

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 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Обзор существующих методов обнаружения RDP-трафика	4
2 Программная реализация метода обнаружения RDP-трафика	7
2.1 Определение активных сессий путем анализа TCP-соединения	12
2.2 Обработка данных и построение графиков для анализа поведения RDP-трафика	16
3 Анализ распределения размера пакетов	18
3.1 Вычисление среднего значения и стандартного отклонения размеров пакетов	18
3.2 Определение пороговых значений для интервалов	19
3.3 Анализ полученных данных на наличие признаков RDP-сессии	19
4 Анализ распределения временных интервалов между пакетами	23
4.1 Вычисление среднего значения и стандартного отклонения интервалов	23
4.2 Определение пороговых значений для интервалов	24
4.3 Анализ полученных данных на наличие признаков RDP-сессии	24
5 Анализ частоты флагов PSH	25
5.1 Расчет частоты флагов PSH для каждого интервала времени	25
5.2 Анализ полученных данных на наличие признаков RDP-сессии	26
6 Некоторые модификации для улучшения обнаружения RDP-сессии	28
7 Тестирование программы на определение наличия или отсутствия RDP-сессии	32
ЗАКЛЮЧЕНИЕ	37
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	38
Приложение А Код traffic-detection.py	39

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 Обзор существующих методов обнаружения RDP-трафика

RDP (Remote Desktop Protocol) — это протокол удаленного рабочего стола, который используется для удаленного управления компьютерами. Протокол RDP позволяет пользователям подключаться к удаленному компьютеру, используя протокол TCP/IP и передавать данные через сеть.

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 PSH 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 ис-

пользует 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.

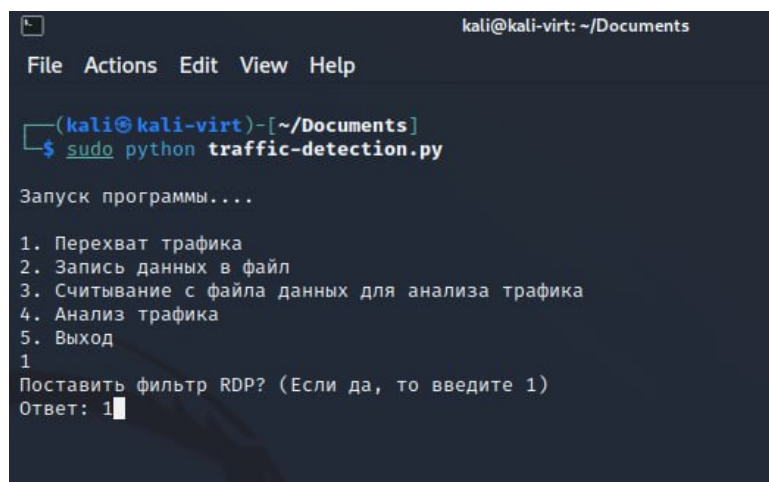
Стоит отметить, что каждый из методов анализа временных интервалов, размеров пакетов и флагов пакетов имеет свои собственные недостатки. Тем не менее, все три метода могут быть реализованы совместно. В данной работе была

создана программа, которая включает все три метода. Далее будет рассмотрен каждый метод более подробно, но перед этим необходимо рассказать немного о самой программе.

2 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 1.



```
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 1 – 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 2 – 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 3.

```
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 3 – 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 4.

```
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 4 – 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 5.

```
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 5 – 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 АСК имеет значение равное единице.

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

где $V_{A_{in}}$ и $V_{A_{out}}$ — число входящих и исходящих АСК-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

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

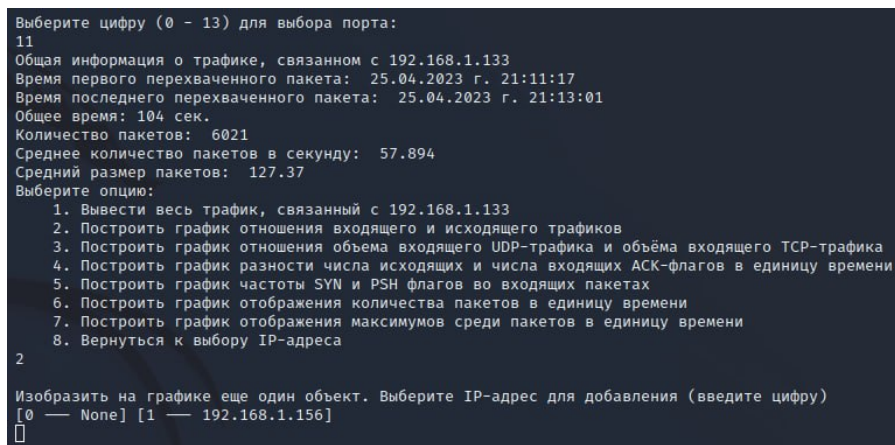
где V_{Pin} число входящих ТСП-пакетов, в которых установлен флаг PSH = 1, V_{tcp} — число входящих ТСП-пакетов в единицу времени. Флаг PSH (Push) в ТСП-заголовке используется для указания конечной точке передачи данных о том, что все буферизованные данные должны быть немедленно отправлены получателю, 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 6. двух



```
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 6 – 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.

2.1 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 7 – 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 8 – 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 9-10 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 9 – 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 10 – 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.

2.2 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. В дальнейшем данные операционные системы будем обозначать как Win и Kali соответственно. В эксперименте всегда участвовали 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.

3 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.

3.1 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.

3.2 Определение пороговых значений для интервалов

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.

3.3 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 11 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:

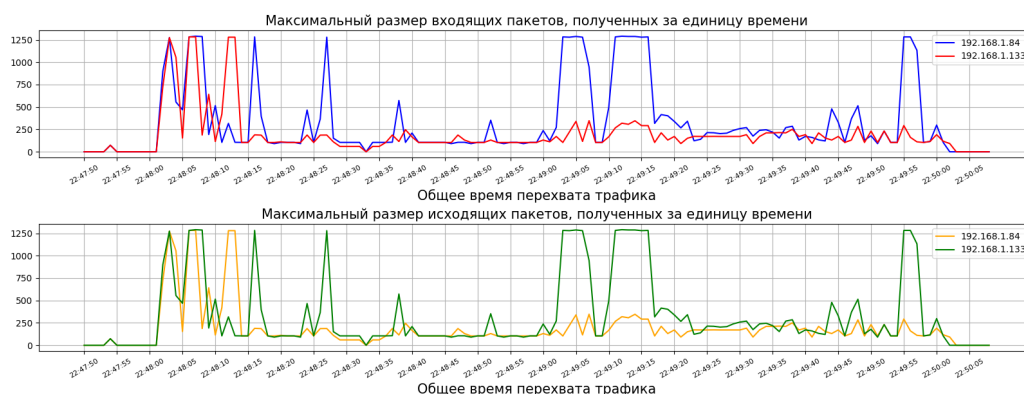


Рисунок 11 – График отображения максимумов среди пакетов, рассчитанных в единицу времени (соединение Win - Win)

- 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 12 - 14. 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.

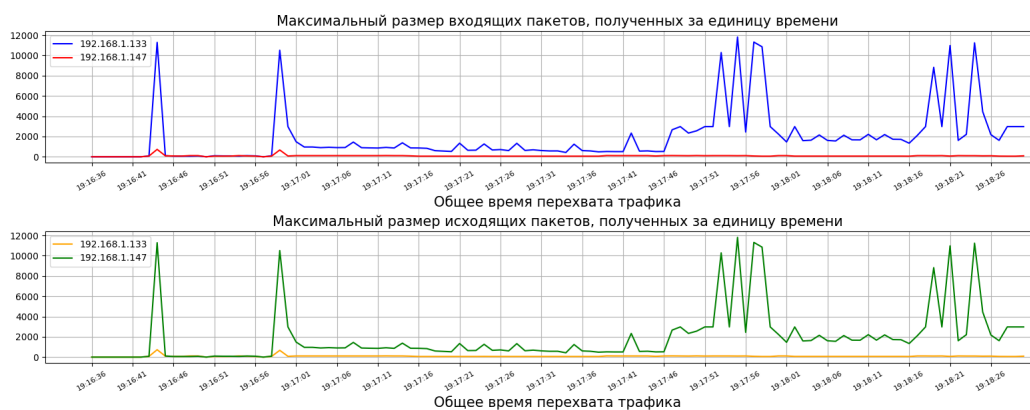


Рисунок 12 – График отображения максимумов среди пакетов в единицу времени (соединение Win - Kali)



Рисунок 13 – График отображения максимумов среди пакетов в единицу времени (соединение Kali - Win)

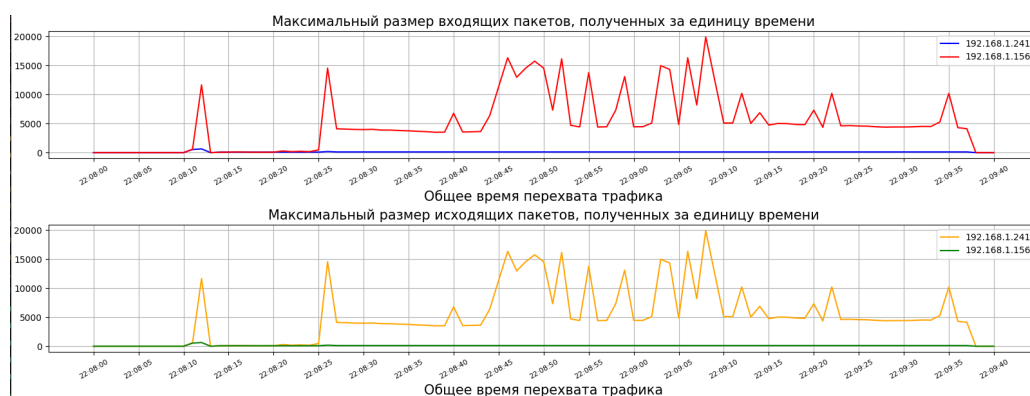


Рисунок 14 – График отображения максимумов среди пакетов в единицу времени (соединение Kali - Kali)

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.

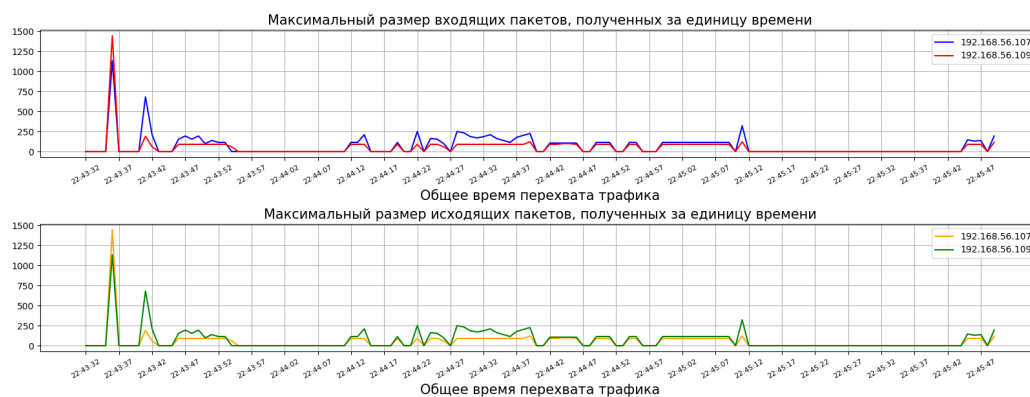


Рисунок 15 – График отображения максимумов среди пакетов в единицу времени (подключение по SSH)

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

4 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.

4.1 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.

4.2 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.

4.3 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.

5 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.

5.1 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.

Коэффициент модуля разности μ_{psh} и cur_{psh} был рассчитан исходя из данных соединений. Можно заметить, что на рисунке 17 при соединении Win - Kali из-за такого коэффициента на некоторых промежутках времени программа не сможет более точно определить, есть ли в данном интервале мышиные или клавиатурные события.

5.2 Анализ полученных данных на наличие признаков RDP-сессии

Просматривая каждый TCP-пакет, программа проверяет наличие установленного флага PSH 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 PSH-флагом.

На рисунках 16 - 19 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 16 – График частоты PSH флагов (соединение Win - Win)

На примере рисунка 17 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 17 – График частоты PSH флагов (соединение Win - Kali)

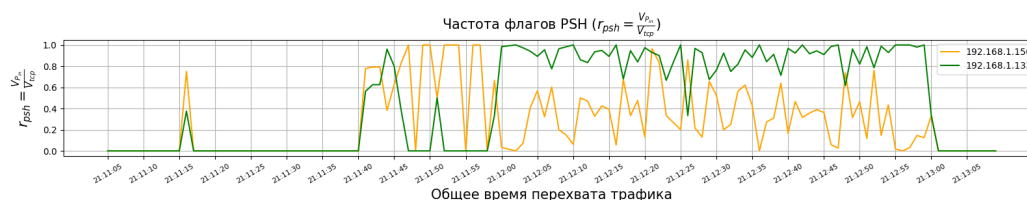


Рисунок 18 – График частоты PSN флагов (соединение Kali - Win)

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 19 видно, что частота флагов PSN не равна нулю. В этом временном интервале передается фиксированное количество PSN-флагов.



Рисунок 19 – График частоты PSN 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.

6 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 20 - 23 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 20 – График отношения объема входящего и исходящего трафика (соединение Win - Win)

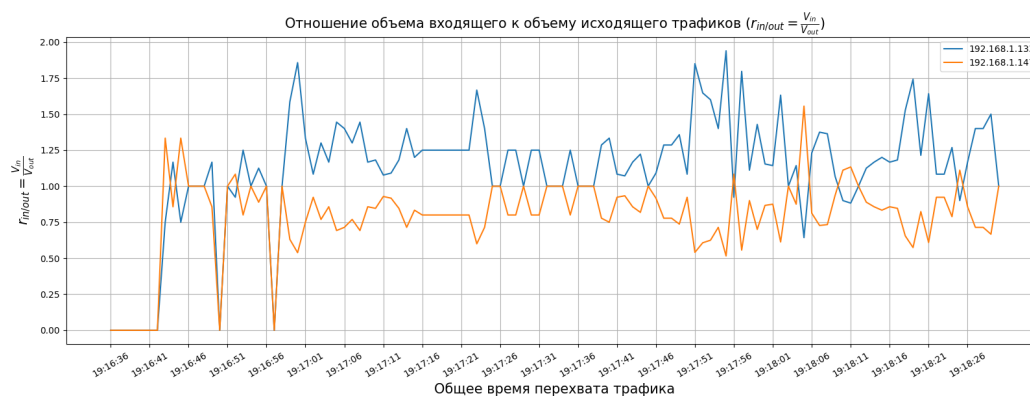


Рисунок 21 – График отношения объема входящего и исходящего трафика (соединение Win - Kali)

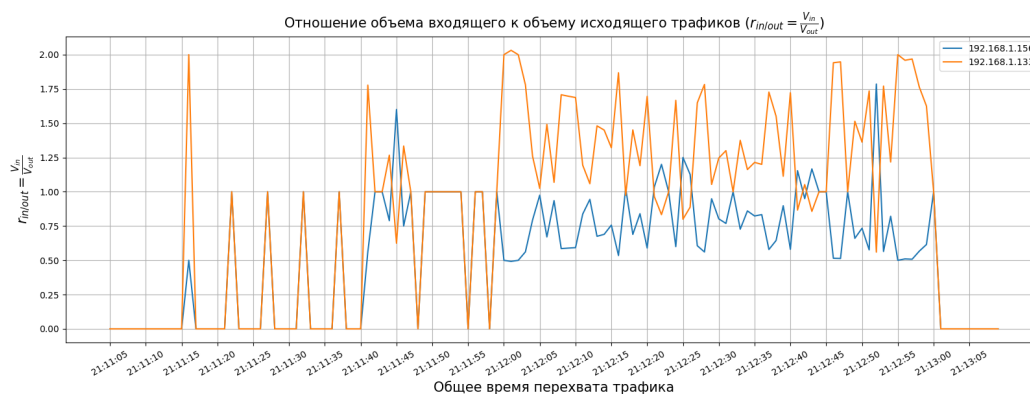


Рисунок 22 – График отношения объема входящего и исходящего трафика (соединение Kali - Win)

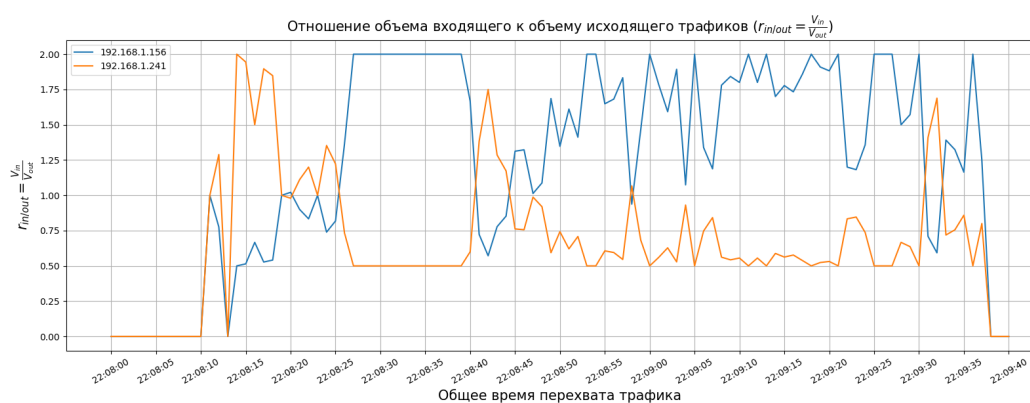


Рисунок 23 – График отношения объема входящего и исходящего трафика (соединение Kali - Kali)

Как можно заметить из рисунков 20 - 23, 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 24 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.

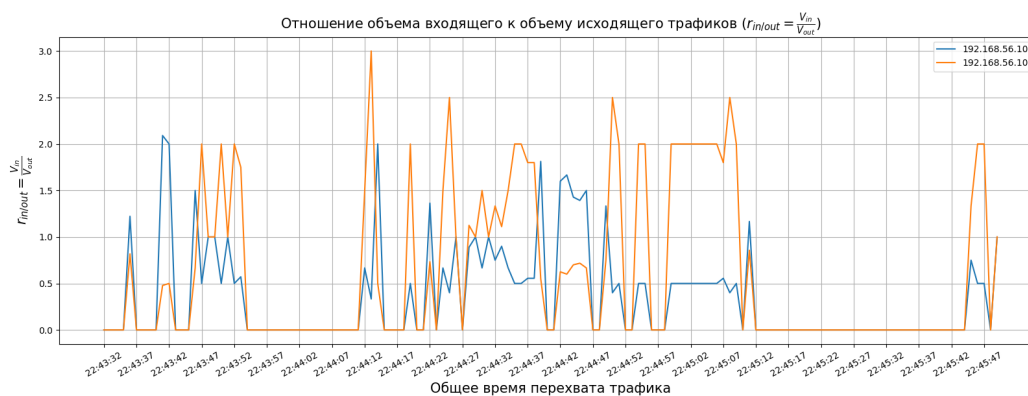


Рисунок 24 – График отношения объема входящего и исходящего трафика (подключение по SSH)

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

Таким образом данная модификация 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 вычисляются 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.

7 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).

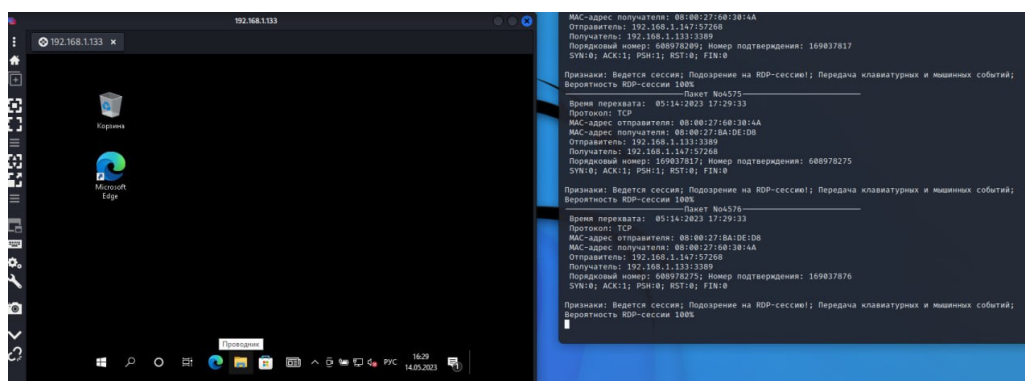


Рисунок 26 – Работа программы при установке RDP-сессии (соединение Kali - Win)

Из рисунка 26 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 27 изображено изменение значения стандартного RDP-порта на 13389. Это операция делается в редакторе реестра Windows. Информацию о том, как это можно сделать описанна в документации Microsoft [7].

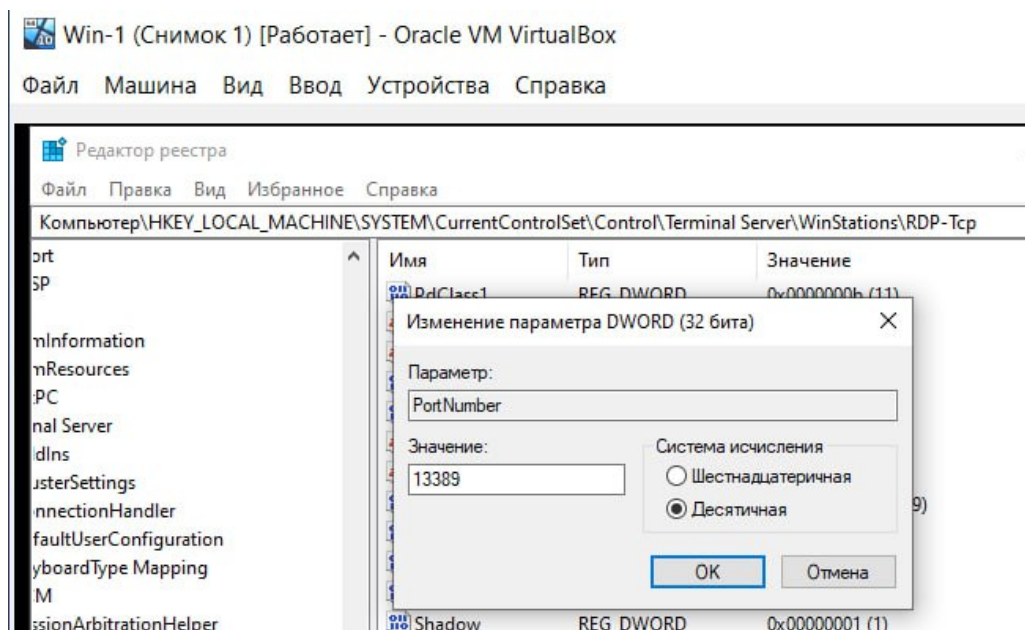


Рисунок 27 – Изменение номера порта по умолчанию на порт 13389

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.

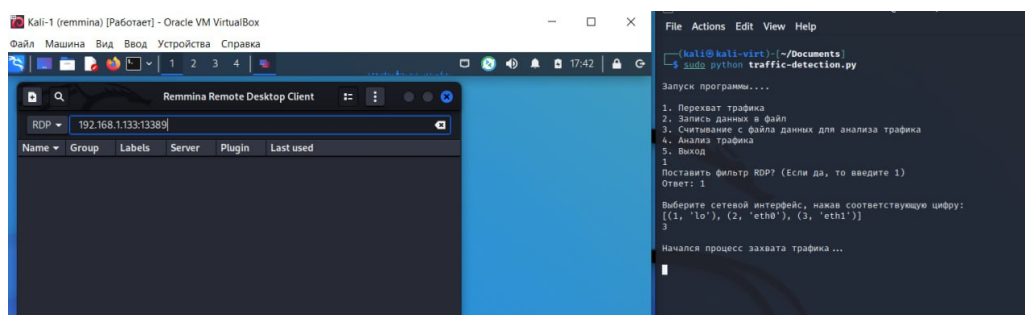


Рисунок 28 – Установка RDP-сессии по порту 13389

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 29 показана вероятность RDP-сессии на текущий момент времени.

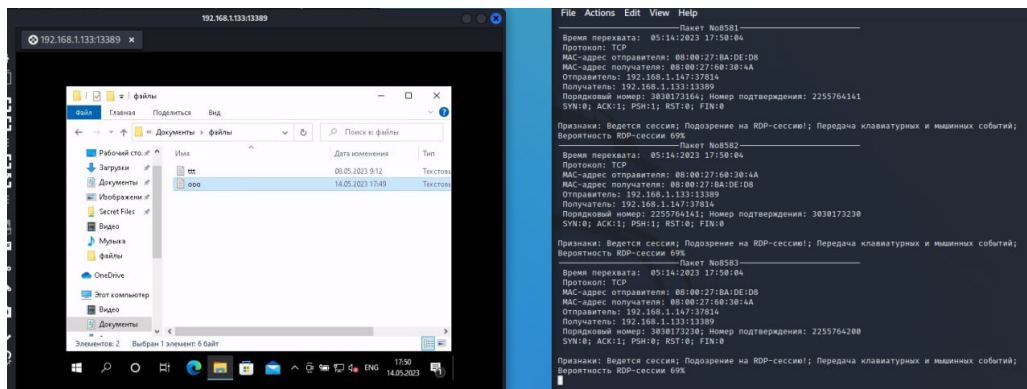


Рисунок 29 – Демонстрация работы программы при активной RDP-сессии

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.

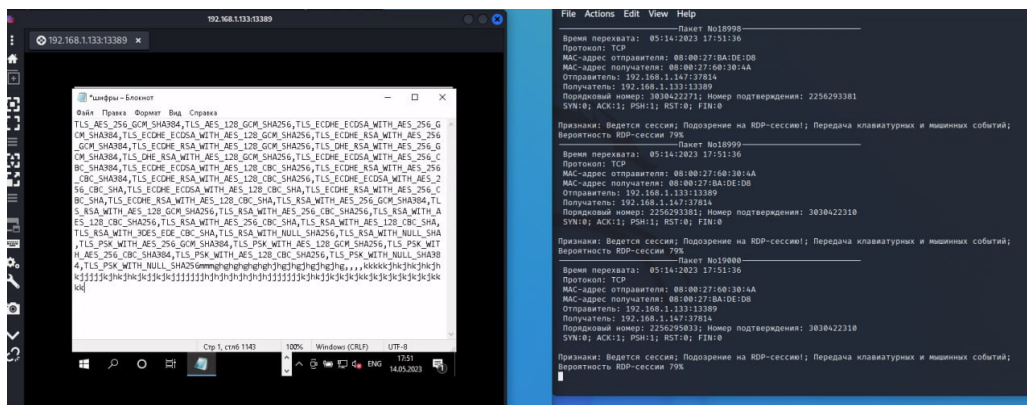


Рисунок 30 – Увеличение вероятности RDP-сессии

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 31.

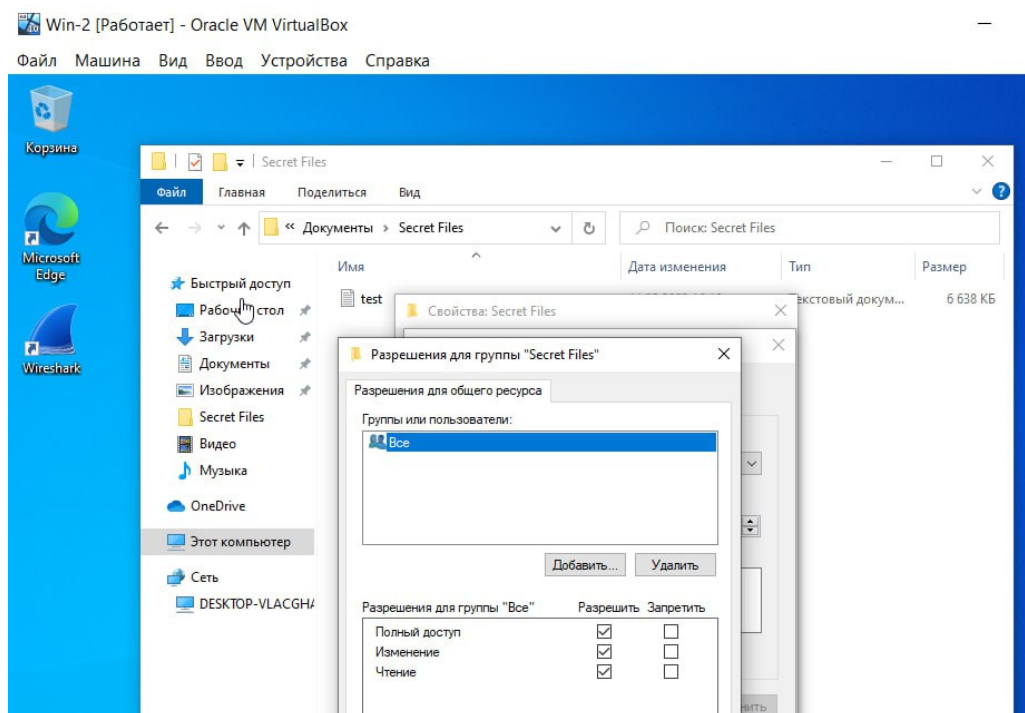


Рисунок 31 – Предоставление папке общего доступа

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

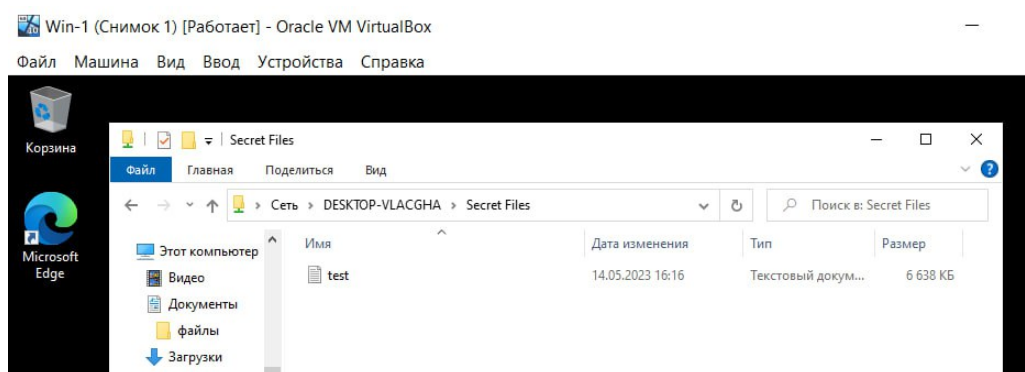


Рисунок 32 – Рабочий стол VM Windows 10 (192.168.1.133)

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 33. 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.

```
kali@kali-virt: ~/Documents
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 33 – Информация об установке SMB-сессии.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы...

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 «TCP flags» [Электронный ресурс] / URL: <https://www.keycdn.com/support/tcp-flags#:~:text=ACK> (дата обращения 31.03.2023), Яз. англ.
- 5 Документация по стандартным библиотекам языка Python [Электронный ресурс] / URL: <https://docs.python.org/3/library/socket.html> (дата обращения 31.03.2023), Яз. англ.
- 6 Статья «Работа с клиентом удаленного рабочего стола Remmina» [Электронный ресурс] / URL: <https://white55.ru/remmina.html> (дата обращения 15.04.2023), Яз. рус.
- 7 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.

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 мышиных 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 class Session:
77
78     def __init__(self, strtTime, init, target, port):
79         self.fl_syn = True
80         self.fl_fin = False
81         self.fl_rst = False
82         self.strtTime = strtTime
83         self.curTime = strtTime + 5
84         self.curSec = strtTime + 1
85         self.finTime = None
86         self.totalTime = None
87         self.initiator = init
88         self.target = target
89         self.port = port
90         self.seq_num = None
91         self.ack_num = None
92         self.is_rdp = False
93         self.is_rdpArr = []
94         self.cntTr = 0

```



```

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

```

```

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

```

```

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

```

```

239         self.trafficInit.append(self.cntInitIn / self.cntInitOut)
240     else:
241         self.trafficInit.append(0.0)
242     if self.cntTargOut != 0:
243         self.trafficTarg.append(self.cntTargIn / self.cntTargOut)
244     else:
245         self.trafficTarg.append(0.0)
246     self.cntInitIn = 0
247     self.cntTargIn = 0
248     self.cntInitOut = 0
249     self.cntTargOut = 0
250     self.curSec += 1
251     if pkt.ip_src == self.initiator:
252         self.cntInitOut += 1
253     if pkt.ip_dest == self.initiator:
254         self.cntInitIn += 1
255     if pkt.ip_src == self.target:
256         self.cntTargOut += 1
257     if pkt.ip_dest == self.target:
258         self.cntTargIn += 1
259
260
261 def rdpArr_check(self):
262     l = len(self.is_rdpArr)
263     if l > 2:
264         return self.cntTr > l - self.cntTr
265     else:
266         return False
267
268
269 def get_average_val(self):
270     n = len(self.pshfreq)
271     if n >= 4:
272         return (self.pshfreq[n - 4] + self.pshfreq[n - 3] + \
273               self.pshfreq[n - 2]) / 3
274     return -10
275
276
277 def rdp_check(self):
278     if self.port == '3389':
279         self.is_rdpArr.append(True)
280         self.cntTr += 1
281         self.prob = 100
282     elif self.prob > 70:
283         self.is_rdpArr.append(True)
284         self.cntTr += 1
285     else:
286         if (self.is_rdpInOut and self.is_rdpIntvl) or \

```

```

287         (self.is_rdpInOut and self.is_rdpPSH and self.is_rdpDev):
288             self.is_rdpArr.append(True)
289             self.cntTr += 1
290         else:
291             if (self.is_rdpInOut or self.is_rdpIntvl):
292                 if (self.is_rdpDev and self.rdpArr_check()) or \
293                     (not self.is_rdpDev and self.rdpArr_check()):
294                     self.is_rdpArr.append(True)
295                     self.cntTr += 1
296                 else:
297                     self.is_rdpArr.append(False)
298             else:
299                 self.is_rdpArr.append(False)
300             if len(self.is_rdpArr) > 4:
301                 self.prob = round((self.cntTr / len(self.is_rdpArr)) * 100)
302
303
304     def fin_rdp_check(self):
305         cnt = 0
306         for el in self.is_rdpArr:
307             if el:
308                 cnt += 1
309         self.is_rdp = (False, True)[cnt > len(self.is_rdpArr) - cnt]
310
311
312     # Получение ethernet-кадра
313     def get_ethernet_frame(data):
314         dest_mac, src_mac, proto = struct.unpack('!6s6sH', data[:14])
315         return get_mac_addr(dest_mac), get_mac_addr(src_mac), socket.htons(proto)
316
317
318     # Получение MAC-адреса
319     def get_mac_addr(mac_bytes):
320         mac_str = ''
321         for el in mac_bytes:
322             mac_str += format(el, '02x').upper() + ':'
323         return mac_str[:len(mac_str) - 1]
324
325
326     # Получение IPv4-заголовка
327     def get_ipv4_data(data):
328         version_header_length = data[0]
329         header_length = (version_header_length & 15) * 4
330         ttl, proto, src, dest = struct.unpack('!8xBB2x4s4s', data[:20])
331         return ttl, proto, ipv4_dec(src), ipv4_dec(dest), data[header_length:]
332
333
334     # Получение IP-адреса формата X.X.X.X

```

```

335 def ipv4_dec(ip_bytes):
336     ip_str = ''
337     for el in ip_bytes:
338         ip_str += str(el) + '.'
339     return ip_str[:-1]
340
341
342 # Получение UDP-сегмента данных
343 def get_udp_segment(data):
344     src_port, dest_port, size = struct.unpack('!HH2xH', data[:8])
345     return str(src_port), str(dest_port), size, data[8:]
346
347
348 # Получение TCP-сегмента данных
349 def get_tcp_segment(data):
350     src_port, dest_port, sequence, ack, some_block = struct.unpack('!HLLH', data[:14])
351     return str(src_port), str(dest_port), str(sequence), str(ack), \
352         some_block, data[(some_block >> 12) * 4:]
353
354
355 # Форматирование данных для корректного представления
356 def format_data(data):
357     if isinstance(data, bytes):
358         data = ''.join(r'\x{:02x}'.format(el) for el in data)
359     return data
360
361
362 # Перехват трафика и вывод информации в консоль
363 def start_to_listen(s_listen):
364     global Packet_list
365     NumPacket = 1
366     curcnt = 1000
367     while True:
368         # Получение пакетов в виде набора hex-чисел
369         raw_data, _ = s_listen.recvfrom(65565)
370         pinfo = [''] * 18
371         pinfo[0], pinfo[1] = NumPacket, time.time()
372         pinfo[2] = len(raw_data)
373         # Если это интернет-протокол четвертой версии
374         pinfo[4], pinfo[3], protocol = get_ethernet_frame(raw_data)
375         if protocol == 8:
376             _, proto, pinfo[6], pinfo[7], data_ipv4 = get_ipv4_data(raw_data[14:])
377             if NumPacket > curcnt:
378                 curcnt += 1000
379                 clear_end_sessions()
380             # Если это UDP-протокол
381             if proto == 17:
382                 NumPacket += 1

```

```

383     pinfo[5] = 'UDP'
384     pinfo[8], pinfo[9], _, data_udp = get_udp_segment(data_ip4)
385     pinfo[10] = len(data_udp)
386     Packet_list.append(PacketInf( pinfo[0], pinfo[1], pinfo[2]
387                                   , pinfo[3], pinfo[4], pinfo[5]
388                                   , pinfo[6], pinfo[7], pinfo[8]
389                                   , pinfo[9], pinfo[10]))
390     mes_prob = find_session_location(Packet_list[-1])
391     print_packet_inf(Packet_list[-1], mes_prob)
392     # Если это TCP-протокол
393     if proto == 6:
394         NumPacket += 1
395         pinfo[5] = 'TCP'
396         pinfo[8], pinfo[9], pinfo[11], \
397         pinfo[12], flags, data_tcp = get_tcp_segment(data_ip4)
398         pinfo[10] = len(data_tcp)
399         pinfo[13] = str((flags & 16) >> 4)
400         pinfo[14] = str((flags & 8) >> 3)
401         pinfo[15] = str((flags & 4) >> 2)
402         pinfo[16] = str((flags & 2) >> 1)
403         pinfo[17] = str(flags & 1)
404         Packet_list.append(PacketInf( pinfo[0], pinfo[1], pinfo[2], pinfo[3]
405                                       , pinfo[4], pinfo[5], pinfo[6], pinfo[7]
406                                       , pinfo[8], pinfo[9], pinfo[10], pinfo[11]
407                                       , pinfo[12], pinfo[13], pinfo[14], pinfo[15]
408                                       , pinfo[16], pinfo[17] ))
409         mes_prob = find_session_location(Packet_list[-1])
410         print_packet_inf(Packet_list[-1], mes_prob)
411     if keyboard.is_pressed('space'):
412         s_listen.close()
413         print('\nЗавершение программы...\n')
414         break
415
416
417 def clear_end_sessions():
418     global Session_list
419     n = len(Session_list)
420     ids = []
421     for i in range(n):
422         if Session_list[i].fl_fin or Session_list[i].fl_rst:
423             if Session_list[i].totalTime < 10:
424                 ids.append(i)
425         # else:
426         # ids.append(i)
427     tmp = Session_list.copy()
428     Session_list.clear()
429     for i in range(n):
430         if i in ids:

```

```

431         continue
432     Session_list.append(tmp[i])
433     for s in Session_list:
434         s.get_rdp_features(Packet_list[-1], True)
435
436
437 def find_session_location(pkt):
438     global Session_list
439     if pkt.protoType == 'UDP':
440         for s in Session_list:
441             if (not s.fl_fin and not s.fl_rst):
442                 if ( (pkt.ip_src == s.initiator and pkt.ip_dest == s.target) or \
443                     (pkt.ip_src == s.target and pkt.ip_dest == s.initiator) ) and \
444                     (pkt.port_src == s.port or pkt.port_dest == s.port):
445                     s.get_in_out_traffic(pkt)
446                     s.get_rdp_features(pkt)
447                     if s.is_rdp:
448                         if s.is_rdpPSH:
449                             return ([4, 5, 8], s.prob)
450                             return ([4, 5], s.prob)
451             return ([0], 0)
452     if pkt.fl_syn == '1' and pkt.fl_ack == '0':
453         Session_list.append(Session( pkt.timePacket, pkt.ip_src
454                                     , pkt.ip_dest, pkt.port_dest ))
455         Session_list[-1].upd_seq_num(pkt.seq)
456         Session_list[-1].get_in_out_traffic(pkt)
457         Session_list[-1].get_rdp_features(pkt)
458         return ([1], Session_list[-1].prob)
459     for s in Session_list:
460         if (not s.fl_fin and not s.fl_rst):
461             if pkt.fl_fin == '1' and pkt.fl_ack == '1' and \
462                 ( (pkt.ip_src == s.initiator and pkt.ip_dest == s.target) or \
463                   (pkt.ip_src == s.target and pkt.ip_dest == s.initiator) ) and \
464                   (pkt.port_src == s.port or pkt.port_dest == s.port):
465                 s.upd_fl_fin(pkt.timePacket)
466                 s.get_in_out_traffic(pkt)
467                 s.get_rdp_features(pkt)
468                 if s.is_rdp:
469                     if s.is_rdpPSH:
470                         return ([5, 6, 8], s.prob)
471                         return ([5, 6], s.prob)
472                 return ([6], s.prob)
473     if pkt.fl_rst == '1' and pkt.fl_ack == '1' and \
474         ( (pkt.ip_src == s.initiator and pkt.ip_dest == s.target) or \
475           (pkt.ip_src == s.target and pkt.ip_dest == s.initiator) ) and \
476           (pkt.port_src == s.port or pkt.port_dest == s.port):
477         s.upd_fl_rst(pkt.timePacket)
478         s.get_in_out_traffic(pkt)

```



```

479         s.get_rdp_features(pkt)
480     if s.is_rdp:
481         if s.is_rdpPSH:
482             return ([5, 7, 8], s.prob)
483         return ([5, 7], s.prob)
484     return ([7], s.prob)
485 if pkt.fl_syn == '1' and pkt.fl_ack == '1' and s.ack_num == None and \
486     pkt.ack == str(s.seq_num + 1) and pkt.ip_src == s.target and \
487     pkt.ip_dest == s.initiator and pkt.port_src == s.port:
488     s.upd_ack_num(pkt.ack)
489     s.upd_seq_num(pkt.seq)
490     s.get_in_out_traffic(pkt)
491     s.get_rdp_features(pkt)
492     if s.is_rdp:
493         if s.is_rdpPSH:
494             return ([2, 5, 8], s.prob)
495         return ([2, 5], s.prob)
496     return ([2], s.prob)
497 elif pkt.fl_syn == '0' and pkt.fl_ack == '1' and pkt.ack == str(s.seq_num + 1) and \
498     pkt.seq == s.ack_num and \
499     pkt.port_dest == s.port and pkt.ip_src == s.initiator and \
500     pkt.ip_dest == s.target:
501     s.get_in_out_traffic(pkt)
502     s.get_rdp_features(pkt)
503     if s.is_rdp:
504         if s.is_rdpPSH:
505             return ([3, 5, 8], s.prob)
506         return ([3, 5], s.prob)
507     return ([3], s.prob)
508 if pkt.fl_ack == '1' and \
509     ( (pkt.ip_src == s.initiator and pkt.ip_dest == s.target) or \
510       (pkt.ip_src == s.target and pkt.ip_dest == s.initiator) ) and \
511     (pkt.port_src == s.port or pkt.port_dest == s.port):
512     s.get_in_out_traffic(pkt)
513     s.get_rdp_features(pkt)
514     if s.is_rdp:
515         if s.is_rdpPSH:
516             return ([4, 5, 8], s.prob)
517         return ([4, 5], s.prob)
518     return ([4], s.prob)
519 return ([0], 0)
520
521
522 def print_inf_about_sessions():
523     cnt = 1
524     print(f'\nБыло перехвачено {len(Session_list)} сессии(-й)')
525     for s in Session_list:
526         print(f'\nИнформация о сессии #{cnt}:')

```

```

527     print(f'Инициатор подключения: {s.initiator}')
528     print(f'Целевое устройство: {s.target}')
529     print(f'Порт подключения: {s.port}')
530     print( f'Время установки соединения:'
531           , time.strftime('%d.%m.%Y г. %H:%M:%S', time.localtime(s.strtTime)) )
532     if s.finTime == None:
533         print(f'Время завершения соединения: нет данных')
534     else:
535         print( f'Время завершения соединения:'
536               , time.strftime('%d.%m.%Y г. %H:%M:%S', time.localtime(s.finTime)))
537         print(f'Общее время соединения: {s.totalTime} сек')
538     if s.is_rdp and s.prob > 50:
539         print(Back.GREEN + Fore.BLACK + f'Найдена RDP-сессия с вероятностью {s.prob}%!!!!')
540     cnt += 1
541     print(f'{line}{line}\n')
542
543
544 def write_to_file(f):
545     if Packet_list == []:
546         return False
547     try:
548         for obj in Packet_list:
549             if obj.protoType == 'UDP':
550                 f.write( f'No:{obj.numPacket};Time:{obj.timePacket};Pac-size:{obj.packetSize};' +
551                       f'MAC-src:{obj.mac_src};MAC-dest:{obj.mac_dest};Type:{obj.protoType};' +
552                       f'IP-src:{obj.ip_src};IP-dest:{obj.ip_dest};Port-src:{obj.port_src};' +
553                       f'Port-dest:{obj.port_dest};Len-data:{obj.len_data};!\n' )
554             else:
555                 f.write( f'No:{obj.numPacket};Time:{obj.timePacket};Pac-size:{obj.packetSize};' +
556                       f'MAC-src:{obj.mac_src};MAC-dest:{obj.mac_dest};Type:{obj.protoType};' +
557                       f'IP-src:{obj.ip_src};IP-dest:{obj.ip_dest};Port-src:{obj.port_src};' +
558                       f'Port-dest:{obj.port_dest};Len-data:{obj.len_data};Seq:{obj.seq};' +
559                       f'Ack:{obj.ack};Fl-ack:{obj.fl_ack};Fl-psh:{obj.fl_psh};' +
560                       f'Fl-rst:{obj.fl_rst};Fl-syn:{obj.fl_syn};Fl-fin:{obj.fl_fin};!\n' )
561         except:
562             return False
563     return True
564
565
566 # Считывание с файла и заполнение массива
567 # Packet_list объектами класса PacketInf
568 def read_from_file(inf):
569     global Packet_list
570     a = []
571     while True:
572         beg = inf.find(':')
573         end = inf.find(';')
574         if beg == -1 and end == -1:

```

```

575         break
576     else:
577         a.append(inf[beg + 1: end])
578         inf = inf[end + 1:]
579     # try:
580     if a[5] == 'TCP':
581         Packet_list.append(PacketInf( a[0], a[1], a[2], a[3], a[4], a[5]
582                                     , a[6], a[7], a[8], a[9], a[10], a[11]
583                                     , a[12], a[13], a[14], a[15], a[16], a[17] ))
584         _ = find_session_location(Packet_list[-1])
585     elif a[5] == 'UDP':
586         Packet_list.append(PacketInf( a[0], a[1], a[2], a[3], a[4], a[5]
587                                     , a[6], a[7], a[8], a[9], a[10] ))
588         _ = find_session_location(Packet_list[-1])
589     # except:
590     #     print('Ошибка при считывании файла...')
591     #     exit(0)
592
593
594 def print_packet_inf(obj, mes_prob):
595     if findRDP:
596         if 5 not in mes_prob[0] or mes_prob[1] <= 50:
597             return
598     print( f'{line}Пакет No{obj.numPacket}{line}\n'
599           , 'Время перехвата: '
600           , time.strftime( '%m:%d:%Y %H:%M:%S'
601                           , time.localtime(obj.timePacket) ) + '\n'
602           , f'Протокол: {obj.protoType}\n'
603           , f'MAC-адрес отправителя: {obj.mac_src}\n'
604           , f'MAC-адрес получателя: {obj.mac_dest}\n'
605           , f'Отправитель: {obj.ip_src}:{obj.port_src}\n'
606           , f'Получатель: {obj.ip_dest}:{obj.port_dest}')
607     if obj.protoType == 'TCP':
608         print( f' Порядковый номер: {obj.seq}; Номер подтверждения: {obj.ack}\n' +
609               f' SYN:{obj.fl_syn}; ACK:{obj.fl_ack}; PSH:{obj.fl_psh}; ' +
610               f'RST:{obj.fl_rst}; FIN:{obj.fl_fin}\n')
611     print('Признаки: ', end='')
612     for i in mes_prob[0]:
613         print(Phrases_signs[i], end='; ')
614     print(f'\nВероятность RDP-сессии {mes_prob[1]}%')
615
616     # Получение общей информации о текущей
617     # попытке перехвата трафика
618     def get_common_data():
619         global Labels_list
620         Labels_list.clear()
621         IPList = set()
622         numPacketsPerSec = []

```

```

623     curTime = Packet_list[0].timePacket + 1
624     fin = Packet_list[-1].timePacket + 1
625     Labels_list.append(time.strftime('%H:%M:%S', time.localtime(Packet_list[0].timePacket)))
626     cntPacket = 0
627     i = 0
628     while curTime < fin:
629         for k in range(i, len(Packet_list)):
630             if Packet_list[k].timePacket > curTime:
631                 numPacketsPerSec.append(cntPacket)
632                 Labels_list.append(time.strftime('%H:%M:%S', time.localtime(curTime)))
633                 cntPacket = 0
634                 i = k
635                 break
636             cntPacket += 1
637             curTime += 1
638         numPacketsPerSec.append(cntPacket)
639     for p in Packet_list:
640         IPList.add(p.ip_src)
641         IPList.add(p.ip_dest)
642     return list(IPList), numPacketsPerSec
643
644
645 def get_common_ports(curIP):
646     ports = set()
647     for pkt in Packet_list:
648         if pkt.ip_src == curIP or pkt.ip_dest == curIP:
649             ports.add(pkt.port_src)
650             ports.add(pkt.port_dest)
651     return list(ports)
652
653
654 # Вывод пар (число, IP-адрес) для
655 # предоставления выбора IP-адреса
656 # пользователю
657 def print_list_of_pairs(IPList, fl=False):
658     num = 0
659     cnt = 1
660     if fl:
661         print ('[' + str(num), '---', 'None', end='] ')
662         cnt += 1
663         num += 1
664     for el in IPList:
665         if cnt > 3:
666             cnt = 0
667             print ('[' + str(num), '---', el, end=']\n')
668         else:
669             print ('[' + str(num), '---', el, end='] ')
670         cnt += 1

```

```

671     num += 1
672     print('')
673
674
675     # Вывод пакетов, связанных с выбранным IP-адресом
676     def print_adjacent_packets(adjcPacketList):
677         cnt = 0
678         for p in adjcPacketList:
679             t = time.strftime('%H:%M:%S', time.localtime(p.timePacket))
680             if cnt % 2 == 1:
681                 print( f'Номер пакета: {p.numPacket};', f' Время: {t};'
682                     , f' Размер: {p.packetSize};', f' MAC-адрес отправителя: {p.mac_src};'
683                     , f' MAC-адрес получателя: {p.mac_dest};', f' Протокол: {p.protoType};'
684                     , f' Отправитель: {p.ip_src}:{p.port_src};'
685                     , f' Получатель: {p.ip_dest}:{p.port_dest};'
686                     , f' Размер поля данных: {p.len_data};', end='' )
687             if p.protoType == 'TCP':
688                 print( f' Порядковый номер: {p.seq}; Номер подтверждения: {p.ack};' +
689                     f' SYN:{p.fl_syn}; ACK:{p.fl_ack}; PSH:{p.fl_psh}; ' +
690                     f'RST:{p.fl_rst}; FIN:{p.fl_fin};')
691             else:
692                 print('')
693         else:
694             print( Back.CYAN + Fore.BLACK + f'Номер пакета: {p.numPacket};' + f' Время: {t};' +
695                 f' Размер: {p.packetSize};' + f' MAC-адрес отправителя: {p.mac_src};' +
696                 f' MAC-адрес получателя: {p.mac_dest};' +
697                 f' Отправитель: {p.ip_src}:{p.port_src};' +
698                 f' Получатель: {p.ip_dest}:{p.port_dest};' +
699                 f' Протокол: {p.protoType};' +
700                 f' Размер поля данных: {p.len_data};', end='' )
701             if p.protoType == 'TCP':
702                 print( Back.CYAN + Fore.BLACK + f' Порядковый номер: {p.seq};' +
703                     f' Номер подтверждения: {p.ack};' +
704                     f' SYN:{p.fl_syn}; ACK:{p.fl_ack}; PSH:{p.fl_psh};' +
705                     f' RST:{p.fl_rst}; FIN:{p.fl_fin};')
706             else:
707                 print('')
708         cnt += 1
709
710
711
712     # Получение данных об отношении входящего
713     # трафика к исходящему в единицу времени
714     def get_in_out_rel(exploreIP, strt, fin, port):
715         cntInput = 0
716         cntOutput = 0
717         rel_list = []
718         curTime = strt + 1

```

```

719     fin += 1
720     pos = 0
721     while curTime < fin:
722         for k in range(pos, len(Packet_list)):
723             if Packet_list[k].timePacket > curTime:
724                 if cntOutput != 0:
725                     rel_list.append(cntInput / cntOutput)
726                 else:
727                     rel_list.append(0.0)
728                 cntInput = 0
729                 cntOutput = 0
730                 pos = k
731                 break
732         if port == None:
733             if Packet_list[k].ip_src == exploreIP:
734                 cntOutput += 1
735             if Packet_list[k].ip_dest == exploreIP:
736                 cntInput += 1
737         else:
738             if Packet_list[k].port_src == port or Packet_list[k].port_dest == port:
739                 if Packet_list[k].ip_src == exploreIP:
740                     cntOutput += 1
741                 if Packet_list[k].ip_dest == exploreIP:
742                     cntInput += 1
743         curTime += 1
744     if cntOutput != 0:
745         rel_list.append(cntInput / cntOutput)
746     else:
747         rel_list.append(0.0)
748     return rel_list
749
750
751     # Получение данных об отношении количества
752     # входящего UDP-трафика на количество
753     # исходящего TCP-трафика в единицу времени
754     def get_udp_tcp_rel(exploreIP, strt, fin, port):
755         cntUDP = 0
756         cntTCP = 0
757         curTime = strt + 1
758         fin += 1
759         pos = 0
760         rel_list = []
761         while curTime < fin:
762             for k in range(pos, len(Packet_list)):
763                 if Packet_list[k].timePacket > curTime:
764                     if cntTCP != 0:
765                         rel_list.append(cntUDP / cntTCP)
766                     else:

```

```

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

```

```

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

```



```

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

```

```

911     pktSizeDstList = []
912     curTime = strt + 1
913     fin += 1
914     pos = 0
915     while curTime < fin:
916         cntpktsrc = 0
917         cntpktdest = 0
918         maxpktsizesrc = 0
919         maxpktsizedst = 0
920         for k in range(pos, len(Packet_list)):
921             if Packet_list[k].timePacket > curTime:
922                 pktAmntSrcList.append(cntpktsrc)
923                 pktAmntDstList.append(cntpktdest)
924                 pktSizeSrcList.append(maxpktsizesrc)
925                 pktSizeDstList.append(maxpktsizedst)
926                 pos = k
927                 break
928         if port == None:
929             if Packet_list[k].ip_src == exploreIP:
930                 cntpktsrc += 1
931                 if maxpktsizesrc < Packet_list[k].packetSize:
932                     maxpktsizesrc = Packet_list[k].packetSize
933             if Packet_list[k].ip_dest == exploreIP:
934                 cntpktdest += 1
935                 if maxpktsizedst < Packet_list[k].packetSize:
936                     maxpktsizedst = Packet_list[k].packetSize
937         else:
938             if Packet_list[k].port_src == port or Packet_list[k].port_dest == port:
939                 if Packet_list[k].ip_src == exploreIP:
940                     cntpktsrc += 1
941                     if maxpktsizesrc < Packet_list[k].packetSize:
942                         maxpktsizesrc = Packet_list[k].packetSize
943                 if Packet_list[k].ip_dest == exploreIP:
944                     cntpktdest += 1
945                     if maxpktsizedst < Packet_list[k].packetSize:
946                         maxpktsizedst = Packet_list[k].packetSize
947         curTime += 1
948     pktAmntSrcList.append(cntpktsrc)
949     pktAmntDstList.append(cntpktdest)
950     pktSizeSrcList.append(maxpktsizesrc)
951     pktSizeDstList.append(maxpktsizedst)
952     return pktAmntSrcList, pktAmntDstList, pktSizeSrcList, pktSizeDstList
953
954
955     # Получение общей информации о трафике,
956     # связанном с выбранным IP-адресом
957     def get_inf_about_IP(exploreIP, port):
958         adjcPacketList = []

```

```

959     adjcIPList = set()
960     if port != None:
961         for p in Packet_list:
962             if p.port_src == port or p.port_dest == port:
963                 if p.ip_src == exploreIP:
964                     adjcPacketList.append(p)
965                     adjcIPList.add(p.ip_dest)
966                 if p.ip_dest == exploreIP:
967                     adjcPacketList.append(p)
968                     adjcIPList.add(p.ip_src)
969     else:
970         for p in Packet_list:
971             if p.ip_src == exploreIP:
972                 adjcPacketList.append(p)
973                 adjcIPList.add(p.ip_dest)
974             if p.ip_dest == exploreIP:
975                 adjcPacketList.append(p)
976                 adjcIPList.add(p.ip_src)
977     return adjcPacketList, list(adjcIPList)
978
979
980 def get_pos_by_IP(curIP):
981     for i in range(len(Object_list)):
982         if Object_list[i].ip == curIP:
983             return i
984     return -1
985
986
987 # Получение меток и "шага" для оси абсцисс
988 def get_x_labels(total_time):
989     global x_axisLabels
990     step = 1
991     if total_time > 600:
992         step = 30
993     elif total_time > 300:
994         step = 10
995     elif total_time > 50:
996         step = 5
997     x_axisLabels.clear()
998     for i in range(0, len(Labels_list), step):
999         x_axisLabels.append(Labels_list[i])
1000     return step
1001
1002
1003 def get_2nd_IP_for_plot(k):
1004     print('\nИзобразить на графике еще один объект. Выберите ' + \
1005           'IP-адрес для добавления (введите цифру)')
1006     print_list_of_pairs(Object_list[k].adjcIPList, True)

```

```

1007     scndIP = 'None'
1008     try:
1009         pos = int(input())
1010     except:
1011         print('Некорректный ввод!')
1012         return -1
1013     else:
1014         if pos < 0 or pos > len(Object_list[k].adjcIPList):
1015             print('Некорректный ввод!')
1016             return -1
1017         if pos != 0:
1018             scndIP = Object_list[k].adjcIPList[pos - 1]
1019     return scndIP
1020
1021
1022     # Выбор опций для выбранного IP-адреса
1023     def choose_options(k, strt, fin, step, port):
1024         curIP = Object_list[k].ip
1025         Object_list[k].adjcPacketList, Object_list[k].adjcIPList = get_inf_about_IP(curIP, port)
1026         Object_list[k].strt_time = time.localtime(Object_list[k].adjcPacketList[0].timePacket)
1027         Object_list[k].fin_time = time.localtime(Object_list[k].adjcPacketList[-1].timePacket)
1028         Object_list[k].amnt_packet = len(Object_list[k].adjcPacketList)
1029         totalTime = round( Object_list[k].adjcPacketList[-1].timePacket - \
1030                             Object_list[k].adjcPacketList[0].timePacket )
1031         if totalTime == 0:
1032             totalTime = 1
1033         Object_list[k].avg_packet_num = round(Object_list[k].amnt_packet / totalTime, 3)
1034         avgSize = 0
1035         for p in Object_list[k].adjcPacketList:
1036             avgSize += p.len_data
1037         Object_list[k].avg_packet_size = round(avgSize / Object_list[k].amnt_packet, 3)
1038         while True:
1039             print(f'Общая информация о трафике, связанном с {curIP}')
1040             print( 'Время первого перехваченного пакета: '
1041                   , time.strftime('%d.%m.%Y г. %H:%M:%S', Object_list[k].strt_time) )
1042             print( 'Время последнего перехваченного пакета: '
1043                   , time.strftime('%d.%m.%Y г. %H:%M:%S', Object_list[k].fin_time) )
1044             print('Общее время:', totalTime, 'сек.')
1045             print('Количество пакетов: ', Object_list[k].amnt_packet)
1046             print('Среднее количество пакетов в секунду: ', Object_list[k].avg_packet_num)
1047             print('Средний размер пакетов: ', Object_list[k].avg_packet_size)
1048             print(f'""Выберите опцию:
1049             1. Вывести весь трафик, связанный с {curIP}
1050             2. Построить график отношения входящего и исходящего трафиков
1051             3. Построить график отношения объема входящего UDP-трафика и объёма входящего TCP-трафика
1052             4. Построить график разности числа исходящих и числа входящих ACK-флагов в единицу времени
1053             5. Построить график частоты SYN и PSN флагов во входящих пакетах
1054             6. Построить график отображения количества пакетов в единицу времени

```

```

1055 7. Построить график отображения максимумов среди пакетов в единицу времени
1056 8. Вернуться к выбору IP-адреса """)
1057 bl = input()
1058 if bl == '1':
1059     print_adjacent_packets(Object_list[k].adjcPacketList)
1060
1061 elif bl == '2':
1062     Object_list[k].in_out_rel_data = get_in_out_rel(curIP, strt, fin, port)
1063     x = [i for i in range(0, len(Object_list[k].in_out_rel_data))]
1064     x_labels = [i for i in range(0, len(x), step)]
1065     scndIP = get_2nd_IP_for_plot(k)
1066     if scndIP == -1:
1067         continue
1068     if scndIP != 'None':
1069         pos = get_pos_by_IP(scndIP)
1070         Object_list[pos].in_out_rel_data = get_in_out_rel(scndIP, strt, fin, port)
1071     fig = plt.figure(figsize=(16, 6), constrained_layout=True)
1072     f = fig.add_subplot()
1073     f.grid()
1074     f.set_title('Отношение объема входящего к объему исходящего трафиков' + \
1075                r' ($r_{in/out} = \frac{V_{in}}{V_{out}}$)', fontsize=15)
1076     f.set_xlabel('Общее время перехвата трафика', fontsize=15)
1077     f.set_ylabel(r'$r_{in/out} = \frac{V_{in}}{V_{out}}$', fontsize=15)
1078     plt.plot(x, Object_list[k].in_out_rel_data, label=curIP)
1079     if scndIP != 'None':
1080         plt.plot(x, Object_list[pos].in_out_rel_data, label=scndIP)
1081     plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=10)
1082     f.legend()
1083     plt.show()
1084 elif bl == '3':
1085     Object_list[k].udp_tcp_rel_data = get_udp_tcp_rel(curIP, strt, fin, port)
1086     x = [i for i in range(0, len(Object_list[k].udp_tcp_rel_data))]
1087     x_labels = [i for i in range(0, len(x), step)]
1088     scndIP = get_2nd_IP_for_plot(k)
1089     if scndIP == -1:
1090         continue
1091     if scndIP != 'None':
1092         pos = get_pos_by_IP(scndIP)
1093         Object_list[pos].udp_tcp_rel_data = get_udp_tcp_rel(scndIP, strt, fin, port)
1094     fig = plt.figure(figsize=(16, 6), constrained_layout=True)
1095     f = fig.add_subplot()
1096     f.grid()
1097     f.set_title('Отношение объема входящего UDP-трафика к объему ' +
1098                'входящего TCP-трафика' + r' ($r_{in} = \frac{V_{udp}}{V_{tcp}}$)'
1099                , fontsize=15)
1100     f.set_xlabel('Общее время перехвата трафика', fontsize=15)
1101     f.set_ylabel(r'$r_{in} = \frac{V_{udp}}{V_{tcp}}$', fontsize=15)
1102     plt.plot(x, Object_list[k].udp_tcp_rel_data, label=curIP)

```

```

1103     if scndIP != 'None':
1104         plt.plot(x, Object_list[pos].udp_tcp_rel_data, label=scndIP)
1105     plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=10)
1106     f.legend()
1107     plt.show()
1108 elif bl == '4':
1109     Object_list[k].ack_flags_diff_data = get_ack_flags_diff(curIP, strt, fin, port)
1110     x = [i for i in range(0, len(Object_list[k].ack_flags_diff_data))]
1111     x_labels = [i for i in range(0, len(x), step)]
1112     scndIP = get_2nd_IP_for_plot(k)
1113     if scndIP == -1:
1114         continue
1115     if scndIP != 'None':
1116         pos = get_pos_by_IP(scndIP)
1117         Object_list[pos].ack_flags_diff_data = get_ack_flags_diff(scndIP, strt, fin, port)
1118     fig = plt.figure(figsize=(16, 6), constrained_layout=True)
1119     f = fig.add_subplot()
1120     f.grid()
1121     f.set_title('Разность числа исходящих и числа входящих АСК-флагов' + \
1122                r' ($r_{ack} = V_{A_{out}} - V_{A_{in}})$', fontsize=15)
1123     f.set_xlabel('Общее время перехвата трафика', fontsize=15)
1124     f.set_ylabel(r'$r_{ack} = V_{A_{out}} - V_{A_{in}}$', fontsize=15)
1125     plt.plot(x, Object_list[k].ack_flags_diff_data, label=curIP)
1126     if scndIP != 'None':
1127         plt.plot(x, Object_list[pos].ack_flags_diff_data, label=scndIP)
1128     plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=10)
1129     f.legend()
1130     plt.show()
1131 elif bl == '5':
1132     data = get_syn_flags_freq(curIP, strt, fin, port)
1133     Object_list[k].syn_flags_freq_data = data
1134     data = get_psh_flags_freq(curIP, strt, fin, port)
1135     Object_list[k].psh_flags_freq_data = data
1136     x = [i for i in range(0, len(Object_list[k].syn_flags_freq_data))]
1137     x_labels = [i for i in range(0, len(x), step)]
1138     scndIP = get_2nd_IP_for_plot(k)
1139     if scndIP == -1:
1140         continue
1141     if scndIP != 'None':
1142         pos = get_pos_by_IP(scndIP)
1143         data = get_syn_flags_freq(scndIP, strt, fin, port)
1144         Object_list[pos].syn_flags_freq_data = data
1145         data = get_psh_flags_freq(scndIP, strt, fin, port)
1146         Object_list[pos].psh_flags_freq_data = data
1147     fig = plt.figure(figsize=(16, 6), constrained_layout=True)
1148     gs = gridspec.GridSpec(ncols=1, nrows=2, figure=fig)
1149     fig_1 = fig.add_subplot(gs[0, 0])
1150     fig_1.grid()

```

```

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

```

```

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

```



```

1247     fig_2.grid()
1248     plt.plot(x, Object_list[k].pkt_size_data_src, 'orange', label=curIP)
1249     fig_2.set_title('Максимальный размер исходящих пакетов, полученных за ' + \
1250                     'единицу времени', fontsize=15)
1251     fig_2.set_xlabel('Общее время перехвата трафика', fontsize=15)
1252     # fig_2.set_ylabel(r'$r_{psh} = \frac{V_{P_{in}}}{V_{tcp}}$', fontsize=15)
1253     if scndIP != 'None':
1254         plt.plot(x, Object_list[pos].pkt_size_data_src, 'g', label=scndIP)
1255     plt.xticks(x_labels, x_axisLabels, rotation=30, fontsize=8)
1256     fig_2.legend()
1257     plt.show()
1258 elif b1 == '8':
1259     break
1260
1261
1262 def choose_mode():
1263     global Packet_list, Object_list, Labels_list, Session_list, findRDP
1264     while True:
1265         print('1. Перехват трафика')
1266         print('2. Запись данных в файл')
1267         print('3. Считывание с файла данных для анализа трафика')
1268         print('4. Анализ трафика')
1269         print('5. Выход')
1270         b1 = input()
1271         if b1 == '1':
1272             Packet_list.clear()
1273             Object_list.clear()
1274             Labels_list.clear()
1275             Session_list.clear()
1276             findRDP = False
1277             print('Поставить фильтр RDP? (Если да, то введите 1)')
1278             f1 = input('Ответ: ')
1279             if f1 == '1':
1280                 findRDP = True
1281             try:
1282                 print('\nВыберите сетевой интерфейс, нажав соответствующую цифру:')
1283                 print(socket.if_nameindex())
1284                 interface = int(input())
1285                 if 0 > interface or interface > len(socket.if_nameindex()):
1286                     print('\nОшибка ввода!!!\n')
1287                     return
1288                 os.system(f'ip link set {socket.if_indextoname(interface)} promisc on')
1289                 s_listen = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
1290             except PermissionError:
1291                 print('\nНедостаточно прав!')
1292                 print('Запустите программу от имени администратора!')
1293                 return
1294         else:

```

```

1295     print('\nНачался процесс захвата трафика...\n')
1296     start_to_listen(s_listen)
1297     print(f'\nДанные собраны. Перехвачено: {len(Packet_list)} пакетов(-a)\n')
1298
1299     print('\nХотите записать перехваченный трафик в файл? (да - нажмите 1)')
1300     bl1 = input('Ответ: ')
1301     if '1' in bl1:
1302         print('Введите название файла (например: data.log)')
1303         FileName = input()
1304         try:
1305             f = open(FileName, 'w')
1306         except:
1307             print('\nНекорректное название файла!\n')
1308             continue
1309         if write_to_file(f):
1310             print(f'\nВ файл {FileName} была успешно записана информация.\n')
1311             f.close()
1312         else:
1313             print(f'\nОшибка записи в файл {FileName}! Возможно нет данных для записи\n')
1314             f.close()
1315     print('')
1316     elif bl == '2':
1317         if Packet_list == []:
1318             print('\nНет данных! Сначала необходимо получить данные!\n')
1319             continue
1320         print('Введите название файла (например: data.log)')
1321         FileName = input()
1322         try:
1323             f = open(FileName, 'w')
1324         except:
1325             print('\nНекорректное название файла!\n')
1326             continue
1327         if write_to_file(f):
1328             print(f'\nВ файл {FileName} была успешно записана информация.\n')
1329             f.close()
1330         else:
1331             print(f'\nОшибка записи в файл {FileName}! Возможно нет данных для записи...\n')
1332             f.close()
1333             continue
1334     elif bl == '3':
1335         Packet_list.clear()
1336         Object_list.clear()
1337         Labels_list.clear()
1338         Session_list.clear()
1339         print('Введите название файла (например: data.log)')
1340         FileName = input()
1341         if not Packet_list:
1342             try:

```

```

1343         f = open(FileName, 'r')
1344     except:
1345         print('\nНекорректное название файла!\n')
1346         continue
1347     while True:
1348         inf = f.readline()
1349         if not inf:
1350             break
1351         read_from_file(inf)
1352     f.close()
1353     print(f'\nДанные собраны. Перехвачено: {len(Packet_list)} пакетов(-a)\n')
1354 elif bl == '4':
1355     if Packet_list == []:
1356         print('\nНет данных! Сначала необходимо получить данные!\n')
1357         continue
1358     IPList, numPacketsPerSec = get_common_data()
1359     clear_end_sessions()
1360     for s in Session_list:
1361         s.fin_rdp_check()
1362     print_inf_about_sessions()
1363     strt = Packet_list[0].timePacket
1364     fin = Packet_list[-1].timePacket
1365     strt_time = time.localtime(strt)
1366     fin_time = time.localtime(fin)
1367     avgNumPacket = 0
1368     for el in numPacketsPerSec:
1369         avgNumPacket += el
1370     avgNumPacket /= len(numPacketsPerSec)
1371     avgSizePacket = 0
1372     for p in Packet_list:
1373         avgSizePacket += p.packetSize
1374     avgSizePacket /= len(Packet_list)
1375
1376     step = get_x_labels(int(fin - strt))
1377     print('Общая информация:')
1378     print('Время первого перехваченного пакета: ',
1379           time.strftime('%d.%m.%Y г. %H:%M:%S', strt_time) )
1380     print('Время последнего перехваченного пакета: ',
1381           time.strftime('%d.%m.%Y г. %H:%M:%S', fin_time) )
1382     print('Количество пакетов: ', len(Packet_list))
1383     print('Общее время перехвата: ', round(fin - strt, 3), 'сек')
1384     print('Среднее количество пакетов в секунду: ', round(avgNumPacket, 3))
1385     print('Средний размер пакетов: ', round(avgSizePacket, 3))
1386     print('Завершить просмотр (нажмите \"q\" для выхода)')
1387     for k in range(len(IPList)):
1388         Object_list.append(ExploreObject(IPList[k]))
1389         Object_list[-1].commonPorts = get_common_ports(IPList[k])
1390     print_list_of_pairs(IPList)

```

```

1391     print(f'\nВыберите цифру (0 - {len(IPList) - 1}) для просмотра IP-адреса:')
1392     k = input()
1393     if k == 'q':
1394         break
1395     try:
1396         k = int(k)
1397     except:
1398         print('\nНекорректный ввод!\n')
1399         continue
1400     else:
1401         if 0 <= k and k < len(IPList):
1402             port = None
1403             print('Список портов которые участвовали в соединении с данным IP-адресом')
1404             print_list_of_pairs(Object_list[k].commonPorts, True)
1405             t = len(Object_list[k].commonPorts)
1406             print(f'\nВыберите цифру (0 - {t}) для выбора порта:')
1407             k1 = input()
1408             if k1 == 'q':
1409                 break
1410             try:
1411                 k1 = int(k1)
1412             except:
1413                 print('Некорректный ввод!\n')
1414                 continue
1415             else:
1416                 if 0 <= k1 and k1 <= t:
1417                     if k1 != 0:
1418                         port = Object_list[k].commonPorts[k1 - 1]
1419                         choose_options(k, strt, fin, step, port)
1420                     else:
1421                         print(f'Введите число в пределах 0 - {t - 1}')
1422             else:
1423                 print(f'Введите число в пределах 0 - {len(IPList) - 1}')
1424         elif b1 == '5':
1425             return
1426
1427
1428 if __name__ == '__main__':
1429     print('\nЗапуск программы...\n')
1430     choose_mode()

```