64:87
Отправитель: 192.168.1.133:49931
Получатель: 192.168.1.84:445
Порядковый номер: 112334628; Номер подтверждения: 3981065271
SYN:0; ACK:1; PSH:0; RST:0; FIN:0

Признаки: Установлена сессия;
Вероятность RDP-сессии 0%
-----Пакет No28-----
```

Рисунок 36 – Информация об установке SMB-сессии

Из рисунка 36 видно, что программа перехватила установку активной SMB-сессии. На следующем рисунке изображен SMB-трафик, из которого видно, что никаких признаков протокола RDP не наблюдалось.

```
-----Пакет No366-----
Время перехвата: 05:14:2023 17:02:37
Протокол: TCP
MAC-адрес отправителя: 08:00:27:AD:64:87
MAC-адрес получателя: 08:00:27:60:30:4A
Отправитель: 192.168.1.84:445
Получатель: 192.168.1.133:49931
Порядковый номер: 3987522979; Номер подтверждения: 112344009
SYN:0; ACK:1; PSH:1; RST:0; FIN:0

Признаки: Ведется сессия;
Вероятность RDP-сессии 0%

-----Пакет No367-----
Время перехвата: 05:14:2023 17:02:37
Протокол: TCP
MAC-адрес отправителя: 08:00:27:60:30:4A
MAC-адрес получателя: 08:00:27:AD:64:87
Отправитель: 192.168.1.133:49931
Получатель: 192.168.1.84:445
Порядковый номер: 112344009; Номер подтверждения: 3987523655
SYN:0; ACK:1; PSH:1; RST:0; FIN:0

Признаки: Ведется сессия;
Вероятность RDP-сессии 0%

-----Пакет No368-----
Время перехвата: 05:14:2023 17:02:37
Протокол: TCP
MAC-адрес отправителя: 08:00:27:AD:64:87
MAC-адрес получателя: 08:00:27:60:30:4A
Отправитель: 192.168.1.84:445
Получатель: 192.168.1.133:49931
Порядковый номер: 3987523655; Номер подтверждения: 112344101
SYN:0; ACK:1; PSH:1; RST:0; FIN:0

Признаки: Ведется сессия;
Вероятность RDP-сессии 0%
```

Рисунок 37 – Информация о перехваченных пакетах, принадлежащих SMB-сессии

Далее были выполнены аналогичные действия, но перехват трафика осуществлялся уже с включенным фильтром RDP. На рисунке 38 показан результат перехвата сетевого трафика.

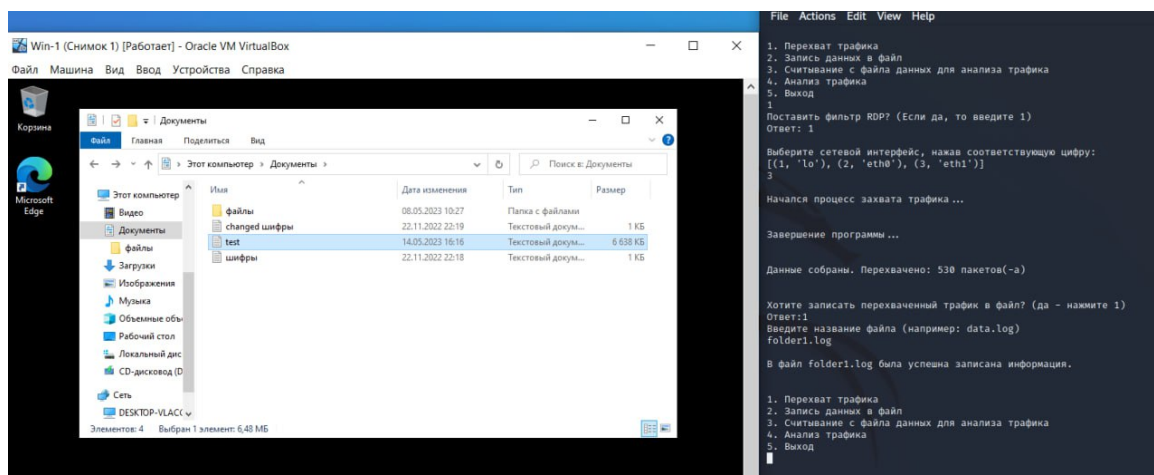


Рисунок 38 – Работа программы с установленным фильтром RDP при SMB-сессии

Согласно рисунку, изображенному на 38, программе не удалось обнаружить пакеты, свойственные протоколу RDP, так как в данном случае они отсутствовали.

Следующим шагом будет проведен эксперимент, чтобы выяснить, вызывает ли программа «traffic-detection.py» ложные срабатывания при обработке HTTP- и HTTPS-трафика. Сначала был запущен перехват сетевого трафика с установленным фильтром RDP. После этого производилось активное взаимодействие с браузером. На рисунке 39 изображено открытие в интернет-браузере сайта «Википедия», и пока что никаких ложных срабатываний программы не обнаружено.

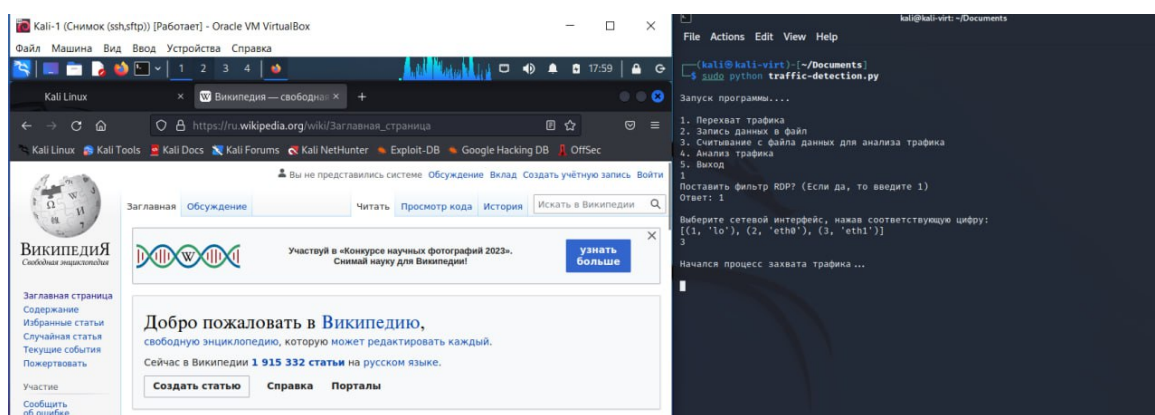


Рисунок 39 – Перехват трафика при работе с интернет-браузером

В течение нескольких минут в браузере открывались различные вкладки, на которых производились активные движения мышкой И нажатия клавиш на клавиатуре. Также на некоторых сайтах осуществлялся просмотр видео. Из рисунка 40 видно, что после завершения перехвата сетевого трафика было получено 38347 пакетов, и среди них программа не обнаружила признаков активной RDP-сессии.

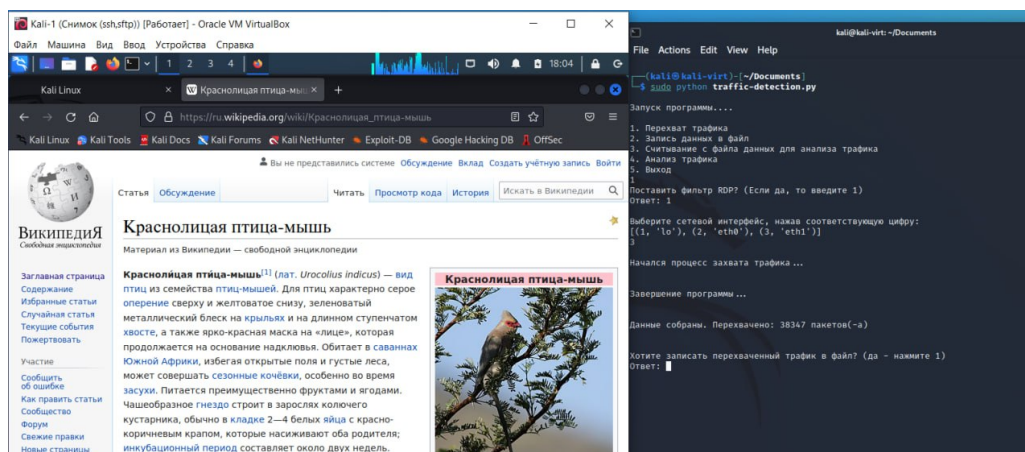


Рисунок 40 – Завершение перехвата трафика с установленным фильтром RDP

Конечно, в данном трафике есть небольшой процент пакетов, которые можно отнести к признакам протокола RDP, но программа оценивает все пяти-секундные интервалы и считает процентное соотношение для каждой установленной активной сессии.

Таким образом, с помощью статистических методов анализа сетевого трафика, реализованных в программе «traffic-detection.py», можно отличить RDP-трафик от других видов трафика. При всех четырех типах соединений программе удалось верно определить как наличие активных RDP-сессии, так и их отсутствие. Однако всё равно нельзя однозначно обнаружить RDP-трафик, так как применение статистических методов анализа сетевого трафика имеет ряд проблем:

- Неоднородность трафика: сетевой трафик может содержать множество различных типов трафика, включая RDP-трафик, и использование статистических методов для обнаружения конкретной сессии может оказаться сложным из-за этой неоднородности.
- Сложность обработки трафика: RDP-сессии могут содержать множество пакетов, и анализ большого объема трафика может быть сложным и требовать значительных вычислительных ресурсов.
- Вариации в сетевых конфигурациях: настройки сетевой конфигурации могут влиять на формат и содержание RDP-трафика, что может затруднить обнаружение RDP-сессий с помощью методов анализа трафика.

ЗАКЛЮЧЕНИЕ

В ходе данной курсовой работы был произведен анализ сетевого трафика для обнаружения активной RDP-сессии с использованием статистических методов. Была разработана программная реализация метода, включающая анализ TCP-соединения, обработку данных и построение графиков для анализа поведения RDP-трафика.

Был произведен анализ распределения размера пакетов, вычисление среднего значения и стандартного отклонения размеров пакетов, определение верхней и нижней границ диапазона значений размеров пакетов для каждого интервала времени и анализ полученных данных на наличие признаков RDP-сессии. Также был проведен анализ распределения временных интервалов между пакетами и частоты флагов PSH.

Были предложены модификации для улучшения обнаружения RDP-сессии, такие как изменение пороговых значений для интервалов и использование множественных методов анализа.

Была произведена проверка разработанной программы на определение наличия или отсутствия RDP-сессии, которая показала эффективность разработанного метода и программной реализации.

Таким образом, на основе анализа статистических характеристик сетевого трафика была разработана методика обнаружения RDP-сессии, которая может быть использована в качестве средства безопасности для защиты информации в компьютерных сетях. Рекомендуется дальнейшее исследование и развитие данного метода для его применения в различных условиях и сценариях использования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Удалённый рабочий стол RDP: как включить и как подключиться по RDP [Электронный ресурс] / URL: <https://hackware.ru/?p=11835> (дата обращения 31.03.2023), Яз. рус.
- 2 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.
- 3 Документация Remote Utilities «RDP» [Электронный ресурс] / URL: <https://www.remoteutilities.com/support/docs/rdp/> (дата обращения 31.03.2023), Яз. англ.
- 4 Статья «Модель OSI» [Электронный ресурс] / URL: http://neerc.ifmo.ru/wiki/index.php?title=OSI_Model (дата обращения 31.03.2023), Яз. рус.
- 5 Статья «TCP flags» [Электронный ресурс] / URL: <https://www.keycdn.com/support/tcp-flags#:~:text=ACK> (дата обращения 31.03.2023), Яз. англ.
- 6 Документация по стандартным библиотекам языка Python [Электронный ресурс] / URL: <https://docs.python.org/3/library/socket.html> (дата обращения 31.03.2023), Яз. англ.
- 7 Документация Microsoft «How Terminal Services Works» [Электронный ресурс] / URL: [https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc755399\(v=ws.10\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc755399(v=ws.10)?redirectedfrom=MSDN) (дата обращения 14.04.2023), Яз. англ.
- 8 Статья «Работа с клиентом удаленного рабочего стола Remmina» [Электронный ресурс] / URL: <https://white55.ru/remmina.html> (дата обращения 15.04.2023), Яз. рус.
- 9 Документация Microsoft «Изменение порта прослушивания для удаленного рабочего стола на компьютере» [Электронный ресурс] / URL: <https://learn.microsoft.com/ru-ru/windows-server/remote/remote-desktop-services/clients/change-listening-port> (дата обращения 28.04.2023), Яз. рус.

- 10 Методическая разработка «Исследование трафика локальной сети посредством сетевого анализатора «Wireshark» » [Электронный ресурс] / URL: <http://ib.psuti.ru/content/metod/методическиеWireshark.pdf> (дата обращения 04.05.2023), Яз. рус.
- 11 Статья из википедии «Standard deviation» [Электронный ресурс] / URL: https://en.wikipedia.org/wiki/Standard_deviation (дата обращения 04.05.2023), Яз. англ.
- 12 Статья «Стандартное отклонение» [Электронный ресурс] / URL: <https://berg.com.ua/indicators-overlays/stddev/> (дата обращения 04.05.2023), Яз. рус.

ПРИЛОЖЕНИЕ А

Код traffic-detection.py

```
1  import matplotlib.pyplot as plt
2  import matplotlib.gridspec as gridspec
3  import time, socket, os, struct, math, keyboard
4  from colorama import init, Back, Fore
5
6  init(autoreset=True)
7
8
9  # Глобальные переменные
10 FileName = ''
11 Packet_list = []
12 Object_list = []
13 Labels_list = []
14 Session_list = []
15 x_axisLabels = []
16 Phrases_signs = [ 'Нет', 'Установка соединения (SYN)'
17                   , 'Подтверждение установки соединения (SYN-ACK)'
18                   , 'Установлена сессия', 'Ведется сессия', 'Подозрение на RDP-сессию!'
19                   , 'Сессия закончена', 'Сессия прервана'
20                   , 'Передача клавиатурных и мышинных событий']
21 findRDP = False
22 line = '-----'
23
24
25 # Класс, содержащий информацию о каком-либо пакете
26 class PacketInf:
27
28     def __init__( self, numPacket, timePacket, packetSize, mac_src, mac_dest, protoType
29                   , ip_src, ip_dest, port_src, port_dest, len_data, seq=None, ack=None
30                   , fl_ack=None, fl_psh=None, fl_rst=None, fl_syn=None, fl_fin=None):
31         self.numPacket = int(numPacket)
32         self.timePacket = float(timePacket)
33         self.packetSize = int(packetSize)
34         self.mac_src = mac_src
35         self.mac_dest = mac_dest
36         self.protoType = protoType
37         self.ip_src = ip_src
38         self.ip_dest = ip_dest
39         self.port_src = port_src
40         self.port_dest = port_dest
41         self.len_data = int(len_data)
42         self.seq = seq
43         self.ack = ack
44         self.fl_ack = fl_ack
45         self.fl_psh = fl_psh
46         self.fl_rst = fl_rst
```



```

47     self.fl_syn = fl_syn
48     self.fl_fin = fl_fin
49
50
51 # Класс, содержащий информацию относительно какого-либо IP-адреса
52 class ExploreObject:
53
54     def __init__(self, ip):
55         self.ip = ip
56         self.strt_time = None
57         self.fin_time = None
58         self.amnt_packet = None
59         self.avg_packet_num = None
60         self.avg_packet_size = None
61
62         self.commonPorts = None
63         self.in_out_rel_data = None
64         self.ack_flags_diff_data = None
65         self.udp_tcp_rel_data = None
66         self.syn_flags_freq_data = None
67         self.psh_flags_freq_data = None
68         self.pkt_amnt_src_data = None
69         self.pkt_amnt_dst_data = None
70         self.pkt_size_data_src = None
71         self.pkt_size_data_dst = None
72         self.adjciPList = None
73         self.adjciPacketList = None
74
75
76 # Класс, содержащий информацию о каждой активной сессии
77 class Session:
78
79     def __init__(self, strtTime, init, target, port):
80         self.fl_syn = True
81         self.fl_fin = False
82         self.fl_rst = False
83         self.strtTime = strtTime
84         self.curTime = strtTime + 5
85         self.curSec = strtTime + 1
86         self.finTime = None
87         self.totalTime = None
88         self.initiator = init
89         self.target = target
90         self.port = port
91         self.seq_num = None
92         self.ack_num = None
93         self.is_rdp = False
94         self.is_rdpArr = []

```

```

95     self.cntTr = 0
96     self.prob = 0
97     self.is_rdpDev = False
98     self.pktSize = []
99     self.is_rdpPSH = False
100    self.cntpsh = 0
101    self.cntPktTCP = 0
102    self.pshfreq = []
103    self.is_rdpInOut = False
104    self.trafficInit = []
105    self.trafficTarg = []
106    self.cntInitIn = 0
107    self.cntTargIn = 0
108    self.cntInitOut = 0
109    self.cntTargOut = 0
110    self.is_rdpIntvl = False
111    self.intervals = []
112    self.prevPktTime = None
113
114
115    # Обновление значения порядкового номера
116    def upd_seq_num(self, seq):
117        self.seq_num = int(seq)
118
119
120    # Обновление значения номера подтверждения
121    def upd_ack_num(self, ack):
122        self.ack_num = ack
123
124
125    # Обновление значения флага FIN
126    def upd_fl_fin(self, fin):
127        self.fl_fin = True
128        self.finTime = fin
129        self.totalTime = round(self.finTime - self.strtTime, 2)
130
131
132    # Обновление значения флага RST
133    def upd_fl_rst(self, fin):
134        self.fl_rst = True
135        self.finTime = fin
136        self.totalTime = round(self.finTime - self.strtTime, 2)
137
138
139    # Вычисление распределений для выявления признаков RDP
140    def get_rdp_features(self, pkt, isfin=False):
141        n = len(self.pktSize)
142        if n != 0 and (pkt.timePacket > self.curTime or isfin):

```

```

143     # Вычисление распределения размеров пакетов
144     sum = 0
145     for el in self.pktSize:
146         sum += el
147     avg = sum / n
148     sum = 0
149     for el in self.pktSize:
150         sum += (el - avg) * (el - avg)
151     dev = math.sqrt(sum / n)
152     cnt = 0
153     for el in self.pktSize:
154         if abs(avg - dev * 4) > el or el > (avg + dev * 4):
155             cnt += 1
156     if cnt * 1.6 > n:
157         self.is_rdpDev = True
158     else:
159         self.is_rdpDev = False
160     self.pktSize.clear()
161     # Вычисление частоты PSH флагов
162     if self.cntPktTCP != 0:
163         self.pshfreq.append(self.cntpsh / self.cntPktTCP)
164     else:
165         self.pshfreq.append(0.0)
166     avg = self.get_average_val()
167     if self.pshfreq[-1] > 0.0 and abs(avg - self.pshfreq[-1]) < 0.3:
168         self.is_rdpPSH = True
169     else:
170         self.is_rdpPSH = False
171     self.cntPktTCP = 0
172     self.cntpsh = 0
173     # Вычисление отношения входящего трафика на исходящий
174     in_len = len(self.trafficInit)
175     out_len = len(self.trafficTarg)
176     if in_len != 0:
177         avg = 0
178         for el in self.trafficInit:
179             avg += el
180         avg = avg / in_len
181         avg1 = 0
182         for el in self.trafficTarg:
183             avg1 += el
184         avg1 = avg1 / out_len
185         if (in_len > 3 and out_len > 3) and \
186             ((1 < avg and avg <= 2.0 and 0.5 <= avg1 and avg1 < 1) or \
187              (0.5 <= avg and avg < 1 and 1 < avg1 and avg1 <= 2.0)) and \
188             (abs(avg - avg1) > 0.2 and abs(avg - avg1) < 1.8):
189             self.is_rdpInOut = True
190     else:

```

```

191         self.is_rdpInOut = False
192     self.cntInitIn = 0
193     self.cntInitOut = 0
194     self.cntTargIn = 0
195     self.cntTargOut = 0
196     self.trafficInit.clear()
197     self.trafficTarg.clear()
198 else:
199     self.is_rdpInOut = False
200     # Вычисление распределения интервалов
201     l = len(self.intervals)
202     if l != 0:
203         sum = 0
204         for el in self.intervals:
205             sum += el
206         avg = sum / l
207         sum = 0
208         for el in self.intervals:
209             sum += (el - avg) * (el - avg)
210         dev = math.sqrt(sum / l)
211         cnt = 0
212         if l > 40:
213             for el in self.intervals:
214                 if el > abs(avg + dev / 1.8) or el < abs(avg - dev / 1.8):
215                     cnt += 1
216             if cnt * 2 > l:
217                 self.is_rdpIntvl = True
218             else:
219                 self.is_rdpIntvl = False
220             self.intervals.clear()
221             self.prevPktTime = None
222         else:
223             self.is_rdpIntvl = False
224         self.curTime += 5
225         self.rdp_check()
226         if len(self.is_rdpArr) == 0:
227             self.is_rdp = False
228         else:
229             self.is_rdp = self.is_rdpArr[-1]
230     self.pktSize.append(pkt.packetSize)
231     if pkt.protoType == 'TCP' and pkt.ip_src == self.initiator:
232         self.cntPktTCP += 1
233         if pkt.fl_psh == '1':
234             self.cntpsh += 1
235     if self.prevPktTime != None:
236         self.intervals.append(pkt.timePacket - self.prevPktTime)
237         self.prevPktTime = pkt.timePacket
238     else:

```

```

239         self.prevPktTime = pkt.timePacket
240
241
242     # Вычисление входящего и исходящего трафика за единицу времени
243     def get_in_out_traffic(self, pkt):
244         if pkt.timePacket > self.curSec:
245             if self.cntInitOut != 0:
246                 self.trafficInit.append(self.cntInitIn / self.cntInitOut)
247             else:
248                 self.trafficInit.append(0.0)
249             if self.cntTargOut != 0:
250                 self.trafficTarg.append(self.cntTargIn / self.cntTargOut)
251             else:
252                 self.trafficTarg.append(0.0)
253             self.cntInitIn = 0
254             self.cntTargIn = 0
255             self.cntInitOut = 0
256             self.cntTargOut = 0
257             self.curSec += 1
258             if pkt.ip_src == self.initiator:
259                 self.cntInitOut += 1
260             if pkt.ip_dest == self.initiator:
261                 self.cntInitIn += 1
262             if pkt.ip_src == self.target:
263                 self.cntTargOut += 1
264             if pkt.ip_dest == self.target:
265                 self.cntTargIn += 1
266
267
268     # Анализ значений списка rdpArr
269     def rdpArr_check(self):
270         l = len(self.is_rdpArr)
271         if l > 2:
272             return self.cntTr > l - self.cntTr
273         else:
274             return False
275
276
277     # Нахождение среднего значения частот PSH-флагов
278     def get_average_val(self):
279         n = len(self.pshfreq)
280         if n >= 4:
281             return (self.pshfreq[n - 4] + self.pshfreq[n - 3] + \
282                     self.pshfreq[n - 2]) / 3
283         return -10
284
285
286     # Осуществление проверки текущего интервала

```

```

287 # времени на наличие RDP-трафика
288 def rdp_check(self):
289     if self.port == '3389':
290         self.is_rdpArr.append(True)
291         self.cntTr += 1
292         self.prob = 100
293     elif self.prob > 70:
294         self.is_rdpArr.append(True)
295         self.cntTr += 1
296         self.prob = round((self.cntTr / len(self.is_rdpArr)) * 100)
297     else:
298         if (self.is_rdpInOut and self.is_rdpIntvl) or \
299             (self.is_rdpInOut and self.is_rdpPSH and self.is_rdpDev):
300             self.is_rdpArr.append(True)
301             self.cntTr += 1
302         else:
303             if (self.is_rdpInOut or self.is_rdpIntvl):
304                 if (self.is_rdpDev and self.rdpArr_check()) or \
305                     (not self.is_rdpDev and self.rdpArr_check()):
306                     self.is_rdpArr.append(True)
307                     self.cntTr += 1
308                 else:
309                     self.is_rdpArr.append(False)
310             else:
311                 self.is_rdpArr.append(False)
312             if len(self.is_rdpArr) > 4:
313                 self.prob = round((self.cntTr / len(self.is_rdpArr)) * 100)
314
315
316 # Подсчет значений списка rdpArr для анализа трафика
317 def fin_rdp_check(self):
318     cnt = 0
319     for el in self.is_rdpArr:
320         if el:
321             cnt += 1
322     self.is_rdp = cnt > len(self.is_rdpArr) - cnt
323
324
325 # Получение ethernet-кадра
326 def get_ethernet_frame(data):
327     dest_mac, src_mac, proto = struct.unpack('!6s6sH', data[:14])
328     return get_mac_addr(dest_mac), get_mac_addr(src_mac), socket.htons(proto)
329
330
331 # Получение MAC-адреса
332 def get_mac_addr(mac_bytes):
333     mac_str = ''
334     for el in mac_bytes:

```

```

335     mac_str += format(e1, '02x').upper() + ':'
336     return mac_str[:len(mac_str) - 1]
337
338
339     # Получение IPv4-заголовка
340     def get_ipv4_data(data):
341         version_header_length = data[0]
342         header_length = (version_header_length & 15) * 4
343         ttl, proto, src, dest = struct.unpack('!8xBB2x4s4s', data[:20])
344         return ttl, proto, ipv4_dec(src), ipv4_dec(dest), data[header_length:]
345
346
347     # Получение IP-адреса формата X.X.X.X
348     def ipv4_dec(ip_bytes):
349         ip_str = ''
350         for e1 in ip_bytes:
351             ip_str += str(e1) + '.'
352         return ip_str[:-1]
353
354
355     # Получение UDP-сегмента данных
356     def get_udp_segment(data):
357         src_port, dest_port, size = struct.unpack('!HH2xH', data[:8])
358         return str(src_port), str(dest_port), size, data[8:]
359
360
361     # Получение TCP-сегмента данных
362     def get_tcp_segment(data):
363         src_port, dest_port, sequence, ack, some_block = struct.unpack('!HLLH', data[:14])
364         return str(src_port), str(dest_port), str(sequence), str(ack), \
365             some_block, data[(some_block >> 12) * 4:]
366
367
368     # Форматирование данных для корректного представления
369     def format_data(data):
370         if isinstance(data, bytes):
371             data = ''.join(r'\x{:02x}'.format(e1) for e1 in data)
372         return data
373
374
375     # Перехват трафика и вывод информации в консоль
376     def start_to_listen(s_listen):
377         global Packet_list
378         NumPacket = 1
379         curcnt = 1000
380         while True:
381             # Получение пакетов в виде набора hex-чисел
382             raw_data, _ = s_listen.recvfrom(65565)

```

```

383 pinf = [''] * 18
384 pinf[0], pinf[1] = NumPacket, time.time()
385 pinf[2] = len(raw_data)
386 # Если это интернет-протокол четвертой версии
387 pinf[4], pinf[3], protocol = get_ethernet_frame(raw_data)
388 if protocol == 8:
389     _, proto, pinf[6], pinf[7], data_ipv4 = get_ipv4_data(raw_data[14:])
390     if NumPacket > curcnt:
391         curcnt += 1000
392         clear_end_sessions()
393     # Если это UDP-протокол
394     if proto == 17:
395         NumPacket += 1
396         pinf[5] = 'UDP'
397         pinf[8], pinf[9], _, data_udp = get_udp_segment(data_ipv4)
398         pinf[10] = len(data_udp)
399         Packet_list.append(PacketInf( pinf[0], pinf[1], pinf[2]
400                                     , pinf[3], pinf[4], pinf[5]
401                                     , pinf[6], pinf[7], pinf[8]
402                                     , pinf[9], pinf[10]))
403         mes_prob = find_session_location(Packet_list[-1])
404         print_packet_inf(Packet_list[-1], mes_prob)
405     # Если это TCP-протокол
406     if proto == 6:
407         NumPacket += 1
408         pinf[5] = 'TCP'
409         pinf[8], pinf[9], pinf[11], \
410         pinf[12], flags, data_tcp = get_tcp_segment(data_ipv4)
411         pinf[10] = len(data_tcp)
412         pinf[13] = str((flags & 16) >> 4)
413         pinf[14] = str((flags & 8) >> 3)
414         pinf[15] = str((flags & 4) >> 2)
415         pinf[16] = str((flags & 2) >> 1)
416         pinf[17] = str(flags & 1)
417         Packet_list.append(PacketInf( pinf[0], pinf[1], pinf[2], pinf[3]
418                                     , pinf[4], pinf[5], pinf[6], pinf[7]
419                                     , pinf[8], pinf[9], pinf[10], pinf[11]
420                                     , pinf[12], pinf[13], pinf[14], pinf[15]
421                                     , pinf[16], pinf[17] ))
422         mes_prob = find_session_location(Packet_list[-1])
423         print_packet_inf(Packet_list[-1], mes_prob)
424 if keyboard.is_pressed('space'):
425     s_listen.close()
426     print('\nЗавершение программы...\n')
427     break
428
429
430 # Обработка значений списка Session_list

```



```

431 def clear_end_sessions():
432     global Session_list
433     n = len(Session_list)
434     ids = []
435     for i in range(n):
436         if Session_list[i].fl_fin or Session_list[i].fl_rst:
437             if Session_list[i].totalTime < 10:
438                 ids.append(i)
439     tmp = Session_list.copy()
440     Session_list.clear()
441     for i in range(n):
442         if i in ids:
443             continue
444         Session_list.append(tmp[i])
445     for s in Session_list:
446         s.get_rdp_features(Packet_list[-1], True)
447
448
449 # Нахождение активных сессий
450 def find_session_location(pkt):
451     global Session_list
452     if pkt.protoType == 'UDP':
453         for s in Session_list:
454             if (not s.fl_fin and not s.fl_rst):
455                 if ( (pkt.ip_src == s.initiator and pkt.ip_dest == s.target) or \
456                     (pkt.ip_src == s.target and pkt.ip_dest == s.initiator) ) and \
457                     (pkt.port_src == s.port or pkt.port_dest == s.port):
458                     s.get_in_out_traffic(pkt)
459                     s.get_rdp_features(pkt)
460                     if s.is_rdp:
461                         if s.is_rdpPSH:
462                             return ([4, 5, 8], s.prob)
463                         return ([4, 5], s.prob)
464         return ([0], 0)
465     if pkt.fl_syn == '1' and pkt.fl_ack == '0':
466         Session_list.append(Session( pkt.timePacket, pkt.ip_src
467                                     , pkt.ip_dest, pkt.port_dest ))
468         Session_list[-1].upd_seq_num(pkt.seq)
469         Session_list[-1].get_in_out_traffic(pkt)
470         Session_list[-1].get_rdp_features(pkt)
471         return ([1], Session_list[-1].prob)
472     for s in Session_list:
473         if (not s.fl_fin and not s.fl_rst):
474             if pkt.fl_fin == '1' and pkt.fl_ack == '1' and \
475                 ( (pkt.ip_src == s.initiator and pkt.ip_dest == s.target) or \
476                   (pkt.ip_src == s.target and pkt.ip_dest == s.initiator) ) and \
477                 (pkt.port_src == s.port or pkt.port_dest == s.port):
478                 s.upd_fl_fin(pkt.timePacket)

```

```

479         s.get_in_out_traffic(pkt)
480         s.get_rdp_features(pkt)
481         if s.is_rdp:
482             if s.is_rdpPSH:
483                 return ([5, 6, 8], s.prob)
484             return ([5, 6], s.prob)
485         return ([6], s.prob)
486     if pkt.fl_rst == '1' and pkt.fl_ack == '1' and \
487        ( (pkt.ip_src == s.initiator and pkt.ip_dest == s.target) or \
488          (pkt.ip_src == s.target and pkt.ip_dest == s.initiator) ) and \
489        (pkt.port_src == s.port or pkt.port_dest == s.port):
490         s.upd_fl_rst(pkt.timePacket)
491         s.get_in_out_traffic(pkt)
492         s.get_rdp_features(pkt)
493         if s.is_rdp:
494             if s.is_rdpPSH:
495                 return ([5, 7, 8], s.prob)
496             return ([5, 7], s.prob)
497         return ([7], s.prob)
498     if pkt.fl_syn == '1' and pkt.fl_ack == '1' and s.ack_num == None and \
499        pkt.ack == str(s.seq_num + 1) and pkt.ip_src == s.target and \
500        pkt.ip_dest == s.initiator and pkt.port_src == s.port:
501         s.upd_ack_num(pkt.ack)
502         s.upd_seq_num(pkt.seq)
503         s.get_in_out_traffic(pkt)
504         s.get_rdp_features(pkt)
505         if s.is_rdp:
506             if s.is_rdpPSH:
507                 return ([2, 5, 8], s.prob)
508             return ([2, 5], s.prob)
509         return ([2], s.prob)
510     elif pkt.fl_syn == '0' and pkt.fl_ack == '1' and pkt.ack == str(s.seq_num + 1) and \
511        pkt.seq == s.ack_num and \
512        pkt.port_dest == s.port and pkt.ip_src == s.initiator and \
513        pkt.ip_dest == s.target:
514         s.get_in_out_traffic(pkt)
515         s.get_rdp_features(pkt)
516         if s.is_rdp:
517             if s.is_rdpPSH:
518                 return ([3, 5, 8], s.prob)
519             return ([3, 5], s.prob)
520         return ([3], s.prob)
521     if pkt.fl_ack == '1' and \
522        ( (pkt.ip_src == s.initiator and pkt.ip_dest == s.target) or \
523          (pkt.ip_src == s.target and pkt.ip_dest == s.initiator) ) and \
524        (pkt.port_src == s.port or pkt.port_dest == s.port):
525         s.get_in_out_traffic(pkt)
526         s.get_rdp_features(pkt)

```

```

527         if s.is_rdp:
528             if s.is_rdpPSH:
529                 return ([4, 5, 8], s.prob)
530             return ([4, 5], s.prob)
531         return ([4], s.prob)
532     return ([0], 0)
533
534
535     # Вывод информации о сессиях
536     def print_inf_about_sessions():
537         cnt = 1
538         print(f'\nБыло перехвачено {len(Session_list)} сессии(-й)')
539         for s in Session_list:
540             print(f'\nИнформация о сессии #{cnt}:')
541             print(f'Инициатор подключения: {s.initiator}')
542             print(f'Целевое устройство: {s.target}')
543             print(f'Порт подключения: {s.port}')
544             print(f'Время установки соединения: '
545                   , time.strftime('%d.%m.%Y г. %H:%M:%S', time.localtime(s.strtTime)) )
546             if s.finTime == None:
547                 print(f'Время завершения соединения: нет данных')
548             else:
549                 print(f'Время завершения соединения: '
550                       , time.strftime('%d.%m.%Y г. %H:%M:%S', time.localtime(s.finTime)))
551                 print(f'Общее время соединения: {s.totalTime} сек')
552             if s.is_rdp and s.prob > 50:
553                 print(Back.GREEN + Fore.BLACK + f'Найдена RDP-сессия с вероятностью {s.prob}%!!!')
554             cnt += 1
555         print(f'{line}{line}\n')
556
557
558     # Запись информации о пакетах в файл
559     def write_to_file(f):
560         if Packet_list == []:
561             return False
562         try:
563             for obj in Packet_list:
564                 if obj.protoType == 'UDP':
565                     f.write( f'No:{obj.numPacket};Time:{obj.timePacket};Pac-size:{obj.packetSize};' +
566                             f'MAC-src:{obj.mac_src};MAC-dest:{obj.mac_dest};Type:{obj.protoType};' +
567                             f'IP-src:{obj.ip_src};IP-dest:{obj.ip_dest};Port-src:{obj.port_src};' +
568                             f'Port-dest:{obj.port_dest};Len-data:{obj.len_data};!\n' )
569                 else:
570                     f.write( f'No:{obj.numPacket};Time:{obj.timePacket};Pac-size:{obj.packetSize};' +
571                             f'MAC-src:{obj.mac_src};MAC-dest:{obj.mac_dest};Type:{obj.protoType};' +
572                             f'IP-src:{obj.ip_src};IP-dest:{obj.ip_dest};Port-src:{obj.port_src};' +
573                             f'Port-dest:{obj.port_dest};Len-data:{obj.len_data};Seq:{obj.seq};' +
574                             f'Ack:{obj.ack};Fl-ack:{obj.fl_ack};Fl-psh:{obj.fl_psh};' +

```

```

575         f'Fl-rst:{obj.fl_rst};Fl-syn:{obj.fl_syn};Fl-fin:{obj.fl_fin};!\n' )
576     except:
577         return False
578     return True
579
580
581     # Считывание с файла и заполнение массива
582     # Packet_list объектами класса PacketInf
583     def read_from_file(inf):
584         global Packet_list
585         a = []
586         while True:
587             beg = inf.find(':')
588             end = inf.find(';')
589             if beg == -1 and end == -1:
590                 break
591             else:
592                 a.append(inf[beg + 1: end])
593             inf = inf[end + 1:]
594         try:
595             if a[5] == 'TCP':
596                 Packet_list.append(PacketInf( a[0], a[1], a[2], a[3], a[4], a[5]
597                                                 , a[6], a[7], a[8], a[9], a[10], a[11]
598                                                 , a[12], a[13], a[14], a[15], a[16], a[17] ))
599                 _ = find_session_location(Packet_list[-1])
600             elif a[5] == 'UDP':
601                 Packet_list.append(PacketInf( a[0], a[1], a[2], a[3], a[4], a[5]
602                                                 , a[6], a[7], a[8], a[9], a[10] ))
603                 _ = find_session_location(Packet_list[-1])
604         except:
605             print('Ошибка при считывании файла...')
606             exit(0)
607
608
609     # Вывод информации о перехваченных пакетах
610     def print_packet_inf(obj, mes_prob):
611         if findRDP:
612             if 5 not in mes_prob[0] or mes_prob[1] <= 50:
613                 return
614         print( f'{line}Пакет No{obj.numPacket}{line}\n'
615               , 'Время перехвата: '
616               , time.strftime( '%m:%d:%Y %H:%M:%S'
617                               , time.localtime(obj.timePacket) ) + '\n'
618               , f'Протокол: {obj.protoType}\n'
619               , f'MAC-адрес отправителя: {obj.mac_src}\n'
620               , f'MAC-адрес получателя: {obj.mac_dest}\n'
621               , f'Отправитель: {obj.ip_src}:{obj.port_src}\n'
622               , f'Получатель: {obj.ip_dest}:{obj.port_dest}')

```

```

623     if obj.protoType == 'TCP':
624         print( f' Порядковый номер: {obj.seq}; Номер подтверждения: {obj.ack}\n' +
625               f' SYN:{obj.fl_syn}; ACK:{obj.fl_ack}; PSH:{obj.fl_psh}; ' +
626               f'RST:{obj.fl_rst}; FIN:{obj.fl_fin}\n')
627     print('Признаки: ', end='')
628     for i in mes_prob[0]:
629         print(Phrases_signs[i], end='; ')
630     print(f'\nВероятность RDP-сессии {mes_prob[1]}%')
631
632
633     # Получение общей информации о текущей
634     # попытке перехвата трафика
635     def get_common_data():
636         global Labels_list
637         Labels_list.clear()
638         IPList = set()
639         numPacketsPerSec = []
640         curTime = Packet_list[0].timePacket + 1
641         fin = Packet_list[-1].timePacket + 1
642         Labels_list.append(time.strftime('%H:%M:%S', time.localtime(Packet_list[0].timePacket)))
643         cntPacket = 0
644         i = 0
645         while curTime < fin:
646             for k in range(i, len(Packet_list)):
647                 if Packet_list[k].timePacket > curTime:
648                     numPacketsPerSec.append(cntPacket)
649                     Labels_list.append(time.strftime('%H:%M:%S', time.localtime(curTime)))
650                     cntPacket = 0
651                     i = k
652                     break
653             cntPacket += 1
654             curTime += 1
655         numPacketsPerSec.append(cntPacket)
656         for p in Packet_list:
657             IPList.add(p.ip_src)
658             IPList.add(p.ip_dest)
659         return list(IPList), numPacketsPerSec
660
661
662     # Получение общих портов относительно текущего IP-адреса
663     def get_common_ports(curIP):
664         ports = set()
665         for pkt in Packet_list:
666             if pkt.ip_src == curIP or pkt.ip_dest == curIP:
667                 ports.add(pkt.port_src)
668                 ports.add(pkt.port_dest)
669         return list(ports)
670

```

```

671
672 # Вывод пар (число, IP-адрес/порт) для
673 # предоставления выбора IP-адреса/порта
674 # пользователю
675 def print_list_of_pairs(IPList, fl=False):
676     num = 0
677     cnt = 1
678     if fl:
679         print ('[' + str(num), '---', 'None', end='] ')
680         cnt += 1
681         num += 1
682     for el in IPList:
683         if cnt > 3:
684             cnt = 0
685             print ('[' + str(num), '---', el, end=']\n')
686         else:
687             print ('[' + str(num), '---', el, end='] ')
688             cnt += 1
689             num += 1
690     print('')
691
692
693 # Вывод пакетов, связанных с выбранным IP-адресом
694 def print_adjacent_packets(adjcPacketList):
695     cnt = 0
696     for p in adjcPacketList:
697         t = time.strftime('%H:%M:%S', time.localtime(p.timePacket))
698         if cnt % 2 == 1:
699             print( f'Номер пакета: {p.numPacket};', f' Время: {t};'
700                 , f' Размер: {p.packetSize};', f' MAC-адрес отправителя: {p.mac_src};'
701                 , f' MAC-адрес получателя: {p.mac_dest};', f' Протокол: {p.protoType};'
702                 , f' Отправитель: {p.ip_src}:{p.port_src};'
703                 , f' Получатель: {p.ip_dest}:{p.port_dest};'
704                 , f' Размер поля данных: {p.len_data};', end='' )
705             if p.protoType == 'TCP':
706                 print( f' Порядковый номер: {p.seq}; Номер подтверждения: {p.ack};' +
707                     f' SYN:{p.fl_syn}; ACK:{p.fl_ack}; PSH:{p.fl_psh}; ' +
708                     f'RST:{p.fl_rst}; FIN:{p.fl_fin};')
709             else:
710                 print('')
711         else:
712             print( Back.CYAN + Fore.BLACK + f'Номер пакета: {p.numPacket};' + f' Время: {t};' +
713                 f' Размер: {p.packetSize};' + f' MAC-адрес отправителя: {p.mac_src};' +
714                 f' MAC-адрес получателя: {p.mac_dest};' +
715                 f' Отправитель: {p.ip_src}:{p.port_src};' +
716                 f' Получатель: {p.ip_dest}:{p.port_dest};' +
717                 f' Протокол: {p.protoType};' +
718                 f' Размер поля данных: {p.len_data};', end='' )

```

```

719         if p.protoType == 'TCP':
720             print( Back.CYAN + Fore.BLACK + f' Порядковый номер: {p.seq};' +
721                   f' Номер подтверждения: {p.ack};' +
722                   f' SYN:{p.fl_syn}; ACK:{p.fl_ack}; PSH:{p.fl_psh};' +
723                   f' RST:{p.fl_rst}; FIN:{p.fl_fin};')
724         else:
725             print('')
726         cnt += 1
727
728
729     # Получение данных об отношении входящего
730     # трафика к исходящему в единицу времени
731     def get_in_out_rel(exploreIP, strt, fin, port):
732         cntInput = 0
733         cntOutput = 0
734         rel_list = []
735         curTime = strt + 1
736         fin += 1
737         pos = 0
738         while curTime < fin:
739             for k in range(pos, len(Packet_list)):
740                 if Packet_list[k].timePacket > curTime:
741                     if cntOutput != 0:
742                         rel_list.append(cntInput / cntOutput)
743                     else:
744                         rel_list.append(0.0)
745                     cntInput = 0
746                     cntOutput = 0
747                     pos = k
748                     break
749             if port == None:
750                 if Packet_list[k].ip_src == exploreIP:
751                     cntOutput += 1
752                 if Packet_list[k].ip_dest == exploreIP:
753                     cntInput += 1
754             else:
755                 if Packet_list[k].port_src == port or Packet_list[k].port_dest == port:
756                     if Packet_list[k].ip_src == exploreIP:
757                         cntOutput += 1
758                     if Packet_list[k].ip_dest == exploreIP:
759                         cntInput += 1
760             curTime += 1
761         if cntOutput != 0:
762             rel_list.append(cntInput / cntOutput)
763         else:
764             rel_list.append(0.0)
765         return rel_list
766

```

```

767
768 # Получение данных об отношении количества
769 # входящего UDP-трафика на количество
770 # исходящего TCP-трафика в единицу времени
771 def get_udp_tcp_rel(exploreIP, strt, fin, port):
772     cntUDP = 0
773     cntTCP = 0
774     curTime = strt + 1
775     fin += 1
776     pos = 0
777     rel_list = []
778     while curTime < fin:
779         for k in range(pos, len(Packet_list)):
780             if Packet_list[k].timePacket > curTime:
781                 if cntTCP != 0:
782                     rel_list.append(cntUDP / cntTCP)
783                 else:
784                     rel_list.append(0.0)
785                 cntTCP = 0
786                 cntUDP = 0
787                 pos = k
788                 break
789             if port == None:
790                 if Packet_list[k].ip_dest == exploreIP:
791                     if Packet_list[k].protoType == 'TCP':
792                         cntTCP += 1
793                     if Packet_list[k].protoType == 'UDP':
794                         cntUDP += 1
795                 else:
796                     if Packet_list[k].port_src == port or Packet_list[k].port_dest == port:
797                         if Packet_list[k].ip_dest == exploreIP:
798                             if Packet_list[k].protoType == 'TCP':
799                                 cntTCP += 1
800                             if Packet_list[k].protoType == 'UDP':
801                                 cntUDP += 1
802             curTime += 1
803             if cntTCP != 0:
804                 rel_list.append(cntUDP / cntTCP)
805             else:
806                 rel_list.append(0.0)
807         return rel_list
808
809
810 # Получение данных о разности количества
811 # исходящих ACK-флагов и количества входящих
812 # ACK-флагов
813 def get_ack_flags_diff(exploreIP, strt, fin, port):
814     cntInput = 0

```



```

815     cntOutput = 0
816     diff_list = []
817     curTime = strt + 1
818     fin += 1
819     pos = 0
820     while curTime < fin:
821         for k in range(pos, len(Packet_list)):
822             if Packet_list[k].timePacket > curTime:
823                 diff_list.append(cntOutput - cntInput)
824                 cntInput = 0
825                 cntOutput = 0
826                 pos = k
827                 break
828             if port == None:
829                 if Packet_list[k].protoType == 'TCP' and Packet_list[k].fl_ack == '1':
830                     if Packet_list[k].ip_src == exploreIP:
831                         cntOutput += 1
832                     if Packet_list[k].ip_dest == exploreIP:
833                         cntInput += 1
834             else:
835                 if Packet_list[k].port_src == port or Packet_list[k].port_dest == port:
836                     if Packet_list[k].protoType == 'TCP' and Packet_list[k].fl_ack == '1':
837                         if Packet_list[k].ip_src == exploreIP:
838                             cntOutput += 1
839                         if Packet_list[k].ip_dest == exploreIP:
840                             cntInput += 1
841             curTime += 1
842     diff_list.append(cntOutput - cntInput)
843     return diff_list
844
845
846     # Получение данных о частоте SYN-флагов
847     def get_syn_flags_freq(exploreIP, strt, fin, port):
848         cntSynTCP = 0
849         cntTCP = 0
850         rel_list = []
851         curTime = strt + 1
852         fin += 1
853         pos = 0
854         while curTime < fin:
855             for k in range(pos, len(Packet_list)):
856                 if Packet_list[k].timePacket > curTime:
857                     if cntTCP != 0:
858                         rel_list.append(cntSynTCP / cntTCP)
859                     else:
860                         rel_list.append(0.0)
861                     cntSynTCP = 0
862                     cntTCP = 0

```

```

863         pos = k
864         break
865     if port == None:
866         if Packet_list[k].ip_dest == exploreIP and Packet_list[k].protoType == 'TCP':
867             cntTCP += 1
868             if Packet_list[k].fl_syn == '1':
869                 cntSynTCP += 1
870         else:
871             if Packet_list[k].port_src == port or Packet_list[k].port_dest == port:
872                 if Packet_list[k].ip_dest == exploreIP and Packet_list[k].protoType == 'TCP':
873                     cntTCP += 1
874                     if Packet_list[k].fl_syn == '1':
875                         cntSynTCP += 1
876         curTime += 1
877     if cntTCP != 0:
878         rel_list.append(cntSynTCP / cntTCP)
879     else:
880         rel_list.append(0.0)
881     return rel_list
882
883
884     # Получение данных о частоте PSH-флагов
885     def get_psh_flags_freq(exploreIP, strt, fin, port):
886         cntPshTCP = 0
887         cntTCP = 0
888         rel_list = []
889         curTime = strt + 1
890         fin += 1
891         pos = 0
892         while curTime < fin:
893             for k in range(pos, len(Packet_list)):
894                 if Packet_list[k].timePacket > curTime:
895                     if cntTCP != 0:
896                         rel_list.append(cntPshTCP / cntTCP)
897                     else:
898                         rel_list.append(0.0)
899                     cntPshTCP = 0
900                     cntTCP = 0
901                     pos = k
902                     break
903             if port == None:
904                 if Packet_list[k].ip_dest == exploreIP and Packet_list[k].protoType == 'TCP':
905                     cntTCP += 1
906                     if Packet_list[k].fl_psh == '1':
907                         cntPshTCP += 1
908             else:
909                 if Packet_list[k].port_src == port or Packet_list[k].port_dest == port:
910                     if Packet_list[k].ip_dest == exploreIP and Packet_list[k].protoType == 'TCP':

```

```

911         cntTCP += 1
912         if Packet_list[k].fl_psh == '1':
913             cntPshTCP += 1
914         curTime += 1
915     if cntTCP != 0:
916         rel_list.append(cntPshTCP / cntTCP)
917     else:
918         rel_list.append(0.0)
919     return rel_list
920
921
922     # Получение данных о количестве пакетов и
923     # о максимумах пакетов в единицу времени
924     def get_pktamnt_and_size_persec(exploreIP, strt, fin, port):
925         pktAmntSrcList = []
926         pktAmntDstList = []
927         pktSizeSrcList = []
928         pktSizeDstList = []
929         curTime = strt + 1
930         fin += 1
931         pos = 0
932         while curTime < fin:
933             cntpktsrc = 0
934             cntpktdest = 0
935             maxpktsizesrc = 0
936             maxpktsizedst = 0
937             for k in range(pos, len(Packet_list)):
938                 if Packet_list[k].timePacket > curTime:
939                     pktAmntSrcList.append(cntpktsrc)
940                     pktAmntDstList.append(cntpktdest)
941                     pktSizeSrcList.append(maxpktsizesrc)
942                     pktSizeDstList.append(maxpktsizedst)
943                     pos = k
944                     break
945             if port == None:
946                 if Packet_list[k].ip_src == exploreIP:
947                     cntpktsrc += 1
948                     if maxpktsizesrc < Packet_list[k].packetSize:
949                         maxpktsizesrc = Packet_list[k].packetSize
950                 if Packet_list[k].ip_dest == exploreIP:
951                     cntpktdest += 1
952                     if maxpktsizedst < Packet_list[k].packetSize:
953                         maxpktsizedst = Packet_list[k].packetSize
954             else:
955                 if Packet_list[k].port_src == port or Packet_list[k].port_dest == port:
956                     if Packet_list[k].ip_src == exploreIP:
957                         cntpktsrc += 1
958                     if maxpktsizesrc < Packet_list[k].packetSize:

```

```

959         maxpktsizesrc = Packet_list[k].packetSize
960         if Packet_list[k].ip_dest == exploreIP:
961             cntpktdest += 1
962             if maxpktsizedst < Packet_list[k].packetSize:
963                 maxpktsizedst = Packet_list[k].packetSize
964         curTime += 1
965         pktAmntSrcList.append(cntpktsrc)
966         pktAmntDstList.append(cntpktdest)
967         pktSizeSrcList.append(maxpktsizesrc)
968         pktSizeDstList.append(maxpktsizedst)
969     return pktAmntSrcList, pktAmntDstList, pktSizeSrcList, pktSizeDstList
970
971
972     # Получение общей информации о трафике,
973     # связанном с выбранным IP-адресом
974     def get_inf_about_IP(exploreIP, port):
975         adjcPacketList = []
976         adjcIPList = set()
977         if port != None:
978             for p in Packet_list:
979                 if p.port_src == port or p.port_dest == port:
980                     if p.ip_src == exploreIP:
981                         adjcPacketList.append(p)
982                         adjcIPList.add(p.ip_dest)
983                     if p.ip_dest == exploreIP:
984                         adjcPacketList.append(p)
985                         adjcIPList.add(p.ip_src)
986         else:
987             for p in Packet_list:
988                 if p.ip_src == exploreIP:
989                     adjcPacketList.append(p)
990                     adjcIPList.add(p.ip_dest)
991                 if p.ip_dest == exploreIP:
992                     adjcPacketList.append(p)
993                     adjcIPList.add(p.ip_src)
994         return adjcPacketList, list(adjcIPList)
995
996
997     # Получение номера по IP-адресу
998     def get_pos_by_IP(curIP):
999         for i in range(len(Object_list)):
1000             if Object_list[i].ip == curIP:
1001                 return i
1002         return -1
1003
1004
1005     # Получение меток и "шага" для оси абсцисс
1006     def get_x_labels(total_time):

```

```

1007     global x_axisLabels
1008     step = 1
1009     if total_time > 600:
1010         step = 30
1011     elif total_time > 300:
1012         step = 10
1013     elif total_time > 50:
1014         step = 5
1015     x_axisLabels.clear()
1016     for i in range(0, len(Labels_list), step):
1017         x_axisLabels.append(Labels_list[i])
1018     return step
1019
1020
1021 # Получение второго IP-адреса
1022 def get_2nd_IP_for_plot(k):
1023     print('\nИзобразить на графике еще один объект. Выберите ' + \
1024           'IP-адрес для добавления (введите цифру)')
1025     print_list_of_pairs(Object_list[k].adjcIPList, True)
1026     scndIP = 'None'
1027     try:
1028         pos = int(input())
1029     except:
1030         print('Некорректный ввод!')
1031         return -1
1032     else:
1033         if pos < 0 or pos > len(Object_list[k].adjcIPList):
1034             print('Некорректный ввод!')
1035             return -1
1036         if pos != 0:
1037             scndIP = Object_list[k].adjcIPList[pos - 1]
1038     return scndIP
1039
1040
1041 # Выбор опций для выбранного IP-адреса
1042 def choose_options(k, strt, fin, step, port):
1043     curIP = Object_list[k].ip
1044     Object_list[k].adjcPacketList, Object_list[k].adjcIPList = get_inf_about_IP(curIP, port)
1045     Object_list[k].strt_time = time.localtime(Object_list[k].adjcPacketList[0].timePacket)
1046     Object_list[k].fin_time = time.localtime(Object_list[k].adjcPacketList[-1].timePacket)
1047     Object_list[k].amnt_packet = len(Object_list[k].adjcPacketList)
1048     totalTime = round( Object_list[k].adjcPacketList[-1].timePacket - \
1049                       Object_list[k].adjcPacketList[0].timePacket )
1050     if totalTime == 0:
1051         totalTime = 1
1052     Object_list[k].avg_packet_num = round(Object_list[k].amnt_packet / totalTime, 3)
1053     avgSize = 0
1054     for p in Object_list[k].adjcPacketList:

```

```

1055     avgSize += p.len_data
1056 Object_list[k].avg_packet_size = round(avgSize / Object_list[k].amnt_packet, 3)
1057 while True:
1058     print(f'Общая информация о трафике, связанном с {curIP}')
1059     print( 'Время первого перехваченного пакета: '
1060           , time.strftime('%d.%m.%Y г. %H:%M:%S', Object_list[k].strt_time) )
1061     print( 'Время последнего перехваченного пакета: '
1062           , time.strftime('%d.%m.%Y г. %H:%M:%S', Object_list[k].fin_time) )
1063     print('Общее время:', totalTime, 'сек.')
1064     print('Количество пакетов: ', Object_list[k].amnt_packet)
1065     print('Среднее количество пакетов в секунду: ', Object_list[k].avg_packet_num)
1066     print('Средний размер пакетов: ', Object_list[k].avg_packet_size)
1067     print(f'""Выберите опцию:
1068     1. Вывести весь трафик, связанный с {curIP}
1069     2. Построить график отношения входящего и исходящего трафиков
1070     3. Построить график отношения объема входящего UDP-трафика и объёма входящего TCP-трафика
1071     4. Построить график разности числа исходящих и числа входящих ACK-флагов в единицу времени
1072     5. Построить график частоты SYN и PSN флагов во входящих пакетах
1073     6. Построить график отображения количества пакетов в единицу времени
1074     7. Построить график отображения максимумов среди пакетов в единицу времени
1075     8. Вернуться к выбору IP-адреса """)
1076     b1 = input()
1077     if b1 == '1':
1078         print_adjacent_packets(Object_list[k].adjcPacketList)
1079
1080     elif b1 == '2':
1081         Object_list[k].in_out_rel_data = get_in_out_rel(curIP, strt, fin, port)
1082         x = [i for i in range(0, len(Object_list[k].in_out_rel_data))]
1083         x_labels = [i for i in range(0, len(x), step)]
1084         scndIP = get_2nd_IP_for_plot(k)
1085         if scndIP == -1:
1086             continue
1087         if scndIP != 'None':
1088             pos = get_pos_by_IP(scndIP)
1089             Object_list[pos].in_out_rel_data = get_in_out_rel(scndIP, strt, fin, port)
1090             fig = plt.figure(figsize=(16, 6), constrained_layout=True)
1091             f = fig.add_subplot()
1092             f.grid()
1093             f.set_title('Отношение объема входящего к объёму исходящего трафиков' + \
1094                       r' ($r_{in/out} = \frac{V_{in}}{V_{out}}$)', fontsize=15 )
1095             f.set_xlabel('Общее время перехвата трафика', fontsize=15)
1096             f.set_ylabel(r'$r_{in/out} = \frac{V_{in}}{V_{out}}$', fontsize=15)
1097             plt.plot(x, Object_list[k].in_out_rel_data, label=curIP)
1098             if scndIP != 'None':
1099                 plt.plot(x, Object_list[pos].in_out_rel_data, label=scndIP)
1100             plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=10)
1101             f.legend()
1102             plt.show()

```

```

1103 elif bl == '3':
1104     Object_list[k].udp_tcp_rel_data = get_udp_tcp_rel(curIP, strt, fin, port)
1105     x = [i for i in range(0, len(Object_list[k].udp_tcp_rel_data))]
1106     x_labels = [i for i in range(0, len(x), step)]
1107     scndIP = get_2nd_IP_for_plot(k)
1108     if scndIP == -1:
1109         continue
1110     if scndIP != 'None':
1111         pos = get_pos_by_IP(scndIP)
1112         Object_list[pos].udp_tcp_rel_data = get_udp_tcp_rel(scndIP, strt, fin, port)
1113         fig = plt.figure(figsize=(16, 6), constrained_layout=True)
1114         f = fig.add_subplot()
1115         f.grid()
1116         f.set_title( 'Отношение объема входящего UDP-трафика к объему ' +
1117                     'входящего TCP-трафика' + r' ($r_{in} = \frac{V_{udp}}{V_{tcp}}$)'
1118                     , fontsize=15 )
1119         f.set_xlabel('Общее время перехвата трафика', fontsize=15)
1120         f.set_ylabel(r'$r_{in} = \frac{V_{udp}}{V_{tcp}}$', fontsize=15)
1121         plt.plot(x, Object_list[k].udp_tcp_rel_data, label=curIP)
1122         if scndIP != 'None':
1123             plt.plot(x, Object_list[pos].udp_tcp_rel_data, label=scndIP)
1124         plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=10)
1125         f.legend()
1126         plt.show()
1127 elif bl == '4':
1128     Object_list[k].ack_flags_diff_data = get_ack_flags_diff(curIP, strt, fin, port)
1129     x = [i for i in range(0, len(Object_list[k].ack_flags_diff_data))]
1130     x_labels = [i for i in range(0, len(x), step)]
1131     scndIP = get_2nd_IP_for_plot(k)
1132     if scndIP == -1:
1133         continue
1134     if scndIP != 'None':
1135         pos = get_pos_by_IP(scndIP)
1136         Object_list[pos].ack_flags_diff_data = get_ack_flags_diff(scndIP, strt, fin, port)
1137         fig = plt.figure(figsize=(16, 6), constrained_layout=True)
1138         f = fig.add_subplot()
1139         f.grid()
1140         f.set_title('Разность числа исходящих и числа входящих АСК-флагов' + \
1141                     r' ($r_{ack} = V_{A_{out}} - V_{A_{in}}$)', fontsize=15)
1142         f.set_xlabel('Общее время перехвата трафика', fontsize=15)
1143         f.set_ylabel(r'$r_{ack} = V_{A_{out}} - V_{A_{in}}$', fontsize=15)
1144         plt.plot(x, Object_list[k].ack_flags_diff_data, label=curIP)
1145         if scndIP != 'None':
1146             plt.plot(x, Object_list[pos].ack_flags_diff_data, label=scndIP)
1147         plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=10)
1148         f.legend()
1149         plt.show()
1150 elif bl == '5':

```

```

1151 data = get_syn_flags_freq(curIP, strt, fin, port)
1152 Object_list[k].syn_flags_freq_data = data
1153 data = get_psh_flags_freq(curIP, strt, fin, port)
1154 Object_list[k].psh_flags_freq_data = data
1155 x = [i for i in range(0, len(Object_list[k].syn_flags_freq_data))]
1156 x_labels = [i for i in range(0, len(x), step)]
1157 scndIP = get_2nd_IP_for_plot(k)
1158 if scndIP == -1:
1159     continue
1160 if scndIP != 'None':
1161     pos = get_pos_by_IP(scndIP)
1162     data = get_syn_flags_freq(scndIP, strt, fin, port)
1163     Object_list[pos].syn_flags_freq_data = data
1164     data = get_psh_flags_freq(scndIP, strt, fin, port)
1165     Object_list[pos].psh_flags_freq_data = data
1166 fig = plt.figure(figsize=(16, 6), constrained_layout=True)
1167 gs = gridspec.GridSpec(ncols=1, nrows=2, figure=fig)
1168 fig_1 = fig.add_subplot(gs[0, 0])
1169 fig_1.grid()
1170 fig_1.set_title('Частота флагов SYN' + \
1171                r' ($r_{syn} = \frac{V_{S_{in}}}{V_{tcp}}$)', fontsize=15)
1172 fig_1.set_xlabel('Общее время перехвата трафика', fontsize=15)
1173 fig_1.set_ylabel(r'$r_{syn} = \frac{V_{S_{in}}}{V_{tcp}}$', fontsize=15)
1174 plt.plot(x, Object_list[k].syn_flags_freq_data, 'b', label=curIP)
1175 if scndIP != 'None':
1176     plt.plot(x, Object_list[pos].syn_flags_freq_data, 'r', label=scndIP)
1177 plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=8)
1178 fig_1.legend()
1179 fig_2 = fig.add_subplot(gs[1, 0])
1180 fig_2.grid()
1181 plt.plot(x, Object_list[k].psh_flags_freq_data, 'orange', label=curIP)
1182 fig_2.set_title('Частота флагов PSH' + \
1183                r' ($r_{psh} = \frac{V_{P_{in}}}{V_{tcp}}$)', fontsize=15)
1184 fig_2.set_xlabel('Общее время перехвата трафика', fontsize=15)
1185 fig_2.set_ylabel(r'$r_{psh} = \frac{V_{P_{in}}}{V_{tcp}}$', fontsize=15)
1186 if scndIP != 'None':
1187     plt.plot(x, Object_list[pos].psh_flags_freq_data, 'g', label=scndIP)
1188 plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=8)
1189 fig_2.legend()
1190 plt.show()
1191 elif bl == '6':
1192     d1, d2, d3, d4 = get_pktamnt_and_size_persec(curIP, strt, fin, port)
1193     Object_list[k].pkt_amnt_src_data = d1
1194     Object_list[k].pkt_amnt_dst_data = d2
1195     Object_list[k].pkt_size_data_src = d3
1196     Object_list[k].pkt_size_data_dst = d4
1197     x = [i for i in range(0, len(Object_list[k].pkt_amnt_src_data))]
1198     x_labels = [i for i in range(0, len(x), step)]

```



```

1199     scndIP = get_2nd_IP_for_plot(k)
1200     if scndIP == -1:
1201         continue
1202     if scndIP != 'None':
1203         pos = get_pos_by_IP(scndIP)
1204         d1, d2, d3, d4 = get_pktamnt_and_size_persec(scndIP, strt, fin, port)
1205         Object_list[pos].pkt_amnt_src_data = d1
1206         Object_list[pos].pkt_amnt_dst_data = d2
1207         Object_list[pos].pkt_size_data_src = d3
1208         Object_list[pos].pkt_size_data_dst = d4
1209     fig = plt.figure(figsize=(16, 6), constrained_layout=True)
1210     gs = gridspec.GridSpec(ncols=1, nrows=2, figure=fig)
1211     fig_1 = fig.add_subplot(gs[0, 0])
1212     fig_1.grid()
1213     fig_1.set_title('Количество входящих пакетов, полученных за ' + \
1214                    'единицу времени', fontsize=15)
1215     fig_1.set_xlabel('Общее время перехвата трафика', fontsize=15)
1216     plt.plot(x, Object_list[k].pkt_amnt_dst_data, 'b', label=curIP)
1217     if scndIP != 'None':
1218         plt.plot(x, Object_list[pos].pkt_amnt_dst_data, 'r', label=scndIP)
1219     plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=8)
1220     fig_1.legend()
1221     fig_2 = fig.add_subplot(gs[1, 0])
1222     fig_2.grid()
1223     plt.plot(x, Object_list[k].pkt_amnt_src_data, 'orange', label=curIP)
1224     fig_2.set_title('Количество исходящих пакетов, полученных за ' + \
1225                    'единицу времени', fontsize=15)
1226     fig_2.set_xlabel('Общее время перехвата трафика', fontsize=15)
1227     if scndIP != 'None':
1228         plt.plot(x, Object_list[pos].pkt_amnt_src_data, 'g', label=scndIP)
1229     plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=8)
1230     fig_2.legend()
1231     plt.show()
1232 elif bl == '7':
1233     d1, d2, d3, d4 = get_pktamnt_and_size_persec(curIP, strt, fin, port)
1234     Object_list[k].pkt_amnt_src_data = d1
1235     Object_list[k].pkt_amnt_dst_data = d2
1236     Object_list[k].pkt_size_data_src = d3
1237     Object_list[k].pkt_size_data_dst = d4
1238     x = [i for i in range(0, len(Object_list[k].pkt_size_data_src))]
1239     x_labels = [i for i in range(0, len(x), step)]
1240     scndIP = get_2nd_IP_for_plot(k)
1241     if scndIP == -1:
1242         continue
1243     if scndIP != 'None':
1244         pos = get_pos_by_IP(scndIP)
1245         d1, d2, d3, d4 = get_pktamnt_and_size_persec(scndIP, strt, fin, port)
1246         Object_list[pos].pkt_amnt_src_data = d1

```

```

1247     Object_list[pos].pkt_amnt_dst_data = d2
1248     Object_list[pos].pkt_size_data_src = d3
1249     Object_list[pos].pkt_size_data_dst = d4
1250     fig = plt.figure(figsize=(16, 6), constrained_layout=True)
1251     gs = gridspec.GridSpec(ncols=1, nrows=2, figure=fig)
1252     fig_1 = fig.add_subplot(gs[0, 0])
1253     fig_1.grid()
1254     fig_1.set_title('Максимальный размер входящих пакетов, полученных за ' + \
1255                    'единицу времени', fontsize=15)
1256     fig_1.set_xlabel('Общее время перехвата трафика', fontsize=15)
1257     plt.plot(x, Object_list[k].pkt_size_data_dst, 'b', label=curIP)
1258     if scndIP != 'None':
1259         plt.plot(x, Object_list[pos].pkt_size_data_dst, 'r', label=scndIP)
1260     plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=8)
1261     fig_1.legend()
1262     fig_2 = fig.add_subplot(gs[1, 0])
1263     fig_2.grid()
1264     plt.plot(x, Object_list[k].pkt_size_data_src, 'orange', label=curIP)
1265     fig_2.set_title('Максимальный размер исходящих пакетов, полученных за ' + \
1266                    'единицу времени', fontsize=15)
1267     fig_2.set_xlabel('Общее время перехвата трафика', fontsize=15)
1268     if scndIP != 'None':
1269         plt.plot(x, Object_list[pos].pkt_size_data_src, 'g', label=scndIP)
1270     plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=8)
1271     fig_2.legend()
1272     plt.show()
1273 elif bl == '8':
1274     break
1275
1276
1277 # Выбор опции (меню)
1278 def choose_mode():
1279     global Packet_list, Object_list, Labels_list, Session_list, findRDP
1280     while True:
1281         print('1. Перехват трафика')
1282         print('2. Запись данных в файл')
1283         print('3. Считывание с файла данных для анализа трафика')
1284         print('4. Анализ трафика')
1285         print('5. Выход')
1286         bl = input()
1287         if bl == '1':
1288             Packet_list.clear()
1289             Object_list.clear()
1290             Labels_list.clear()
1291             Session_list.clear()
1292             findRDP = False
1293             print('Поставить фильтр RDP? (Если да, то введите 1)')
1294             fl = input('Ответ: ')

```

```

1295     if fl == '1':
1296         findRDP = True
1297     try:
1298         print('\nВыберите сетевой интерфейс, нажав соответствующую цифру:')
1299         print(socket.if_nameindex())
1300         interface = int(input())
1301         if 0 > interface or interface > len(socket.if_nameindex()):
1302             print('\nОшибка ввода!!!\n')
1303             return
1304         os.system(f'ip link set {socket.if_indextoname(interface)} promisc on')
1305         s_listen = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
1306     except PermissionError:
1307         print('\nНедостаточно прав!')
1308         print('Запустите программу от имени администратора!')
1309         return
1310     else:
1311         print('\nНачался процесс захвата трафика...\n')
1312         start_to_listen(s_listen)
1313     print(f'\nДанные собраны. Перехвачено: {len(Packet_list)} пакетов(-a)\n')
1314
1315     print('\nХотите записать перехваченный трафик в файл? (да - нажмите 1)')
1316     bl1 = input('Ответ: ')
1317     if '1' in bl1:
1318         print('Введите название файла (например: data.log)')
1319         FileName = input()
1320         try:
1321             f = open(FileName, 'w')
1322         except:
1323             print('\nНекорректное название файла!\n')
1324             continue
1325         if write_to_file(f):
1326             print(f'\nВ файл {FileName} была успешно записана информация.\n')
1327             f.close()
1328         else:
1329             print(f'\nОшибка записи в файл {FileName}! Возможно нет данных для записи\n')
1330             f.close()
1331     print('')
1332     elif bl == '2':
1333         if Packet_list == []:
1334             print('\nНет данных! Сначала необходимо получить данные!\n')
1335             continue
1336         print('Введите название файла (например: data.log)')
1337         FileName = input()
1338         try:
1339             f = open(FileName, 'w')
1340         except:
1341             print('\nНекорректное название файла!\n')
1342             continue

```

```

1343     if write_to_file(f):
1344         print(f'\nВ файл {FileName} была успешно записана информация.\n')
1345         f.close()
1346     else:
1347         print(f'\nОшибка записи в файл {FileName}! Возможно нет данных для записи...\n')
1348         f.close()
1349         continue
1350 elif bl == '3':
1351     Packet_list.clear()
1352     Object_list.clear()
1353     Labels_list.clear()
1354     Session_list.clear()
1355     print('Введите название файла (например: data.log)')
1356     FileName = input()
1357     if not Packet_list:
1358         try:
1359             f = open(FileName, 'r')
1360         except:
1361             print('\nНекорректное название файла!\n')
1362             continue
1363         while True:
1364             inf = f.readline()
1365             if not inf:
1366                 break
1367             read_from_file(inf)
1368         f.close()
1369     print(f'\nДанные собраны. Перехвачено: {len(Packet_list)} пакетов(-а)\n')
1370 elif bl == '4':
1371     if Packet_list == []:
1372         print('\nНет данных! Сначала необходимо получить данные!\n')
1373         continue
1374     IPList, numPacketsPerSec = get_common_data()
1375     clear_end_sessions()
1376     for s in Session_list:
1377         s.fin_rdp_check()
1378     print_inf_about_sessions()
1379     strt = Packet_list[0].timePacket
1380     fin = Packet_list[-1].timePacket
1381     strt_time = time.localtime(strt)
1382     fin_time = time.localtime(fin)
1383     avgNumPacket = 0
1384     for el in numPacketsPerSec:
1385         avgNumPacket += el
1386     avgNumPacket /= len(numPacketsPerSec)
1387     avgSizePacket = 0
1388     for p in Packet_list:
1389         avgSizePacket += p.packetSize
1390     avgSizePacket /= len(Packet_list)

```

```

1391
1392     step = get_x_labels(int(fin - strt))
1393     print('Общая информация:')
1394     print('Время первого перехваченного пакета: '
1395           , time.strftime('%d.%m.%Y г. %H:%M:%S', strt_time) )
1396     print('Время последнего перехваченного пакета: '
1397           , time.strftime('%d.%m.%Y г. %H:%M:%S', fin_time) )
1398     print('Количество пакетов: ', len(Packet_list))
1399     print('Общее время перехвата: ', round(fin - strt, 3), 'сек')
1400     print('Среднее количество пакетов в секунду: ', round(avgNumPacket, 3))
1401     print('Средний размер пакетов: ', round(avgSizePacket, 3))
1402     print('Завершить просмотр (нажмите \"q\" для выхода)')
1403     for k in range(len(IPList)):
1404         Object_list.append(ExploreObject(IPList[k]))
1405         Object_list[-1].commonPorts = get_common_ports(IPList[k])
1406     print_list_of_pairs(IPList)
1407     print(f'\nВыберите цифру (0 - {len(IPList) - 1}) для просмотра IP-адреса:')
1408     k = input()
1409     if k == 'q':
1410         break
1411     try:
1412         k = int(k)
1413     except:
1414         print('\nНекорректный ввод!\n')
1415         continue
1416     else:
1417         if 0 <= k and k < len(IPList):
1418             port = None
1419             print('Список портов которые участвовали в соединении с данным IP-адресом')
1420             print_list_of_pairs(Object_list[k].commonPorts, True)
1421             t = len(Object_list[k].commonPorts)
1422             print(f'\nВыберите цифру (0 - {t}) для выбора порта:')
1423             k1 = input()
1424             if k1 == 'q':
1425                 break
1426             try:
1427                 k1 = int(k1)
1428             except:
1429                 print('Некорректный ввод!\n')
1430                 continue
1431             else:
1432                 if 0 <= k1 and k1 <= t:
1433                     if k1 != 0:
1434                         port = Object_list[k].commonPorts[k1 - 1]
1435                         choose_options(k, strt, fin, step, port)
1436                     else:
1437                         print(f'Введите число в пределах 0 - {t - 1}')
1438             else:

```

```
1439         print(f'Введите число в пределах 0 - {len(IPList) - 1}')
1440     elif b1 == '5':
1441         return
1442
1443
1444 if __name__ == '__main__':
1445     print('\nЗапуск программы...\n')
1446     choose_mode()
```