

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

**РАЗРАБОТКА ПРОГРАММЫ ДЛЯ ВЫЯВЛЕНИЯ RDP-СЕССИЙ С
ИСПОЛЬЗОВАНИЕМ НЕЙРОННЫХ СЕТЕЙ**

КУРСОВАЯ РАБОТА

студента 6 курса 631 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Токарева Никиты Сергеевича

Научный руководитель
доцент

Гортинский А. В.

Заведующий кафедрой

Абросимов М. Б.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 Протокол RDP: место в сетях и особенности реализации	5
1.1 Обзор протокола RDP	5
1.2 Место RDP в стеке протоколов	6
1.2.1 Структура Ethernet-кадра	8
1.2.2 Структура IPv4-заголовка	8
1.2.3 Структура заголовков TCP и UDP	9
2 Особенности и характерные признаки RDP-трафика	14
2.1 Признаки на основе анализа временных интервалов между сетевыми пакетами	14
2.1.1 Вычисление задержки и стандартного отклонения временных интервалов между пакетами	15
2.1.2 Вычисление среднего джиттера и медианы временных интервалов	15
2.2 Признаки, связанные с количественными характеристиками трафика	16
2.2.1 Вычисление отношения объема входящего на исходящий трафик	17
2.2.2 Вычисление отношения объема UDP- и TCP-трафиков	18
2.2.3 Вычисление среднего значения объема пакетов	18
2.3 Признаки на основе анализа параметров TCP-соединений	19
2.3.1 Вычисление частоты флагов PSH и ACK	19
2.3.2 Вычисление отношения ACK/PSH	21
2.3.3 Вычисление разности числа исходящих и входящих ACK-флагов	21
2.3.4 Вычисление отношения числа флагов SYN на сумму флагов FIN + RST	22
2.3.5 Вычисление среднего значения размера окна и частоты его обновления	23
3 Модель LSTM для анализа трафика	25
3.1 Особенности архитектуры LSTM	25
3.1.1 Введение в рекуррентные нейронные сети	25
3.1.2 Структура ячейки LSTM	26

3.1.3	Преимущества LSTM перед другими моделями	28
3.2	Применение LSTM в задаче анализа трафика	29
4	Программная реализация	32
4.1	Общая структура программы.....	32
4.2	Обучение модели LSTM и её применение в программе	34
4.3	Демонстрация работы программы	36
4.3.1	Сравнение поведения прикладных протоколов	37
4.3.2	Сравнительный анализ альтернативных программ удаленного доступа	43
	ЗАКЛЮЧЕНИЕ	47
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	48
Приложение А	Листинг sniffer.py	50
Приложение Б	Листинг common_methods.py	54
Приложение В	Листинг package_parameters.py	56
Приложение Г	Листинг traffic_analysis.py	57
Приложение Д	Листинг chart_creation.py	61
Приложение Е	Листинг session_creation.py.....	73
Приложение Ж	Листинг main.py	82

ВВЕДЕНИЕ

В условиях современной цифровой экономики удалённый доступ к информационным системам играет важную роль в обеспечении удобства, гибкости и эффективности работы. Одним из наиболее популярных протоколов для удалённого управления компьютерами является Remote Desktop Protocol (RDP). Этот протокол широко используется в корпоративных и частных сетях для предоставления безопасного доступа к рабочим станциям, серверам и другим устройствам.

Однако вместе с удобством использования RDP остаётся объектом пристального внимания со стороны злоумышленников. Уязвимости протокола часто используются для реализации атак, включая подбор паролей методом перебора (brute force), использование слабых паролей, кражу данных и распространение вредоносного ПО. В таких условиях задача мониторинга и выявления RDP-трафика в общем потоке сетевых данных приобретает особую актуальность.

Основной целью данной работы является разработка программы для автоматического выявления RDP-сессий в сетевом трафике с использованием нейронных сетей, а именно модели LSTM (Long Short-Term Memory). Эта модель, благодаря своей способности обрабатывать последовательности данных, предоставляет мощный инструмент для анализа характеристик сетевого трафика и классификации его по типу.

Для достижения цели были поставлены следующие задачи:

- изучение особенностей протокола RDP и его признаков в сетевом трафике;
- разработка метрик для анализа и классификации RDP-трафика;
- использование модели LSTM для решения задачи классификации;
- реализация программного продукта, способного анализировать сетевой трафик в режиме реального времени и выявлять RDP-сессии;
- тестирование программы и оценка её эффективности.

Результаты данной работы могут быть полезны для специалистов в области информационной безопасности, системных администраторов, а также разработчиков инструментов мониторинга и анализа сетевого трафика.

1 Протокол RDP: место в сетях и особенности реализации

Концепция удалённого доступа к вычислительным ресурсам зародилась в стремлении управлять компьютерами и серверами, находящимися на расстоянии. Это позволило пользователям быть независимыми от физического местоположения оборудования, обеспечивая гибкость и эффективность в использовании вычислительных мощностей.

Протокол удаленного рабочего стола (Remote Desktop Protocol, RDP) был разработан корпорацией Microsoft в конце 1990-х годов как решение для удаленного доступа к компьютерам и серверам. Истоки технологии удаленного доступа уходят в 90-е годы XX века, когда компании начали создавать протоколы для управления вычислительными системами на расстоянии. Одним из первых таких протоколов был Citrix ICA, который стал основой для создания RDP. Microsoft внедрила эту технологию в свои продукты, начиная с Windows NT Terminal Server 4.0, что значительно упростило удалённую работу.

1.1 Обзор протокола RDP

Идея создания RDP была основана на стремлении обеспечить пользователям возможность удалённого управления вычислительными ресурсами, независимо от их физического расположения. Эта технология стала важной частью корпоративных сетей, позволив администраторам эффективно управлять серверами, а пользователям – работать с удалёнными рабочими столами. Вначале RDP был ориентирован на работу в режиме точка-точка, но со временем получил поддержку многоточечных соединений, что сделало его более универсальным инструментом для совместной работы и администрирования [3].

RDP является расширением семейства протоколов Т.120 и базируется на стандарте T.Share. Его ключевые особенности:

1. Многоканальная архитектура: RDP поддерживает до 64 000 виртуальных каналов для передачи различных типов данных, таких как:
 - данные презентации;
 - управление периферийными устройствами;
 - лицензирование;
 - зашифрованные команды клавиатуры и мыши.
2. Безопасность: протокол включает механизмы шифрования, обеспечивающие защиту передаваемых данных. Это делает RDP надёжным выбором

для корпоративных сетей;

3. Универсальность: RDP изначально был разработан для работы с различными сетевыми топологиями, включая ISDN, POTS и TCP/IP. Современные версии фокусируются на TCP/IP, обеспечивая широкую совместимость;
4. Оптимизация сетевого трафика: протокол использует компрессию данных и механизмы кадрирования для минимизации сетевых задержек;
5. Поддержка мультимедиа: современные версии RDP включают функции передачи аудио, видео и взаимодействия с периферийными устройствами, такими как принтеры и USB-устройства.

1.2 Место RDP в стеке протоколов

Стек протоколов представляет собой набор взаимосвязанных стандартов и правил, определяющих взаимодействие между различными уровнями сетевой архитектуры. Каждый уровень отвечает за выполнение определённых функций, начиная от передачи данных через физические среды до обеспечения высокого уровня абстракции для приложений. Существует несколько моделей сетевых взаимодействий, каждая из которых предлагает свой подход к организации передачи данных между устройствами в сети.

Наиболее известным примером стека протоколов является модель OSI (Open Systems Interconnection), которая состоит из семи уровней:

1. Физический уровень, который отвечает за передачу последовательности битов через канал связи.
2. Канальный уровень, где осуществляется разбиение данных на «кадры», размер которых обычно достигает от нескольких сотен до нескольких тысяч байтов.
3. Сетевой уровень, на котором осуществляется структуризация и маршрутизация пакетов от отправителя к получателю.
4. Транспортный уровень, функцией которого является передача надежных последовательностей данных произвольной длины через коммуникационную сеть от отправителя к получателю.
5. Сеансовый уровень, на котором происходит поддержка сессии связи, управление взаимодействием между приложениями.
6. Уровень представления, который представляет данные в понятном для какой-либо конкретной машины виде.

7. Прикладной уровень, предоставляющий набор интерфейсов для взаимодействия пользовательских процессов с сетью [5].

Вследствие этого, RDP является непосредственно протоколом прикладного уровня модели OSI, наряду с HTTP, FTP, SSH и многими другими. Стоит отметить, что в основном OSI применяется преимущественно в образовательных и теоретических целях. На практике используется модель TCP/IP, которая отражает реальную организацию современных компьютерных сетей [4]. Она была разработана как часть стека протоколов, лежащих в основе интернета. Название TCP/IP связано с двумя ключевыми протоколами этого семейства — Transmission Control Protocol (TCP) и Internet Protocol (IP). Именно они были впервые разработаны и задокументированы в этом стандарте. Иногда эту модель называют моделью DOD (Department of Defense). TCP/IP используется повсеместно, поскольку большинство современных протоколов (HTTP, FTP, RDP) работают в рамках этой модели. Модель TCP/IP разделяет сетевое взаимодействие на четыре уровня, каждый из которых выполняет определённые функции:

1. Сетевой интерфейс/канальный уровень (объединяет уровни 1 и 2 OSI): отвечает за физическую передачу данных.
2. Сетевой уровень (аналог уровня 3 OSI): используется для маршрутизации данных (например, протокол IP).
3. Транспортный уровень (аналог уровня 4 OSI): обеспечивает надежность передачи данных с помощью протоколов TCP и UDP.
4. Прикладной уровень (соответствует уровням 5, 6 и 7 OSI): поддерживает приложения и службы, такие как HTTP, FTP, RDP и т.д.

Основное преимущество TCP/IP в том, что это практическая модель, на основе которой построен интернет. Все современные сети работают с протоколами TCP/IP, поэтому в реальных системах она имеет приоритет.

Здесь важно понимать, что в компьютерных сетях все данные инкапсулируются и декапсулируются на различных уровнях модели TCP/IP. Процесс инкапсуляции происходит так, что на каждом уровне стека протоколы добавляют свои заголовки: от уровня приложений до канального уровня. На этапе приема данные декапсулируются в обратном порядке, начиная с канального уровня и заканчивая уровнем приложений.

Исходя из описания уровней, RDP в модели TCP/IP также относится к при-

кладному уровню. Поэтому, данные RDP сначала инкапсулируются в сегменты TCP, которые затем обрабатываются в IP-пакеты и кадры Ethernet для передачи по сети. На принимающем устройстве данные декапсулируются в обратном порядке, начиная с канального уровня и заканчивая уровнем приложений, уровнем RDP.

Для понимания процесса передачи данных от отправителя к получателю важно рассмотреть структуру пакетов, используемых в сетевых взаимодействиях. Далее будет рассмотрена четырехуровневая структура модели TCP/IP от канального до прикладного.

1.2.1 Структура Ethernet-кадра

На канальном уровне передача данных осуществляется с использованием Ethernet-кадра. Его структура представлена на рисунке 1.



Рисунок 1 – Структура Ethernet кадра

Ключевыми полями Ethernet-кадра являются:

- MAC-адреса источника и назначения – позволяют определить отправителя и получателя на канальном уровне.
- Поле «Тип» – указывает номер инкапсулированного сетевого протокола (например, IPv4 или IPv6).
- Поле данных – содержит инкапсулированные данные более высокого уровня, включая сетевые пакеты.

1.2.2 Структура IPv4-заголовка

Для передачи TCP-сегментов на сетевом уровне используется IP-протокол. В данной работе рассматривается версия IPv4, так как её возможностей достаточно для анализа RDP-трафика. Структура IPv4-заголовка представлена на рисунке 2.

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

Рисунок 2 – Структура IPv4-заголовка

Особо важны следующие поля IPv4-заголовка:

- IP-адреса отправителя и получателя – позволяют идентифицировать устройства, участвующие в обмене данными.
- Поле «Протокол» – определяет протокол транспортного уровня. Например, значение 6 указывает на TCP, а 17 – на UDP.

1.2.3 Структура заголовков TCP и UDP

Известно, что протокол RDP взаимодействует с транспортным уровнем через TCP и UDP. В основном данный протокол использует TCP для установления надёжного соединения и передачи данных, что позволяет обеспечить стабильное взаимодействие между клиентом и сервером удалённого рабочего стола. Сама структура TCP-заголовка показана на рисунке 3.

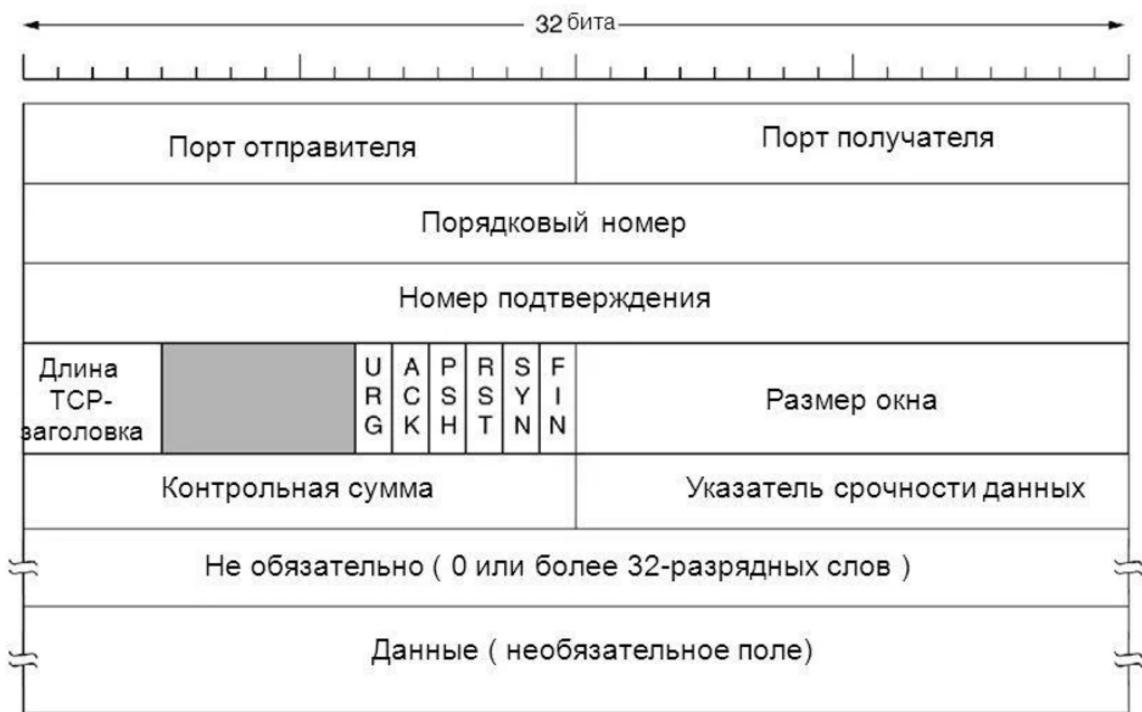


Рисунок 3 – Структура TCP-заголовка

В данном сегменте одними из интересных полей является информация о портах отправителя и получателя. Стоит отметить, что при подключении к удаленному рабочему столу по умолчанию используется порт 3389, что помогает идентифицировать RDP-трафик.

Не менее интересным полем является поле «Размер окна» – это объем данных приема (в байтах), которые можно буферизировать во время подключения. Узел отправки может отправлять только этот объем данных, прежде чем он должен ожидать подтверждения и обновления окна от принимающего узла [6]. Другими словами, данная величина указывает количество байт, которое может быть отправлено без подтверждения от получателя до того момента, когда необходимо будет ожидать нового подтверждения о получении данных.

Также важное значение имеют флаги, содержащиеся в поле флагов. В нем хранятся следующие управляемые биты:

1. NS – одноразовая сумма (Nonce Sum). По-прежнему является экспериментальным флагом, используемым для защиты от случайного злонамеренного сокрытия пакетов от отправителя [7]. Используется для улучшения работы механизма явного уведомления о перегрузке (Explicit Congestion Notification, ECN).
2. CWR – окно перегрузки уменьшено (Congestion Window Reduced). Дан-

ный флаг устанавливается (принимает значение равной единице) отправителем, чтобы показать, что TCP-фрагмент был получен с установленным полем ECE.

3. ECE – ECN-Эхо (ECN-Echo). Этот флаг показывает, поддерживает ли TCP-отправитель ECN.
4. URG – устанавливается, если необходимо передать ссылку на поле указателя срочности (Urgent pointer).
5. ACK – флаг подтверждения используется для подтверждения успешного получения пакета.
6. PSH – инструктирует получателя протолкнуть данные, накопившиеся в приемном буфере, в приложение пользователя.
7. RST – флаг сброса отправляется от получателя к отправителю, когда пакет отправляется на конкретный хост, который этого не ожидал.
8. SYN – начинает соединение и синхронизирует порядковые номера. Первый пакет, отправленный с каждой стороны, должен в обязательном порядке иметь установленным этот флаг.
9. FIN – означает, что данных от отправителя больше нет. Поэтому он используется в последнем пакете, отправленном отправителем.

Они используются для управления соединением, передачи данных и завершения сессий. Благодаря этим флагам можно определить текущее состояние соединения и характер обмена данными.

Поле «Данные» содержит информацию, которую передает приложение, использующее транспортный уровень. То есть, данные, которые находятся в этом поле, принадлежат уровню выше транспортного – прикладному уровню. Таким образом, именно в этом поле хранятся данные о протоколе RDP.

Начиная с версии RDP 8.0 (введенной с Windows Server 2012 и Windows 8), протокол стал поддерживать UDP (англ. User Datagram Protocol – протокол пользовательских датаграмм) как дополнительный транспортный протокол для оптимизации работы в условиях нестабильных сетей [8]. Поэтому на структуру UDP протокола тоже стоит обратить внимание. В отличие от TCP протокол UDP имеет минималистичный заголовок. Его можно увидеть на следующем рисунке.

Заголовок UDP



Рисунок 4 – Структура UDP-заголовка

Из полезной информации можно выделить только номера портов и поле «Данные».

Стоит отметить, что RDP использует UDP в случаях, когда нужно минимизировать задержки и повысить производительность, особенно для мультимедийных задач или нестабильных сетевых условий [8]. Однако TCP остаётся основным протоколом для критически важных операций, таких как управление соединением и передача данных с высокой надежностью.

Рассмотрение структуры пакетов, представленной в данном разделе, позволяет получить общее представление о принципах организации передачи данных в сетях и месте протокола RDP в стеке сетевых протоколов. Однако для решения задачи идентификации RDP-трафика среди общего потока данных возникают некоторые сложности.

Во-первых, RDP-трафик обычно передаётся в зашифрованном виде. Даже если программно получить доступе к полю данных в заголовках TCP и UDP, то расшифровка содержимого без наличия ключа шифрования становится практически невозможной. Это делает анализ RDP-заголовков недостаточно эффективным для точного определения наличия RDP-трафика.

Во-вторых, использование портов для идентификации RDP также не является надёжным методом. Злоумышленники могут изменить стандартный порт, применяемый для RDP-соединений, с целью избежать обнаружения. Более того, если на одном устройстве функционирует несколько RDP-сессий, для каждой из

них могут использоваться разные порты, что дополнительно усложняет задачу анализа.

Таким образом, возникает вопрос: каким образом можно надёжно отличить трафик протокола удалённого рабочего стола от других протоколов, наблюдаемых в сетевой среде?

Ответ на данный вопрос будет рассмотрен в следующем разделе, где проанализированы ключевые метрики и особенности, характерные для RDP-трафика.

2 Особенности и характерные признаки RDP-трафика

В рамках моей предыдущей курсовой работы, посвящённой теме «Статистический анализ сетевого трафика для обнаружения активной RDP-сессии», были изучены и применены различные методы статистического анализа для выявления RDP-трафика. В частности, рассматривались такие подходы, как:

- анализ распределения временных интервалов между пакетами;
- исследование распределения размера пакетов;
- анализ частоты флагов PSH;
- нахождение отношения входящего и исходящего трафика.

Результаты курсовой работы показали, что использование этих методов в совокупности позволяет с высокой степенью вероятности идентифицировать RDP-трафик в сетевой среде.

В рамках настоящего исследования статистические методы рассматриваются более детально. Они дополняются новыми метриками, учитывающими динамические особенности RDP-сессий. Эти признаки используются в качестве входных данных для обучения нейронной сети, что позволяет значительно повысить точность и надёжность обнаружения RDP.

В данном разделе подробно описываются признаки, характерные для RDP-трафика и способы расчета метрик на основе этих признаков. Все представленные метрики сопровождаются формулами и пояснениями их реализации в программной системе. Стоит отметить, что вся информация, используемая при вычислении метрик, собирается самой программой. Программа извлекает из перехваченных пакетов исключительно те данные, которые требуются для расчёта метрик.

2.1 Признаки на основе анализа временных интервалов между сетевыми пакетами

Анализ временных интервалов между пакетами может помочь выявить RDP-сессии. Как правило, промежутки между пакетами RDP-трафика короче, чем у других видов трафика. Это объясняется тем, что протокол RDP рассчитан на передачу данных в реальном времени и требует быстрой доставки информации для поддержания стабильного соединения с удаленным рабочим столом. Соответственно, если в сети наблюдается большое количество пакетов с короткими временными интервалами, это может свидетельствовать о нали-

ции активной RDP-сессии. Тем не менее, важно помнить, что существуют и другие виды трафика, использующие высокие скорости передачи данных и характеризующиеся малыми интервалами между пакетами. Поэтому вычисляется несколько метрик, получаемые из промежутков временных интервалов, чтобы улучшить процесс определения RDP-трафика.

2.1.1 Вычисление задержки и стандартного отклонения временных интервалов между пакетами

Под задержкой будем понимать среднее время, которое проходит между отправкой и получением пакетов. Пусть t_1, t_2, \dots, t_n , где t_i – это временная метка i -го пакета. Разработанная программа запоминает все эти временные метки и находит разницу (Δt_j) между двумя пакетами следующим образом:

$$\Delta t_j = t_{i+1} - t_i, \text{ для всех } i = \overline{1, n-1} \quad (1)$$

Тогда задержка вычисляется по следующей формуле:

$$\mu_L = \frac{1}{n-1} \sum_{i=1}^{n-1} \Delta t_i \quad (2)$$

Исходя из формулы (2) можно заметить, что задержка – это среднее значение временных интервалов между пакетами.

Вычисление стандартного отклонения, получается из значения задержки по следующей формуле:

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n-1} (\Delta t_i - \mu_L)^2} \quad (3)$$

Среднее значение (задержка) и стандартное отклонение позволяет оценить стабильность интервалов времени и, следовательно, стабильность передачи данных, что является одной из характерных черт RDP-трафика.

2.1.2 Вычисление среднего джиттера и медианы временных интервалов

Джиттером называется изменение временной задержки между передачей данных и их получением по сетевому соединению. Проще говоря, джиттер отражает нестабильность задержки. Данная величина вычисляется по следующей

формуле:

$$J_i = |\Delta t_{i+1} - \Delta t_i|, \quad \text{для } i = \overline{1, n-2}. \quad (4)$$

Тогда средний джиттер вычисляется следующим образом:

$$\mu_J = \frac{1}{n-2} \sum_{i=1}^{n-2} J_i \quad (5)$$

Этот показатель характеризует, насколько стабильны интервалы между пакетов, что важно для оценки качества RDP-сессий, поскольку низкий джиттер необходим для стабильной работы удаленного рабочего стола.

Исходя из формул вычисления задержки и джиттера может показаться, что эти величины похожи, но это не так. Они измеряют разные аспекты передачи данных, и каждая из этих метрик имеет свое значение в контексте анализа RDP-трафика. Задержка показывает, как долго передаются пакеты, а джиттер измеряет колебания этого времени.

Медиана временных интервалов является также полезной метрикой, которая может помочь сгладить влияние выбросов и экстремальных значений. Медиана представляет собой среднее значение в отсортированном ряду интервалов времени [9]. Вследствие этого временные метки t_i ($i = \overline{1, n}$) перехваченных пакетов сортируются по возрастанию.

Далее если количество интервалов n нечетное, то медианой будет центральный элемент отсортированного ряда. Если n четное, медианой будет среднее значение двух центральных элементов.

Таким образом медиана вычисляется следующим образом:

$$m_t = \begin{cases} t_{n/2}, & \text{если } n \text{ нечетное} \\ (t_{n/2} + t_{n/2+1})/2, & \text{если } n \text{ четное} \end{cases} \quad (6)$$

2.2 Признаки, связанные с количественными характеристиками трафика

Помимо анализа временных интервалов, важную информацию о характере трафика можно получить из количественных характеристик передаваемых пакетов. Анализ объёма и распределения трафика позволяет выявить ключевые закономерности, присущие протоколу RDP. Такие признаки, как соотношение

входящего и исходящего трафика, распределение объёмов между UDP- и TCP-пакетами, а также средний размер передаваемых пакетов, отражают особенности поведения RDP-сессий. Каждый сетевой протокол обладает уникальным паттерном обмена данными, и изучение этих характеристик играет важную роль в их идентификации.

2.2.1 Вычисление отношения объема входящего на исходящий трафик

Разработанная программа сохраняет пакеты относительно инициатора подключения (клиента) и целевого устройства (сервера). В процессе работы накапливаются две величины: объем исходящего трафика (V_{src}) — количество данных, отправляемых клиентом, и объем входящего трафика (V_{dest}) — количество данных, отправляемых сервером.

Отношение объема входящего на исходящий трафик вычисляется по формуле:

$$r_{dest/src}^{init} = \frac{V_{dest}}{V_{src}} \quad (7)$$

Аналогично отношение объема трафика относительно сервера вычисляется как:

$$r_{dest/src}^{targ} = \frac{V_{src}}{V_{dest}} \quad (8)$$

Эти две величины позволяют оценить симметричность обмена данными между клиентом и сервером. Для протокола RDP характерен асимметричный обмен трафиком: большая часть данных передается от сервера к клиенту, что обусловлено характером работы протокола, где сервер передает графическую информацию, а клиент отправляет команды управления.

Рассмотрение обеих величин важно, поскольку оно обеспечивает всесторонний анализ взаимодействия между клиентом и сервером. Например, высокое значение $r_{dest/src}^{init}$ (преобладание входящего трафика) может указывать на активность RDP-сессии, тогда как отклонения в этих показателях могут сигнализировать о другом типе соединения. Метрика особенно полезна в контексте других признаков, так как её устойчивость к шуму и вариативность в зависимости от типа трафика делает её важным элементом для классификации с использованием нейронной сети [1].

2.2.2 Вычисление отношения объема UDP- и TCP-трафиков

В предыдущем разделе упоминалось, что начиная с версии 8.0, протокол RDP поддерживает передачу данных по UDP. Это особенно заметно в сетевом обмене между компьютерами с установленной операционной системой Windows. По протоколу UDP в основном передаются графические данные, а также действия, совершаемые мышью и клавиатурой, что делает его важной частью работы RDP. Учитывая это, исключать анализ UDP-трафика из общей картины трафика было бы неправильно.

Разработанная программа отслеживает объем переданных пакетов по протоколам UDP и TCP, а также вычисляет их соотношение по следующей формуле:

$$r_{udp/tcp} = \frac{V_{udp}}{V_{tcp}} \quad (9)$$

Таким образом, данная метрика отражает баланс использования двух транспортных протоколов в рамках одного соединения [2]. Для RDP характерно заметное преобладание трафика по протоколу TCP, однако наличие UDP-трафика с характерным объемом может служить дополнительным индикатором активности RDP-сессии.

Эта метрика становится особенно эффективной в сочетании с другими признаками, улучшая точность классификации сетевого трафика и выявления RDP.

2.2.3 Вычисление среднего значения объема пакетов

При работе протоколов прикладного уровня данные передаются пакетами различных объемов. Размер пакетов может варьироваться в зависимости от характера передаваемой информации: от небольших управляющих сообщений до крупных сегментов данных. Исследование размеров передаваемых пакетов может оказаться полезным инструментом при анализе сетевого трафика, особенно когда речь идет об определении аномалий или подозрительных активностей.

В разработанной программе анализируется размер блока «Данные», содержащегося в заголовках протоколов TCP и UDP. В процессе перехвата трафика программа сохраняет размеры полезной нагрузки каждого пакета:

$p_{s_1}, p_{s_2}, \dots, p_{s_n}$. На основании этих данных вычисляется среднее значение объема передаваемых пакетов:

$$\mu_s = \frac{1}{n} \sum_{i=1}^n \Delta p_{s_i} \quad (10)$$

Эта метрика предоставляет информацию о характере передаваемого трафика. Для протокола RDP характерен обмен пакетами малого и среднего размера, отражающий отправку управляющих сигналов, таких как движения мыши, нажатия клавиш, или обновления небольших частей экрана. В отличие от этого, потоки мультимедиа или файлы, передаваемые другими протоколами, обычно характеризуются значительно большими средними размерами пакетов.

Значимость среднего значения объема пакетов заключается в его способности идентифицировать типичные характеристики RDP-сессий и исключать несоответствующий трафик, тем самым улучшая точность анализа и классификации. В сочетании с другими метриками эта характеристика позволяет более точно выделять RDP-трафик среди общего потока сетевых данных.

2.3 Признаки на основе анализа параметров TCP-соединений

Протокол TCP предоставляет множество полей в своих заголовках, каждое из которых несет важную информацию о состоянии соединения, обмене данными и управлении потоком. Эти параметры играют ключевую роль в обеспечении надёжности передачи данных, но также могут служить источником ценной информации для анализа сетевого трафика.

В результате детального изучения особенностей сетевого взаимодействия по протоколу RDP были выявлены признаки, которые позволяют отличить данный тип трафика от других. Эти признаки связаны с использованием управляющих флагов TCP (PSH, ACK, SYN, FIN, RST), размером окна и другими параметрами, которые характеризуют поведение соединения.

2.3.1 Вычисление частоты флагов PSH и ACK

В контексте протокола RDP флаг PSH активно применяется для передачи пользовательских действий, таких как нажатия клавиш и движение мыши, а также для пересылки буферизованных изображений или звуковых данных. Наличие PSH-флагов может быть индикатором активности RDP-сессий, что делает их анализ полезным инструментом для идентификации такого трафика.

Для анализа частоты PSH-флагов разработанная программа подсчитывает количество TCP-пакетов с установленным флагом PSH для каждой стороны

соединения. Это позволяет выделить:

- $V_{P_{init}}$: объем входящего трафика с установленным флагом PSH относительно клиента;
- $V_{P_{targ}}$: объем входящего трафика с установленным флагом PSH относительно сервера;
- V_{tcp}^{init} : общее количество входящих TCP-пакетов, получаемых клиентом;
- V_{tcp}^{targ} : общее количество входящих TCP-пакетов, получаемых сервером.

На основе этих данных вычисляются частоты флагов PSH для каждой стороны.

Для клиента:

$$r_{psh}^{init} = \frac{V_{P_{init}}}{V_{tcp}^{init}} \quad (11)$$

Для сервера:

$$r_{psh}^{targ} = \frac{V_{P_{targ}}}{V_{tcp}^{targ}} \quad (12)$$

Анализ частоты PSH-флагов позволяет выявить характерный для RDP трафик, где их интенсивность выше, чем у большинства других протоколов. Рассмотрение этих метрик как относительно клиента, так и относительно сервера важно для учёта специфики взаимодействия между сторонами: данные клиента часто включают команды, а данные сервера — графическую или звуковую информацию. Такой двусторонний анализ помогает точнее охарактеризовать тип и особенности обмена данными.

Частота флагов ACK вычисляется аналогично частоте PSH-флагов. Программа накапливает данные о числе TCP-пакетов с установленным флагом ACK относительно клиента ($V_{A_{init}}$) и сервера ($V_{A_{targ}}$). После этого рассчитываются частоты ACK-флагов.

Для клиента:

$$r_{ack}^{init} = \frac{V_{A_{init}}}{V_{tcp}^{init}} \quad (13)$$

Для сервера:

$$r_{ack}^{targ} = \frac{V_{A_{targ}}}{V_{tcp}^{targ}} \quad (14)$$

Анализ частоты ACK-флагов предоставляет информацию о количестве

подтверждений, отправляемых и получаемых сторонами соединения. В контексте RDP трафика частота ACK может отражать характер обмена данными, где сервер часто отправляет подтверждения для поступающих от клиента команд и событий.

2.3.2 Вычисление отношения ACK/PSH

Для более детального анализа характерных признаков RDP-сессий программа также вычисляет отношения частот ACK- и PSH-флагов для клиента и сервера.

Для клиента:

$$r_{ack/psh}^{init} = \frac{r_{ack}^{init}}{r_{psh}^{init}} \quad (15)$$

Для сервера:

$$r_{ack/psh}^{init} = \frac{r_{ack}^{targ}}{r_{psh}^{targ}} \quad (16)$$

Данный подход объединяет две ключевые особенности RDP-трафика: интенсивное использование PSH-флагов для передачи данных сервером и частое подтверждение этих данных клиентом с помощью ACK-флагов.

RDP генерирует активный двусторонний трафик, где сервер отправляет значительные объемы данных (в том числе с PSH-флагами), а клиент, в свою очередь, регулярно отвечает подтверждениями (ACK-флагами). Вычисление отношения ACK/PSH позволяет выявить баланс между передачей данных и подтверждениями, что особенно важно для идентификации интерактивных сессий.

Эта комбинация признаков позволяет более точно отличать RDP-трафик от других видов трафика, где подобное поведение не столь выражено.

2.3.3 Вычисление разности числа исходящих и входящих ACK-флагов

Баланс между количеством подтверждений (ACK), отправленных и полученных в рамках TCP-сессии, может отражать динамику взаимодействия сторон. В интерактивных протоколах, таких как RDP, где клиент активно подтверждает получение данных от сервера, разность числа ACK-флагов может быть индикатором асимметрии трафика. Этот показатель предоставляет дополнительную

информацию о характере передачи данных, что полезно для анализа таких сессий.

В программе вычисляется модуль разности числа TCP-пакетов с установленным флагом ACK, причем рассматривается только относительно клиента:

$$d_{ack}^{init} = |V_{A_{src}} - V_{A_{dest}}| \quad (17)$$

где $V_{A_{src}}$ – число исходящих ACK-флагов от клиента, а $V_{A_{dest}}$ – число входящих ACK-флагов, получаемых клиентом. В данном случае необязательно еще вычислять разность исходящих и входящих ACK-флагов относительно сервера, так как данные величины будут одинаковы. Ведь вычисления проводятся под модулем.

Данная метрика полезна тем, что она позволяет выявить асимметрию в подтверждениях, характерную для RDP. Например, в типичной RDP-сессии клиент отправляет больше ACK-флагов, подтверждая получение данных от сервера. Напротив, другие протоколы могут демонстрировать иной баланс или меньшую интенсивность подтверждений. В сочетании с другими метриками разность числа ACK-флагов помогает уточнить модель поведения трафика и улучшает точность обнаружения RDP.

2.3.4 Вычисление отношения числа флагов SYN на сумму флагов FIN + RST

Для протоколов удалённого доступа, таких как RDP, характерно длительное и стабильное соединение, которое устанавливается один раз и остается активным в течение продолжительного времени. В отличие от других приложений, где соединения часто открываются и закрываются, RDP демонстрирует низкую частоту использования флагов FIN и RST, завершающих соединение, после начального установления связи с помощью SYN. Таким образом, соотношение числа флагов SYN к сумме флагов FIN и RST может быть полезным индикатором.

В программе подсчитываются TCP-пакеты с установленными флагами SYN ($V_{S_{init}}, V_{S_{targ}}$), FIN ($V_{F_{init}}, V_{F_{targ}}$) и RST ($V_{R_{init}}, V_{R_{targ}}$) для клиента и сервера. После завершения временного интервала вычисляются отношения:

$$r_{syn/fin+rst}^{init} = \frac{V_{S_{init}} + 1}{V_{F_{init}} + V_{R_{init}} + 1}$$

$$r_{syn/fin+rst}^{targ} = \frac{V_{S_{init}} + 1}{V_{F_{init}} + V_{R_{init}} + 1}$$
(18)

Единица добавляется к числителю и знаменателю для избежания деления на ноль.

Это отношение полезно для анализа, так как в типичной RDP-сессии большая часть трафика проходит в рамках установленного соединения с минимальным использованием флагов FIN и RST.

2.3.5 Вычисление среднего значения размера окна и частоты его обновления

В ходе сеанса TCP-соединения размер окна может изменяться в зависимости от состояния сети и загруженности буфера принимающей стороны:

- когда принимающая сторона запрашивает больше данных (увеличивая размер окна), это считается обновлением окна.
- если окно уменьшается (когда буфер заполнен), это также обновление окна.

Программа при перехвате n пакетов сохраняет значения размера окна $p_{w_1}, p_{w_2}, \dots, p_{w_n}$ и вычисляет среднее значение по формуле:

$$\mu_w = \frac{1}{n} \sum_{i=1}^n \Delta p_{w_i}$$
(19)

В интерактивных протоколах, таких как RDP, изменения сетевой нагрузки и взаимодействия с пользователем приводят к частым изменениям размера окна, поскольку требуется быстрая передача данных с минимальной задержкой. Следовательно, частота обновлений окна может помочь в определении протокола удаленного рабочего стола.

Данная величина будет зависеть от некоторого промежуточного интервала τ и количества изменений размера окна k . Таким образом частота вычисляется по формуле:

$$r_w = \frac{k}{\tau}$$
(20)

Изменения размера окна характерны для интерактивных протоколов, та-

ких как RDP, где нагрузка на сеть и активность пользователя могут приводить к частым обновлениям. Высокая частота обновлений и характер изменения размера окна могут указывать на интенсивность взаимодействий, что делает эти метрики полезными для идентификации RDP-трафика.

3 Модель LSTM для анализа трафика

Для анализа сетевого трафика и выявления специфических протоколов, таких как RDP, необходимо применять подходы, которые способны обрабатывать последовательные данные с учетом их временной структуры. Одной из наиболее подходящих моделей для этой задачи является рекуррентная нейронная сеть на основе LSTM (Long Short-Term Memory).

Модель LSTM обладает уникальной способностью учитывать как кратковременные, так и долговременные зависимости в данных. Это делает ее особенно эффективной для анализа сетевого трафика, который состоит из последовательностей пакетов с временной корреляцией. В данном разделе описывается, почему именно LSTM была выбрана для задачи анализа трафика, как она реализована в программе, а также каким образом модель обучалась для выявления трафика RDP.

3.1 Особенности архитектуры LSTM

3.1.1 Введение в рекуррентные нейронные сети

Рекуррентные нейронные сети (Recurrent Neural Networks, RNN) занимают особое место в области машинного обучения благодаря своей способности обрабатывать последовательные данные. В отличие от традиционных нейронных сетей, которые рассматривают входные данные как независимые и статичные, RNN способны учитывать контекст предыдущих шагов, что делает их незаменимыми для анализа временных рядов, текстов, сигналов и других данных с временной структурой [11].

Ключевая особенность RNN заключается в наличии петли обратной связи, которая позволяет передавать информацию о предыдущих состояниях на следующие шаги. Это позволяет сети обучаться выявлять зависимости во времени, что особенно важно для анализа сетевого трафика, где порядок и временные связи между событиями играют критическую роль.

Однако классические RNN сталкиваются с проблемой обучения на длинных последовательностях из-за эффекта «затухания градиентов». В свою очередь, градиентом называется производная функции ошибки по параметрам модели. В процессе обратного распространения ошибки (backpropagation) градиенты используются для корректировки весов нейронов таким образом, чтобы минимизировать ошибку предсказаний модели. Когда ошибка распространяет-

ся обратно через множество слоев или временных шагов, её величина может значительно уменьшаться. Это приводит к тому, что веса при обновлении изменяются на слишком малые значения, и обучение проходит неэффективно или останавливается, то есть алгоритм обучения не сходится [10]. Эта проблема затрудняет захват долгосрочных зависимостей, что ограничивает их применение в сложных задачах.

Для решения этой проблемы была предложена архитектура Long Short-Term Memory (LSTM) немецкими исследователями Зеппом Хохрайтером (Sepp Hochreiter) и Юргеном Шмидхубером (Jürgen Schmidhuber) в 1997 году. Их работа, опубликованная в статье «Long Short-Term Memory», стала основой для дальнейшего развития рекуррентных нейронных сетей и нашла широкое применение в самых разнообразных областях, включая обработку естественного языка, распознавание речи и анализ временных рядов.

Благодаря введению специальных механизмов управления памятью LSTM способна сохранять информацию на протяжении долгих временных интервалов, что сделало эту модель чрезвычайно эффективной для работы с последовательными данными.

В данной работе основное внимание будет уделено архитектуре LSTM, так как она является наиболее подходящей для анализа сетевого трафика, где важно учитывать как краткосрочные, так и долгосрочные зависимости между событиями. LSTM позволяет извлечь максимальную информацию из временных метрик, обеспечивая высокую точность в задачах идентификации протоколов, таких как RDP.

3.1.2 Структура ячейки LSTM

Основной элемент модели – ячейка LSTM, которая состоит из нескольких ключевых компонентов, взаимодействующих через специальные механизмы управления потоком информации.

Каждая ячейка LSTM содержит следующие основные элементы:

1. Забывающий вентиль (Forget Gate).

Этот вентиль определяет, какая информация из состояния памяти на предыдущем шаге (C_{t-1}) должна быть забыта. Это важно для предотвращения «загрязнения» памяти нерелевантной информацией.

Формула забывающего вентиля:

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f), \quad (21)$$

где x_t – входной вектор (входные данные на текущем шаге), h_{t-1} – скрытое состояние с предыдущего шага, W_{xf} и W_{hf} – матрицы весов, b_i – смещение, σ – сигмоида, которая сжимает значения в диапазон $[0, 1]$, задавая «вес» забывания.

2. Входной вентиль (Input Gate).

Данная компонента управляет количеством новой информации, поступающей в ячейку, а также решает, какую часть новой информации из входных данных x_t и скрытого состояния h_{t-1} следует записать в память [14].

Входной вентиль определяется следующей формулой:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i). \quad (22)$$

Стоит отметить, что параллельно создаётся «кандидат» (Candidate state) новой информации (\tilde{C}_t):

$$\tilde{C}_t = \tanh(W_{xC}x_t + W_{hC}h_{t-1} + b_C). \quad (23)$$

3. Обновление состояния ячейки (Cell state). После работы забывающего и входного вентилей состояние памяти (ячейки) обновляется:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \quad (24)$$

где \odot обозначает поэлементное произведение.

Таким образом, модель сохраняет важную информацию из прошлого ($f_t \odot C_{t-1}$) и добавляет новую информацию ($i_t \odot \tilde{C}_t$).

4. Выходной вентиль (Output Gate).

Этот вентиль управляет тем, какая информация из текущего состояния ячейки должна быть передана в следующее состояние или использована для предсказания [13].

Формула выходного вентиля:

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o), \quad (25)$$

где o_t – выход выходного вентиля.

5. Скрытое состояние (Hidden State).

Состояние h_t Представляет информацию, используемую для выходного значения ячейки. Оно вычисляется как:

$$h_t = o_t \odot \tanh(C_t) \quad (26)$$

Каждый шаг работы LSTM можно описать следующим образом:

1. Сначала забывающий вентиль (f_t) удаляет ненужную информацию из состояния памяти.
2. Затем вентиль записи (i_t) решает, какую новую информацию добавить, и создаёт кандидата новой информации (\tilde{C}_t).
3. Обновляется состояние памяти (C_t) с учётом старой и новой информации.
4. Выходной вентиль (o_t) управляет тем, какая часть информации из памяти используется для обновления скрытого состояния (h_t).
5. Скрытое состояние (h_t) передаётся дальше в цепочке, одновременно выступая как выход текущей ячейки.

Таким образом, LSTM эффективно решает проблему долговременной зависимости, обеспечивая контроль над сохранением, забыванием и использованием информации.

3.1.3 Преимущества LSTM перед другими моделями

LSTM (Long Short-Term Memory) – одна из наиболее подходящих моделей для анализа последовательных данных, таких как сетевой трафик, благодаря ряду её ключевых преимуществ:

1. Учёт временных зависимостей. LSTM обладает механизмом памяти, позволяющим учитывать как краткосрочные, так и долгосрочные зависимости. В анализе сетевого трафика это крайне важно, так как порядок и время между событиями могут свидетельствовать о типе используемого протокола. Например, RDP имеет специфические временные закономерности и последовательности пакетов, которые LSTM может захватывать.
2. Устойчивость к затуханию градиентов. Благодаря использованию специальных элементов управления (входной, выходной и забывающий вентили), LSTM справляется с проблемой затухания градиентов, характерной для классических RNN. Это делает модель устойчивой при работе с длин-

ными временными последовательностями, что критично при анализе претяжённых сессий.

3. Способность обрабатывать данные с разной длиной последовательностей. В сетевом трафике разные сессии могут иметь разное количество пакетов и временных интервалов. LSTM может обрабатывать такие данные, не требуя строгой унификации длины, что делает её гибкой и адаптивной.
4. Эффективная работа с метриками. Модель способна извлекать скрытые паттерны из временных метрик, таких как частота флагов, размер окна, джиттер и другие. Эти метрики становятся входными признаками, которые LSTM анализирует, чтобы классифицировать протоколы.

Таким образом, среди различных подходов к анализу сетевого трафика, LSTM выделяется благодаря своей способности учитывать временные зависимости, устойчивости к шумам и высокой точности работы с разнообразными временными метриками. Однако при усложнении задачи идентификации RDP и увеличении требований к вычислительным ресурсам можно рассмотреть использование альтернативных моделей. Например, GRU, являющаяся упрощённой версией LSTM, может быть эффективна для менее сложных задач, поскольку работает быстрее за счёт меньшего количества параметров. Также можно обратить внимание на свёрточные нейронные сети (1D-CNN), которые хорошо справляются с извлечением признаков, но уступают в учёте долгосрочных зависимостей. В текущем контексте LSTM представляет собой сбалансированное и оптимальное решение для определения RDP. Стоит отметить, что в дальнейшей перспективе возможно исследование комбинаций LSTM с другими подходами для повышения точности и производительности.

3.2 Применение LSTM в задаче анализа трафика

В данном подразделе рассматривается реализация модели LSTM для задачи анализа сетевого трафика, а именно выявления протокола RDP среди общего объёма данных. Модель была реализована с использованием языка программирования Python и библиотек Keras. Ниже представлено описание модели и её структуры.

Модель LSTM была определена следующим образом:

```
1
2 # Определение модели LSTM
3 def define_model(self):
4     self.model = Sequential()
```

```

5      # Входной слой LSTM
6      self.model.add(LSTM(units=64, return_sequences=True))
7      # Полносвязный слой для классификации
8      self.model.add(Dense(units=32, activation='relu'))
9      # Выходной слой: два нейрона для предсказания классов
10     # [1, 0] (RDP) и [0, 1] (не RDP)
11     self.model.add(Dense(units=2, activation='softmax'))
12     # Компиляция модели
13     self.model.compile(optimizer='adam',
14                         , loss='categorical_crossentropy',
15                         , metrics=['accuracy'])
16     print('\nМодель LSTM определена успешно')

```

Листинг 1: Определение модели LSTM

Модель состоит из трёх основных слоёв:

- Входной слой LSTM. Слой определён с помощью функции *LSTM* из библиотеки Keras, в которой количество нейронов установлено равным 64. Выбор такого количества обусловлен необходимостью балансировать между вычислительными затратами и способностью модели захватывать временные зависимости в сетевом трафике. Параметр *return_sequences=True* позволяет передавать последовательности данных следующему слою, что важно для анализа временных зависимостей.
- Полносвязный слой для классификации. В данном слое используются 32 нейрона с функцией активации ReLU (*Rectified Linear Unit*). Эта функция активации обеспечивает линейность при положительных значениях входного сигнала, что позволяет нейронам эффективно обрабатывать данные и минимизировать проблему исчезающего градиента. Полносвязный слой преобразует выходы LSTM в более компактное представление, что позволяет выделить наиболее значимые признаки и передать их в следующий слой. Это упрощает задачу классификации, снижая размерность данных.
- Выходной слой. Содержит 2 нейрона, что соответствует двум классам: [1, 0] для протокола RDP и [0, 1] для всего остального трафика. В качестве функции активации используется *Softmax*, которая интерпретирует выходные значения как вероятности принадлежности входных данных к каждому из классов.

Модель была оптимизирована с помощью алгоритма *Adam*, который является эффективным методом стохастической оптимизации, адаптирующим скорость обучения для каждого параметра. В качестве функции потерь использует-

ся *Categorical Crossentropy*, применяемая для многоклассовой классификации, поскольку она сравнивает распределение вероятностей, возвращаемое моделью, с истинными метками.

Стоит отметить, что все слои и параметры были выбраны с учётом специфики задачи анализа трафика.

В LSTM-слое выбрано 64 нейрона, так как этого оказалось достаточно для моделирования сложных временных зависимостей в данных сетевого трафика. Полносвязный слой имеет 32 нейрона для снижения размерности и увеличения способности модели выделять важные признаки. Совокупность использования слоя LSTM и Dense позволяет модели выделять наиболее важные признаки, характерные для классов данных.

Таким образом, представленная модель LSTM образует гибкую и мощную архитектуру для анализа сетевого трафика и выявления протокола RDP. В следующем разделе будет детально описан функционал программы, эксплуатирующей модель нейронной сети LSTM.

4 Программная реализация

В рамках последней курсовой работы, посвящённой теме «Статистический анализ сетевого трафика для обнаружения активной RDP-сессии», была разработана программа, предоставляющая следующие возможности:

1. Перехват трафика:
 - Перехват всего сетевого трафика с выводом информации о пакетах в консоль.
 - Фильтрация пакетов с признаками активной RDP-сессии на основе статистических методов.
2. Сохранение и загрузка данных:
 - Сохранение перехваченного трафика в файл с ключевой информацией о пакетах.
 - Загрузка и анализ ранее собранных данных, сохранённых этой программой.
3. Анализ трафика:
 - Просмотр активных сессий (обмен трафиком длительностью более 10 секунд).
 - Детальный анализ трафика по IP-адресам и построение графиков.

На базе программы «traffic-detection.py» создана новая версия с расширенным функционалом, включающим интеграцию модели LSTM для автоматического обнаружения активных RDP-сессий. Подробности новой структуры описаны в следующих разделах.

4.1 Общая структура программы

Для интеграции модели нейронной сети в программу «traffic-detection.py» было принято решение провести полный рефакторинг исходной логики программы. Исходно вся функциональность находилась в одном файле «traffic-detection.py», что ограничивало гибкость и масштабируемость. После тщательного пересмотра и рефакторинга, была добавлена логика для работы с нейронными сетями, что позволило значительно улучшить эффективность программы.

Помимо пересмотра логики, в новой версии программы были внесены следующие изменения:

- Изменение алгоритмов классификации пакетов. Одной из ключевых задач при анализе сетевого трафика является правильная классификация

пакетов. В старой версии программы для классификации пакетов использовался алгоритм, который создавал структуры, называемые «сессиями». Эти сессии связывались с конкретными пакетами, основанными на информации о взаимодействии между двумя устройствами через TCP-протокол. Процесс был основан на принципе «трехстороннего рукопожатия» (Three-way Handshake), который включает следующие этапы:

1. Инициация соединения: клиент отправляет серверу пакет с флагом SYN, предлагая начать соединение.
2. Ответ сервера: сервер подтверждает получение запроса, отправляя клиенту пакет с флагами SYN и ACK.
3. Завершение соединения: клиент подтверждает установление связи, отправив пакет с флагом ACK.

Программа отслеживала эти этапы и фиксировала активные сессии, создавая соответствующие структуры. В новой версии программы логика изменилась. Вместо создания сессий на основе трехстороннего рукопожатия программа сохраняет в структуру пару IP-адресов (клиента и сервера), пару портов и время перехвата. При перехвате новых пакетов программа проверяет IP-адреса и порты, сопоставляя их с уже существующими структурами или создавая новые. Это позволило значительно упростить алгоритм и повысить его эффективность, что было также достигнуто благодаря добавлению многопоточности для параллельной обработки пакетов.

- Добавление динамического определения общего порта. Общий порт – это порт получателя, указанный в первом пакете при установке соединения. В ходе тестирования программы было выявлено, что в процессе активной сессии клиент или сервер могут изменять свои порты передачи данных, в то время как порт получателя остается неизменным. Это явление наблюдается в протоколах RDP и HTTP/HTTPS, где данные могут передаваться через несколько портов одновременно. Для эффективного отслеживания таких случаев был разработан алгоритм динамического определения общего порта. Это решение позволяет программе правильно отслеживать пакеты, даже если клиент или сервер меняют порты, но сохраняют общий порт получателя.
- Изменение логики обнаружения признаков RDP-сессии. Ранее програм-

ма для выявления признаков RDP-сессий использовала статистический анализ на основе расчета различных параметров сетевого трафика, таких как время между пакетами, размер пакетов и другие метрики. Каждые 5 секунд программа вычисляла параметры для каждой активной сессии и на основе пороговых значений принимала решение о наличии признаков RDP. В новой версии программы функцию выявления RDP-сессий теперь выполняет нейронная сеть, обученная на метриках, основанных на статистических признаках трафика. Как именно нейронная сеть осуществляет классификацию и как она была интегрирована в программу, будет подробно рассмотрено далее.

- Изменение построения графиков и добавление новых. В старой версии программы анализ сетевого трафика осуществлялся с помощью построения графиков, которые визуализировали различные параметры, такие как задержка, частота флагов и другие метрики. На основе этих графиков выявлялись потенциальные признаки RDP-сессий. В новой версии программы были добавлены дополнительные графики для улучшения анализа, а также для более детального отслеживания динамики сетевого трафика и его связи с активностью RDP.

Эти изменения обеспечили более эффективную работу программы и улучшили её способность к анализу сетевого трафика, позволяя выявлять признаки активных RDP-сессий с высокой точностью. Программа теперь использует современные методы обработки данных, включая нейронные сети, что делает её более гибкой и мощной для решения поставленной задачи.

4.2 Обучение модели LSTM и её применение в программе

Основная идея использования нейронной сети заключается в том, чтобы с фиксированным временным интервалом τ определять, является ли каждая активная сессия RDP. Для этого было выбрано значение $\tau = 15$ секунд. Таким образом, каждые 15 секунд программа анализирует накопленные данные, вычисляет метрики и выполняет предсказание для всех активных сессий. Это позволяет своевременно идентифицировать RDP-сессии в потоке сетевого трафика.

Однако для того чтобы использовать модель нейронной сети в выявлении RDP-трафика, первым этапом необходимо было её обучить. Для этого была проведена серия экспериментов по сбору сетевого трафика с различных устройств

и протоколов.

Собранные данные сохранялись в файлы с помощью переработанной программы. Самой сложной частью оказался процесс разметки данных. Благодаря сохранённой информации о временных метках каждого перехваченного пакета удалось восстановить последовательность событий в сетевом трафике. Данные были разбиты на пятнадцатисекундные интервалы, в рамках которых рассчитывались ключевые метрики, необходимые для обучения модели.

Исследование поведения протокола RDP позволило выявить двадцать одну метрику, которые позволяют определять RDP-сессии в сетевом трафике. Каждые 15 секунд для каждой активной сессии рассчитываются следующие показатели:

1. Средняя задержка.
2. Стандартное отклонение временных интервалов между пакетами.
3. Средний джиттер.
4. Медиана временных интервалов.
5. Отношение объёма входящего к исходящему трафику относительно клиента.
6. Отношение объёма входящего к исходящему трафику относительно сервера.
7. Отношение объёма UDP-трафика к объёму TCP-трафика.
8. Среднее значение объёма пакетов, полученных клиентом.
9. Среднее значение объёма пакетов, полученных сервером.
10. Частота флагов PSH относительно клиента.
11. Частота флагов PSH относительно сервера.
12. Частота флагов ACK относительно клиента.
13. Частота флагов ACK относительно сервера.
14. Отношение флагов ACK к флагам PSH относительно клиента.
15. Отношение флагов ACK к флагам PSH относительно сервера.
16. Разница между числом исходящих и входящих флагов ACK относительно клиента.
17. Отношение количества флагов SYN к сумме флагов FIN и RST относительно клиента.
18. Отношение количества флагов SYN к сумме флагов FIN и RST относительно сервера.

19. Среднее значение размера окна.
20. Частота изменения размера окна.
21. Количество пакетов.

После расчёта всех этих метрик для каждой активной сессии создаётся вектор x_t , который затем подается на вход нейронной сети. Каждый такой вектор сохраняется в файл вместе с соответствующей меткой «RDP» или «не RDP». Такая разметка позволяет сформировать высококачественный обучающий набор данных.

Обучение модели проводится на размеченных данных, после чего обученная модель сохраняется в файл формата *.keras*. Основная программа загружает эту модель с помощью метода *load_model* из библиотеки *keras.models*. После загрузки модель переключается в режим предсказания, что позволяет ей обрабатывать данные в реальном времени. Используя метод *predict()*, каждые 15 секунд модель анализирует метрики активных сессий и делает предсказания об их принадлежности к RDP.

4.3 Демонстрация работы программы

Тестирование программы проводилось на виртуальных машинах (ВМ), работающих с различными операционными системами и подключённых к реальной локальной сети. Для эксперимента использовались следующие системы: Windows 10 Professional версии 22H2 (обозначим как Win), Kali Linux 2024.3 Release (Kali) и Ubuntu 22.04.5 LTS (Ubuntu). Эксперимент заключался в установке соединений между двумя ВМ, использующими различные протоколы, и запуске программы с обученной моделью нейронной сети на третьей ВМ для анализа сетевого трафика.

Особенностью эксперимента стало то, что поведение протокола RDP при соединении между ВМ с разными операционными системами оказалось различным. Это повлияло на показатели признаков и метрик, используемых моделью для классификации трафика. Поэтому для повышения точности обучения и предсказаний в модели учитывались особенности взаимодействия различных операционных систем.

В этом разделе будет продемонстрирована работа программы с разными видами прикладных протоколов и проведен сравнительный анализ поведения RDP-сессий при взаимодействии разных ОС и альтернативных программ удаленного доступа.

4.3.1 Сравнение поведения прикладных протоколов

Для начала был запущен перехват трафика с фильтром RDP, как показано на рисунке 5. То есть сейчас включен режим, когда в консоль будет выводиться информация о пакетах, принадлежащие активным RDP-сессиям. Стоит отметить, что при перехвате теперь можно заметить индикаторы выполнения прямоугольной формы. На данном рисунке их количество сейчас равно семи. Они символизируют о том, что уже семь раз было совершено предсказание активных сессий с помощью модели LSTM. То есть с момента запуска программы прошло уже 105 секунд.

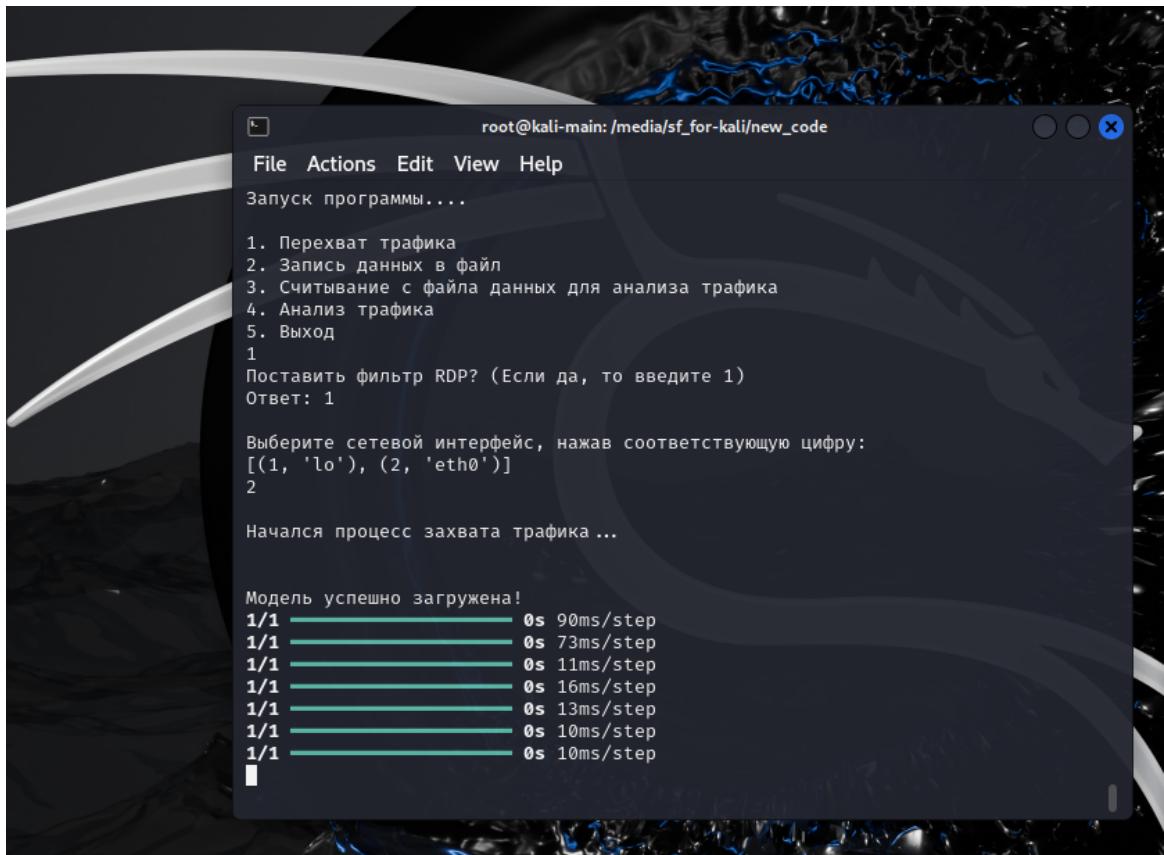


Рисунок 5 – Запуск перехвата трафика с фильтром RDP

На следующем рисунке показано подключение по RDP. В данном случае соединение устанавливается между Win (клиент) и Kali (сервер). Стоит отметить, что для осуществления такого подключения на Kali запускался сервис XRDP, бесплатный протокол удаленного доступа, основанный на протоколе RDP. Из рисунка 6 видно, что программа нашла активную RDP-сессию.

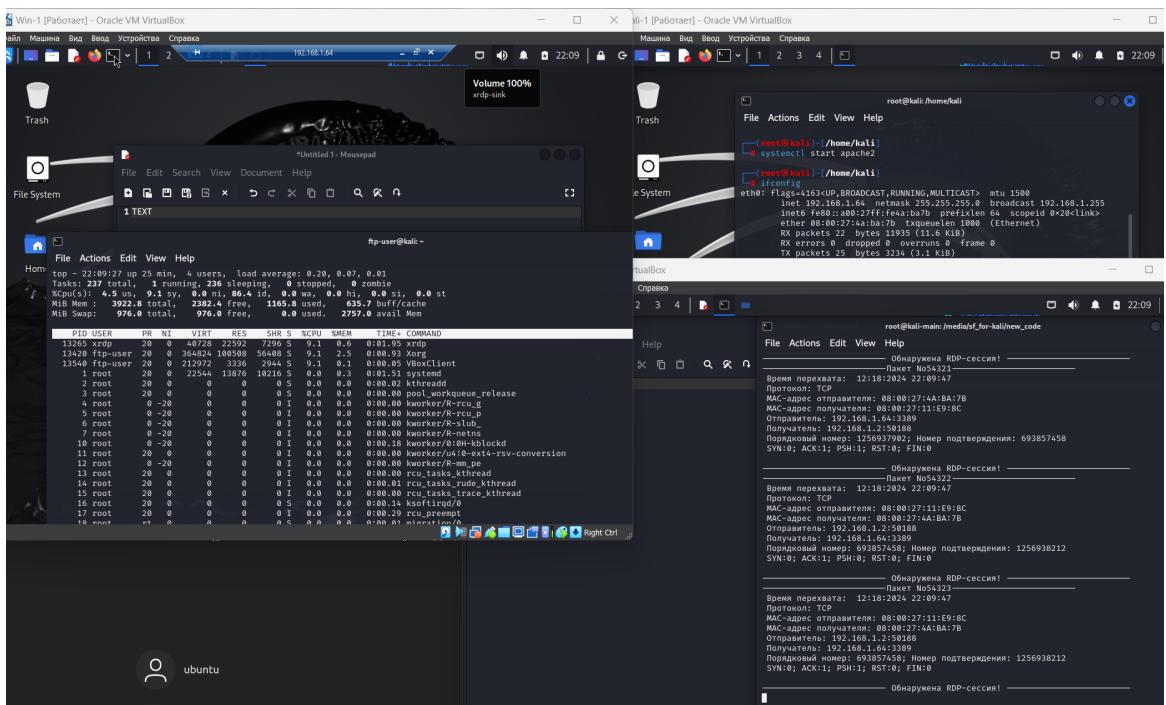


Рисунок 6 – Состояние программы при RDP-трафике

При подключении по RDP совершались движения мышкой и нажатия клавиш клавиатуры. Буквально после 30 секунд модель смогла определить RDP-трафик. После завершения RDP-сессии проводилась работа с другими протоколами.

На ВМ Kali был предварительно запущен HTTP-сервер, на котором хранилось несколько файлов разного размера, а также видео которое также можно было посмотреть. Открытие страницы HTTP-сервера совершилось с другой ВМ Win. На Win совершились активные попытки скачивания большого файла с сервера и вместе с этим просматривалось видео. Из рисунка 7 видно, что ложных срабатываний за весь период активного взаимодействия с HTTP сервером не наблюдалось.

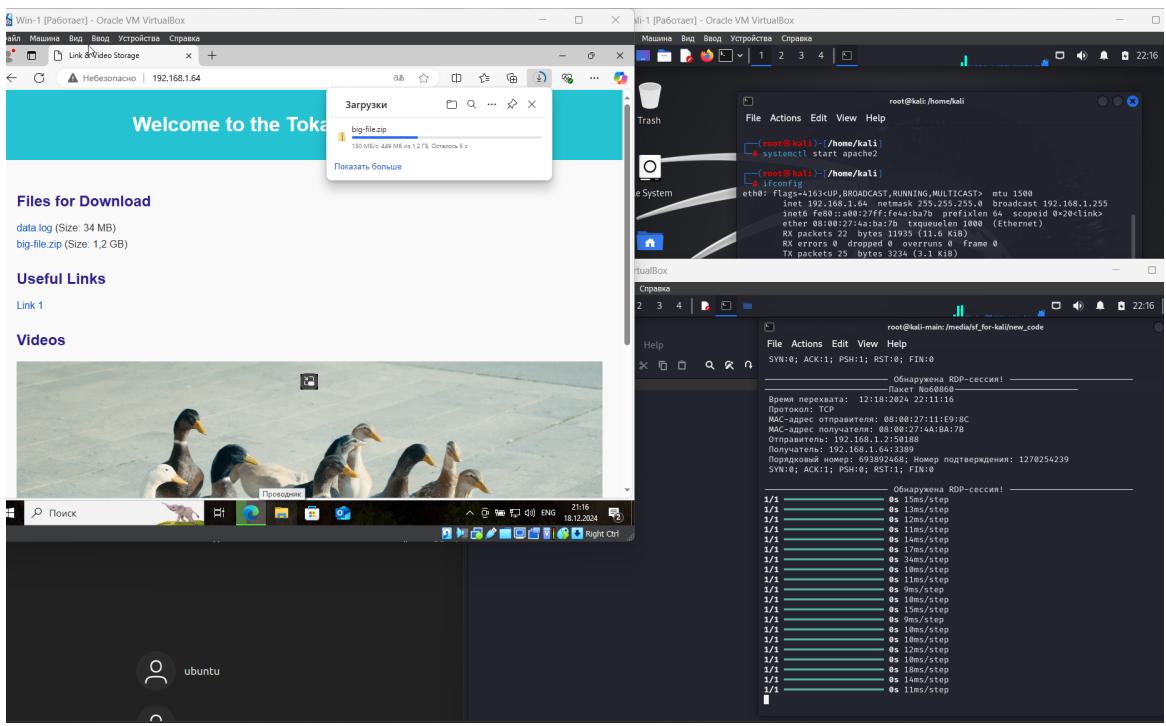


Рисунок 7 – Состояние программы при HTTP-трафике

Далее был рассмотрен еще один протокол прикладного уровня, а именно протокол FTP. Этот протокол полезен для загрузки и выгрузки файлов. Подключение осуществлялось по протоколу FTP между Win и Kali. После успешной установки соединения пользователем были сделаны различные команды, после того как пользователь нашел большой файл, то он начал его скачивать, как показано на рисунке 8.

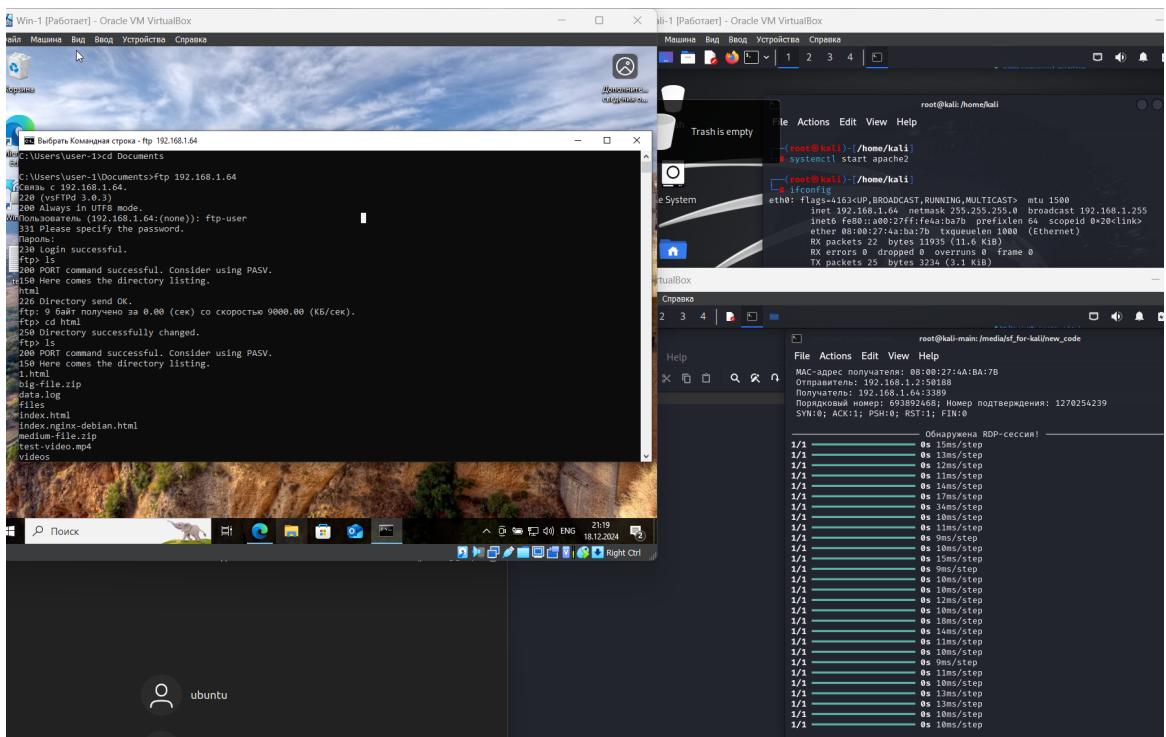


Рисунок 8 – Соединение по протоколу FTP

После нескольких успешных попыток скачивания большого файла ложных срабатываний также не наблюдалось.

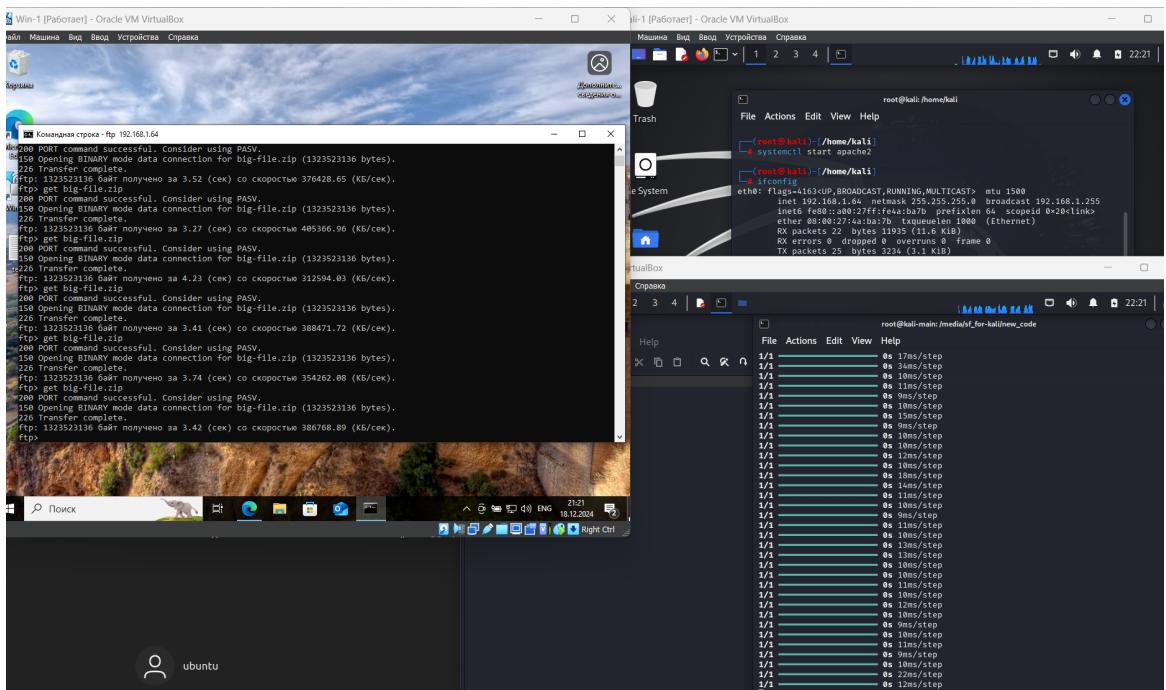


Рисунок 9 – Состояние программы при FTP-трафике

На следующих двух рисунках показаны графики одной из метрик.

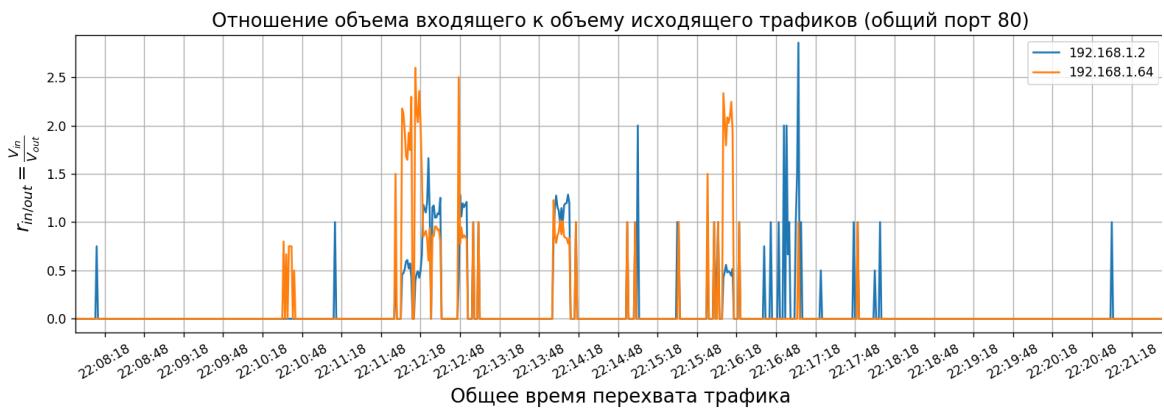


Рисунок 10 – График отношения объема входящего и исходящего трафиков (HTTP)

Из второго графика следует, что при FTP-сессии клиент и сервер при отправке и при получении получают почти одинаковое количество пакетов.

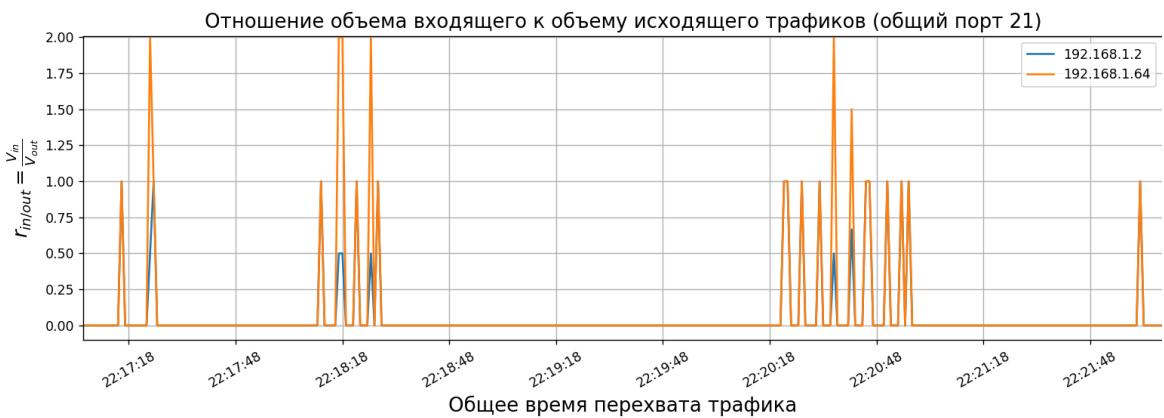


Рисунок 11 – График отношения объема входящего и исходящего трафиков (FTP)

Для сравнения на рисунке 12 показан следующий график.



Рисунок 12 – График отношения объема входящего и исходящего трафиков (RDP)

Из рисунков 10 - 12 видно, что по метрике отношения объемов входящего и исходящего трафиков поведение RDP различается от поведений HTTP и FTP.

Далее был рассмотрен еще один протокол прикладного уровня – SSH. На следующем рисунке показано состояние программы при активной работе с протоколом SSH.

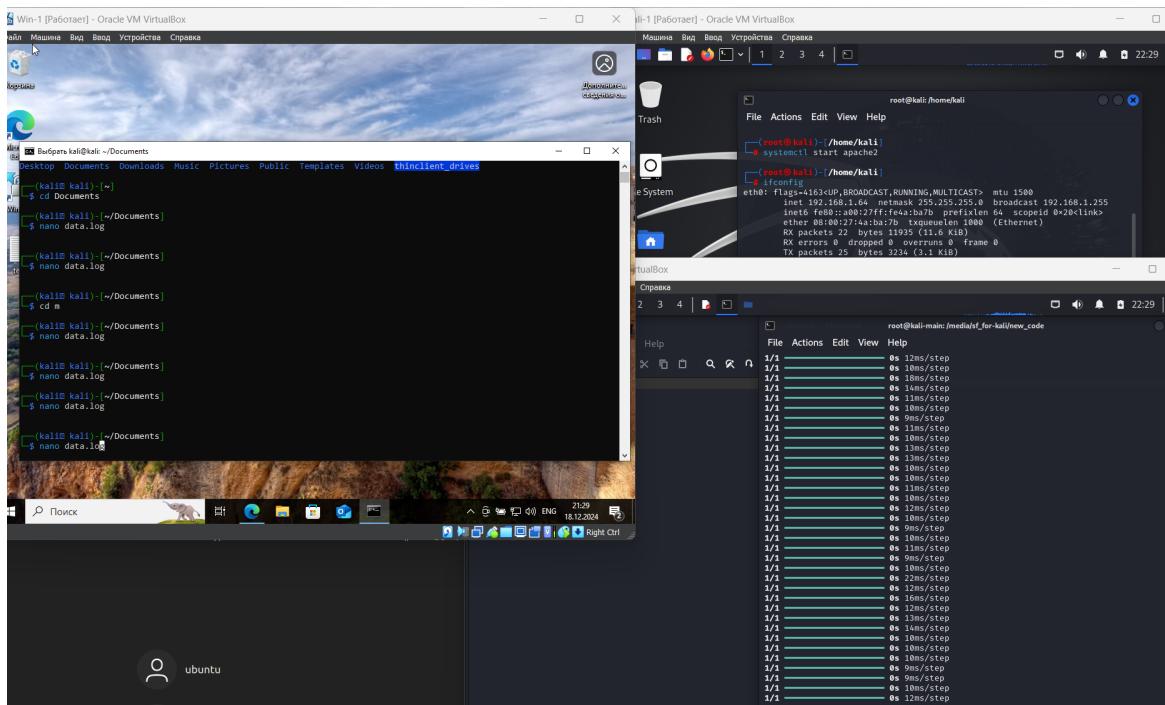


Рисунок 13 – Состояние программы при SSH-трафике

Стоит отметить, что если исследовать графики, показанные на рисунках 12 и 14, то поведение SSH и RDP достаточно похоже. Особенno если при подключении по SSH выполнять какие-либо непрерывные действия, например открыть большой файл и листать его строки. Если сравнить графики SSH и RDP по метрике отношения объемов входящего и исходящего трафиков, то можно заметить некоторую схожесть.



Рисунок 14 – График отношения объема входящего и исходящего трафиков (SSH)

Однако, если смотреть по другим признакам, как показано на следую-

ших двух рисунках, то можно заметить небольшое различие в этих двух видах протоколов.



Рисунок 15 – График отображения количества входящих пакетов (SSH)

Частота флагов PSH является одним из ключевых признаков, по которым можно сформировать небольшое представление о поведении протоколов. Особенно, если это касается протокола RDP, так как этот протокол активно использует флаги PSH для передачи данных. Флаги PSH сигнализируют о том, что данные должны быть немедленно переданы приложению, что важно для обеспечения быстрой реакции интерфейса пользователя. В данном случае из рисунка 16 видно, как клиент с IP-адресом 192.168.1.2 активно получает входящие TCP-пакеты с установленным флагом PSH от сервера.



Рисунок 16 – График отображения количества входящих пакетов (RDP)

Исходя из рассмотренных рисунков программа способна по определенным признакам различать один тип трафика от другого и демонстрировать надежность анализа даже в сложных случаях, когда различия между протоколами минимальны. Таким образом, программа может быть использована как для исследовательских целей, так и для практического применения в области сетевой безопасности.

4.3.2 Сравнительный анализ альтернативных программ удаленного доступа

Каждый сетевой протокол разработан для решения специфических задач, используя уникальные технологии. Тем не менее существуют протоколы,

которые имеют схожую структуру и поведение. В рамках данной работы важно также изучить альтернативные программы удаленного доступа, такие как Ammyy Admin, Remmina и RAdmin. Основной задачей этого раздела является демонстрация работы программы при использовании этих приложений.

Прежде чем перейти к анализу альтернативных программ, необходимо рассмотреть случай изменения стандартного порта для RDP. На виртуальной машине с операционной системой Windows порт по умолчанию для RDP-соединения был изменен с 3389 на 53389 [15]. На рисунке 17 показан процесс установления RDP-соединения, а также вывод программы с информацией о пакетах этой сессии. Программа успешно идентифицировала RDP-сессию, что подтверждает её способность работать независимо от номера порта. Это особенно важно, так как использование анализа, основанного только на портах, является ненадежным методом, как упоминалось ранее.

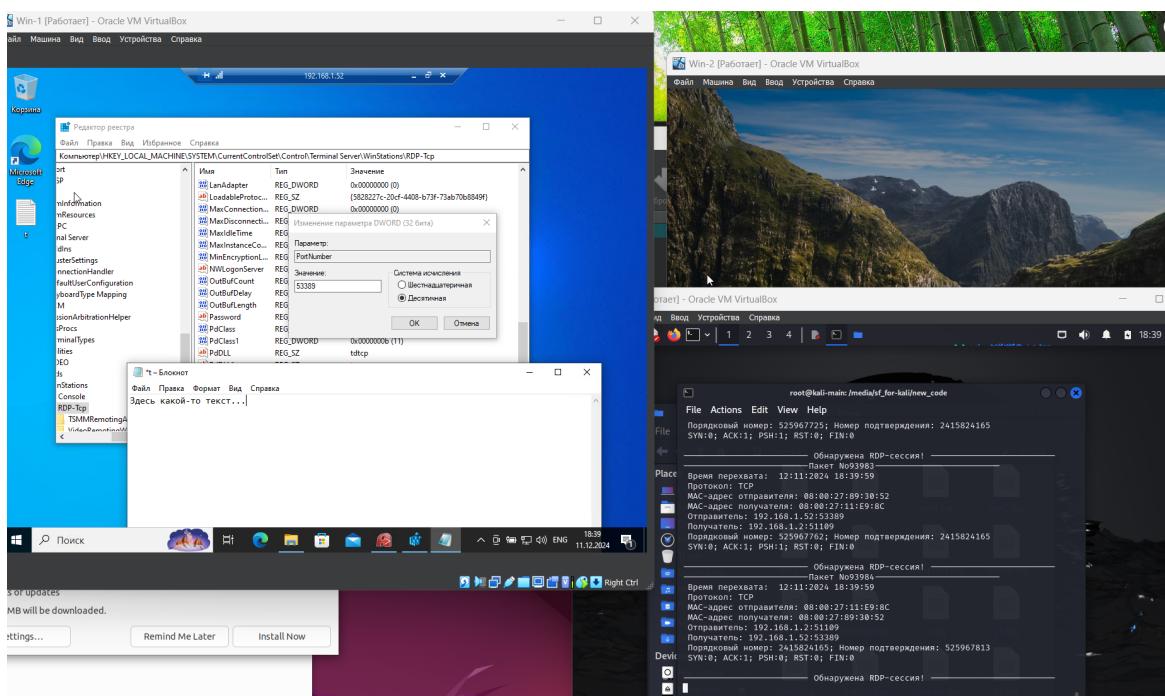


Рисунок 17 – Состояние программы при RDP-трафике (смена порта)

Ammyy Admin – это программа удаленного доступа и администрирования, разработанная компанией Ammyy Group. Она позволяет пользователям получать удаленный доступ к компьютерам и серверам через интернет и управлять ими в реальном времени. На следующем рисунке показано подключение с помощью Ammyy Admin между двумя ВМ операционной системы Windows. Так как данное приложение поддерживает протокол RDP, то программа смогла распознать признаки RDP-сессии.

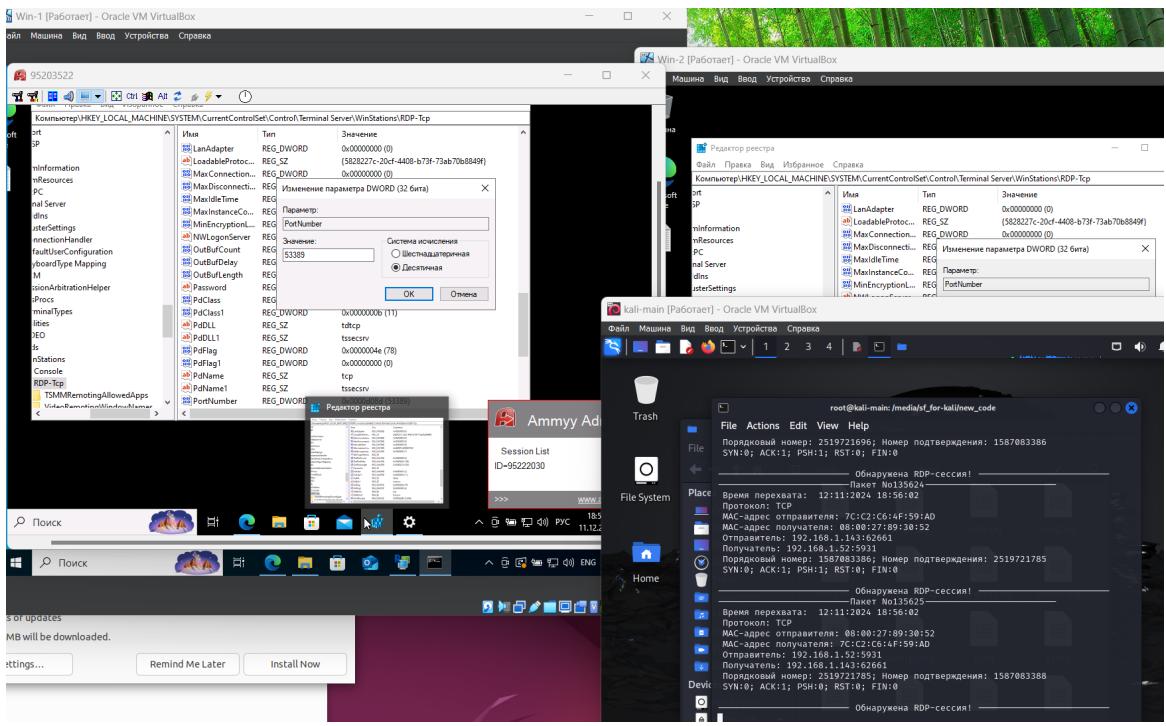


Рисунок 18 – Состояние программы при RDP-трафике (использование Ammyy Admin)

Аналогичное поведение наблюдается при использовании Remmina, бесплатной программы для удаленного доступа, ориентированной на пользователей операционной системы Linux. На рисунке 19 видно, как было установлено подключение между Ubuntu (клиента) и Win (сервера).

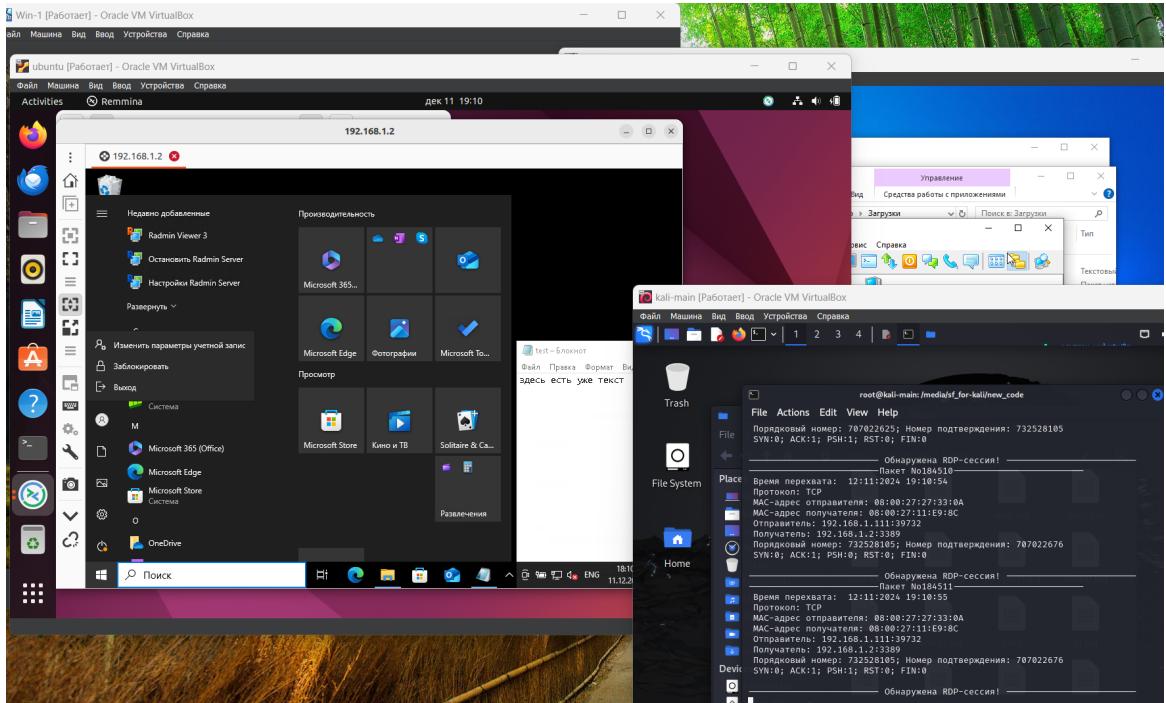


Рисунок 19 – Состояние программы при RDP-трафике (использование Remmina)

RAdmin – это программа удалённого администрирования для платформ

Windows, разработанная компанией Famatech. Она является одной из альтернатив подключения к удаленному рабочему столу. Исходя из следующего рисунка, подключение с помощью RAdmin также распознается программой.

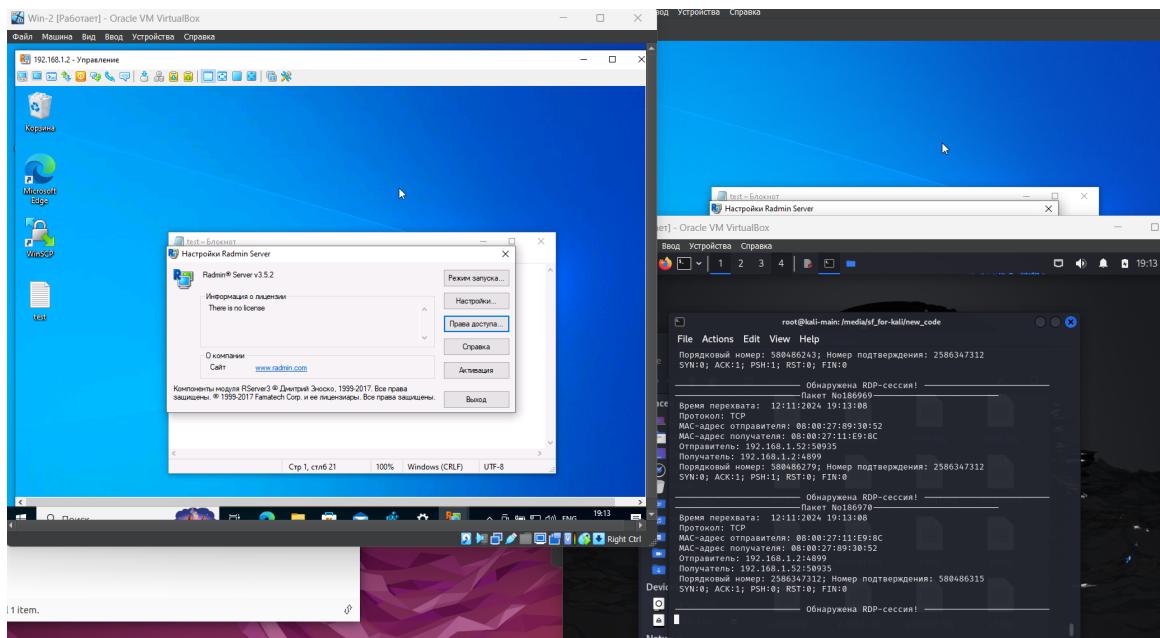


Рисунок 20 – Состояние программы использований RAdmin

Стоит отметить, что первые попытки работы с такими сервисами, как Ammyy Admin, Remmina и RAdmin, приводили к идентификации их трафика как RDP. Таким образом, разработанная программа способна распознавать трафик альтернативных приложений как RDP благодаря их схожести в поведении. Это указывает на универсальность подхода, позволяющего идентифицировать не только известные, но и неизвестные сервисы с аналогичной структурой передачи данных. Такой результат особенно ценен для задач мониторинга и анализа сетевого трафика, где важна способность адаптироваться к изменениям и новым угрозам.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы была разработана программа для автоматического выявления RDP-сессий в сетевом трафике с использованием нейронных сетей, в частности модели LSTM. Данная работа позволила углубленно изучить особенности протокола RDP и выявить его характерные признаки в сетевом трафике. На основе этих знаний были разработаны метрики для анализа и классификации RDP-трафика, что позволило эффективно использовать возможности модели LSTM для решения задачи классификации.

С помощью данной модели программа получила способность идентифицировать трафик, схожий с RDP, что позволяет применять её для обнаружения альтернативных приложений удалённого доступа. Программа работает в режиме реального времени, что обеспечивает её практическую применимость для задач мониторинга корпоративных и частных сетей.

Полученные результаты подчеркивают перспективы использования нейронных сетей для задач сетевой безопасности. Разработанное решение может быть полезным инструментом для специалистов в области информационной безопасности и системных администраторов, обеспечивая эффективное выявление RDP-трафика и предотвращение возможных атак. Дальнейшее развитие программы может включать адаптацию модели для работы с новыми протоколами и приложениями, а также улучшение её производительности и точности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Карабанская, Е. В. Метод выявления аномалий сетевого трафика, основанный на его самоподобной структуре / Е.В. Карабанская, Н.И. Соседова // Безопасность информационных технологий. - 2019. - Т. 26. - № 1. - С. 98-110.
- 2 Фаткиева, Р. Р. Разработка метрик для обнаружения атак на основе анализа сетевого трафика / Р. Р. Фаткиева // Вестник Бурятского государственного университета. Математика, информатика. - 2013. - № 9. - С. 81-86.
- 3 Liang, H. Understanding the Remote Desktop Protocol (RDP) / H. Liang, L. Chen // [Электронный ресурс] : описание протокола удаленного рабочего стола RDP. - URL: <https://learn.microsoft.com/en-us/troubleshoot/windows-server/remote/understanding-remote-desktop-protocol> (дата обращения: 30.11.2024). - Загл. с экрана. - Яз. англ.
- 4 Кручинин, С. В. Стеки сетевых технологий TCP/IP и OSI/ISO / С. В. Кручинин // Вопросы науки. - 2015. - Т. 3. - С. 145-147.
- 5 Maintenance script Модель OSI [Электронный ресурс] : описание модели OSI. - Университет ИТМО. - URL: http://neerc.ifmo.ru/wiki/index.php?title=OSI_Model (дата обращения: 30.11.2024). - Загл. с экрана. - Яз. рус.
- 6 Liang, H. Описание функций Windows TCP / H. Liang, L. Chen, A. Li // [Электронный ресурс] : описание параметров TCP-заголовка . - URL: <https://learn.microsoft.com/ru-ru/troubleshoot/windows-server/networking/description-tcp-features> (дата обращения: 01.12.2024). - Загл. с экрана. - Яз. рус.
- 7 KeyCDN TCP flags [Электронный ресурс] : описание TCP-флагов. - proinity LLC 2023. - URL: <https://www.keycdn.com/support/tcp-flags\#:~:text=ACK> (дата обращения: 01.12.2024). - Загл. с экрана. - Яз. англ.
- 8 Remote Desktop Protocol: UDP Transport Extension [Электронный ресурс] : описание подключения к удаленному рабочему столу по протоколу UDP. - Microsoft 2024. - https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-rdpeudp/2744a3ee-04fb-407b-a9e3-b3b2ded422b1 (дата обращения: 07.12.2024). - Загл. с экрана. - Яз. англ.
- 9 Киреева, Н. В Исследование трафика локальной сети посредством сетевого анализатора «Wireshark» [Электронный ресурс] : ме-

тодическая разработка к лабораторной работе / Н. В. Киреева, В.П. Зайкин, Н. Н. Васин // Поволжский государственный университет телекоммуникаций и информатики, Самара, 2010. - URL: <http://ib.psuti.ru/content/metod/методическиеWireshark.pdf> (дата обращения 08.12.2024). - Загл. с экрана. - Яз. рус.

- 10 Проблемы нейронных сетей [Электронный ресурс] : описание взрывающегося и затухающего градиента. - Университет ИТМО. - URL: https://neerc.ifmo.ru/wiki/index.php?title=Проблемы_нейронных_сетей (дата обращения: 08.12.2024). - Загл. с экрана. - Яз. рус.
- 11 Созыкин, А. В. Обзор методов обучения глубоких нейронных сетей / А. В. Созыкин // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2017. - Т. 6. - № 3. - С. 28-59.
- 12 Ketkar, N. Introduction to keras / N. Ketkar // Deep learning with python: a hands-on introduction. - 2017. - С. 97-111.
- 13 Schmidhuber, J. Long short-term memory / S. Hochreiter, J. Schmidhuber // Neural Comput. - 1997. - Т. 9. - №. 8. - С. 1735-1780.
- 14 Olah, C. Understanding LSTM Networks / C. Olah // Colah's Blog. - 2015. - Сведения доступны также по Интернет: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> (дата обращения: 09.12.2024). - Загл. с экрана. - Яз. англ.
- 15 Jenks, A. Change the listening port for Remote Desktop on your computer / A. Jenks, S. Manheim, H. Lohr // [Электронный ресурс] : изменение порта прослушивания для RDP. - Microsoft, 2024. - URL: <https://learn.microsoft.com/ru-ru/windows-server/remote/remote-desktop-services/clients/change-listening-port> (дата обращения: 15.12.2024). - Загл. с экрана. - Яз. англ.

ПРИЛОЖЕНИЕ А

Листинг sniffer.py

```
1 import socket, struct, keyboard, os
2 import threading
3 import queue
4 from time import time
5 from common_methods import write_to_file, Packet_list
6 from package_parameters import PacketInf
7 from session_creation import SessionInitialization
8
9 class Sniffer:
10
11     def __init__(self) -> None:
12         self.connection = None
13         self.findRDP = False
14         self.packet_queue = queue.Queue()
15         self.error_load_model = False
16
17     # Получение ethernet-кадра
18     def get_ethernet_frame(self, data):
19         dest_mac, src_mac, proto = struct.unpack('!6s6sH', data[:14])
20         return self.get_mac_addr(dest_mac), self.get_mac_addr(src_mac),
21         socket.htons(proto)
22
23     # Получение MAC-адреса
24     def get_mac_addr(self, mac_bytes):
25         mac_str = ''
26         for el in mac_bytes:
27             mac_str += format(el, '02x').upper() + ':'
28         return mac_str[:len(mac_str) - 1]
29
30     # Получение IPv4-заголовка
31     def get_ipv4_data(self, data):
32         version_header_length = data[0]
33         header_length = (version_header_length & 15) * 4
34         ttl, proto, src, dest = struct.unpack('!8xBB2x4s4s', data[:20])
35         return ttl, proto, self.ipv4_dec(src), self.ipv4_dec(dest),
36         data[header_length:]
37
38     # Получение IP-адреса формата X.X.X.X
39     def ipv4_dec(self, ip_bytes):
40         ip_str = ''
41         for el in ip_bytes:
42             ip_str += str(el) + '.'
43         return ip_str[:-1]
44
45     # Получение UDP-сегмента данных
```

```

44     def get_udp_segment(self, data):
45         src_port, dest_port, size = struct.unpack('!HH2xH', data[:8])
46         return str(src_port), str(dest_port), size, data[8:]
47
48     # Получение TCP-сегмента данных
49     def get_tcp_segment(self, data):
50         src_port, dest_port, sequence, ack, \
51         offset_flags, win_size = struct.unpack('!HLLHH', data[:16])
52         offset = (offset_flags >> 12) * 4
53         return str(src_port), str(dest_port), str(sequence), \
54                 str(ack), offset_flags, win_size, data[offset:]
55
56     # Форматирование данных для корректного представления
57     def format_data(self, data):
58         if isinstance(data, bytes):
59             data = ''.join(r'\x{:02x}'.format(el) for el in data)
60         return data
61
62     # Перехват трафика и вывод информации в консоль
63     def start_to_listen(self):
64
65         def packet_processing():
66             while True:
67                 pkt = self.packet_queue.get()
68                 if pkt is None:
69                     break # Завершаем обработку
70                 fl = si.find_session_location(pkt)
71                 si.print_packet_information(pkt, fl)
72                 self.packet_queue.task_done()
73
74                 global Packet_list
75                 NumPacket = 1
76                 curcnt = 1000
77                 pinf = ['] * 19
78                 si = SessionInitialization(self.findRDP, False)
79                 if self.findRDP:
80                     if not si.load_LSTM_model():
81                         self.error_load_model = True
82                         return
83
84                 # Запускаем поток для обработки пакетов
85                 processing_thread = threading.Thread(target=packet_processing)
86                 processing_thread.start()
87
88                 while True:
89                     raw_data, _ = self.connection.recvfrom(65565)
90                     pinf[0], pinf[1] = NumPacket, time()
91                     pinf[2] = len(raw_data)

```

```

92         if si.curTime is None:
93             si.add_start_time(pinf[1])
94
95         pinf[4], pinf[3], protocol = self.get_ethernet_frame(
96         raw_data)
97         if protocol == 8:
98             _, proto, pinf[6], pinf[7], data_ipv4 = self.
99             get_ipv4_data(raw_data[14:])
100
101             if proto == 17: # UDP
102                 NumPacket += 1
103                 pinf[5] = 'UDP',
104                 pinf[8], pinf[9], _, data_udp = self.
105                 get_udp_segment(data_ipv4)
106                 pinf[10] = len(data_udp)
107                 Packet_list.append(PacketInf(pinf))
108                 self.packet_queue.put(Packet_list[-1])
109
110             if proto == 6: # TCP
111                 NumPacket += 1
112                 pinf[5] = 'TCP',
113                 pinf[8], pinf[9], pinf[11], \
114                 pinf[12], flags, pinf[18], data_tcp = self.
115                 get_tcp_segment(data_ipv4)
116                 pinf[10] = len(data_tcp)
117                 pinf[13] = str((flags & 16) >> 4)
118                 pinf[14] = str((flags & 8) >> 3)
119                 pinf[15] = str((flags & 4) >> 2)
120                 pinf[16] = str((flags & 2) >> 1)
121                 pinf[17] = str(flags & 1)
122                 Packet_list.append(PacketInf(pinf))
123                 self.packet_queue.put(Packet_list[-1])
124
125             if keyboard.is_pressed('space'):
126                 self.connection.close()
127                 self.packet_queue.put(None)
128                 processing_thread.join()
129                 si.packet_preparation()
130                 print ('\nЗавершение программы...\n')
131                 break
132
133             # Определение параметров перехвата трафика
134             def traffic_interception(self):
135                 try:
136                     print('Поставить фильтр RDP? (Если да, то введите 1)')
137                     fl = input('Ответ: ')
138                     if fl == '1':
139                         self.findRDP = True

```

```

136         print ('\nВыберите сетевой интерфейс, нажав соответствующую
цифру: ')
137         print (socket.if_nameindex())
138         interface = int(input())
139         if 0 > interface or interface > len(socket.if_nameindex()):
140             print ('\nОшибка ввода!!!\n')
141             return
142         os.system(f'ip link set {socket.if_indextoname(interface)}'
143         promisc on')
144         self.connection = socket.socket(socket.AF_PACKET, socket.
145         SOCK_RAW, socket.ntohs(3))
146         except PermissionError:
147             print ('\nНедостаточно прав!')
148             print ('Запустите программу от имени администратора!')
149             return
150         else:
151             print ('\nНачался процесс захвата трафика...\n')
152             self.start_to_listen()
153             if not self.error_load_model:
154                 print(f'\nДанные собраны. Перехвачено: {len(Packet_list)} п
акетов (-a)\n')
155                 write_to_file()

```

ПРИЛОЖЕНИЕ Б

Листинг common_methods.py

```
1 from session_creation import SessionInitialization
2 from package_parameters import PacketInf
3
4 # Список перехваченных пакетов
5 Packet_list = []
6
7 # Запись информации о пакетах в файл
8 def write_to_file():
9     if not Packet_list:
10         print('Нет данных для записи в файл!')
11         return
12
13     try:
14         FileName = input('Введите название файла (например: data.log):')
15         with open(FileName, 'w') as f:
16             for obj in Packet_list:
17                 base_info = (
18                     f"No:{obj.numPacket};Time:{obj.timePacket};Pac-size"
19                     f":{obj.packetSize};"
20                     f"MAC-src:{obj.mac_src};MAC-dest:{obj.mac_dest};"
21                     f"Type:{obj.protoType};"
22                     f"IP-src:{obj.ip_src};IP-dest:{obj.ip_dest};Port-"
23                     f"src:{obj.port_src};"
24                     f"Port-dest:{obj.port_dest};Len-data:{obj.len_data}"
25                     f");"
26
27                 if obj.protoType == 'TCP':
28                     tcp_info = (
29                         f"Seq:{obj.seq};Ack:{obj.ack};Fl-ack:{obj."
30                         f"fl_ack};"
31                         f"Fl-psh:{obj.fl_psh};Fl-rst:{obj.fl_rst};Fl-"
32                         f"syn:{obj.fl_syn};"
33                         f"Fl-fin:{obj.fl_fin};Win-size:{obj.win_size};"
34                     )
35                     f.write(base_info + tcp_info + " !\n")
36                 else:
37                     f.write(base_info + " !\n")
38             print(f'\nИнформация успешно записана в файл {FileName}.\n')
39
40     except Exception as e:
41         print(f'\nОшибка записи в файл {FileName}: {e}\n')
42
43 # Обработка строки с данными
44 def row_processing(inf):
```

```

38     global Packet_list
39     data = [field.split(';', 1)[1] for field in inf.split(';') if ';' 
40             in field]
41     if data:
42         Packet_list.append(PacketInf(data))
43
44 # Проверка, является ли сессия новой
45 def is_new_session(pkt, iplist):
46     return (pkt.ip_src, pkt.ip_dest) not in iplist and (pkt.ip_dest, 
47     pkt.ip_src) not in iplist
48
49 # Считывание с файла и заполнение массива Packet_list
50 def read_from_file():
51     try:
52         fileName = input('Введите название файла (например: data.log): ')
53         if Packet_list:
54             print('Список Packet_list уже содержит данные.')
55             return
56         si = SessionInitialization(True, False)
57         si.load_LSTM_model()
58         iplist = set()
59
60         with open(fileName, 'r') as f:
61             for inf in f:
62                 if not inf.strip():
63                     continue
64                 row_processing(inf)
65                 packet = Packet_list[-1]
66
67                 if is_new_session(packet, iplist):
68                     iplist.add((packet.ip_src, packet.ip_dest))
69
70                 if si.curTime is None:
71                     si.add_start_time(packet.timePacket)
72
73                 si.find_session_location(packet)
74
75                 si.packet_preparation()
76                 print(f'\nДанные успешно считаны из файла {fileName}.\n')
77             except Exception as e:
78                 print(f'\nОшибка обработки файла {fileName} или загрузки модели 
: {e}\n')

```

ПРИЛОЖЕНИЕ В

Листинг package_parameters.py

```
1 from math import sqrt
2
3 # Класс, содержащий информацию о каком-либо пакете
4 class PacketInf:
5
6     def __init__(self, lst):
7         self.numPacket = int(lst[0])
8         self.timePacket = float(lst[1])
9         self.packetSize = int(lst[2])
10        self.mac_src, self.mac_dest, self.protoType = lst[3], lst[4],
11        lst[5]
12        self.ip_src, self.ip_dest = lst[6], lst[7]
13        self.port_src, self.port_dest, self.len_data = int(lst[8]), int
14        (lst[9]), int(lst[10])
15
16        if self.protoType == 'TCP':
17            self.seq, self.ack = lst[11], lst[12]
18            self.fl_ack, self.fl_psh = lst[13], lst[14]
19            self.fl_rst, self.fl_syn = lst[15], lst[16]
20            self.fl_fin, self.win_size = lst[17], 0
21            if len(lst) > 18:
22                self.win_size = int(lst[18])
23
24 # Класс, содержащий информацию относительно какого-либо IP-адреса
25 class ExploreObject:
26
27     def __init__(self, ip):
28         self.ip = ip
29         self strt_time, self.fin_time = None, None
30         self.amnt_packet, self.avg_packet_num = None, None
31         self.avg_packet_size = None
32
33         self.commonPorts = None
34         self.in_out_rel_data, self.ack_flags_diff_data = None, None
35         self.avg_time_intervals, self.dev_time_intervals = None, None
36         self.psh_flags_freq_data, self.ack_flags_freq_data = None, None
37         self(pkt_amnt_src_data, self.pkt_amnt_dst_data = None, None
38         self.adjcIPList, self.adjcPacketList, self.avg_winsize_dest = None,
39         None, None
```

37

ПРИЛОЖЕНИЕ Г

Листинг traffic_analysis.py

```
1 import time
2 from common_methods import Packet_list
3 from session_creation import SessionInitialization, Session_list
4 from package_parameters import ExploreObject
5 from chart_creation import ChartCreation, Object_list
6
7
8 class TrafficAnalysis:
9
10     def __init__(self) -> None:
11         self.IPList = None
12         self.numPacketsPerSec = None
13         self.labels_list = []
14         self.obj_list = []
15
16     # Получение общей информации о текущей
17     # попытке перехвата трафика
18     def get_common_data(self):
19         self.IPList = set()
20         self.numPacketsPerSec = []
21         curTime = Packet_list[0].timePacket + 1
22         fin = Packet_list[-1].timePacket + 1
23         self.labels_list.append(time.strftime('%H:%M:%S',
24                                         time.localtime(Packet_list[0].
25                                         timePacket)))
26         cntPacket = 0
27         i = 0
28         while curTime < fin:
29             for k in range(i, len(Packet_list)):
30                 if Packet_list[k].timePacket > curTime:
31                     self.numPacketsPerSec.append(cntPacket)
32                     self.labels_list.append(time.strftime('%H:%M:%S',
33                                         time.localtime(curTime)))
34                     cntPacket = 0
35                     i = k
36                     break
37             cntPacket += 1
38             curTime += 1
39         self.numPacketsPerSec.append(cntPacket)
40         for p in Packet_list:
41             self.IPList.add(p.ip_src)
42             self.IPList.add(p.ip_dest)
43         self.IPList = sorted(list(self.IPList),
44                             key=lambda ip: list(map(int, ip.split(
45                             ',')))) )
```

```

43
44 # Получение общих портов относительно текущего IP-адреса
45 def get_common_ports(self, curIP):
46     ports = set()
47     for pkt in Packet_list:
48         if pkt.ip_src == curIP or pkt.ip_dest == curIP:
49             ports.add(pkt.port_src)
50             ports.add(pkt.port_dest)
51     return sorted(list(ports))
52
53 # Вывод пар (число, IP-адрес/порт) для
54 # предоставления выбора IP-адреса/порта
55 # пользователю
56 def print_list_of_pairs(self, IPLList, fl=False):
57     num = 0
58     cnt = 1
59     if fl:
60         print('[' + str(num), '---', 'None', end='] ')
61         cnt += 1
62         num += 1
63     for el in IPLList:
64         if cnt > 3:
65             cnt = 0
66             print('[' + str(num), '---', el, end=']\n')
67         else:
68             print('[' + str(num), '---', el, end='] ')
69             cnt += 1
70             num += 1
71     print('')
72
73 # Обработка общих данных
74 def start_to_analyse(self):
75     if Packet_list == []:
76         print('\nНет данных! Сначала необходимо получить данные!\n')
77     return
78     self.get_common_data()
79     si = SessionInitialization(False, False)
80     print(f'Sessions len = {len(Session_list)}:')
81     si.clear_unwanted_sessions()
82     print(f'After Sessions len = {len(Session_list)}:')
83     si.print_inf_about_sessions()
84     strt = Packet_list[0].timePacket
85     fin = Packet_list[-1].timePacket
86     strt_time = time.localtime(strt)
87     fin_time = time.localtime(fin)
88     avgNumPacket = 0
89     for el in self.numPacketsPerSec:

```

```

90         avgNumPacket += el
91     avgNumPacket /= len(self.numPacketsPerSec)
92     avgSizePacket = 0
93     for p in Packet_list:
94         avgSizePacket += p.packetSize
95     avgSizePacket /= len(Packet_list)
96     print('Общая информация:')
97     print('Время первого перехваченного пакета: '
98           , time.strftime('%d.%m.%Y г. %H:%M:%S', strt_time) )
99     print('Время последнего перехваченного пакета: '
100        , time.strftime('%d.%m.%Y г. %H:%M:%S', fin_time) )
101    print('Количество пакетов: ', len(Packet_list))
102    print('Общее время перехвата: ', round(fin - strt, 3), 'сек')
103    print('Среднее количество пакетов в секунду: ', round(
104        avgNumPacket, 3))
105    print('Средний размер пакетов: ', round(avgSizePacket, 3))
106    print('Завершить просмотр (нажмите "q" для выхода)')
107    for k in range(len(self.IPList)):
108        Object_list.append(ExploreObject(self.IPList[k]))
109        Object_list[-1].commonPorts = self.get_common_ports(self.
110 IPList[k])
111        self.print_list_of_pairs(self.IPList)
112        print(f'\nВыберите цифру (0 - {len(self.IPList)} - 1) для просм
отра IP-адреса: ')
113        k = input()
114        if k == 'q':
115            return
116        try:
117            k = int(k)
118        except:
119            print('\nНекорректный ввод!\n')
120            return
121        else:
122            if 0 <= k and k < len(self.IPList):
123                port = None
124                print('Список портов которые участвовали в соединении
с данным IP-адресом')
125                self.print_list_of_pairs(Object_list[k].commonPorts,
126 True)
127                t = len(Object_list[k].commonPorts)
128                print(f'\nВыберите цифру (0 - {t}) для выбора порта: ')
129                k1 = input()
130                if k1 == 'q':
131                    return
132                try:
133                    k1 = int(k1)
134                except:
135                    print('Некорректный ввод!\n')

```

```
133         return
134     else:
135         if 0 <= k1 and k1 <= t:
136             if k1 != 0:
137                 port = Object_list[k].commonPorts[k1 - 1]
138                 ChartCreation(k, strt, fin, port, self.
139 labels_list).start_to_plot()
140             else:
141                 print(f'Введите число в пределах 0 - {t - 1}')
142         else:
143             print(f'Введите число в пределах 0 - {len(self.IPList)
- 1}')
```

ПРИЛОЖЕНИЕ Д

Листинг chart_creation.py

```
1 import time
2 import matplotlib.pyplot as plt
3 import matplotlib.gridspec as gridspec
4 from colorama import Back, Fore
5 from common_methods import Packet_list
6 from math import sqrt
7
8 # Список исследуемых объектов (IP-порт)
9 Object_list = []
10
11 class ChartCreation():
12
13     def __init__(self, k, strt, fin, port, lbls_lst) -> None:
14         self.k = k
15         self.strt_time = strt
16         self.fin_time = fin
17         self.curPort = port
18         self.step = None
19         self.curIP = None
20         self.labels_list = lbls_lst
21         self.x_axisLabels = []
22
23     # Получение меток и "шага" для оси абсцисс
24     def get_x_labels(self):
25         total_time = int(self.fin_time - self.strt_time)
26         self.step = 1
27         if total_time > 600:
28             self.step = 30
29         elif total_time > 300:
30             self.step = 10
31         elif total_time > 50:
32             self.step = 5
33         self.x_axisLabels.clear()
34         for i in range(0, len(self.labels_list), self.step):
35             self.x_axisLabels.append(self.labels_list[i])
36
37     # Получение общей информации о трафике,
38     # связанном с выбранным IP-адресом
39     def get_inf_about_IP(self):
40         adjcPacketList = []
41         adjcIPList = set()
42         if self.curPort != None:
43             for p in Packet_list:
44                 if p.port_src == self.curPort or p.port_dest == self.
curPort:
```

```

45             if p.ip_src == self.curIP:
46                 adjcPacketList.append(p)
47                 adjcIPList.add(p.ip_dest)
48             if p.ip_dest == self.curIP:
49                 adjcPacketList.append(p)
50                 adjcIPList.add(p.ip_src)
51         else:
52             for p in Packet_list:
53                 if p.ip_src == self.curIP:
54                     adjcPacketList.append(p)
55                     adjcIPList.add(p.ip_dest)
56                 if p.ip_dest == self.curIP:
57                     adjcPacketList.append(p)
58                     adjcIPList.add(p.ip_src)
59     return adjcPacketList, list(adjcIPList)
60
61 # Вывод пакетов, связанных с выбранным IP-адресом
62 def print_adjacent_packets(self):
63     adjcPacketList = Object_list[self.k].adjcPacketList
64
65     def format_packet(packet):
66         t = time.strftime('%H:%M:%S', time.localtime(packet.
timePacket))
67         base_info = (
68             f'Номер пакета: {packet.numPacket}; Время: {t}; ,
69             f'Размер: {packet.packetSize}; MAC-адрес отправителя: {packet.mac_src}; '
70             f'MAC-адрес получателя: {packet.mac_dest}; ,
71             f'Отправитель: {packet.ip_src}:{packet.port_src}; ,
72             f'Получатель: {packet.ip_dest}:{packet.port_dest}; ,
73             f'Протокол: {packet.protoType}; Размер поля данных: {packet.len_data}; '
74         )
75         return base_info
76
77     def format_tcp_details(packet):
78         return (
79             f' Порядковый номер: {packet.seq}; Номер подтверждения: {packet.ack}; ,
80             f'SYN: {packet.fl_syn}; ACK: {packet.fl_ack}; PSH: {packet.fl_psh}; ,
81             f'RST: {packet.fl_RST}; FIN: {packet.fl_fin}; '
82         )
83
84     for idx, packet in enumerate(adjcPacketList):
85         base_info = format_packet(packet)
86         style = Back.CYAN + Fore.BLACK if idx % 2 == 0 else ""
87         print(style + base_info, end=' ')

```

```

88
89     if packet.protoType == 'TCP':
90         tcp_details = format_tcp_details(packet)
91         print(style + tcp_details)
92     else:
93         print(' ')
94
95 # Выход пар (число, IP-адрес/порт) для
96 # предоставления выбора IP-адреса/порта
97 # пользователю
98 def print_list_of_pairs(self, IPList, fl=False):
99     num = 0
100    cnt = 1
101    if fl:
102        print('[' + str(num), '---', 'None', end='] ')
103        cnt += 1
104        num += 1
105    for el in IPList:
106        if cnt > 3:
107            cnt = 0
108            print('[' + str(num), '---', el, end=']\n')
109        else:
110            print('[' + str(num), '---', el, end='] ')
111        cnt += 1
112        num += 1
113    print(' ')
114
115 # Получение второго IP-адреса
116 def get_2nd_IP_for_plot(self):
117     print('\nИзобразить на графике еще один объект. Выберите ' + \
118             'IP-адрес для добавления (введите цифру)')
119     self.print_list_of_pairs(Object_list[self.k].adjcIPList, True)
120     scndIP = 'None'
121     try:
122         pos = int(input())
123     except:
124         print('Некорректный ввод!')
125         return -1
126     else:
127         if pos < 0 or pos > len(Object_list[self.k].adjcIPList):
128             print('Некорректный ввод!')
129             return -1
130         if pos != 0:
131             scndIP = Object_list[self.k].adjcIPList[pos - 1]
132     return scndIP
133
134 # Получение номера по IP-адресу
135 def get_pos_by_IP(self, exploreIP):

```

```

136     for i in range(len(Object_list)):
137         if Object_list[i].ip == exploreIP:
138             return i
139     return -1
140
141 # Получение данных об отношении входящего
142 # трафика к исходящему в единицу времени
143 def get_in_out_rel(self, exploreIP):
144     cntInput = 0
145     cntOutput = 0
146     rel_list = []
147     curTime = self.strt_time + 1
148     finTime = self.fin_time + 1
149     pos = 0
150     while curTime < finTime:
151         for k in range(pos, len(Packet_list)):
152             if Packet_list[k].timePacket > curTime:
153                 if cntOutput != 0:
154                     rel_list.append(cntInput / cntOutput)
155                 else:
156                     rel_list.append(0.0)
157                 cntInput = 0
158                 cntOutput = 0
159                 pos = k
160                 break
161             if self.curPort == None:
162                 if Packet_list[k].ip_src == exploreIP:
163                     cntOutput += 1
164                 if Packet_list[k].ip_dest == exploreIP:
165                     cntInput += 1
166                 else:
167                     if Packet_list[k].port_src == self.curPort or
Packet_list[k].port_dest == self.curPort:
168                         if Packet_list[k].ip_src == exploreIP:
169                             cntOutput += 1
170                         if Packet_list[k].ip_dest == exploreIP:
171                             cntInput += 1
172                         curTime += 1
173             if cntOutput != 0:
174                 rel_list.append(cntInput / cntOutput)
175             else:
176                 rel_list.append(0.0)
177     return rel_list
178
179 # Универсальный метод для получения частоты флагов
180 def get_flags_freq(self, exploreIP, flag_type):
181     cntFlagTCP = 0
182     cntTCP = 0

```

```

183     rel_list = []
184     curTime = self.strt_time + 1
185     finTime = self.fin_time + 1
186     pos = 0
187     while curTime < finTime:
188         for k in range(pos, len(Packet_list)):
189             if Packet_list[k].timePacket > curTime:
190                 if cntTCP != 0:
191                     rel_list.append(cntFlagTCP / cntTCP)
192                 else:
193                     rel_list.append(0.0)
194                 cntFlagTCP = 0
195                 cntTCP = 0
196                 pos = k
197                 break
198             if self.curPort is None:
199                 if Packet_list[k].ip_dest == exploreIP and
Packet_list[k].protoType == 'TCP':
200                     cntTCP += 1
201                     if getattr(Packet_list[k], flag_type) == '1':
202                         cntFlagTCP += 1
203                     else:
204                         if Packet_list[k].port_src == self.curPort or
Packet_list[k].port_dest == self.curPort:
205                             if Packet_list[k].ip_dest == exploreIP and
Packet_list[k].protoType == 'TCP':
206                                 cntTCP += 1
207                                 if getattr(Packet_list[k], flag_type) == '1
':
208                                     cntFlagTCP += 1
209                                     curTime += 1
210                                     if cntTCP != 0:
211                                         rel_list.append(cntFlagTCP / cntTCP)
212                                     else:
213                                         rel_list.append(0.0)
214     return rel_list
215
216 # Получение данных о количестве пакетов и
217 # о максимумах пакетов в единицу времени
218 def get_pktnamnt_and_size_persec(self, exploreIP):
219     pktAmntSrcList = []
220     pktAmntDstList = []
221     pktSizeSrcList = []
222     pktSizeDstList = []
223     curTime = self.strt_time + 1
224     finTime = self.fin_time + 1
225     pos = 0
226     while curTime < finTime:

```

```

227         cntpktsrc = 0
228         cntpktdest = 0
229         maxpktsizesrc = 0
230         maxpktsizedst = 0
231         for k in range(pos, len(Packet_list)):
232             if Packet_list[k].timePacket > curTime:
233                 pktAmntSrcList.append(cntpktsrc)
234                 pktAmntDstList.append(cntpktdest)
235                 pktSizeSrcList.append(maxpktsizesrc)
236                 pktSizeDstList.append(maxpktsizedst)
237                 pos = k
238                 break
239             if self.curPort == None:
240                 if Packet_list[k].ip_src == exploreIP:
241                     cntpktsrc += 1
242                     if maxpktsizesrc < Packet_list[k].packetSize:
243                         maxpktsizesrc = Packet_list[k].packetSize
244                 if Packet_list[k].ip_dest == exploreIP:
245                     cntpktdest += 1
246                     if maxpktsizedst < Packet_list[k].packetSize:
247                         maxpktsizedst = Packet_list[k].packetSize
248                 else:
249                     if Packet_list[k].port_src == self.curPort or
Packet_list[k].port_dest == self.curPort:
250                         if Packet_list[k].ip_src == exploreIP:
251                             cntpktsrc += 1
252                             if maxpktsizesrc < Packet_list[k].
packetSize:
253                                 maxpktsizesrc = Packet_list[k].
packetSize
254                                 if Packet_list[k].ip_dest == exploreIP:
255                                     cntpktdest += 1
256                                     if maxpktsizedst < Packet_list[k].
packetSize:
257                                         maxpktsizedst = Packet_list[k].
packetSize
258                                         curTime += 1
259                                         pktAmntSrcList.append(cntpktsrc)
260                                         pktAmntDstList.append(cntpktdest)
261                                         pktSizeSrcList.append(maxpktsizesrc)
262                                         pktSizeDstList.append(maxpktsizedst)
263                                         return pktAmntSrcList, pktAmntDstList, pktSizeSrcList,
pktSizeDstList
264
265     # Вычисление среднего размера окна
266     def get_avg_window_size(self, exploreIP):
267         avgWindowSizeDest = []
268         sumDest = 0

```

```

269     cntDest = 0
270     curTime = self.strt_time + 1
271     finTime = self.fin_time + 1
272     pos = 0
273     while curTime < finTime:
274         for k in range(pos, len(Packet_list)):
275             if Packet_list[k].protoType == "UDP":
276                 continue
277             if Packet_list[k].timePacket > curTime:
278                 if cntDest != 0:
279                     avgWindowSizeDest.append(sumDest / cntDest)
280                 else:
281                     avgWindowSizeDest.append(0)
282                     sumDest = 0
283                     cntDest = 0
284                     pos = k
285                     break
286             if self.curPort == None:
287                 if Packet_list[k].ip_dest == exploreIP:
288                     sumDest += Packet_list[k].win_size
289                     cntDest += 1
290             else:
291                 if Packet_list[k].port_src == self.curPort or
292                     Packet_list[k].port_dest == self.curPort:
293                     if Packet_list[k].ip_src == exploreIP or
294                         Packet_list[k].ip_dest == exploreIP:
295                             sumDest += Packet_list[k].win_size
296                             cntDest += 1
297                             curTime += 1
298             if cntDest != 0:
299                 avgWindowSizeDest.append(sumDest / cntDest)
300             else:
301                 avgWindowSizeDest.append(0)
302     return avgWindowSizeDest
303
304     # Выбор опций для выбранного IP-адреса
305     def start_to_plot(self):
306         self.get_x_labels()
307         self.curIP = Object_list[self.k].ip
308         Object_list[self.k].adjcPacketList, Object_list[self.k].
309         adjcIPList = self.get_inf_about_IP()
310         Object_list[self.k].strt_time = time.localtime(Object_list[self.
311             k].adjcPacketList[0].timePacket)
312         Object_list[self.k].fin_time = time.localtime(Object_list[self.
313             k].adjcPacketList[-1].timePacket)
314         Object_list[self.k].amnt_packet = len(Object_list[self.k].
315             adjcPacketList)
316         totalTime = round( Object_list[self.k].adjcPacketList[-1].

```

```

timePacket - \
Object_list[self.k].adjcPacketList[0] .
311
timePacket )
312     if totalTime == 0:
313         totalTime = 1
314     Object_list[self.k].avg_packet_num = round(Object_list[self.k].
amnt_packet / totalTime , 3)
315     avgSize = 0
316     for p in Object_list[self.k].adjcPacketList:
317         avgSize += p.len_data
318     Object_list[self.k].avg_packet_size = round(avgSize /
Object_list[self.k].amnt_packet , 3)
319     while True:
320         print(f'Общая информация о трафике, связанном с {self.curIP
}')
321         print( 'Время первого перехваченного пакета: ' ,
time.strftime('%d.%m.%Y г. %H:%M:%S' , Object_list[
self.k].strt_time) )
322         print( 'Время последнего перехваченного пакета: ' ,
time.strftime('%d.%m.%Y г. %H:%M:%S' , Object_list[
self.k].fin_time) )
323         print('Общее время: ' , totalTime , 'сек.')
324         print('Количество пакетов: ' , Object_list[self.k].
amnt_packet)
325         print('Среднее количество пакетов в секунду: ' ,
Object_list
[self.k].avg_packet_num)
326         print('Средний размер пакетов: ' , Object_list[self.k].
avg_packet_size)
327         print(f"""Выберите опцию:
328             1. Вывести весь трафик, связанный с {self.curIP}
329             2. Построить график отношения входящего и исходящего трафик
ов
330             3. Построить график частоты ACK и PSH флагов во входящих па
кетах
331             4. Построить график отображения количества пакетов в единиц
у времени
332             5. Построить график отображения среднего количества значени
я размеров окна
333             6. Вернуться к выбору IP-адреса """
)
334             bl = input()
335             if bl == '1':
336                 self.print_adjacent_packets()
337             elif bl == '2':
338                 Object_list[self.k].in_out_rel_data = self.
get_in_out_rel(self.curIP)
339                 x = [i for i in range(0, len(Object_list[self.k].
in_out_rel_data))]
340                 x_labels = [i for i in range(0, len(x) , self.step)]
341

```

```

343         scndIP = self.get_2nd_IP_for_plot()
344         if scndIP == -1:
345             continue
346         if scndIP != 'None':
347             pos = self.get_pos_by_IP(scndIP)
348             Object_list[pos].in_out_rel_data = self.
349             get_in_out_rel(scndIP)
350             fig = plt.figure(figsize=(16, 6), constrained_layout=
351             True)
352             f = fig.add_subplot()
353             f.grid()
354             f.set_title('Отношение объема входящего к объему исход-
355             ящего трафиков (общий порт {self.curPort})', fontsize=15)
356             f.set_xlabel('Общее время перехвата трафика', fontsize=
357             15)
358             f.set_ylabel(r'$r_{in/out} = \frac{V_{in}}{V_{out}}$', font-
359             size=15)
360             plt.plot(x, Object_list[self.k].in_out_rel_data, label=
361             self.curIP)
362             if scndIP != 'None':
363                 plt.plot(x, Object_list[pos].in_out_rel_data, label=
364                 scndIP)
365             plt.xticks(x_labels, self.x_axisLabels, rotation=30,
366             fontsize=10)
367             f.legend()
368             plt.show()
369             elif bl == '3':
370                 data = self.get_flags_freq(self.curIP, 'fl_ack')
371                 Object_list[self.k].ack_flags_freq_data = data
372                 data = self.get_flags_freq(self.curIP, 'fl_psh')
373                 Object_list[self.k].psh_flags_freq_data = data
374                 x = [i for i in range(0, len(Object_list[self.k].
375                 ack_flags_freq_data))]
376                 x_labels = [i for i in range(0, len(x), self.step)]
377                 scndIP = self.get_2nd_IP_for_plot()
378                 if scndIP == -1:
379                     continue
380                 if scndIP != 'None':
381                     pos = self.get_pos_by_IP(scndIP)
382                     data = self.get_flags_freq(scndIP, 'fl_ack')
383                     Object_list[pos].ack_flags_freq_data = data
384                     data = self.get_flags_freq(scndIP, 'fl_psh')
385                     Object_list[pos].psh_flags_freq_data = data
386                     fig = plt.figure(figsize=(16, 6), constrained_layout=
387                     True)
388                     gs = gridspec.GridSpec(ncols=1, nrows=2, figure=fig)
389                     fig_1 = fig.add_subplot(gs[0, 0])
390                     fig_1.grid()

```

```

381             fig_1.set_title('Частота флагов ACK (общий порт {self.
382 curPort})', fontsize=15 )
383             fig_1.set_xlabel('Общее время перехвата трафика',
384 fontsize=15)
385             fig_1.set_ylabel(r'$r_{ack} = \frac{V_{S_{in}}}{V_{tcp}}$',
386 fontsize=15)
387             plt.plot(x, Object_list[self.k].ack_flags_freq_data, 'b
388 ', label=self.curIP)
389             if scndIP != 'None':
390                 plt.plot(x, Object_list[pos].ack_flags_freq_data, 'r
391 ', label=scndIP)
392             plt.xticks(x_labels, self.x_axisLabels, rotation=30,
393 fontsize=8)
394             fig_1.legend()
395             fig_2 = fig.add_subplot(gs[1, 0])
396             fig_2.grid()
397             plt.plot(x, Object_list[self.k].psh_flags_freq_data, 'orange
398 ', label=self.curIP)
399             fig_2.set_title('Частота флагов PSH (общий порт {self.
400 curPort})', fontsize=15 )
401             fig_2.set_xlabel('Общее время перехвата трафика',
402 fontsize=15)
403             fig_2.set_ylabel(r'$r_{psh} = \frac{V_{P_{in}}}{V_{tcp}}$',
404 fontsize=15)
405             if scndIP != 'None':
406                 plt.plot(x, Object_list[pos].psh_flags_freq_data, 'g
407 ', label=scndIP)
408             plt.xticks(x_labels, self.x_axisLabels, rotation=30,
409 fontsize=8)
410             fig_2.legend()
411             plt.show()
412         elif bl == '4':
413             d1, d2, d3, d4 = self.get_pktnum_and_size_persec(self.
414 curIP)
415             Object_list[self.k].pkt_amnt_src_data = d1
416             Object_list[self.k].pkt_amnt_dst_data = d2
417             Object_list[self.k].pkt_size_data_src = d3
418             Object_list[self.k].pkt_size_data_dst = d4
419             x = [i for i in range(0, len(Object_list[self.k].
420 pkt_amnt_src_data))]
421             x_labels = [i for i in range(0, len(x), self.step)]
422             scndIP = self.get_2nd_IP_for_plot()
423             if scndIP == -1:
424                 continue
425             if scndIP != 'None':
426                 pos = self.get_pos_by_IP(scndIP)
427                 d1, d2, d3, d4 = self.get_pktnum_and_size_persec(
428 scndIP)

```

```

414             Object_list[pos].pkt_amnt_src_data = d1
415             Object_list[pos].pkt_amnt_dst_data = d2
416             Object_list[pos].pkt_size_data_src = d3
417             Object_list[pos].pkt_size_data_dst = d4
418             fig = plt.figure(figsize=(16, 6), constrained_layout=True)
419             gs = gridspec.GridSpec(ncols=1, nrows=2, figure=fig)
420             fig_1 = fig.add_subplot(gs[0, 0])
421             fig_1.grid()
422             fig_1.set_title('Количество входящих пакетов, полученных за единицу времени (общий порт {self.curPort})', fontsize=15)
423             fig_1.set_xlabel('Общее время перехвата трафика', fontsize=15)
424             plt.plot(x, Object_list[self.k].pkt_amnt_dst_data, 'b',
425             label=self.curIP)
426             if scndIP != 'None':
427                 plt.plot(x, Object_list[pos].pkt_amnt_dst_data, 'r',
428                 , label=scndIP)
429             plt.xticks(x_labels, self.x_axisLabels, rotation=30,
430             fontsize=8)
431             fig_1.legend()
432             fig_2 = fig.add_subplot(gs[1, 0])
433             fig_2.grid()
434             plt.plot(x, Object_list[self.k].pkt_amnt_src_data, 'orange',
435             label=self.curIP)
436             fig_2.set_title('Количество исходящих пакетов, полученных за единицу времени (общий порт {self.curPort})', fontsize=15)
437             fig_2.set_xlabel('Общее время перехвата трафика', fontsize=15)
438             if scndIP != 'None':
439                 plt.plot(x, Object_list[pos].pkt_amnt_src_data, 'g',
440                 , label=scndIP)
441             plt.xticks(x_labels, self.x_axisLabels, rotation=30,
442             fontsize=8)
443             fig_2.legend()
444             plt.show()
445             elif bl == '5':
446                 d = self.get_avg_window_size(self.curIP)
447                 Object_list[self.k].avg_winsize_dest = d
448                 x = [i for i in range(0, len(Object_list[self.k].
449                 avg_winsize_dest))]
450                 x_labels = [i for i in range(0, len(x), self.step)]
451                 scndIP = self.get_2nd_IP_for_plot()
452                 if scndIP == -1:
453                     continue
454                 if scndIP != 'None':
455                     pos = self.get_pos_by_IP(scndIP)
456                     d = self.get_avg_window_size(scndIP)

```

```

450             Object_list[pos].avg_winsize_dest = d
451             fig = plt.figure(figsize=(16, 6), constrained_layout=
452                           True)
453                 f = fig.add_subplot()
454                 f.grid()
455                 f.set_title( 'Среднее количество размера окна, полученн
456                 ых за единицу времени (общий порт {self.curPort})', fontsize=15 )
457
458                 f.set_xlabel('Общее время перехвата трафика', fontsize
459                           =15)
460                 plt.plot(x, Object_list[self.k].avg_winsize_dest, label
461                           =self.curIP + '(получатель)')
462                 if scndIP != 'None':
463                     plt.plot(x, Object_list[pos].avg_winsize_dest,
464                               label=scndIP + '(получатель)')
465                 plt.xticks(x_labels, self.x_axisLabels, rotation=30,
466                             fontsize=10)
467                 f.legend()
468                 plt.show()
469             elif bl == '6':
470                 break
471

```

ПРИЛОЖЕНИЕ Е

Листинг session_creation.py

```
1 import time
2 import numpy as np
3 from colorama import init, Back, Fore
4 from math import sqrt
5 from keras.models import load_model
6
7 # Глобальный список сессий
8 Session_list = []
9
10 init(autoreset=True)
11
12 # Класс, содержащий информацию о каждой активной сессии
13 class Session:
14
15     def __init__(self, strt_time, ips, ports) -> None:
16         self.strt_time = strt_time
17         self.ips = ips
18         self.ports = ports
19         self.stateActive = True
20         self.forceFin = False
21         self.isRDP = False
22
23         self.prevTimePkt = None
24         self.lastTimePkt = None
25         self.totalTime = None
26         self.intervalsList = []
27
28         # Для подсчета количества/размера пакетов
29         self.cntPktSrcIP1 = self.cntPktDestIP1 = 0
30         self.pktSizeDestIP1 = []
31         self.pktSizeDestIP2 = []
32
33         # Для подсчета флагов PSH
34         self.cntPSHDestIP1 = self.cntPSHDestIP2 = 0
35
36         # Для подсчета флагов ACK
37         self.cntACKDestIP1 = self.cntACKDestIP2 = self.cntACKSrcIP1
38         = 0
39
40         # Для подсчета всего TCP-трафика
41         self.cntPktTCPDestIP1 = self.cntPktTCPDestIP2 = 0
42
43         # Для подсчета флагов SYN, FIN, RST
44         self.cntSYNSrc = self.cntSYNDest = self.cntFINSrc = 0
        self.cntFINDest = self.cntRSTSsrc = self.cntRSTDest = 0
```

```

45
46         self.cntTr = self.CNT = self.cntPkt = self.cntPktUDP = 0
47         self.winSizeList = []
48         self.rdpProb = []
49
50     # Сбор данных о сессии, извлекаемых из пакетов
51     def update_data(self, pkt):
52         # Вычисление временных интервалов
53         if self.prevTimePkt is None:
54             self.prevTimePkt = pkt.timePacket
55         else:
56             self.intervalsList.append(pkt.timePacket - self.prevTimePkt
57 )
58             self.prevTimePkt = pkt.timePacket
59     # Подсчет параметров входящего и исходящего трафиков для IP1
60     # и размера входящих пакетов для IP1 и IP2
61     if pkt.ip_src == self.ips[0]:
62         self.cntPktSrcIP1 += 1
63         self.pktSizeDestIP2.append(pkt.packetSize)
64     else:
65         self.cntPktDestIP1 += 1
66         self.pktSizeDestIP1.append(pkt.packetSize)
67     # Подсчет флагов PSH и ACK
68     if pkt.protoType == 'TCP':
69         if pkt.ip_dest == self.ips[0]:
70             if pkt.fl_psh == '1':
71                 self.cntPSHDestIP1 += 1
72             if pkt.fl_ack == '1':
73                 self.cntACKDestIP1 += 1
74             if pkt.fl_syn == '1':
75                 self.cntSYNDest += 1
76             if pkt.fl_fin == '1':
77                 self.cntFINDest += 1
78             if pkt.fl_RST == '1':
79                 self.cntRSTDest += 1
80             self.cntPktTCPDestIP1 += 1
81         if pkt.ip_dest == self.ips[1]:
82             if pkt.fl_psh == '1':
83                 self.cntPSHDestIP2 += 1
84             if pkt.fl_ack == '1':
85                 self.cntACKDestIP2 += 1
86             if pkt.fl_syn == '1':
87                 self.cntSYNSrc += 1
88             if pkt.fl_fin == '1':
89                 self.cntFINSrc += 1
90             if pkt.fl_RST == '1':
91                 self.cntRSTSsrc += 1

```

```

92             self.cntPktTCPDestIP2 += 1
93             # TODO надо подумать что делать с этими флагами соединения
94             if pkt.fl_fin == '1' or pkt.fl_rst == '1':
95                 self.forceFin = True
96             # Подсчет размеров окна
97             self.winSizeList.append(pkt.win_size)
98         else:
99             self.cntPktUDP += 1
100            self.lastTimePkt = pkt.timePacket
101            self.cntPkt += 1
102            self.CNT += 1
103
104            # Обнуление накопленных данных после предсказания
105        def clean_all_parameters(self):
106            # Очистка временных данных
107            self.prevTimePkt = None
108            self.intervalsList.clear()
109
110            # Сброс счетчиков и списков
111            self.cntPktSrcIP1 = self.cntPktDestIP1 = 0
112            self.pktSizeDestIP1.clear()
113            self.pktSizeDestIP2.clear()
114            self.cntPSHDestIP1 = self.cntPSHDestIP2 = 0
115            self.cntACKDestIP1 = self.cntACKDestIP2 = self.cntACKSrcIP1 = 0
116            self.cntPktTCPDestIP1 = self.cntPktTCPDestIP2 = 0
117            self.cntSYNSrc = self.cntSYNDest = self.cntFINSrc = 0
118            self.cntFINDest = self.cntrRSTSsrc = self.cntrRSTDest = 0
119
120            self.winSizeList.clear()
121            self.cntPktUDP = self.cntPkt = 0
122
123            # Получение входного вектора x_t
124        def get_result(self):
125
126            def ratio_calc(num, denom):
127                if denom == 0:
128                    return 0
129                return num / denom
130
131            result = []
132            if not self.stateActive:
133                return None
134            l = len(self.intervalsList)
135            self.totalTime = round(self.lastTimePkt - self.strt_time, 2)
136            if self.stateActive and self.cntPkt < 2:
137                self.stateActive = False
138            return None
139            # Вычисление средней задержки

```

```

140     sum = 0
141     for el in self.intervalsList:
142         sum += el
143     result.append(sum / 1)
144     # Вычисление стандартного отклонения
145     sum = 0
146     for el in self.intervalsList:
147         sum += (el - result[0]) * (el - result[0])
148     result.append(sqrt(sum / 1))
149     # Вычисление среднего отклонения (джиттера)
150     sum = 0
151     if l < 2:
152         result.append(0)
153     else:
154         for i in range(1, l):
155             sum += abs(self.intervalsList[i] - self.intervalsList[i
156             - 1])
156             result.append(sum / (l - 1))
157             # Вычисление медианы временных интервалов
158             tmp = sorted(self.intervalsList)
159             if l % 2 == 0:
160                 result.append((tmp[(l // 2) - 1] + tmp[l // 2]) / 2)
161             else:
162                 result.append(tmp[l // 2])
163                 # Вычисление отношения объема входящего на исходящий трафик для
164                 IP1 и IP2
165                 result.append(ratio_calc(self.cntPktDestIP1, self.cntPktSrcIP1)
166             )
166                 result.append(ratio_calc(self.cntPktSrcIP1, self.cntPktDestIP1)
167             )
167                 # Вычисление отношения объема UDP-трафика и TCP-трафика
168                 result.append(ratio_calc(self.cntPktUDP, self.cntPkt - self.
169                 cntPktUDP))
169                 # Вычисление среднего значения объема пакетов получаемого IP1
170                 l = len(self.pktSizeDestIP1)
171                 sum = 0
172                 for el in self.pktSizeDestIP1:
173                     sum += el
174                     if l != 0:
175                         result.append(sum / l)
176                     else:
177                         result.append(0)
177                         # Вычисление среднего значения объема пакетов получаемого IP2
178                         l = len(self.pktSizeDestIP2)
179                         sum = 0
180                         for el in self.pktSizeDestIP2:
181                             sum += el
182                             if l != 0:

```

```

183         result.append(sum / 1)
184     else:
185         result.append(0)
186     # Вычисление частоты флагов PSH для IP1 и IP2
187     result.append(ratio_calc(self.cntPSHDestIP1, self.
188 cntPktTCPDestIP1))
188     result.append(ratio_calc(self.cntPSHDestIP2, self.
189 cntPktTCPDestIP2))
190     # Вычисление частоты флагов ACK для IP1 и IP2
191     result.append(ratio_calc(self.cntACKDestIP1, self.
192 cntPktTCPDestIP1))
193     result.append(ratio_calc(self.cntACKDestIP2, self.
194 cntPktTCPDestIP2))
195     # Вычисление отношения ACK/PSH для IP1 и IP2
196     result.append(ratio_calc(self.cntPSHDestIP1, self.cntACKDestIP1
197 ))
197     result.append(ratio_calc(self.cntPSHDestIP2, self.cntACKDestIP2
198 ))
199     # Вычисление разности числа исходящих и входящих ACK-флагов IP1
200     result.append(abs(self.cntACKDestIP1 - self.cntACKSrcIP1))
201     # Вычисление отношения количества флагов SYN, FIN, RST
202     result.append(ratio_calc(self.cntSYNSrc + 1
203 , self.cntFINSrc + self.cntRSTSsrc + 1
204 ))
205     result.append(ratio_calc(self.cntSYNDest + 1
206 , self.cntFINDest + self.cntRSTDest + 1
207 ))
208     # Вычисление среднего размера экрана
209     l = len(self.winSizeList)
210     if l != 0:
211         sum = 0
212         for el in self.winSizeList:
213             sum += el
214         result.append(sum / l)
215     # Вычисление частоты обновления окна
216     cntRatio = 1
217     prev = self.winSizeList[0]
218     for winSize in self.winSizeList[1:]:
219         if prev != winSize:
220             prev = winSize
221             cntRatio += 1
222             result.append(cntRatio / 15)
223     else:
224         result.extend([0, 0])
225     result.append(self.cntPkt)
226     self.clean_all_parameters()
227     return result
228

```

```

223 # Оценка результата предсказания
224 def rdp_prob_check(self, val0, val1):
225     if val0 > 0.5 and val1 < 0.5:
226         self.rdpProb.append((True, [val0, val1]))
227         self.cntTr += 1
228     else:
229         self.rdpProb.append((False, [val0, val1]))
230     l = len(self.rdpProb)
231     if not self.isRDP and l >= 2:
232         self.isRDP = self.cntTr > l - self.cntTr
233
234
235 # Обработка получаемых пакетов
236 class SessionInitialization:
237
238     def __init__(self, fl_find_rdp=False, fl_train=True) -> None:
239         self.cur_ports = set()
240         self.curTime = None
241         self.model = None
242         self.train_mode = fl_train
243         self.findRDP = fl_find_rdp
244         self.x_input = []
245         self.cntPeriods = 0
246         self.line = '-----',
247
248     # Инициализация времени
249     def add_start_time(self, strt):
250         self.curTime = strt + 15
251
252     # Запись входных векторов в файл
253     def write_data_to_file(self, filename='x_input.log'):
254         with open(filename, 'a+') as f:
255             f.write(f'{self.cntPeriods}-th interval\n')
256             for ports, row in self.x_input:
257                 f.write(f'{ports}:')
258                 for el in row:
259                     f.write(f'{el},')
260                 f.write('!\n')
261
262     # Загрузка модели для выявления RDP сессий
263     def load_LSTM_model(self, filename='model.keras'):
264         try:
265             self.model = load_model(f'../model_directory/{filename}')
266         except Exception as ex:
267             print(ex)
268             print('Модель должна лежать в каталоге model.keras')
269             return False
270         else:

```

```

271         print(' \nМодель успешно загружена!')
272         return True
273
274     # Выполнение предсказания по каждой активной сессии
275     def get_prediction(self, indexes):
276         pred = self.model.predict(self.x_input)
277         j = 0
278         for i in indexes:
279             Session_list[i].rdp_prob_check(pred[0, j, 0], pred[0, j,
280                                             1])
281             j += 1
282
283     # Обработка режимов работы с моделью
284     def packet_preparation(self):
285         self.x_input = []
286         self.cntPeriods += 1
287         # Если поставлен флаг для выявления RDP-трафика
288         # то происходит работа с нейронной сетью
289         if self.findRDP:
290             ids = []
291             for i in range(len(Session_list)):
292                 vec = Session_list[i].get_result()
293                 if vec is not None:
294                     ids.append(i)
295                     self.x_input.append(vec)
296             self.x_input = np.array(self.x_input)
297             self.x_input = np.expand_dims(self.x_input, axis=0)
298             self.get_prediction(ids)
299
300         # Режим обучения
301         elif self.train_mode:
302             for s in Session_list:
303                 vec = s.get_result()
304                 if vec is not None:
305                     self.x_input.append(((s.ips, s.ports), vec))
306             self.write_data_to_file()
307
308         else:
309             for s in Session_list:
310                 vec = s.get_result()
311
312     # Классификация пакетов по сессиям
313     def find_session_location(self, pkt) -> bool:
314         global Session_list
315         isNewSession = True
316         if pkt.timePacket > self.curTime:
317             self.packet_preparation()
318             self.curTime += 15
319         for s in Session_list:
320             if s.stateActive and pkt.ip_src in s.ips and pkt.ip_dest in

```

```

    s.ips:
318             if (s.ports[1] is None and ( pkt.port_src == s.ports[0]
or \
319                                         pkt.port_dest == s.
ports[0]) ) or \
320                                         (pkt.port_src in s.ports and pkt.port_dest in s.
ports):
321                 isNewSession = False
322                 s.update_data(pkt)
323                 return s.isRDP
324             elif s.ports[1] is not None:
325                 if (pkt.port_dest in s.ports and pkt.port_src not
in s.ports):
326                     isNewSession = False
327                     s.ports = (s.ports[1], None)
328                     self.cur_ports.add(pkt.port_dest)
329                     s.update_data(pkt)
330                     return s.isRDP
331             elif (pkt.port_src in s.ports and pkt.port_dest not
in s.ports):
332                     isNewSession = False
333                     s.ports = (s.ports[0], None)
334                     self.cur_ports.add(pkt.port_src)
335                     s.update_data(pkt)
336                     return s.isRDP
337         if isNewSession:
338             if pkt.protoType == 'TCP' and pkt.fl_syn == '1' and pkt.
fl_ack == '0':
339                 self.cur_ports.add(pkt.port_dest)
340                 Session_list.append(Session( pkt.timePacket
341                                         , (pkt.ip_src, pkt.ip_dest)
342                                         , (pkt.port_dest, None)))
343         else:
344             Session_list.append(Session( pkt.timePacket
345                                         , (pkt.ip_src, pkt.ip_dest)
346                                         , (pkt.port_src, pkt.
port_dest)))
347             Session_list[-1].update_data(pkt)
348         return False
349
350 # Вывод информации о перехваченных пакетах
351 def print_packet_information(self, pkt, pred_res):
352     if self.findRDP and not pred_res:
353         return
354     packet_info = [
355         f'{self.line}Пакет №{pkt.numPacket}{self.line}',

356         f'Время перехвата: {time.strftime("%m:%d:%Y %H:%M:%S", time
.localtime(pkt.timePacket))}',


```

```

357         f'Протокол: {pkt.protoType},',
358         f'MAC-адрес отправителя: {pkt.mac_src},',
359         f'MAC-адрес получателя: {pkt.mac_dest},',
360         f'Отправитель: {pkt.ip_src}:{pkt.port_src},',
361         f'Получатель: {pkt.ip_dest}:{pkt.port_dest},'
362     ]
363     if pkt.protoType == 'TCP':
364         tcp_flags = f'SYN:{pkt.fl_syn}; ACK:{pkt.fl_ack}; PSH:{pkt.
365         fl_psh}; RST:{pkt.fl_rst}; FIN:{pkt.fl_fin}'
366         packet_info.append(f'Порядковый номер: {pkt.seq}; Номер под
367        тверждения: {pkt.ack}')
368         packet_info.append(tcp_flags)
369         print("\n".join(packet_info))
370         if self.findRDP and pred_res:
371             print(f'{self.line} Обнаружена RDP-сессия! {self.line}')
372
373     # Обработка значений списка Session_list
374     def clear_unwanted_sessions(self):
375         global Session_list
376         Session_list = [
377             session for session in Session_list
378             if session.CNT >= 20 and session.totalTime >= 10
379         ]
380
381     # Вывод информации о сессиях
382     def print_inf_about_sessions(self):
383         print(f'\nБыло перехвачено {len(Session_list)} сессии(-й)')
384         for cnt, s in enumerate(Session_list, start=1):
385             session_info = [
386                 f'\nИнформация о сессии #{cnt}:',
387                 f'IP-адреса: {s.ips}',
388                 f'Порт подключения: {s.ports[0]}', if s.ports[1] is None
389                 else f'Порты подключения: {s.ports}',
390                 f'Время перехвата первого пакета: {time.strftime("%d.%m
391                 .%Y г. %H:%M:%S", time.localtime(s.strt_time))}',
392                 f'Количество перехваченных пакетов: {s.CNT}',
393                 f'Общее время перехвата: {s.totalTime}'
394             ]
395             if s.isRDP:
396                 session_info.append(Back.GREEN + Fore.BLACK + 'Найдена
397                 RDP-сессия!!!')
398                 print("\n".join(session_info))
399                 print(f'{self.line}{self.line}\n')

```

ПРИЛОЖЕНИЕ Ж

Листинг main.py

```
1 import os
2 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
3 import tensorflow as tf
4 from session_creation import Session_list
5 from chart_creation import Object_list
6 from common_methods import read_from_file, write_to_file, Packet_list
7 from sniffer import Sniffer
8 from traffic_analysis import TrafficAnalysis
9
10
11 # Выбор опции (меню)
12 def choose_mode():
13     global Packet_list, Object_list, Session_list
14     while True:
15         print('1. Перехват трафика')
16         print('2. Запись данных в файл')
17         print('3. Считывание с файла данных для анализа трафика')
18         print('4. Анализ трафика')
19         print('5. Выход')
20         bl = input()
21         if bl == '1':
22             Packet_list.clear()
23             Object_list.clear()
24             Session_list.clear()
25             Sniffer().traffic_interception()
26         elif bl == '2':
27             write_to_file()
28         elif bl == '3':
29             Packet_list.clear()
30             Object_list.clear()
31             Session_list.clear()
32             read_from_file()
33             print(f'\nДанные собраны. Перехвачено: {len(Packet_list)} пакетов(-а)\n')
34         elif bl == '4':
35             TrafficAnalysis().start_to_analyse()
36         elif bl == '5':
37             return
38
39
40 if __name__ == '__main__':
41     print('\nЗапуск программы....\n')
42     choose_mode()
43
```