МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки)

Компьютерные технологии и математическое моделирование
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Анализ DNS-трафика в режиме реального времени с целью выявления образования DNS туннелей

Обучающийся М.А. Шипицин
(Инициалы Фамилия) (личная подпись)

Руководитель к.ф.-м.н., Г.А. Тырыгина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант к.п.н., Т.С. Якушева
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2023

Аннотация

Тема бакалаврской работы: «Анализ DNS-трафика в режиме реального времени с целью выявления образования DNS туннелей».

Актуальность работы подкрепляется появлением новых вирусов, использующих технику DNS-туннелирования.

Объект исследования – техника обнаружения DNS-туннелей.

Предметом работы являются методы, алгоритмы и программное обеспечение компьютерных систем для решения задач обнаружения передачи несанкционированного трафика по компьютерной сети посредством туннелирования DNS.

Целью работы является создание системы, способной анализировать в режиме реального времени DNS-трафик для распознавания формирования DNS-туннелей при помощи методов машинного обучения.

Для достижения цели выпускной квалификационной работы, необходимо решить ряд задач:

1. провести анализ предметной области,
2. реализовать систему по обнаружению DNS-туннелей,
3. провести тестирование и замеры эффективности разработанной системы;

Выпускная квалификационная работа состоит из введения, трех разделов и заключения.

Результаты бакалаврской работы представляют практический интерес и могут быть использованы в целях кибер-защиты.

В работе использовано 19 рисунков, 26 источников используемой литературы. Общий объем ВКР составляет 42 страницы.

Abstract

The title of the graduation work is "Real-time analysis of DNS traffic to detect DNS tunnel formation". The purpose of this work is to create a model system capable of analyzing DNS traffic in real-time and recognizing the formation of DNS tunnels using machine learning methods.

The object of the research is the detection of DNS tunneling technique, while the subject of the work is the methods, algorithms, and software of computer systems to solve the problem of detecting unauthorized traffic transmission through a computer network using DNS tunneling.

The senior paper consists of an introduction, three sections, and a conclusion. The first section discusses existing methods of creating DNS tunnels and their detection. The second section designs software modules and implements the software. The third section performs testing and analysis of the effectiveness of the developed system.

The aim of the graduation project includes studying existing approaches, developing a new methodology, conducting experimental research, analyzing the results, and comparing them with existing methods.

In conclusion, I would like to emphasize that this work is relevant for increasing the level of cyber security. It can be used to prevent potential security threats, such as data exfiltration and unauthorized access to network resources.

Содержание

Введение

Скрытые кибератаки могут иметь серьезные последствия для компаний. В частности, злоумышленники могут получить конфиденциальную информацию, которая может быть использована в корыстных целях. Это может привести к утрате доверия пользователей и, в конечном итоге, к снижению рыночной стоимости организации. Кроме того, компания может столкнуться с судебными и финансовыми последствиями в случае утечки конфиденциальной информации [1].

DNS-туннелирование - это метод передачи данных через протокол DNS, который используется для решения сетевых запросов, связанных с доменными именами. При использовании DNS-туннелирования данные кодируются в формат DNS-запроса, который затем отправляется на DNS-сервер. Данные извлекаются из ответа, который возвращается отправителю, и расшифровываются. DNS-туннелирование используется для передачи данных через сеть, которая обычно блокирует определенные протоколы, такие как FTP, SSH и т.д.

DNS-серверы могут быть скомпрометированы злоумышленниками, которые могут использовать различные методы, такие как взлом пароля администратора, использование уязвимостей в программном обеспечении сервера или ввод вредоносного кода через DNS-туннелирование [4].

Для защиты DNS-серверов от таких атак, была реализована система, основанная на сверточной нейронной сети, способная обнаруживать формирование DNS-туннелей. Этим подтверждается актуальность работы.

1 Исследование предметной области

1.1 Постановка задачи

Первое известное обсуждение DNS-туннелирования было получено от Оскара Пирсона в списке рассылки Bugtraq в апреле 1998 года (Pearson, 1998) [11]. С тех пор было разработано несколько утилит DNS для туннелирования DNS. Все утилиты используют схожие основные методы, но имеют различия в кодировке и других деталях реализации. Основные методы, используемые всеми утилитами туннелирования DNS, включают контролируемый домен или поддомен, компонент на стороне сервера, компонент на стороне клиента и данные, закодированные в полезной нагрузке DNS (рисунок 1).

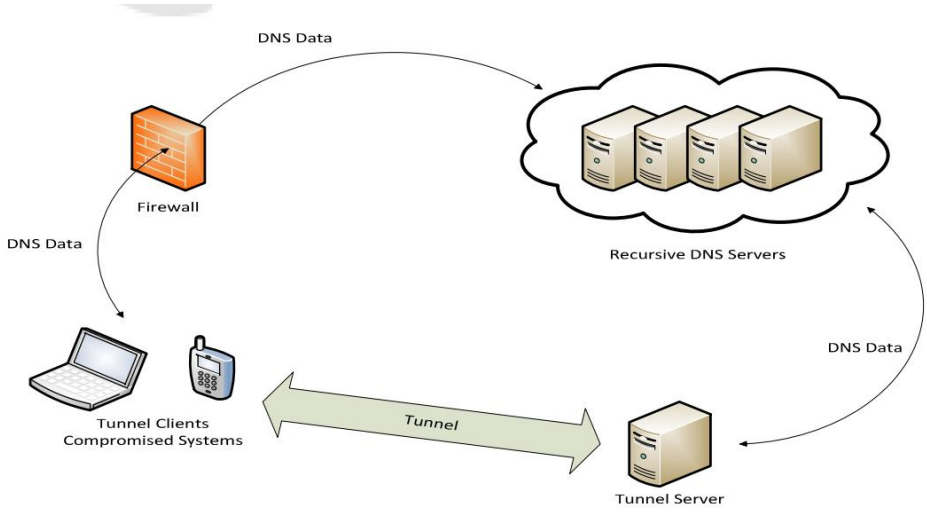

Рисунок 1 – Принципиальная схема DNS туннелирования.

За последние десять лет были обнаружены два вида сетевых вирусов-червей: Morto и Federbot. Они используют технологию DNS-туннелирования для передачи команд управления зараженными узлами. Кроме того, специалисты в области кибербезопасности обнаружили вредоносное программное обеспечение BernhardPOS и FrameworkPOS, которое также использует DNS-туннелирование для кражи информации о кредитных картах с платежных терминалов [2]. Это подтверждает актуальность техники DNS-туннелирования.

Для моделирования новой методики, были проанализированы существующие методы создания DNS-туннелей и их обнаружения. Изучены основные средства анализа сетевого трафика, используемые для обнаружения DNS-туннелей, а также методы машинного обучения, применяемые для автоматического распознавания таких туннелей.

1.1.1 Основы DNS

DNS (Domain Name System) является системой, предназначенной для перевода доменных имен в IP-адреса, которые используются для обмена данными в компьютерных сетях (рисунок 2). Злоумышленники могут использовать DNS для скрытия кибератаки, инкапсулируя протоколы разных уровней модели OSI в доменное имя DNS-запроса. Это позволяет им обойти защитные механизмы, которые основаны на отслеживании трафика сети и протоколов, используемых при обмене данными [3].

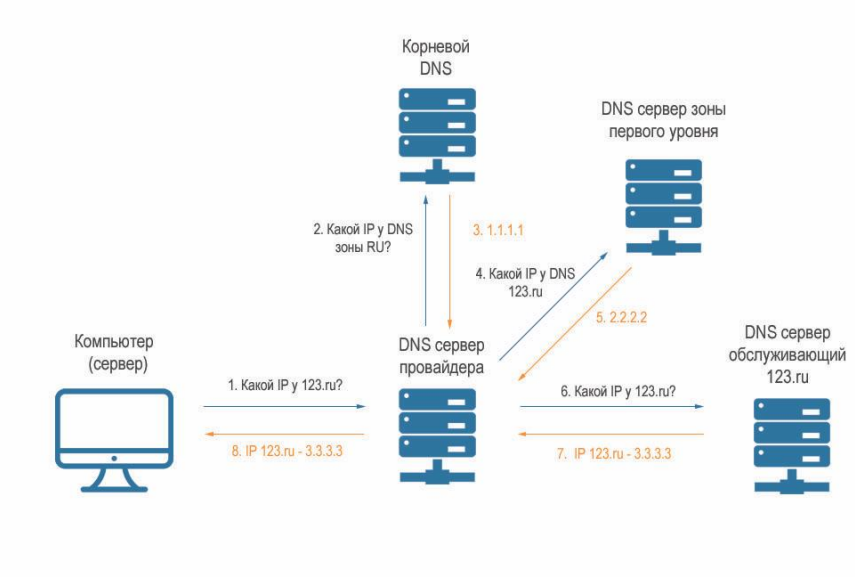

Рисунок 2 – Принцип работы DNS.

DNS имеет более 30 типов записей, многие из которых являются критическими для основных интернет-сервисов. Тип записи A преобразует доменное имя в IPv4-адрес. Запись AAAA используется для преобразования

домена в IPv6-адрес. Тип записи CNAME используется для преобразования доменного имени в каноническое имя. Тип записи MX используется для определения почтовых серверов для домена. Тип записи NS используется для определения авторитетных серверов имен для домена. Запись PTR или указатель обычно используется для преобразования IP-адреса в его доменное имя. Это обычно называется обратным поиском [22].

DNS использует как UDP-серверный порт 53, так и TCP-серверный порт 53 для связи. Обычно используется UDP, но TCP будет использоваться для передачи зон или с полезной нагрузкой более 512 байт. Также есть расширенные механизмы для DNS (EDNS) (Vixie, 1999). Если EDNS поддерживается обоими узлами в DNS-связи, то можно использовать UDP-полезную нагрузку более 512 байт. EDNS - это функция, которую можно использовать для улучшения пропускной способности для туннелирования DNS.

DNS - иерархическая система, в которой каждый уровень может обслуживаться другим сервером с различной владельческой принадлежностью. Благодаря этой иерархической системе, владелец домена может определить авторитетные серверы для своего домена, то есть контролировать конечный хост назначения для запросов DNS для своего домена. В типичном предприятии конечные точки не запрашивают DNS напрямую из интернета. У них есть внутренние DNS-серверы, которые обеспечивают DNS-сервисы для конечных точек [5].

Однако, поскольку DNS будет перенаправлять запросы до тех пор, пока не будет обращаться к авторитетному DNS-серверу, злоумышленник, имеющий доступ к внутренней конечной точке, может использовать инфраструктуру DNS предприятия для создания DNS-туннеля к домену, который он контролирует.

1.1.2 Эксфильтрация данных

Кодирование данных с помощью алфавитного кодировщика - это метод перевода данных из одной формы представления в другую. В данном случае данные переводятся в формат DNS-запроса, который используется в DNS-туннелировании. Алфавитный кодировщик используется для перевода данных в последовательность символов, которые можно закодировать в DNS-запросе (рисунок 3).

Рисунок 3 – Закодированная информация в доменном имени

Передача данных без разрешения владельца, также известная как эксфильтрация данных, может быть рассмотрена как кража информации. В данной работе исследуется конкретный случай утечки данных, который включает в себя злонамеренную кражу корпоративных секретов, интеллектуальной собственности и другой ценной информации у владельца. Эта утечка данных осуществляется с использованием техники DNS-туннелирования, как показано на рисунке 4.

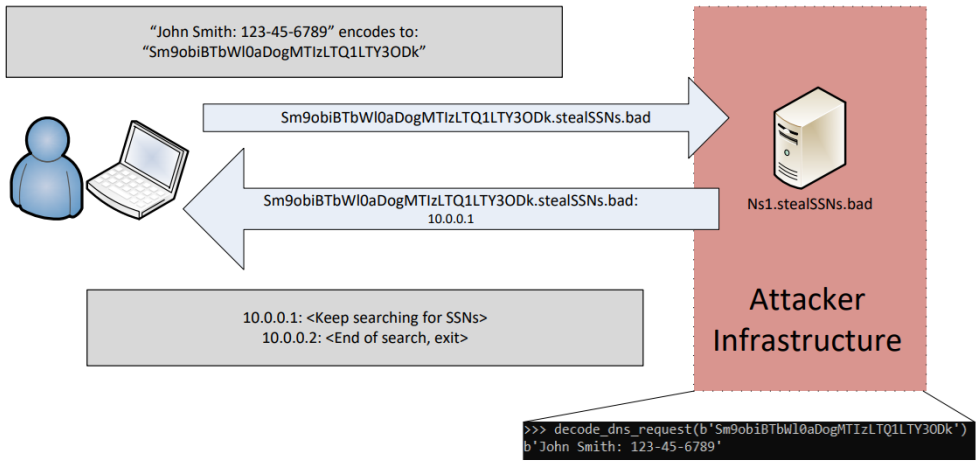

Рисунок 4 – Эксфильтрация данных через DNS туннель.

Среди различных прикладных методов обнаружения эксфильтрации данных через DNS-туннелирование одним из них является анализ длины доменных имен [9]. В соответствии с исследованием, проведенным авторами этого метода, обычно длина субдоменов не превышает 30 символов. На основе этого наблюдения можно установить правила блокировки DNS-запросов, длина которых превышает этот порог. Это позволяет идентифицировать потенциально подозрительные запросы и предотвратить эксфильтрацию данных через DNS-туннелирование. Однако следует отметить, что такой метод не является универсальным, поскольку программы, такие как Iodine, позволяют установить максимальное значение единицы передачи (MTU) и таким образом обойти такие ограничения [7, 20].

Для борьбы с проблемой эксфильтрации данных через DNS-протокол предлагается метод бинарной классификации DNS-запросов на безопасные и небезопасные. Запросы, используемые для туннелирования системы доменных имен, считаются небезопасными, в то время как все остальные запросы считаются безопасными. На основе этого метода можно принимать меры для предотвращения эксфильтрации данных.

Данный классификатор представляет собой сверточную нейронную сеть, обученную на доменных именах из открытых репозиториев,

используемых при DNS-туннелировании. Такой подход основан на предположении, что безопасные DNS-запросы составляют подмножество естественных языковых выражений, в то время как запросы, используемые для DNS-туннелирования, не обладают этим свойством.

Межсетевое экранирование представляет собой процесс, при котором создается межсетевой экран. Этот экран обеспечивает безопасность сетевых соединений, разделяет различные сетевые зоны и контролирует трафик между ними.

Межсетевой экран (англ. firewall) функционирует как система безопасности, защищающая компьютерные сети от несанкционированного доступа и контролирующая передачу данных между сетями (рисунок 5). Он обычно располагается на границе сети и осуществляет контроль входящего и исходящего сетевого трафика. Firewall использует определенные правила и политики, основываясь на источнике, направлении, протоколе, портах и других параметрах, для разрешения или блокирования трафика. Таким образом обеспечивается безопасность сети и предотвращаются внешние атаки, такие как вирусы, черви, троянские программы, взломы и другие формы киберпреступности.

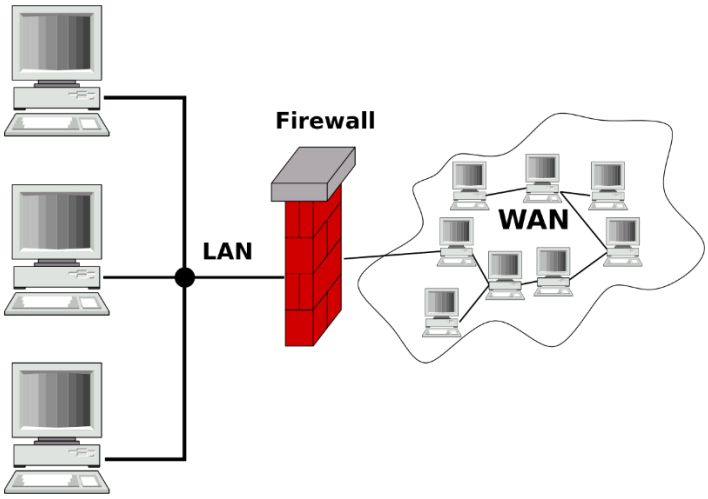

Рисунок 5 – Брандмауэр (англ. firewall)

DNS-туннелирование может быть использовано для кражи данных, например, для передачи украденных данных через DNS-запросы, обходя сетевые фильтры и межсетевые экраны. Для защиты от таких атак рекомендуется использовать методы защиты DNS-серверов, такие как контроль доступа и обновление программного обеспечения. Злоумышленник получает доступ к данным через DNS-туннелирование, потому что этот протокол не блокируется межсетевыми экранами, что делает его популярным средством для кибератак. Кроме того, методика DNS-туннелирования может использоваться для обхода систем защиты и фильтрации, что делает ее еще более опасной [21].

Мониторинг сетевого трафика является критически важным для сетевых администраторов. Он позволяет быстро обнаруживать и устранять проблемы, связанные с сетью, что обеспечивает бесперебойную работу трафика и защиту от нарушений безопасности.

1.2 Существующие методы

1.2.1 Анализ биграмм

В работах [10, 11] исследуется метод обнаружения DNS-туннелей, основанный на анализе частоты использования биграмм в доменных именах. Авторы предполагают, что безопасные доменные имена следуют закону Ципфа, в то время как имена, используемые для туннелирования, распределены равномерно [19].

Исходя из этой гипотезы, каждое доменное имя оценивается на основе частоты биграмм в естественном языке. Для этой цели используется предварительно подготовленная таблица с частотами биграмм. Если оценка не превышает заданного порога, алгоритм считает доменное имя небезопасным. Применение этого подхода позволило достичь точности обнаружения DNS-туннелей на уровне 98,74%.

Несмотря на высокую точность, этот метод имеет некоторые недостатки:

– требуется построение таблицы с частотами биграмм для каждого естественного языка, что на практике затруднительно,
– анализ проводился только на доменных именах, созданных программой туннелирования DNSCat, которая является одной из множества доступных программ для создания DNS-туннелей;

1.2.2 Статистический метод

Другой метод обнаружения DNS-туннелирования основан на предположении, что статистические характеристики определенных числовых параметров DNS-запросов при туннелировании отличаются от характеристик безопасных запросов. Например, исследования [7, 8] рассматривают такие параметры, как энтропия доменного имени, объем трафика к домену, соотношение длины доменного имени к субдомену и другие. Авторы предлагают использовать алгоритм изолирующего леса (isolation forest) для обнаружения аномалий и подчеркивают эффективность этого метода в обнаружении DNS-туннелирования и высокую точность классификации [15].

Однако проблема такого подхода заключается в его практическом применении. Если провайдер DNS, предоставляющий услуги разрешения доменных имен, решит использовать подобный детектор для вычисления общего трафика к вредоносному домену, ему потребуется собирать параметры со всех DNS-резолверов. Это требует распределенного хранения таблицы частот запросов, что не всегда осуществимо из-за сложности инженерных задач и финансовых затрат на поддержку такой инфраструктуры.

1.2.3 Метод опорных векторов (SVM)

Метод опорных векторов (Support Vector Machine, SVM), может применяться как для классификации, так и для регрессии. В задаче классификации SVM ищет гиперплоскость в пространстве признаков, которая наилучшим образом разделяет объекты разных классов (рисунок 6).

Этот метод позволяет решать задачи многоклассовой классификации, включая туннелированные и нетуннелированные случаи. В данной работе осуществляется поиск различных типов DNS-туннелей (FTP, POP3, HTTP, HTTPs) по отдельности.

Данный подход основан на использовании данных NetFlow, которые не содержат информации о передаваемых данных или доменных именах. Однако преимуществом решения на основе NetFlow является высокая производительность, так как нет необходимости распаковывать каждый отдельный запрос, что требовало бы значительных вычислительных ресурсов. NetFlow предоставляет информацию о суммарном количестве запросов в одном IP-потоке [9].

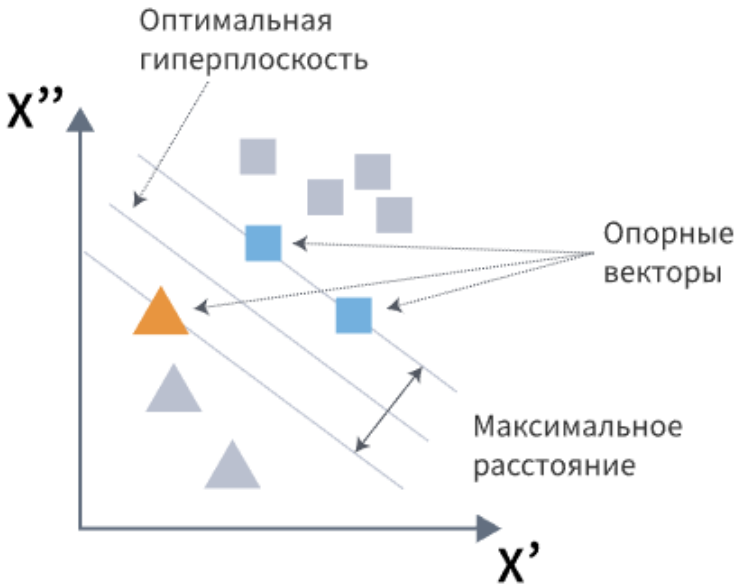

Рисунок 6 – Классификация SVM.

Однако, несмотря на то, что исследователи работают с инструментами туннелирования, они проводят мало экспериментов с различным программным обеспечением, фокусируясь на одном или двух из них.

1.2.4 Метод случайного леса (RF)

Случайный лес (Random Forest) – алгоритм машинного обучения, применяемый для классификации, регрессии и кластеризации данных. В случае классификации, случайный лес состоит из ансамбля решающих деревьев, каждое из которых голосует за определенный класс объекта (рисунок 7).

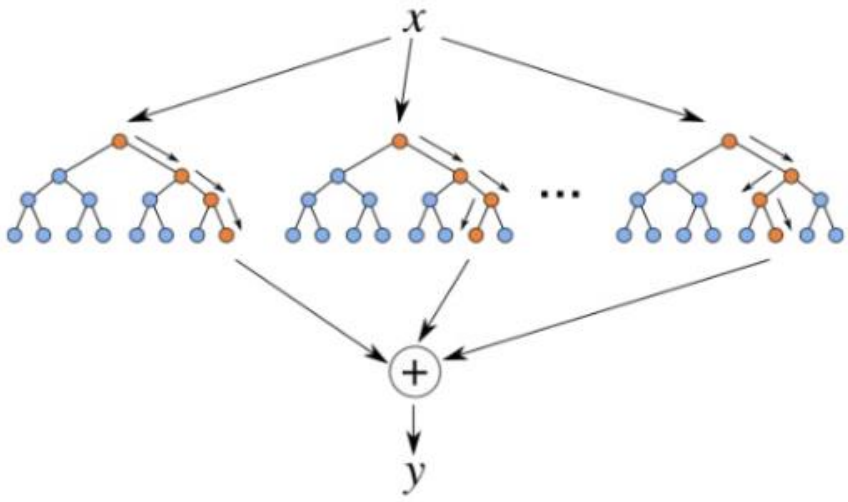

Рисунок 7 – Принципиальная схема метода случайного леса.

Рассмотрим процесс работы случайного леса в задаче классификации:

- входные данные представляются в виде признаков и меток классов, где метки указывают, к какому классу относится каждый объект,
- алгоритм создает несколько (обычно от 10 до 100) решающих деревьев, каждое из которых обучается на случайном подмножестве данных и случайном подмножестве признаков. Этот подход помогает снизить переобучение модели и улучшить ее обобщающую способность,
- для каждого дерева алгоритм выбирает наиболее важные признаки, которые используются при построении дерева,

– после построения всех деревьев каждое из них голосует за класс объекта. Обычно выбирается класс с наибольшим количеством голосов в качестве итогового классификационного результата,
– после обучения случайного леса, он может быть использован для классификации новых объектов путем применения каждого дерева к этим объектам и подсчета голосов для каждого класса;

В итоге, метод случайного леса предоставляет более точные результаты по сравнению с отдельными деревьями, так как он объединяет множество деревьев и применяет статистические методы для уменьшения ошибок.

Важно отметить, что метод случайного леса имеет свои ограничения, включая высокую вычислительную сложность и необходимость настройки параметров модели. Однако он все еще широко применяется в задачах классификации из-за своей высокой точности и способности обрабатывать большие объемы данных [18].

В исследовании Buczak [11] были рассмотрены различные типы характеристик для обнаружения туннелирования DNS. Они использовали случайный лес в качестве классификатора для сравнения этих характеристик. В числе этих характеристик были количество ответов, предоставленных в ответе, время между двумя последовательными запросами для определенного домена и время между двумя последовательными ответами для определенного домена. Также было рассмотрено время между двумя последовательными пакетами для определенного домена.

1.3 Существующие утилиты по созданию DNS-туннелей

На данный момент существует более 14 ПО, находящихся в свободном доступе, использующие различные техники DNS-туннелирования. Наиболее популярные из них:

– dns2tcp: была написана Оливье Дембуром и Николя Коллиньоном. Она написана на языке C и работает под управлением Linux. Клиент может работать под Windows. Поддерживает запросы типа KEY и TXT (Dembour, 2008),
– DNScapy: был разработан Пьером Биенаймом. Он использует Scapy для генерации пакетов. DNScapy поддерживает SSH туннелирование через DNS, включая Socks-прокси. Его можно настроить на использование CNAME или TXT записей, или и того, и другого в произвольном порядке (Bienaime, 2011),
– DNScat (1): был первоначально выпущен в 2004 году, а самая последняя версия была выпущена в 2005 году. Он был написан Тадеушем Пьетрашеком. DNScat представлен как универсальный инструмент с множеством применений, включая двунаправленную коммуникацию через DNS. DNScat основан на Java и работает на Unix-подобных системах. DNScat поддерживает записи A и запросы на запись CNAME (Pietraszek, 2004),
– DNScat (2): был написан Роном Боусом. Самый ранний известный публичный релиз был выпущен в 2010 году. Он работает под управлением Linux, Mac OS X и Windows. DNScat кодирует запросы либо в кодировке NetBIOS, либо в шестнадцатеричной кодировке. DNScat может использовать A, AAAA, CNAME, NS, TXT и MX. Он обеспечивает дейтаграммный и потоковый режим. Существует также полезная нагрузка Metasploit на основе DNScat-B (Bowes, 2010),
– Iodine: программа туннелирования DNS, впервые выпущенная в 2006 году с обновлениями до 2010 года. Она была разработана Бьорном Андерссоном и Эриком Экманом. Iodine написана на языке C и работает под управлением Linux, Mac OS X, Windows и других операционных систем. Iodine был перенесен на Android. Он использует интерфейс tun или tap на конечной точке (Andersson, 2010);

1.4 Математическая модель

Сформулируем задачу обнаружения DNS-туннеля следующим образом: пусть $\mathrm{X} = \{x_0, x_1, \dots, x_n\}$ – запрашиваемое доменное имя, с которым ассоциирована метка $y \in Y$ с вероятностью p_y . Тогда стоит задача нахождения функции

$$y: \mathrm{X} \to Y, \tag{1}$$

Пусть дан алфавит символов A, из которого состоит доменное имя, такое, что $x_i \in \mathrm{A}$, а само доменное имя представляет собой подмножество размещений алфавита $\mathrm{X} \subset \mathrm{A}^{\mathrm{k}}$, при этом $k = |\mathrm{X}| = 256$.

Таким образом, задача заключается в классификации слов из заданного алфавита в соответствии с классами, определенными в множестве Y. При этом необходимо учитывать максимальную длину доменного имени, установленную в спецификации протокола DNS [10].

Применим метод, описанный в работе [2], для обнаружения функции (1), при этом в качестве классификатора будет использоваться сверточная нейронная сеть.

1.4.1 Сверточная нейронная сеть

Сверточная нейронная сеть (СНС) (англ. CNN) в области обработки естественного языка (ЕЯ) (англ. NLP) является одним из самых популярных методов анализа текстовых данных. Изначально разработанная для работы с изображениями, СНС была успешно адаптирована для обработки текста. В области ЕЯ сверточная нейронная сеть использует операцию свертки для извлечения характеристик из текстовых данных. Эта сеть может использоваться для классификации текста на основе извлеченных характеристик, которые затем передаются в полносвязную сеть для окончательной классификации [23].

Процесс работы сверточной нейронной сети в области ЕЯ начинается с применения нескольких операций свертки к текстовым данным, где каждая операция использует свое определенного размера ядро свертки.

После применения операций свертки полученные характеристики обрабатываются с использованием активационных функций, таких как ReLU (Rectified Linear Unit), которые добавляют нелинейность в выходные данные. Затем выполняется операция подвыборки (Pooling), которая позволяет снизить размерность данных и сделать их более устойчивыми к шуму.

Одна из особенностей сверточных нейронных сетей в NLP заключается в том, что они могут работать со входными данными различной длины. Это достигается с помощью использования фиксированного размера ядер свертки и операций субдискретизации, которые позволяют снизить размерность данных [24].

Также, CNN способны автоматически выделять наиболее важные признаки из последовательностей символов, представляющих DNS-запросы, без необходимости явного определения их формы. Это особенно важно для задачи классификации DNS-туннелирования, так как DNS-запросы могут содержать множество различных признаков, которые могут быть сложно определить вручную.

Кроме того, CNN могут обрабатывать входные данные разной длины, что делает их подходящими для классификации DNS-запросов, которые могут иметь различную длину в зависимости от запроса. CNN также позволяют автоматически определять оптимальные признаки для классификации, что может снизить необходимость в ручном отборе признаков.

Сверточные нейронные сети в NLP применяются в различных задачах, включая классификацию текстов, определение тональности текста, распознавание именованных сущностей и машинный перевод. Они показали высокую точность и скорость работы в сравнении с другими методами обработки текста, такими как рекуррентные нейронные сети (Recurrent Neural Networks, RNN) [25].

Слой Conv1D относится к сверточному слою нейронной сети в одномерном пространстве. В контексте задачи обнаружения туннелирования, данное одномерное пространство представляет длину последовательности символов доменного имени. Свёрточный слой состоит из набора сверточных фильтров, которые применяются к разным участкам доменного имени.

Процесс извлечения признаков для 1D сверточного слоя иллюстрируется на рисунке 8. На нём изображён слой свёртки 1D, создающий 256 фильтров (признаков) (nf = 256) с размером окна (ядром) 4 (ks = 4) и значением шага (stride length) 1 (sl = 1). Слой выбирает группы из 4 символов для применения сверточных фильтров и продолжает перемещаться на один символ вправо (значение шага), применяя те же свертки к остальной части последовательности. В результате нейронная сеть создаёт признаки 4-грамм. Эти признаки представляют дискриминирующую силу этих групп символов в доменных именах.

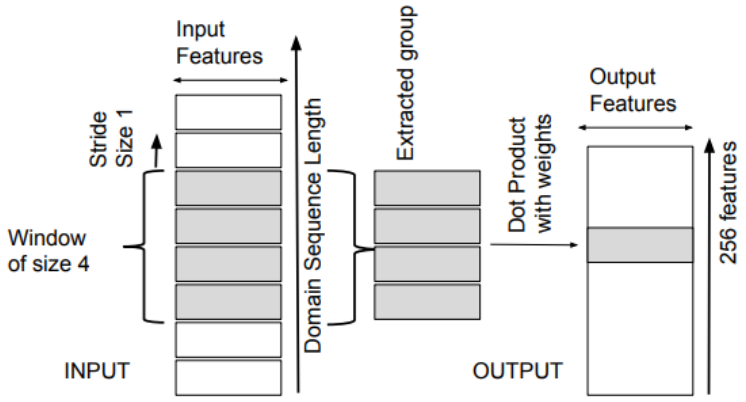

Рисунок 8 – Выделение признаков в одномерном слое свертки.

Применение одного и того же фильтра ко всей последовательности значительно сокращает время вычислений по сравнению с традиционными слоями многослойного персептрона. Кроме того, поскольку сверточное ядро независимо работает с каждым 4-граммом, возможно одновременно пройти по всему входному слою. Это параллелизация и ее следующее низкое время вычислений - одно из основных преимуществ использования сверточных сетей вместо других подходов глубокого обучения, обычно используемых для обработки текста, таких как Long Short Term Memory (LSTM) [26].

Использование сверточных нейронных сетей для классификации DNS туннелирования может быть реализовано следующим образом:

– подготовка данных: Необходимо подготовить набор данных, включающий DNS-запросы и соответствующие метки классов (нормальные запросы и запросы, связанные с туннелированием). Каждый DNS-запрос может быть представлен в виде строки символов. Эти строки нужно преобразовать в числовой формат, например, с помощью кодирования one-hot. Датасет разбивается на обучающую и тестовую выборки,
– создание модели CNN: Модель CNN состоит из слоев свертки, слоев пулинга и полносвязных слоев. Сверточные слои обрабатывают входные данные с использованием сверточных фильтров, которые извлекают различные признаки из данных. Пулинговые слои уменьшают размерность карт признаков, уменьшая количество параметров и улучшая обобщающую способность модели. Полносвязные слои используются для классификации данных,
– обучение модели: Модель обучается с помощью оптимизации функции потерь, которая минимизирует разницу между предсказанными и истинными метками классов на обучающей выборке. Для улучшения качества модели можно применять регуляризацию, аугментацию данных и другие методы,

– оценка модели: После обучения модель необходимо оценить на тестовой выборке. Для этого могут использоваться метрики, такие как точность (accuracy), полнота (recall), точность (precision) и F1-мера. При необходимости можно проанализировать ошибки модели и внести корректировки в процесс обучения;

Для успешной классификации DNS туннелирования с использованием сверточных нейронных сетей (CNN) необходимо правильно выбрать архитектуру модели, параметры обучения и оптимальные гиперпараметры. Также важно учитывать возможность столкновения модели с новыми типами туннелирования, которых нет в обучающей выборке, поэтому важно постоянно обновлять выборку и обучать модель на новых данных.

Применение сверточной нейронной сети в задаче классификации DNS туннелирования способно улучшить качество классификации, так как такие сети используют многослойные архитектуры, способные моделировать более сложные взаимосвязи между признаками [12].

1.4.2 Архитектура нейронной сети

Поскольку входы нейронной сети представляют собой векторы действительных чисел, требуется определить отображение символьного множества X в пространство R^k. В этом случае, возможно использование модифицированного метода встраивания слов (word embedding), описанного в статье [2], который заменяет словарь слов использованием алфавита символов.

Word embedding в задаче классификации NLP с помощью CNN - это процесс преобразования слов в векторные представления, которые затем используются как входные данные для сверточной нейронной сети. Это необходимо, так как сверточные нейронные сети работают с числовыми данными, а слова в текстах представлены в виде строк символов. Word embedding позволяет преобразовать слова в числовые векторы фиксированной длины, которые могут быть использованы в сверточных нейронных сетях.

Есть функция I, которая присваивает каждому элементу из множества A определенный индекс. В данном случае встраивание является процессом преобразования вектора символов в вектор целых чисел, где каждому символу соответствует индекс из алфавита A:

$$I\colon \mathrm{A} \to \{0, \dots, |\mathrm{A}|\}, \mathrm{cemb}(\mathrm{X}) = I^{-1}(X) = \mathrm{E}. \quad (2)$$

Понятно, что массив E, полученный в результате, будет содержать большое количество нулевых элементов из-за редкости доменных имен, превышающих 256 символов в длину. Чтобы уменьшить количество нулей в массиве E, мы будем использовать метод встраивания, предложенный в статье [13].

$$E_{k\to\frac{k}{2}}\colon \mathrm{R}^k \to \mathrm{R}^{\frac{k}{2}}, \mathrm{Q} = E_{k\to\frac{k}{2}}(\mathrm{E}). \quad (3)$$

После сокращения размерности входного вектора применим сверточную фильтрацию

$$\mathrm{conv1d}(\mathrm{Q}; \mathrm{H}, w) = \langle \mathrm{Q}_{i,i+w}, \mathrm{H} \rangle, \mathrm{h_j} = \mathrm{conv1d}(\mathrm{Q}; \mathrm{H}, w) + b, \quad (4)$$

где H – ядро свертки,

w – размер ядра свертки,

Q – встраивание символьного представления доменного имени $\mathrm{Q} \in R^{\frac{k}{2}}$ в пространство действительных чисел размерности $k/2$.

Для удобства последующего объяснения, определим функцию активации сверточного слоя i:

$$a_{\mathrm{conv1d}}\left(\mathrm{h}^{(i)}\right) = \tanh\left(h^{(i)}\right). \quad (5)$$

Чтобы существенно сократить объем числового представления доменного имени, предлагается добавить слой понижающей дискретизации после каждого сверточного слоя. В этом случае, мы будем выбирать максимальное значение в каждой строке из предыдущего сверточного слоя [17].

$$a_{\max pool}\left(\mathrm{z}^{i}\right) = \max\left(\mathrm{z}^{i}\right), \quad (6)$$

где $\mathrm{z}^{(i)} = \mathrm{W}^{(i)}\mathrm{x}^{(i)} + \mathrm{b}^{(i)}$.

Основная идея состоит в том, что сверточный слой уже выявляет определенные характеристики, связанные с доменным именем, поэтому подробное объяснение и дальнейшая обработка большого объема данных не нужны.

Дополнительно определим слой, используемый для самонормализации сверточной сети [14]:

$$\mathrm{selu}(x;\alpha) = \lambda\begin{cases} x, & x > 0 \\ \alpha e^{x} - \alpha, & x \le 0, \end{cases} \quad (7)$$

$$a_{\mathrm{selu}}\left(\mathrm{z}^{i}\right) = \mathrm{selu}\left(\mathrm{z}^{i}\right). \quad (8)$$

В данном подходе представлена простая техника, которая обеспечивает нормализацию выходов сверточного слоя нейронной сети. Это позволяет поддерживать математическое ожидание и дисперсию выходов слоя в определенном диапазоне значений [16].

Разработаем механизм регуляризации для скрытых слоев нейронной сети, используя оператор прореживания. Пусть R – случайная дискретная величина, принимающая значения из множества $\{0,1\}$, а p – наперед заданная константа, лежащая в диапазоне [0,1]. Тогда закон распределения случайной величины R выглядит следующим образом

$$F_R = \begin{cases} p, R = 1; \\ 1 - p, R = 0, \end{cases} \tag{9}$$

а функция прореживания определяется выражением

$$\mathrm{d} = \mathrm{J}_{1,n} * \mathrm{r}_p, D_p(\mathrm{Z}) = \mathrm{Z} * \mathrm{d}^T, \tag{10}$$

где $\mathrm{r}_p \in \mathrm{R}^n$ – вектор случайных величин;

J – вектор единиц;

* - произведение Адамара.

Функция softmax применяется к выходному слою нейронной сети для обеспечения того, чтобы сумма всех вероятностей, относящихся к каждому классу из Y, составляла единицу:

$$\mathrm{softmax}(\mathrm{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{|\mathrm{x}|} e^{x_j}}, a_{\mathrm{softmax}}^{(i)} = \mathrm{softmax}\left(\mathrm{z}^{(i)}\right). \tag{11}$$

Итого, архитектура нейронной сети, применяемой для классификации символьных строк доменных имен, выглядит следующим образом:

$$\mathrm{h}^{(1)} = E_{k \to \frac{k}{2}}(\mathrm{E}),$$

$$\mathrm{h}^{(2)} = \left[a_{\mathrm{conv1d(Q,H,10)}} \circ a_{\mathrm{maxpool}}\right]\left(\mathrm{h}^{(1)}\right),$$

$$\mathrm{h}^{(3)} = \left[a_{\mathrm{conv1d(Q,H,7)}} \circ a_{\mathrm{maxpool}}\right]\left(\mathrm{h}^{(2)}\right),$$

$$\mathrm{h}^{(4)} = \left[a_{\mathrm{conv1d(Q,H,5)}} \circ a_{\mathrm{maxpool}}\right]\left(\mathrm{h}^{(3)}\right),$$

$$\mathrm{h}^{(5)} = \left[a_{\mathrm{conv1d(Q,H,3)}} \circ a_{\mathrm{maxpool}}\right]\left(\mathrm{h}^{(4)}\right),$$

$$\mathrm{z}^{(6)} = a_{\mathrm{selu}}\left(\mathrm{h}^{(5)}\right), \mathrm{z}^{(7)} = D_{0,1}\left(\mathrm{z}^{(6)}\right),$$

$$\hat{y}(\mathrm{E}) = a_{\mathrm{softmax}}\left(\mathrm{h}^{(5)} * \mathrm{z}^{(6)} * \mathrm{z}^{(7)}\right). \quad (12)$$

Стоит отметить, что классификатор $\hat{y}$ определен относительно встраивания символьного представления E вместо доменного имени X. Приведем конечный вид классификатора:

$$y^* = \hat{y}\left(\mathrm{cemb}(\mathrm{X})\right). \quad (13)$$

Для предлагаемой нейронной сети мы будем использовать перекрестную энтропию в качестве функции потерь, а в качестве оптимизационного алгоритма – Adam [24]. Алгоритм Adam основан на вычислении градиента и имеет значительно более высокую скорость сходимости по сравнению с классическим алгоритмом стохастического градиентного спуска.

Выводы:

Сформулирована постановка задачи, определены основные понятия, проанализированы существующие методы, представлена математическая модель, спроектирована архитектура нейронной сети.

2 Реализация системы распознавания DNS-туннелей

С использованием этих технологий была реализована система распознавания DNS-туннелей, которая позволяет анализировать DNS-трафик в режиме реального времени и выявлять признаки образования туннелей:

– Python - язык программирования, выбранный для реализации данной системы. Python имеет множество библиотек и фреймворков, позволяющих быстро и эффективно реализовывать машинное обучение и анализ данных [6];
– TensorFlow - фреймворк для глубокого обучения, который используется для построения сверточных нейронных сетей (CNN). TensorFlow позволяет легко и быстро создавать, обучать и развертывать модели глубокого обучения;
– Keras - открытая библиотека, написанная на языке Python и обеспечивающая взаимодействие с искусственными нейронными сетями. Дополнение к фреймворку TensorFlow;
– Pandas - библиотека для анализа и обработки данных. Pandas предоставляет удобные средства для работы с таблицами и временными рядами, что позволяет легко проводить предобработку данных;
– Scikit-learn - библиотека для машинного обучения, которая предоставляет множество алгоритмов классификации, регрессии и кластеризации. Scikit-learn используется для оценки качества моделей и выбора наилучших гиперпараметров;
– Wireshark - программа для анализа сетевых протоколов, используемая для сбора данных и анализа DNS-трафика;
– NumPy - библиотека для работы с массивами данных в Python, которая предоставляет быстрые и эффективные алгоритмы для обработки массивов;

– Matplotlib - библиотека для визуализации данных в Python, используется для отображения результатов анализа данных и оценки качества моделей;

Программа состоит из нескольких подмодулей:

– создание модели нейронной сети;

– механизм встраивания слов;

– модуль работы с тренировочными и валидационными выборками;

– модуль обучения нейронной сети;

– методы анализа эффективности;

Для реализации и обучения нейронной сети была выбрана библиотека машинного обучения TensorFlow, включающую в себя множество решений и алгоритмов.

TensorFlow предоставляет гибкий набор инструментов и возможностей для разработки различных моделей глубокого обучения. Он поддерживает создание и обучение разнообразных нейронных сетей, включая сверточные нейронные сети (CNN), рекуррентные нейронные сети (RNN) и генеративно-состязательные сети (GAN), а также позволяет легко масштабировать обучение на больших объемах данных и использовать графические процессоры (GPU) для ускорения вычислений.

Для генерации модели сверточной нейронной сети, описанной в подразделе 1.3.2, воспользуемся фреймворком TensorFlow. Часть кода представлена на рисунке 9.

```python
def create_cnn(num_classes: int = 2) -> tf.keras.Model:
    x = Input(shape=(256,), dtype="int64")
    h = Embedding(en2vec.corpus_size + 1, 128, input_length=256)(x)

    conv1 = Convolution1D(filters=256, kernel_size=10, activation="tanh")(h)
    conv2 = Convolution1D(filters=256, kernel_size=7, activation="tanh")(h)
    conv3 = Convolution1D(filters=256, kernel_size=5, activation="tanh")(h)
    conv4 = Convolution1D(filters=256, kernel_size=3, activation="tanh")(h)

    h = Concatenate()(
        [
            GlobalMaxPooling1D()(conv1),
            GlobalMaxPooling1D()(conv2),
            GlobalMaxPooling1D()(conv3),
            GlobalMaxPooling1D()(conv4),
        ]
    )

    h = Dense(1024, activation="selu", kernel_initializer="lecun_normal")(h)
    h = AlphaDropout(0.1)(h)
    h = Dense(1024, activation="selu", kernel_initializer="lecun_normal")(h)
    h = AlphaDropout(0.1)(h)

    y = Dense(num_classes, activation="softmax")(h)

    model = Model(x, y)
    model.compile(
        optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy", AUC()]
    )
    return model
```

Рисунок 9 – Создание модели CNN.

Также, из подраздела 1.3.2 следует, что на вход нейронной сети должен подаваться трансформированный вектор, полученный при встраивании слов в вектор. Реализация данной утилиты представлена на рисунке 10.

```python
class str2vec:
    """Encode string to the vector of integers using an alphabet."""

    EN_US_ALPHABET = string.printable

    TL_DOMAINS = ["tunnel.tuns.org.", "hidemyself.org.", "tunnel.example.org."]

    def __init__(self, alphabet: str, maxlen: int = 256) -> None:
        self.corpus = {ch: pos + 1 for pos, ch in enumerate(alphabet)}
        self.alphabet = alphabet
        self.maxlen = maxlen

    @property
    def corpus_size(self):
        return len(self.corpus)

    def convert(self, s: bytes) -> np.array:
        for d in self.TL_DOMAINS:
            s = s.replace(d, "")

        maxlen = min(len(s), self.maxlen)
        str2idx = np.zeros(self.maxlen, dtype="int64")

        for i in range(1, maxlen + 1):
            c = s[-i]
            str2idx[i - 1] = self.corpus[c] if c in self.corpus else 0
        return str2idx

    def decode(self, i: int) -> str:
        return self.alphabet[i - 1] if 0 < i < len(self.alphabet) else " "
```

Рисунок 10 – Утилита встраивания символов в вектор.

Для обучения нейронной сети, необходимо обработать и разделить выборку. Для этого использовалась библиотека pandas. В первую очередь нужно закодировать категориальный признак и произвести встраивание домена в вектор. Данная функция представлена на рисунке 11.

```python
def cast_dtqbc2(
    df: pd.DataFrame, binary: bool = False, pos_class: int = 0, encode=True
) -> pd.DataFrame:
    """Prepare IRDTUN-2 dataset for training a model.

    The dataset contains domain names produced by various tunneling programs,
    like `iodine`, `dnstun`, `tuns`, `dns2tcp`.

    The dataset is used for domain names text analysis.
    """
    df = df.sample(frac=1.0)
    if binary:
        df.loc[df["label"] != pos_class, "label"] = 1 - pos_class

    if encode:
        df["x_true"] = df["qname"].map(en2vec.convert)
    else:
        df["x_true"] = df["qname"]
    df["y_true"] = df["label"]

    x_true = df["x_true"].to_numpy().tolist()
    y_true = to_categorical(df["y_true"].to_numpy()).tolist()
    x_true = np.asarray(x_true, dtype="int64") if encode else np.asarray(x_true)

    return x_true, np.asarray(y_true)
```

Рисунок 11 – Преобразование выборки.

Для обучения построенной модели нейронной сети, необходимо отслеживать ее точность или же некоторую показательную величину ошибки классификации. Для этого реализован отдельный модуль, включающий в себя несколько утилит.

Метрики могут служить целевыми функциями для оптимизации моделей. Например, в процессе обучения модели можно использовать метрики для настройки гиперпараметров или выбора лучших весов модели. Они являются стандартизированным способом описания производительности моделей и обмена результатами между исследователями и практиками машинного обучения.

Эти метрики – точность (Precision), полнота (Recall) или показатель истинно положительных результатов (TPR), показатель ложных положительных результатов (FPR) и F1-оценка. Точность вычисляется как отношение правильно идентифицированных элементов, отнесенных к положительному классу, ко всем элементам, отнесенным к положительному классу, в то время как полнота вычисляется как отношение правильно

идентифицированных элементов, отнесенных к положительному классу, к общему количеству истинных положительных результатов. F1-оценка представляет собой гармоническое среднее точности и полноты. Наконец, показатель ложных положительных результатов вычисляется как отношение неправильно идентифицированных элементов, отнесенных к положительному классу, к общему числу фактических отрицательных предсказаний.

В целом, метрики в машинном обучении играют важную роль в измерении и сравнении производительности моделей, а также в поддержке принятия решений о выборе, настройке и оптимизации моделей.

Все перечисленные метрики реализованы в библиотеке sklearn. На рисунках 12-13 представлена программная реализация оценки модели во время ее обучения.

```python
def eval_class(y_pred: np.array, y_true: np.array, average: str = "binary") -> None:
    """Print metrics and confusion matrix for the given predictions."""
    print(
        f"accuracy: {accuracy_score(y_true, y_pred):9.4f} - "
        f"precision: {precision_score(y_true, y_pred, average=average):9.4f} - "
        f"recall: {recall_score(y_true, y_pred, average=average):9.4f} - ",
        f"f1: {f1_score(y_true, y_pred, average=average):9.4f}",
        end=" - ",
    )

    if average == "binary":
        tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
        print(
            f"true_positives: {tp} - true_negatives: {tn} - "
            f"false_positives: {fp} - false_negatives: {fn}"
        )
    else:
        print()
```

Рисунок 12 – Метод eval_class.

```python
def eval(
    model: tf.keras.Model,
    test_path: str,
    cast_dataset: Callable,
    binary: bool = True,
    prepare_predict: bool = True,
    prepare_skip: bool = False,
    average: str = "binary",
    root_dir: str = "",
    encode: bool = True,
    limit: Optional[int] = None,
) -> Tuple[np.array, np.array]:
    """Evaluate the model on the test dataset.

    Method prints the basic metrics of the model and returns true and predicted
    results of the test data classification.
    """
    path = f"{root_dir}/{test_path}"
    if path not in _dataset_cache:
        _dataset_cache[path] = pd.read_csv(path)

    dataset_df = _dataset_cache[path]
    dataset_df = dataset_df.iloc[:limit] if limit else dataset_df

    x_test, y_test = cast_dataset(dataset_df, binary=binary, encode=encode)

    y_prob = model.predict(x_test)
    y_pred = np.argmax(y_prob, axis=-1) if prepare_predict else y_prob

    if prepare_skip:
        return y_test, y_prob

    # Convert from categorical to numerical represetnation of the label.
    if binary:
        y_true = np.argmax(y_test, axis=-1)
        eval_class(y_pred, y_true, average=average)
    else:
        y_pred = to_categorical(y_pred)
        for cls in range(y_pred.shape[1]):
            print(f"metrics for class {cls}:")
            y_pr = y_pred[:, cls].astype("int64")
            y_true = y_test[:, cls].astype("int64")
            eval_class(y_pr, y_true)

    return y_test, y_prob
```

Рисунок 13 – Метод оценки классификации.

Непосредственно сам запуск обучения производится методом fit_multilabel, представленном на рисунках 14-15.

```python
def fit_multilabel(factory: Callable, h5_path: str) -> Callable:
    model, _ = nn.train(
        model_factory=partial(factory, num_classes=len(class_names)),
        model_h5_path=h5_path,
        train_epochs=20,
        root_dir=ROOT_DIR,
        **datasets.umudga_m.train,
    )
    return partial(metrics.eval, model, root_dir=ROOT_DIR, **datasets.umudga_m.test)
```

Рисунок 14 – Запуск обучения нейронной сети.

```python
y_true, y_pred = fit_multilabel(factory, h5_path)()
y_true, y_pred = np.argmax(y_true, axis=-1), np.argmax(y_pred, axis=-1)

conf_mat = wandb.plot.confusion_matrix(
    probs=None,
    y_true=y_true,
    preds=y_pred,
    class_names=class_names,
)

wandb.log({"conf_mat": conf_mat})
```

Рисунок 15 – Запуск обучения нейронной сети.

Для практического применения смоделированной и обученной нейронной сети, подключим модуль dnstun (URL: https://github.com/netrack/dnstun) к DNS-серверу CoreDNS. CoreDNS является одним из популярных DNS-серверов, который используется как в облачных инфраструктурах, включая Kubernetes, так и в качестве самостоятельного сервера. Модуль dnstun выполняет процесс разрешения доменных имен, включая дополнительный анализ для обнаружения туннелирования (рисунок 16).

Рисунок 16 – Схема работы модуля CoreDNS.

Часто используется метод блокировки DNS-запросов для предотвращения утечки данных из компьютерной сети. Этот метод направлен на предотвращение скрытой передачи информации путем идентификации категории доменного имени в запросе и принятия решения о блокировке DNS-запроса при помощи специально разработанного классификатора.

Применение классификатора для блокировки DNS-запросов помогает обеспечить безопасность сети и предотвратить потенциальную утечку данных или доступ к вредоносным ресурсам. Это один из инструментов, которые используются в комплексных системах безопасности для защиты компьютерных сетей и информации.

Для практической реализации блокировки доменных имен широко используются зоны с политикой ответов (RPZ). Эти зоны могут быть настроены вручную, если требуется ограничить доступ к определенным запросам, или могут быть созданы динамически, если список запрещенных зон неизвестен заранее.

Выводы:

Реализована и обучена сверточная нейронная сеть, разработана система по обнаружению DNS-туннелей с использованием нейронной сети.

Далее необходимо произвести замеры эффективности разработанной системы по обнаружению DNS-туннелей.

3 Тестирование и анализ эффективности

При создании или в процессе использования нейронной сети необходимо учитывать несколько параметров, которые оказывают влияние на ее работу. Эти параметры включают в себя количество эпох, размер пакета обучающих данных и количество итераций.

Пакет обычно представляет собой небольшой набор обучающих примеров, на основе которых модель подстраивается. Весь набор данных разделяется на несколько пакетов, которые передаются в нейронную сеть последовательно.

Итерация указывает, сколько пакетов должно пройти через модель, чтобы завершить одну эпоху. Эпоха представляет собой полный проход обучающей модели. Эпоха считается завершенной, когда модель больше не обрабатывает обучающие данные.

Обучение нейронной сети обычно проходит в два этапа: сначала на тренировочном наборе данных, затем на проверочном наборе данных. Нейронная сеть останавливает свое обучение, если не наблюдает ошибок или, когда любая ошибка нейронной сети перестает уменьшаться и при коррекции коэффициентов уже наблюдается рост ошибки [24].

Если нейронная сеть демонстрирует высокую точность только на тренировочных данных, это может указывать на переобучение. Переобучение происходит, когда нейронная сеть слишком точно подстраивается под конкретный набор тренировочных данных и не способна обобщать свои знания на новые данные. В таких случаях нейронная сеть может неправильно классифицировать новые данные, которые не были представлены во время обучения. Причины переобучения могут быть связаны с неправильной архитектурой нейронной сети, избыточным числом параметров или недостаточным объемом тренировочных данных.

Кроме переобучения, существует проблема недообучения, когда нейронная сеть не смогла полностью выучить закономерности в

тренировочных данных. Это может быть вызвано слабой моделью, сильной регуляризацией, ограниченным разнообразием данных или недостаточной длительностью тренировки. В таких случаях нейронная сеть может не распознать некоторые закономерности, которые были присутствовали в тренировочных данных, и допустить ошибки при обработке новых данных [26].

Созданная модель обучалась на выборке, состоящей из 16000 DNS-пакетов, из которых 12000 – безопасны, 4000 – небезопасны (рисунок 17). Анализ эффективности проводился на выборке из 4000 пакетов.

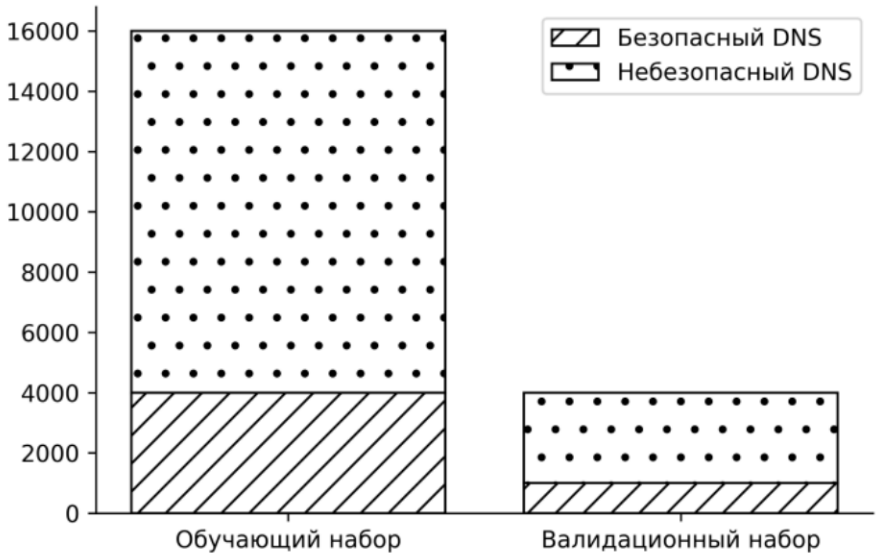

Рисунок 17 – Выборки данных.

Во время обучения, модель совершила 10 итераций. При этом уже после 4 итерации обучения ошибка нейронной сети была минимальной (рисунок 18).

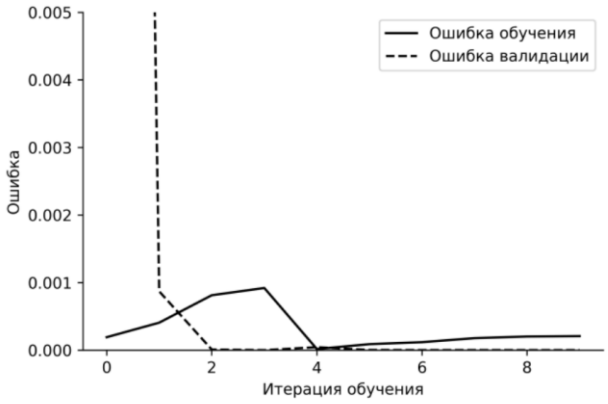

Рисунок 18 – Итерации обучения.

В результате обучения, были проанализированы метрики, перечисленные в разделе 2. ROC-кривая представлена на рисунке 19.

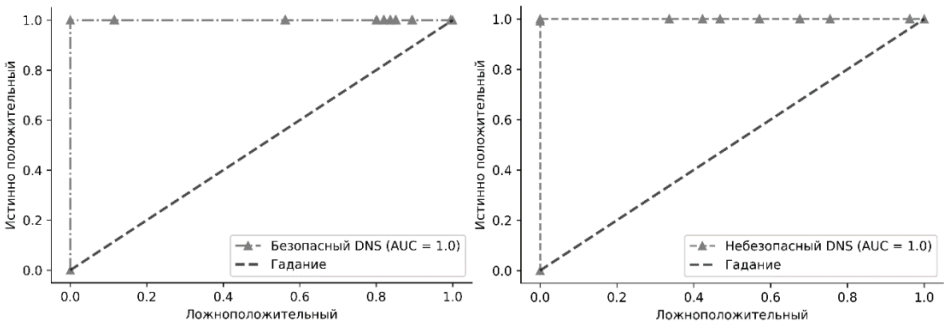

Рисунок 19 – ROC-кривая классификации DNS-запросов.

Согласно этой метрике, модель демонстрирует практически идеальную точность. Это подтверждает изначальную гипотезу о различиях между доменными именами, используемыми в туннелировании, и естественными словами, которые составляют безопасные доменные имена. В противном случае, обученная модель не смогла бы правильно классифицировать доменные имена на разные категории.

Выводы:

Система по обнаружению DNS-туннелей протестирована. Подтверждена гипотеза об эффективности сверточных нейронных сетей в задаче классификации.

Заключение

В результате данной выпускной квалификационной работы были выполнены все поставленные задачи.

Получены следующие результаты:

– проведена аналитическая работа по изучению проблемы обнаружения DNS туннелей в реальном времени. В рамках работы были рассмотрены существующие методы анализа DNS трафика, а также реализован подход к решению данной проблемы, основанный на машинном обучении;
– реализована система по обнаружению DNS-туннелей с использованием различных технологий;
– в результате тестирования разработанной системы не было обнаружено ошибок, а результаты взаимодействия с программой подтвердили правильность ее работы;

В ходе исследования предметной области были указаны материалы, позволяющие ознакомиться с теоретическими данными и реализацией сверточной нейронной сети в области обработки естественного языка.

Итак, использование сверточной нейронной сети для текстового анализа доменных имен в обнаружении DNS-туннелирования предоставляет преимущества по сравнению с применением других методов. В отличие от статистических методов, такой подход не требует полного анализа содержимого DNS-запроса, включая временные характеристики и накопление исторических данных о сети. Более того, текстовый анализ доменных имен превосходит анализ биграмм, поскольку не требует построения таблицы частот для каждого языка.

Список используемой литературы

1. Башмаков, С.А. Анализ DNS-трафика для обнаружения утечек конфиденциальной информации / С.А. Башмаков, А.А. Журчев // Информатика и её применения. – 2017. – Т.11. – №11. – С. 5-12.
2. Бубнов, Я. В. Текстовый анализ DNS-запросов для защиты компьютерных сетей от эксфильтрации данных / Я. В. Бубнов, Н. Н. Иванов // Информатика. – 2020. − Т. 17, № 3. – С. 78–86.
3. Ветров, А. А. Анализ трафика на основе DNS-запросов / А. А. Ветров // Молодой ученый. – 2017. – Т.37. – №37. – С. 225-228.
4. Гелиг А., Матвеев А. Введение в математическую теорию обучаемых распознающих систем и нейронных сетей. Учебное пособие / Аркадий Гелиг, Алексей Матвеев - Издательство СПбГУ, 2014.
5. Глушенко, В. А. Анализ DNS-трафика для выявления и классификации сетевых атак / В. А. Глушенко, О. В. Макаров // Научный журнал КубГАУ. – 2018. – Т.145. – №145. – С. 1-15.
6. Документация по языку программирования Python // Python [Электронный ресурс]: официальный сайт Python. URL: https://www.python.org/doc/.
7. Дятлов, Ю. С. Методы обнаружения и защиты от DNS-атак / Ю. С. Дятлов, С. И. Волков // Труды Института системного программирования РАН. – 2016. – Т.28. – №2. – С. 47-64.
8. Захаров, А. В. Обнаружение и анализ DNS-туннелирования / А. В. Захаров, Е. В. Соколова, Ю. Г. Шерстобитов // Информационные технологии. – 2017. – Т.23. – №10. – С. 662-671.
9. Калабухов Е.В., Недведский А.Ю., Масензов В.В., & Якубович Ф.В. Обнаружение передачи несанкционированного трафика посредством туннелирования DNS. // Журнал «Наука, образование.

10. Липеев, В. А. Обзор технологий DNS-фильтрации и их сравнительный анализ / В. А. Липеев, А. А. Литвинов // Вестник Московского университета МВД России. – 2016. – Т.4. – №4. – С. 130-138.

11. Bianco, D. (2006, May 3). A traffic-analysis approach to detecting dns tunnels.

12. Bondarenko, A. & Shtovba, S. Analysis of Existing Approaches to DNS Tunneling Detection. // 2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT). – 2019. – P. 1-6.

13. Chen, C.-H., Tseng, H.-F., & Chen, C.-Y. Network Traffic Analysis for Detecting DNS Tunneling. // 2019 2nd International Conference on Communication Engineering and Technology (ICCET). – 2019. – P. 179-182.

14. Das, A., Shen, M.Y., Shashanka, M., & Wang, J. Detection of exfiltration and tunneling over DNS. // 16th IEEE International Conference on Machine Learning and Applications. – 2017. – P. 529-534.

15. Eddison, L. Python Machine Learning: A Technical Approach To Python Machine Learning For Beginners. // CreateSpace Independent Publishing Platform. 2018. P. 237.

16. Ellens, W., Zuraniewski, P., Sperotto, A., Schotanus, H., Mandjes, M., & Meeuwissen, E. Flow-Based Detection of DNS Tunnels. // Emerging Management Mechanisms for the Future Internet. AIMS 2013. Lecture Notes in Computer Science. – 2013. – Vol. 7943. – P. 183-194.

17. Farnham, G. Detecting DNS tunneling. // SANS Institute Information Security Reading Room. – 2013. – P. 1-14.

18. Geneiatakis, D., Gritzalis, S., Kambourakis, G., & Moschos, T. Detecting DNS amplification attacks. // International Workshop on Critical Information Infrastructures Security. – 2007. – P. 60-71.

19. Qi, C., Chen, X., Xu, C., Shi, J., Liu, P.: A bigram based real time dns tunnel detection approach. Procedia Computer Science (2013)

20. Kelleher, J. D., Mac Namee, B., & D'Arcy, A. Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies. // The MIT Press. 2015. P. 441.

21. Nadler, A., Aminov, A., & Shabtai, A. Detection of malicious and low throughput data exfiltration over the DNS protocol // Computers & Security. 2019. Vol. 80. P. 36-53.

22. Saxe, J., & Berlin, K. eXpose: A character-level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys.

23. Shalev-Shwartz, S., & Ben-David, S. Understanding Machine Learning // From Theory to Algorithms. Cambridge University Press. 2014. P. 397.

24. Witten, I., Frank, E., & Hall, M. Data Mining: Practical Machine Learning Tools and Techniques. // Morgan Kaufmann. 2011. P. 654.

25. Yu, B., Smith, L., Threefoot, M., & Olumofin, F. Behavior analysis based DNS tunneling detection and classification with big data technologies // IoTBD. 2016. P. 175-182.

26. Zhang, J., Yang, L., Yu, S., & Ma, J. A DNS tunneling detection method based on deep learning models to prevent data exfiltration // International Conference on Network and System Security. 2019. P. 520-535.