**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc9464621)

[ПОСТАНОВКА ЗАДАЧИ 5](#_Toc9464622)

[АКТУАЛЬНОСТЬ ИССЛЕДОВАНИЯ 7](#_Toc9464623)

[1. ПОДХОД К РЕШЕНИЮ ЗАДАЧИ 13](#_Toc9464624)

[1.1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ 13](#_Toc9464625)

[1.2 ВЫБОР МЕТОДА ОБНАРУЖЕНИЯ НЕИЗВЕСТНЫХ УГРОЗ 15](#_Toc9464626)

[1.3 ЦИФРОВОЙ ОТПЕЧАТОК ЗАПРОСА 21](#_Toc9464627)

[1.4 ОБЕСПЕЧЕНИЕ ПРОСТОЙ ИНТЕГРАЦИИ С РАБОТАЮЩИМ ВЕБ-ПРИЛОЖЕНИЕМ 27](#_Toc9464628)

[1.5 ВЫБОР СИСТЕМЫ КОНТЕЙНЕРИЗАЦИИ МЕЖСЕТЕВОГО ЭКРАНА УРОВНЯ ВЕБ-ПРИЛОЖЕНИЯ 29](#_Toc9464629)

[1.6 ХРАНЕНИЕ ДАННЫХ О ЗАПРОСАХ 33](#_Toc9464630)

[1.7 ТЕХНОЛОГИЧЕСКОЕ ИСПОЛНЕНИЕ МЕЖСЕТЕВОГО ЭКРАНА УРОВНЯ ВЕБ-ПРИЛОЖЕНИЯ 36](#_Toc9464631)

[1.8 ПОДГОТОВКА И ОБРАБОТКА ДАННЫХ ДЛЯ АЛГОРИТМА ОПРЕДЕЛЕНИЯ ВРЕДОНОСНОСТИ ЗАПРОСА 41](#_Toc9464632)

[2. ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ. ЗАКЛЮЧЕНИЕ 44](#_Toc9464633)

[2.1 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ 44](#_Toc9464634)

[2.2 ДАЛЬНЕЙШЕЕ РАЗВИТИЕ 47](#_Toc9464635)

[3. СПИСОК ЛИТЕРАТУРЫ 49](#_Toc9464636)

[ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД 52](#_Toc9464637)

# ВВЕДЕНИЕ

## ПОСТАНОВКА ЗАДАЧИ

Современные веб-приложения подвергаются атакам со стороны злоумышленников. Существует по меньшей мере три разных типа атак.

Атаки первого типа, распределённый отказ в обслуживании (Distributes Denial of Service, DDoS), являются самыми разрушительными. Они направлены на то, чтобы нарушить обычную работу веб-приложения, чтобы веб-приложение перестало обрабатывать запросы других пользователей. Атаки данного типа чаще всего отражаются программными или аппаратными межсетевыми экранами на четвёртом, транспортном уровне модели OSI [1]. Такие межсетевые экраны фильтруют IP- или MAC- адреса, обнаруживают трафик, нетипичный для данного веб-приложения и т.д. В связи с тем, что такие межсетевые экраны не имеют логической связи с самими веб-приложениями, существует несколько проблем. Во-первых, блокировка выглядит чужеродно по отношению к самому веб-приложению и его среде. Пользователь работает в определённом интерфейсе веб-приложения, а в результате ложноположительной блокировки он может увидеть пустую страницу с сообщением браузера о том, что запрашиваемая страница недоступна. Во-вторых, пользователь не может сообщить через привычный канал взаимодействия о том, что произошла ложноположительная блокировка. В результате каждая такая блокировка приносит репутационные потери веб-приложению и его владельцу. В-третьих, обработка большого количества IPv6 адресов в быстрой оперативной памяти представляет собой нерешённую проблему.

Второй тип атак направлен на взлом самого веб-приложения. Это может быть инъекция кода (SQL injection), межсайтовый скриптинг (Cross-Site Scripting, XSS), попытка подбора пароля полным перебором или по словарю (Brute-force) и другие атаки, направленные на хищение данных. Атаки данного типа отражаются межсетевыми экранами уровня веб-приложения на основе правил [2][3]. Набор правил содержит основной шаблонный код и техники, которые позволяют внедрять вредоносный код или проводить межсайтовый скриптинг. Кроме того, в правилах может быть описана частота использования точек доступа API для предотвращения атак типа подбор пароля полным перебором. Однако, даже с обновляемой базой правил, она остаётся статичной и новые или слегка изменённые методы атак успешно обходят такие межсетевые экраны [4].

Третий тип атак – это интернет-мошенничество с использованием веб-приложения. Одни из самых главных ценностей веб-приложения – это данные и пользователи. Формально, при использовании интернет-мошенничества нормальная работа веб-приложения не нарушается, в связи с этим обнаружить атаку непросто. Вредоносные пользователи или боты могут вести себя очень похоже на обычных пользователей, но в их поведении так или иначе будут наблюдаться некоторые странности, нетипичные шаблоны поведения [5][6].

Задача данного исследования заключается в создании способа обнаружения атак третьего типа с некоторой вероятностью.

## АКТУАЛЬНОСТЬ ИССЛЕДОВАНИЯ

Согласно исследованию Лаборатории Касперского о DDoS-атаках в четвёртом квартале 2018 года[[1]](#footnote-1), наибольшее количество атак составляют именно атаки первого типа, как показано на рисунке 1.

A close up of a logo

Description automatically generated

Рисунок 1 - Распределение атак по типам в четвёртом квартале 2018 года

И только 2,2% от всех атак составляют атаки по протоколу HTTP на уровне приложения, то есть атаки второго и третьего типа.

Однако, согласно исследованию Positive Technologies за 2018 год[[2]](#footnote-2), веб-приложения по-прежнему уязвимы к атакам разных типов на уровне приложения.

Кроме того, согласно исследованию Anti-Malware за октябрь 2017 года[[3]](#footnote-3), на российском рынке межсетевых экранов представлены такие продукты как PT Application Firewall, Imperva Web Application Firewall, F5 Web Application Firewall. В данных продуктах заявлено наличие как средства борьбы с атаками типа отказ в обслуживании (первый тип), так и средства по борьбе со взломами (второй тип), которые включают в себя уже известные правила для атак и механизмы создания новых правил для новых атак. Кроме того, существует свободный продукт ModSecurity, который имеет большую базу правил и позволяет создавать собственные правила определения того, на сколько запрос является вредоносным.

Помимо коробочных продуктов, согласно исследованию межсетевых экранов от Gartner за 2018 год[[4]](#footnote-4), начинают набирать популярность межсетевые экраны как сервисы, как показано на рисунке 2.

К таким продуктам относятся Cloudflare Cloud Web Application Firewall, Amazon AWS WAF, Microsoft Azure Application Gateway с функцией Web Application Firewall.

A screenshot of a cell phone

Description automatically generated

Рисунок 2 - Magic Quadrant межсетевых экранов от Gartner за 2018 год

Таким образом, современные межсетевые экраны умеют бороться с атаками первых двух типов из описанных выше.

Однако, ни в одном из перечисленных продуктов не заявлена интеграция с самим веб-приложением. Чаще всего, перечисленные межсетевые экраны устанавливаются вразрез между интернетом и веб-приложением, но существуют варианты зеркалирования траффика канала интернет-веб-приложение к межсетевому экрану. В результате этого межсетевые экраны вынуждены предугадывать действия пользователя и определять их подозрительность без знания конкретных алгоритмов работы самого веб-приложения. Такой подход не всегда является эффективным, особенно с учётом изменений в логике работы веб-приложения. Сравнение межсетевых экранов приведено на рисунке 3.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Межсетевой экран /  Возможности | PT Application Firewall | Imperva Web Application Firewall | F5 Web Application Firewall | ModSecurity | Cloudflare Cloud Web Application Firewall | Amazon AWS WAF | Microsoft Azure Application Gateway с WAF |
| Защита от DDoS | Да | Да | Да | Нет | Да | Да | Да |
| Механизм правил | Да | Да | Да | Да | Да | Да | Да |
| Интеграция с веб-приложением | Нет | Нет | Нет | Да | Нет | Нет | Нет |
| Механизмы машинного обучения | Да | Нет | Нет | Нет | Да | Да | Да |

Рисунок 3 – Сравнение имеющихся на российском рынке межсетевых экранов уровня веб-приложения

С учётом замеченного выше, а также информации из рисунка 3, создание межсетевого экрана уровня веб-приложения как сервиса, с интеграцией с самим веб-приложением является актуальным исследованием.

# 1. ПОДХОД К РЕШЕНИЮ ЗАДАЧИ

## 1.1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

В набор технических характеристик описанных выше межсетевых экранов входят следующие функции:

* Установка как вразрез каналу интернет-веб-приложение, так и параллельно ему
* Поставка как в виде аппаратного комплекса, так и в виде виртуальной машины и подпрограммы для облачного сервиса
* Наличие базы данных вредоносных IP-адресов и URL-адресов
* Обнаружение ботов на основе последовательности переходов внутри веб-приложения
* Обнаружение атаки типа полный перебор
* Разграничение доступа к ресурсам на основе идентификаторов пользователей и ресурсов
* Безсигнатурные методы защиты от уязвимостей веб-приложений
* Возможность создания пользовательских правил

Таким образом, перечисленные выше функции уже реализованы, и для создаваемого межсетевого экрана их реализация не является обязательной.

С другой стороны, в межсетевых экранах отсутствуют возможность по интеграции с самим веб-приложением, в том числе для корректного сопоставления запросов, выполненных конкретным пользователем. Это также не позволяет накапливать репутацию по конкретному пользователю.

Другой проблемой является способность перечисленных выше межсетевых экранов противостоять новым и изменяющимся угрозам. Даже в случае модели распространения межсетевого экрана как сервиса, проходит некоторое время от начала атаки с новой угрозой, до момента её обнаружения сотрудниками производителя и добавления сигнатуры. Для решения этой проблемы рационально использовать технологии машинного обучения, которые по имеющемуся набору данных позволят классифицировать запрос с некоторой вероятностью на подозрительный или не подозрительный.

Суммируя сказанное выше, создаваемый межсетевой экран должен обладать следующими характеристиками:

* Возможностью работы в имеющейся инфраструктуре веб-приложения или лёгкой интеграцией с ней
* Интеграцией с самим веб-приложением. Окончательное решение о вредоносности/не вредоносности запроса принимает веб-приложение с учётом бизнес-требований
* Возможностью обнаруживать заранее неизвестные вредоносные запросы на основе имеющейся истории запросов

## 1.2 ВЫБОР МЕТОДА ОБНАРУЖЕНИЯ НЕИЗВЕСТНЫХ УГРОЗ

Согласно требованиям, определённым ранее, сигнатурный метод обнаружения неизвестных угроз не подходит для межсетевого экрана. Даже с накоплением качественной базы признаков вредоносного запроса, каждый новый вредоносный запрос, непохожий на предыдущие, легко сведёт защиту на нет. Кроме того, даже с учётом быстрого обнаружения новых вредоносных запросов со стороны разработчика межсетевого экрана, распространение сигнатур на всех клиентов является медленным, в результате клиенты всё равно могут пострадать от атаки.

Одним из вариантов решения подобной задачи является экспертная система, в которой предопределена база знаний, характеризующая и описывающая признаки, встречающиеся у вредоносных запросов. Такая система в целом не будет страдать от неактуальности базы данных, при условии её обновления с обычной частотой. Однако, применение такой системы будет проблематично в силу того, что часть запросов может быть ошибочно отнесена к вредоносным, и система не сможет учесть данный опыт для своего совершенствования. Кроме того, экспертная система всё же не лишена главного недостатка сигнатурного метода, и требует периодического обновления базы данных.

Таким образом, становится понятно, что единственный возможный метод для обнаружения неизвестных угроз в межсетевом экране уровня веб-приложения – это метод, основанный на машинном обучении. Согласно описанию свойств методов машинного обучения, они решают задачу не напрямую, а обучаются на некотором наборе данных, и способны с некоторой вероятностью решить задачу как на известных, так и на заранее неизвестных наборах данных. Далее приводятся разные алгоритмы машинного обучения, их плюсы и минусы с учётом текущей задачи разработки межсетевого экрана [7]. Выбор метода машинного обучения и правильное его применение являются ключевой задачей данного исследования, от метода зависит успешность исследования и разработки.

Искусственная нейронная сеть представляет собой программное воплощение биологических нейронных сетей – нервных клеток организма. При этом, её структура и методы функционирования схожи с такими же в биологических нейронных сетях [8]. Нейронные сети состоят из множества нейронов, находящихся в разных слоях, отличающихся по назначению. Процесс обучения искусственной нейронной сети сводится к подбору коэффициентов связи между нейронами.

Плюсы искусственной нейронной сети заключаются в практически неограниченной способности к выделению значимых признаков и свойств, которые могут быть неочевидными. Такие нейронные сети достаточно универсальны, могут использоваться для решения различных задач.

Минусами является невозможность точного определения, почему нейронная сеть сделала именно такое заключение об объекте, достаточная сложность обучения, архитектуры и использования искусственной нейронной сети в целом.

Линейная и логистическая регрессия являются методами, при помощи которых устанавливается зависимость некоторой переменной (вероятности вредоносности запроса) от некоторого множества факторов (характеристик запроса). Эти методы характеризуются некоторой прямой, отображающей зависимость между указанными выше переменными. Для обучения модели необходимо подобрать такие коэффициенты при факторах, при которых значение некоторой переменной будет ближе всего к корректному на всём протяжении прямой.

Плюсы линейной и логистической регрессии заключаются в их простоте и лёгкости для понимания и обучения.

С другой стороны, чаще всего предметная область является сложной, в результате некоторая переменная не является линейно или логистически зависимой от всех параметров, и модель в таком случае будет выдавать много ошибочных результатов.

Наивный байесовский классификатор позволяет проводить классификацию при известных значениях вероятностей принадлежности к определённым классам. Таким образом, если в совокупности вероятностей по признакам объект скорее всего относится к определённому классу, классификатор выдаст соответствующее решение.

Плюсы наивного байесовского классификатора в его простоте, в скорости обучения.

Минусы заключаются в том, что набор признаков должен быть как можно более независимым, что не всегда удаётся определить эксперту.

Метод k-ближайших соседей позволяет проводить как классификацию, так и регрессию объектов. По совокупности признаков и соседям относительно этих признаков (различия между признаками несущественные) алгоритм относит объект к одному из классов. Признаки при этом обычно нормируются до значений от нуля до единицы, определяется некоторая функция определения дистанции, чаще всего расстояние между точками в N-мерном пространстве. Кроме того, в качестве одного из улучшений алгоритма можно ввести веса для признаков, чтобы можно было выделять более значимые признаки.

Плюс метода заключается в скорости его работы, потому что, например, определение расстояния до точки в пространстве является быстрым алгоритмом.

Минусом является тот факт, что алгоритм k-ближайших соседей начинает работать хуже, когда размерность пространства N сильно разрастается.

Деревья решений позволяют также проводить как классификацию, так и регрессию объектов. Они представляют собой деревья с точки зрения структуры данных, в которых в каждом узле хранится текущий проверяемый признак, а рёбра проводят к нужному следующему узлу в зависимости от значения признака.

Плюсы деревьев решений связаны с тем, что такая модель легко может интерпретироваться человеком, человек в дальнейшем может вносить правки для её улучшения. Кроме того, деревья решений не предъявляют особых требований к данным, нет необходимости нормировать входные данные.

Главный минус деревьев решений состоит в том, что они могут переобучаться. В результате глубина дерева неконтролируемо вырастает, а прогноз становится точным только для обучающей выборки, новые объекты классифицируются или регрессируют значительно хуже.

Согласно исследованиям [9][10], применение таких методов машинного обучения как метод k-ближайших соседей, метод классификации на основе последовательной минимальной оптимизации, метод деревьев решений, построенный по алгоритму C4.5 или CART, к задаче построения межсетевого экрана уровня веб-приложения является возможным, и, кроме того, оно оказывается достаточно эффективным.

Однако, как показывается в исследовании [4], такие методы обхода межсетевого экрана уровня веб-приложения на основе методов машинного обучения как замена проверяемых по словарю ключевых слов на их аналоги без изменения семантики, использование особых строковых последовательностей, интерпретация которых изменяет семантику выражения, переобучение обученной модели межсетевого экрана на новые вредоносные запросы с последующим вымыванием знаний о старых вредоносных запросах, тоже могут быть достаточно эффективными.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алгоритм / Свойство | Искусственная нейронная сеть | Линейная и логистическая регрессия | Наивный байесовский классификатор | Метод k-ближайших соседей | Деревья решений |
| Простота использования | Нет | Да | Да | Да | Да |
| Простота интерпретации результата | Нет | Да | Да | Да | Да |
| Низкие требования к данным для обучения | Да | Нет | Нет | Да | Да |
| Высокая скорость работы | Нет | Да | Да | Да | Да |
| Работа с большим количеством признаков | Да | Да | Да | Нет | Да |
| Низкая вероятность переобучения | Нет | Да | Да | Да | Нет |

Рисунок 4 – Сравнение методов машинного обучения

С учётом всех приведённых на рисунке 4 плюсов и минусов алгоритмов машинного обучения, для задачи создания межсетевого экрана уровня веб-приложения лучше других подходит алгоритм деревьев решений. Таким образом, в дальнейшем он будет использован для задачи регрессии при построении программы.

Задача защиты от атак на межсетевые экраны должна решаться параллельно с созданием самих межсетевых экранов с учётом особенностей их работы. Такая задача не входит в данное исследование, она может быть рассмотрена позднее в продолжении исследования.

## 1.3 ЦИФРОВОЙ ОТПЕЧАТОК ЗАПРОСА

Метод для определения вредоносности запроса был выбран ранее, теперь необходимо подготовить данные для этого метода.

Очевидно, что минимальной единицей взаимодействия клиента и веб-приложения является веб-запрос, или транзакция. В данном исследовании будут рассматриваться классические HTTP/HTTPS запросы. Другие протоколы поверх TCP/UDP, такие, как WebSocket, FTP, SMTP вполне могут быть рассмотрены с учётом принципов, изложенных в данном исследовании. Для простоты исследования будет использоваться гипертекстовый протокол HTTP/1.1, определённый в стандарте RFC 7230[[5]](#footnote-5) с примечаниями об использовании протокола защиты транспортного уровня TLS 1.3, определённого в стандарте RFC 8446[[6]](#footnote-6). Протоколы HTTP/2 (RFC 7540[[7]](#footnote-7)) и HTTP/3 (черновик стандарта №20 от 23 апреля 2019 года[[8]](#footnote-8)) являются бинарными протоколами, что усложняет их анализ, но, в сущности, не является ограничением при использовании описываемого здесь метода.

В рамках данного исследования все веб-запросы при использовании TLS 1.3 терминируются до межсетевого экрана уровня веб-приложения, однако этот межсетевой экран полностью проинформирован относительно всего процесса взаимодействия в рамках протокола TLS 1.3.

Кроме того, межсетевой экран уровня веб-приложения может быть проинформирован относительно процесса взаимодействия операционной системы и веб-клиента по стеку протокола TCP/IP.

Также, для простоты примем, что для каждого запроса можно однозначно установить его авторство, то есть автор запроса является зарегистрированным пользователем в веб-приложении, и каждый запрос однозначно идентифицируется как принадлежащий одному из пользователей. Несмотря на то, что это не всегда так, методы, описанные ниже, могут с достаточно высокой вероятностью установить принадлежность запроса. Все описанные методы применимы для браузера Google Chrome версии 74 с настройками по-умолчанию без дополнительных действий со стороны клиента. Очевидно, что, если веб-приложение является только JSON-API без клиентской части, то набор характеристик значительно сокращается.

Цифровые отпечатки запроса, или, расширенно, цифровые отпечатки устройства представляют собой набор характеристик запроса, который позволяет идентифицировать запрос или устройство, и с достаточно высокой вероятностью отличить один запрос или устройство от другого. Для алгоритма определения вредоносности запроса необходим набор этих характеристик.

Начиная с протоколов более низкого уровня, доступны как минимум следующие характеристики запроса или устройства [11][12]:

* Тип канала подключения к интернету, определяемый по некоторым константам в алгоритмах работы TCP/IP, по самим алгоритмам TCP/IP (алгоритм установки соединения, размер окна, алгоритм увеличения окна, количество перезапрашиваемых пакетов, MTU, количество промежуточных точек обмена траффиком между клиентом и сервером) [13]. Все перечисленные характеристики в совокупности позволяют с большой долей вероятности установить, является ли соединение проводным или беспроводным. Кроме того, можно узнать и особенности программной реализации протокола, и, как следствие, операционную систему клиента
* IPv4 или IPv6 адрес и порт. Несмотря на то, что такой адрес может принадлежать множеству клиентов за счёт использования технологии NAT со стороны интернет-провайдера, это одна из основных характеристик запроса [14][15]. По IP-адресу можно установить страну и город пользователя, интернет-провайдера
* Используемый клиентом DNS-сервер. Получить эту характеристику довольно нетривиально, но, тем не менее, возможно. Для этого необходимо создать свой собственный DNS-сервер, а браузеру отдать для загрузки URL с неизвестным доменом. Браузер попытается определить IP-адрес этого URL, для этого он обратится к DNS-серверу интернет-провайдера, а тот в свою очередь может обратиться к созданному собственному DNS-серверу. Характеристика позволяет точнее определить интернет-провайдера
* Используемые алгоритмы шифрования, их порядок и предпочтительный алгоритм для установки соединения по протоколу TLS. Набор алгоритмов шифрования обновляется с некоторой периодичностью, уязвимые и потенциально уязвимые алгоритмы удаляются, добавляются более совершенные алгоритмы. С помощью данной характеристики можно определить версию веб-браузера, операционную систему и версию операционной системы
* Шрифты веб-браузера и то, как они отображаются на странице. Размер символов, их количество, форма, сглаживание и другие характеристики могут сильно отличаться как в разных веб-браузерах, так и в одних и тех же браузерах, но в разных операционных системах [16]. По рендерингу шрифтов можно с некоторой точностью определить браузер и операционную систему, в которой он работает
* Особенности выполнения JavaScript-кода браузером. С учётом того, что существует три основных движка для выполнения JavaScript-кода (Chrome V8, SpiderMonkey и Chakra), каждый из этих движков обладает некоторыми специфичными особенностями. Какие-то движки лучше используют JIT, какие-то в целом работают медленнее. На основе некоторых тестов можно с некоторой точностью определить движок исполнения JavaScript-кода
* Заголовки браузера, в частности User-Agent. Этот заголовок запроса создан для того, чтобы веб-сервер знал, с какой программы к нему обращается клиент. Позволяет однозначно определить веб-браузер, его версию, а также операционную систему
* Информация о браузере, предоставляемая JavaScript API. По спецификации языка JavaScript, браузер раскрывает некоторые характеристики, которые помогают веб-страницам отображаться корректно. К такой информации относятся: разрешение экрана, физические размеры экрана в дюймах, глубина цвета, текущая ориентация страницы (портретная или альбомная), дата и время операционной системы с часовым поясом, формат отображения даты и времени операционной системы, язык и локализация, аппаратная платформа (информация о видеоадаптере, который производит рендеринг), размер оперативной памяти, количество логических процессоров, информация о батарее (только в Google Chrome – процент заряда, время заряда/разряда, заряжается ли в данный момент), установленные плагины веб-браузера (для чтения PDF, воспроизведения Flash и т.д.), и другая информация [17]. По этим характеристикам в целом можно узнать о физическом устройстве, с которого осуществляется выход в интернет, о текущем географическом местоположении клиента
* Поддерживаемые JavaScript, HTML и CSS API. Браузеры реализуют спецификации API постепенно, от версии к версии добавляя новые возможности. По тестам на наличие некоторых возможностей в текущем веб-браузере можно очень точно определить текущую версию веб-браузера
* Рендеринг и отображение Canvas. Как и в характеристике со шрифтами, веб-браузеры по-разному отображают изображения на виртуальных холстах из-за особенностей как самого веб-браузера, его движка построения и сжатия изображений, так и аппаратного обеспечения [18]. Эта характеристика позволяет довольно точно определять как веб-браузер, так и операционную систему, на которой он запущен
* Рендеринг и отображение WebGL. WebGL API позволяет веб-браузеру строить 3D-объекты с использованием видеоадаптера. Как и в характеристике с Canvas, разные видеоадаптеры и веб-браузеры производят различные объекты, по которым их можно различать [19]. Характеристика также позволяет определить как веб-браузер, так и видеоадаптер, с которым веб-браузер работает в данный момент
* Определение входа в некоторые социальные сети. Метод основан на том, что ссылки на некоторые страницы социальных сетей доступны только пользователям, выполнившим вход. За счёт этого браузер клиента пытается выполнить такое действие, результат действия показывает, есть ли куки от нужной социальной сети. Характеристика определяет, в каких социальных сетях клиент выполнил вход, потенциально определяет его принадлежность к некоторым социальным группам
* Установленные и настроенные расширения по влиянию на веб-страницы. При известных данных о том, что некоторое веб-браузерное расширение определённым образом влияет на веб-страницы, можно делать вывод о наличии установленного и включенного расширения. Для примера хорошо подходит блокировщик рекламы, имеющий в своих настройках некоторое количество списков рекламы для блокирования [20]. На основании блокирования элементов из разных списков можно сделать вывод о том, какие конкретно списки настроены в текущем веб-браузере. Характеристика позволяет потенциально определить принадлежность клиента к некоторой языковой группе

С учётом всех перечисленных возможностей для сбора характеристик веб-запроса, составить его цифровой отпечаток не составляет существенной проблемы [21].

В данном исследовании будут применяться следующие характеристики веб-запросов для определения их вредоносности:

* IP-адрес
* Заголовок User-Agent
* Идентификатор пользователя, выполнившего запрос

Перечисленных данных достаточно для проверки и доказательства работоспособности такого межсетевого экрана уровня веб-приложения.

## 1.4 ОБЕСПЕЧЕНИЕ ПРОСТОЙ ИНТЕГРАЦИИ С РАБОТАЮЩИМ ВЕБ-ПРИЛОЖЕНИЕМ

Согласно современным практикам создания веб-приложений, например, The Twelve-Factor App[[9]](#footnote-9), веб-приложения должны выполняться в изолированной и самодостаточной среде, которую чаще всего называют контейнером, который в свою очередь создаётся на основе образа. Образ – это некий набор данных, который может включать в себя операционную систему, сторонние программные библиотеки, данные и саму программу в скомпилированном или исходном виде. Образ после создания не изменяется, что гарантирует, что приложение будет работать всегда в одном окружении. На основе образа создаётся контейнер, в котором и происходит выполнение программы. Контейнеры, таким образом, являются единицей исполнения – их можно создавать (что равносильно запуску), останавливать, удалять [22]. Кроме того, согласно The Twelve-Factor App, приложения должны взаимодействовать при помощи унифицированного интерфейса. Для UNIX-подобных операционных систем таким интерфейсом является сокет, что для веб-приложения выражается номером порта (возможно, с IP-адресом, хостом, сетевым протоколом и т.д.). Таким образом, взаимодействие приложений, работающих в виде контейнеров, осуществляется при помощи сети: одно из веб-приложений знает адрес другого, и совершает обращение при помощи HTTP- или другого типа запроса к другому веб-приложению. Работу по корректному определению адресов и распределению запросов выполняет исполняющая среда, которая и запустила контейнеры или связанна с ними.

В связи с этим, возможны две ситуации:

1. Текущая инфраструктура веб-приложения не использует средств контейнеризации. В таком случае для интеграции межсетевого экрана уровня веб-приложения достаточно в инфраструктуре развернуть систему контейнеризации, и запустить в ней межсетевой экран. Веб-приложение будет взаимодействовать с ним через порт.
2. Текущая инфраструктура веб-приложения уже использует средства контейнеризации. В таком случае достаточно запустить контейнер из образа межсетевого экрана уровня веб-приложения и обращаться к нему через порт.

Таким образом, в любом случае интеграция текущего веб-приложения с межсетевым экраном уровня веб-приложения, описанным в данном исследовании, является процессом с небольшим временем завершения. Это является одной из характеристик межсетевого экрана, и она выполняется за счёт системы контейнеризации приложений.

## 1.5 ВЫБОР СИСТЕМЫ КОНТЕЙНЕРИЗАЦИИ МЕЖСЕТЕВОГО ЭКРАНА УРОВНЯ ВЕБ-ПРИЛОЖЕНИЯ

Различные системы контейнеризации приложений стали появляться сравнительно давно. Например, ещё в 2008 году, до массового наступления эпохи контейнеризации, появилась система LXC (Linux Container)[[10]](#footnote-10). Эта система является комбинацией нескольких программ и API из ядра Linux, таких как cgroups, которые позволяют ограничивать выделение ресурсам определённым процессам. В связи с этим, LXC может работать только в операционных системах на базе ядра Linux, в которых есть все необходимые компоненты.

Позднее, в 2013 году, появилась программа Docker[[11]](#footnote-11), которая была основана также на cgroups, kernel namespaces, однако имела более удобный интерфейс, хранилище образов. Вкупе с тем, что технология появилась в очень удачный момент широкого распространения микросервисов и схожих принципов, она стала стандартом де-факто как система контейнеризации. Кроме непосредственно программы для контейнеризации, компания-разработчик предложила также решения для запуска одной программы как набора нескольких контейнеров Docker Compose, а также кластеризацию контейнеров на несколько серверов Docker Swarm. Позднее, произошло выделение программы Moby[[12]](#footnote-12) как ядра Docker в отдельный свободно развиваемый проект с коммерциализацией части уже разработанных технологий.

Кроме того, существуют и альтернативные системы контейнеризации. Например, rkt[[13]](#footnote-13), от CoreOS, в дальнейшем RedHat, в дальнейшем IBM. Программа предоставляет собой интерфейс, очень похожий на Docker. Из-за несущественных отличий от Docker не имеет широкого распространения и фактически не развивается. Также, существует ориентированный на крупный бизнес проект Singularity[[14]](#footnote-14), который также имеет похожий интерфейс, и который также не используется большинством разработчиков.

В связи с тем, что контейнеры в целом стали играть существенную роль в жизненном цикле многих приложений, в частности, веб-приложений, было бы неразумно зависеть от конкретной реализации контейнеров и образов в одной из выбранных программ. Для преодоления этой проблемы инициативная группа во главе с Linux Foundation создала проект Open Container Initiative (OCI)[[15]](#footnote-15). Проект был запущен в 2015 году с поддержкой компаний Docker и CoreOS.

В рамках проекта были разработаны спецификации образов и исполнения образов. Спецификация образов детально описывает, что образ состоит из манифеста, индекса (для перечисления нескольких манифестов), множества слоёв файловой системы и конфигурации. На основе имеющейся спецификации можно создавать свои собственные образы, которые можно будет единообразно интерпретировать и запускать. Спецификация исполнения образов детально описывает, как должна быть сконфигурирована среда в разных операционных системах, как она должна взаимодействовать с файловой системой, какой жизненный цикл у контейнеров и как проводить контейнеры через него. Кроме того, существует спецификация распространения образов, которая описывает то, как образы должны сохраняться, храниться и скачиваться. Кроме того, следует отметить, что в рамках проекта была также разработана программа runc, которая представляет собой референсную реализацию создания образов и управления жизненным циклом контейнеров. На ней можно проверить свою реализацию, насколько точно она соответствует спецификации.

В целом проект Open Container Initiative открыл путь к существованию таких программ как kubernetes[[16]](#footnote-16), которая позволяет развёртывать контейнеризированные приложения в кластерах. При этом, среда исполнения контейнеров, как и сами контейнеры, могут быть совершенно разными.

В силу всех перечисленных выше причин, для создания образов межсетевого экрана уровня веб-приложения выбрана программа Docker как средство, которое соответствует стандарту Open Container Initiative и широко используется в сообществе. Системы исполнения контейнеров в данном случае не относится к предмету исследования, и её выбор возлагается на пользователей данного контейнера.

Для выбранной программы Docker были разработаны скрипты (файлы Dockerfile), которые позволяют собрать контейнер с нужными данными. Кроме того, был разработан скрипт для запуска нескольких контейнеров Docker (файл docker-compose.yml) с нужными настройками, представленный на рисунке 5.

A screenshot of a cell phone

Description automatically generated

Рисунок 5 – docker-compose.yml файл, используемый в межсетевом экране

## 1.6 ХРАНЕНИЕ ДАННЫХ О ЗАПРОСАХ

Для эффективной реализации межсетевого экрана уровня веб-приложения необходимо сохранять историю запросов пользователя. Это позволит выявлять аномалии в новых запросах и реагировать на них.

Очевидно, что количество запросов к веб-приложению может быть очень большим за малый промежуток времени. Для таких случаев лучшим решением является база данных на основе NoSQL-технологий [23]. Ключевыми требованиями к ней будут следующие:

* Горизонтальная масштабируемость на запись и чтение данных. Это необходимо для обработки большого числа запросов в малый промежуток времени
* Распределённое хранилище данных. Из-за высоких объёмов данные невозможно будет хранить в рамках одного сервера
* Быстрый доступ к данным, относящимся к конкретному пользователю. Необходимо для принятия решения относительно вредоносности запроса конкретного пользователя

С учётом перечисленных требований, для хранилища данных хорошо подходит система управления базами данных Apache Cassandra[[17]](#footnote-17). Она является распределённой, данные хранятся в виде столбцов. Таким образом, можно удовлетворить все требования, предъявляемые к базе данных. С другой стороны, Apache Cassandra является сравнительно новой технологией, и её развёртывание и поддержка могут быть связаны с дополнительными затратами времени из-за неподготовленности инженеров.

Для подготовки работающего прототипа была выбрана классическая реляционная система управления базами данных PostgreSQL[[18]](#footnote-18). Это хорошо проработанная база данных, которая вместе с тем является свободной и бесплатной. PostgreSQL является одной из лидирующих бесплатных систем хранения данных, в том числе для веб-приложений. Однако, межсетевой экран уровня веб-приложения был создан с расчётом на то, что система хранения данных может быть легко заменена с PostgreSQL на Apache Cassandra без изменения бизнес-логики приложения.

Для нужд межсетевого экрана уровня веб-приложения в сервере баз данных была создана база данных «kwaf» с одной таблицей «events», схема которой представлена на рисунке 6.

A screenshot of a cell phone

Description automatically generated

Рисунок 6 - Структура базы данных в приложении

Описание некоторых столбцов из таблицы «events»:

* «user\_id»: идентификатор пользователя, выполнившего запрос. Данный идентификатор имеет формат Universally unique identifier (UUID) v4. Само веб-приложение должно иметь точно такие же идентификаторы пользователей, или может обеспечивать взаимно-однозначное отображение идентификаторов пользователей веб-приложения на пользователей межсетевого экрана уровня веб-приложения
* «user\_agent»: заголовок User-Agent браузера, выполнившего запрос. Имеет формат строки
* «host»: IPv4 или IPv6 адрес источника, выполнившего запрос. Имеет формат строки
* «rating»: число с плавающей точкой от 0 до 100 включительно, которое определяет вероятность вредоносности запроса. При этом 0 – запрос однозначно является не вредоносным, 100 – запрос однозначно является вредоносным

PostgreSQL обеспечивает хороший уровень производительности, которого хватает для демонстрационных возможностей данного межсетевого экрана уровня веб-приложения.

## 1.7 ТЕХНОЛОГИЧЕСКОЕ ИСПОЛНЕНИЕ МЕЖСЕТЕВОГО ЭКРАНА УРОВНЯ ВЕБ-ПРИЛОЖЕНИЯ

Межсетевой экран уровня веб-приложения, разработанный в рамках данного исследования и называющийся «kwaf», состоит из двух программных компонентов.

Первый компонент является веб-приложением, разработанным на языке программирования Kotlin[[19]](#footnote-19) с использованием каркаса Spring Boot[[20]](#footnote-20). Язык программирования Kotlin выполняется на платформе Java Virtual Machine (JVM), что обеспечивает его хорошую производительность в связи с тем, что платформа JVM развивается несколько десятилетий такими компаниями как Sun Microsystems и Oracle, а также сообществом программистов. Программы на платформе JVM чаще всего используются в тех случаях, когда необходима платформа, которая имеет предсказуемое поведение, которая будет поддерживаться сравнительно длительное время. Практически все крупные программы, в том числе корпоративные, разработаны для выполнения на платформе Java Virtual Machine. Выбор конкретно этого языка на платформе JVM обусловлен тем, что язык программирования Java является в целом довольно многословным, поэтому для выражения одних и тех же семантических конструкций чаще всего требуется больше синтаксических конструкций по сравнению с языком программирования Kotlin.

Веб-приложения содержит в себе две входные точки доступа:

1. <Любой HTTP-метод> /ingress/<любая допустимая последовательность символов> <Любое тело запроса>. Данная точка доступа необходима для интеграции межсетевого экрана уровня веб-приложения с самим веб-приложением в автоматическом режиме вразрез каналу интернет – веб-приложение или зеркалированием трафика из интернета к веб-приложению. Очевидно, что данный метод позволяет принимать абсолютно любой HTTP-запрос, соответствующий спецификации. На основе данных из принятого запроса производится обработка и расчёт вредоносности запроса
2. POST /events <Тело в формате JSON, в котором содержится необходимая информация о запросе>. Данная точка доступа необходима для интеграции межсетевого экрана уровня веб-приложения с самим веб-приложением в ручном режиме. На основе тела запроса производится обработка и расчёт вредоносности запроса. Позволяет делать выборочную проверку запросов, и, в целом, настраивать межсетевой экран для работы в произвольном необходимом режиме

Вторым компонентом является программа, которая и занимается определением вредоносности запроса. Входные и выходные данные для неё передаются через протокол HTTP, поэтому в некотором смысле эту программу можно назвать веб-приложением. Программа написана на языке программирования Python. Данный язык программирования был выбран в связи с тем, что на нём реализовано или сравнительно просто подключить и использовать много программных модулей с алгоритмами машинного обучения, искусственных нейронный сетей и обработки данных. К таким модулям относятся: TensorFlow[[21]](#footnote-21), Keras[[22]](#footnote-22), scikit-learn[[23]](#footnote-23),XGBoost[[24]](#footnote-24), CatBoost[[25]](#footnote-25). Для использования протокола HTTP не применяются какие-либо веб-фреймворки, потому что они в данном случае будут избыточными. Программа содержит в себе только одну точку доступа:

* POST /rating <Тело в формате JSON, в котором содержится как информация о текущем запросе, так и информация по предыдущим запросам пользователя>. Данная точка доступа предназначена для обмена данными в межсетевом экране. Она получает данные, направляет их на вход программе, программа производит расчёт рейтинга, и он возвращается в качестве ответа. Метод POST был использован вместо очевидного GET ввиду того, что программе необходимы входные данные, которые невозможно передать через GET запрос, потому что у него отсутствует тело

Для определения вредоносности запроса на основе характеристик, описанных ранее, был использован алгоритм машинного обучения дерево решений (decision tree) в режиме регрессии. Данный алгоритм позволяет построить дерево, которое будет выдавать некоторую количественную характеристику в зависимости от количественных характеристик обучающей выборки и примера для определения характеристики [24].

В связи с тем, что деревья решений появились более 35 лет назад [25], реализация простого варианта исполнения не является нерешённой проблемой и не будет рассмотрена в данном исследовании. Программа на языке программирования Python использует модуль scikit-learn, в котором содержится класс tree.DecisionTreeRegressor, который представляет собой реализацию дерева решений для задачи регрессии. В качестве алгоритма дерева используется оптимизированный алгоритм CART.

Кроме того, оба компонента межсетевого экрана спроектированы по технологии предметно-ориентированной архитектуры (domain-driven design) [26]. Использование именно такой технологии позволяет программам быть легко поддерживаемыми, быть тесно связанными с предметной областью, а также быть легко расширяемыми. Для данного межсетевого экрана это означает, в частности, что может быть заменён компонент перехвата входящих веб-запросов, так и компонент определения их вредоносности. Кроме того, может быть заменён алгоритм определения вредоносности запроса на другой, более совершенный. При этом API программы не нужно будет изменять.

A picture containing screenshot

Description automatically generated

Рисунок 7 - Архитектура межсетевого экрана уровня веб-приложения

Высокоуровневая архитектура взаимодействия двух компонентов межсетевого экрана уровня веб-приложения и самого веб-приложения представлена на рисунке 7. Так, запрос пользователя в режиме установки вразрез каналу попадает в компонент перехвата запросов. Данный компонент выделяет необходимые данные из самого запроса, сохраняет их в базу данных и вызывает компонент анализа запросов для выполнения расчёта репутации запроса. Сохранение данных в базу данных, как и вызов компонента анализа могут выполняться в асинхронном режиме, чтобы не задерживать проброс оригинального запроса в само веб-приложение. Далее, веб-приложение получает запрос пользователя, и может запросить репутацию данного запроса из компонента анализа запросов в случае, если это необходимо по бизнес-правилам. В случае, если такой задачи нет, веб-приложение формирует ответ на пользовательский запрос и возвращает его.

Основные фрагменты исходного кода представлены в приложении А.

## 1.8 ПОДГОТОВКА И ОБРАБОТКА ДАННЫХ ДЛЯ АЛГОРИТМА ОПРЕДЕЛЕНИЯ ВРЕДОНОСНОСТИ ЗАПРОСА

Как уже было сказано ранее, для определения вредоносности запроса используются следующие характеристики запроса:

* IP-адрес
* Заголовок User-Agent
* Идентификатор пользователя, выполнившего запрос

Считается, что идентификатор пользователя был получен из аутентификационных данных пользователя. В рамках данного исследования это число в формате UUID, по которому можно относить запросы к конкретному пользователю.

Для алгоритма определения вредоносности необходимо, чтобы каждый признак мог быть измерен в числовом значении. В простейшем случае по тому, насколько сильно отличаются числа, алгоритм определяет, насколько сильно отличается итоговый результат.

IP-адрес представляет собой 32-битное число в особой записи. Для алгоритма определения вредоносности положим, что все IP-адреса данного интернет-провайдера в данном городе отличаются не более чем на 1; все IP-адреса данного города, но других интернет-провайдеров отличаются не более чем на 5; все IP-адреса другого города, но этой же страны отличаются не более чем на 10; все IP-адреса другой страны отличаются не более чем на 100. Данный метод кодирования позволяет представить все IP-адреса в виде чисел, которые могут быть сравнены между собой, с явной зависимостью от их возможной вредоносности. Определение принадлежности IP-адреса конкретному интернет-провайдеру, городу или стране выходит за рамки данного исследования, существуют свободные базы данных GeoIP, например, от MaxMind[[26]](#footnote-26) с использованием которых можно узнать принадлежность IP-адреса.

Заголовок User-Agent является строкой с особой структурой, в которой записаны, как правило, название текущего веб-браузера, его версия, название движка веб-браузера, его версия, название операционной системы, её версия. Различные заголовки User-Agent могут говорить о том, что пользователь работает с веб-приложением через несколько разных устройств или браузеров. Или же, что профилем пользователя владеют несколько человек. Кроме того, существует проект Tor Browser[[27]](#footnote-27), который позволяет скрыть реальный IP-адрес и может устанавливать недостоверный заголовок User-Agent. Такие программы и сервисы часто используют злоумышленники, потому что они помогают не раскрывать реальные данные пользователей, кроме того, делают это бесплатно. Для алгоритма определения вредоносности положим, что все версии данного браузера на данной платформе отличаются не более чем на 1; все другие браузеры на данной платформе отличаются не более чем на 5; все другие платформы и браузеры отличаются не более чем на 10.

В качестве базовых версий IP-адресов и заголовков User-Agent выбираются данные, предоставленные при регистрации профиля пользователя в веб-приложении. Со временем старые записи теряют актуальность, потому что версии браузеров и операционных систем обновляются, пользователи меняют сами браузеры и операционные системы, меняют интернет-провайдеров и меняют места жительства. Поэтому, в перспективе целесообразно использовать в качестве базовых версий те IP-адреса и заголовки User-Agent, которые находятся в некотором небольшом временном окне (например, за последний месяц), и оценка вредоносности которых минимальна.

Кроме того, не все запросы в веб-приложении могут быть одинаково вредоносными. Например, мошенники почти наверняка не будут осуществлять платежи из своих средств в пользу веб-приложения, не будут интересоваться данными, которые им не принесут выгоды. Можно с большой уверенностью считать все запросы, которые раскрывают данные для фишинга или спама потенциально вредоносными. Например, поисковые запросы в социальных сетях, которые раскрывают личные данные пользователя, к которым относятся дата рождения, город проживания, мобильный телефон, электронная почта, идентификаторы в других социальных сетях сильно заинтересуют злоумышленников в связи с тем, что такой набор данных открывает практически неограниченный доступ к созданию цифровых идентификаторов реальных людей и дальнейшим мошенническим действиям. Таким образом, URL-адреса запросов должны также участвовать в методе определения вредоносности запроса, но проставление числовых коэффициентов для разных URL-адресов зависит полностью от веб-приложения, числовые коэффициенты должны быть выбраны на этапе интеграции межсетевого экрана уровня веб-приложения и самого веб-приложения, и должны периодически пересматриваться.

# 2. ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ. ЗАКЛЮЧЕНИЕ

## 2.1 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

Разработанный алгоритм обнаружения вредоносных запросов показал свою работоспособность в межсетевом экране уровня веб-приложения. Межсетевой экран обнаруживает подозрительные запросы, сильно отличающиеся от типичных запросов пользователя по выбранным характеристикам запроса.

Для сравнения был выбран свободный межсетевой экран ModSecurity версии 3.0.3 с набором правил OWASP ModSecurity Core Rule Set (CRS) версии 3.1.0. Была произведена имитация взлома пользователя: сначала пользователь впервые производил запрос к веб-приложению (имитация регистрации в веб-приложении) с IP-адресом из Санкт-Петербурга, России, и заголовком User-Agent со значением «Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_14\_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36», что является обычным заголовком распространённого браузера Google Chrome. Второй запрос производился от имени этого же пользователя через 10 минут, к другому URL, с IP-адресом из Афганистана (по данным Kaspersky Cyberthreat[[28]](#footnote-28) на момент проведения сравнения Афганистан был страной с самым высоким риском заражения компьютерными угрозами), заголовок User-Agent был выбран как «Mozilla/5.0 (Windows NT 6.1; rv:60.0) Gecko/20100101 Firefox/60.0» (такой же заголовок используется в Tor Browser).

В результате ModSecurity никак не отреагировал на оба запроса. Разработанный в рамках данного исследования межсетевой экран уровня веб-приложения «kwaf» первому запросу присвоил вероятность вредоносности 0, то есть запрос был классифицирован как безвредный. Второму запросу была присвоена вероятность вредоносности 100, то есть запрос был классифицирован как однозначно вредоносный. Ответ межсетевого экрана на получение репутации второго запроса представлен на рисунке 8.

A screenshot of a social media post

Description automatically generated

Рисунок 8 - Пример ответа межсетевого экрана уровня веб-приложения

Однако, вместе с тем стало понятно, что выбранных признаков недостаточно для полноценного определения вредоносности запроса. Для подготовки межсетевого экрана уровня веб-приложения не только к опытной эксплуатации, но и к полноценному внедрению необходимо добавить другие характеристики запроса с их числовыми характеристиками. В таком случае алгоритм получит гораздо больше данных о запросе и сможет определять вредоносность эффективнее.

Кроме того, шкала оценки вероятности вредоносности запроса от 0 до 1 (или от 0 до 100 в процентах) не является абсолютной для межсетевого экрана уровня веб-приложения. Под каждое веб-приложение, с которым интегрируется экран, значения данной шкалы необходимо переоценивать. Данная характеристика относится к свойствам разработанного межсетевого экрана, который может интегрироваться с любым веб-приложением, и не является ошибкой или недостатком. Единственная рекомендация для интеграторов межсетевого экрана может быть такой: если запрос отнесён к вредоносным с вероятностью от 0.5 (50%), то необходимо вручную проанализировать запрос для уточнения правил работы межсетевого экрана; если запрос отнесён к вредоносным с вероятностью от 0.75 (75%), то он с большой долей вероятности является вредоносным, и может быть отнесён к таким в автоматическом режиме. В процессе интеграции межсетевого экрана уровня веб-приложения и самого веб-приложения необходимо скорректировать данные значения вероятностей для правильной работы с веб-приложением.

## 2.2 ДАЛЬНЕЙШЕЕ РАЗВИТИЕ

С учётом всех разработанных в данном исследовании алгоритмов и принципов, возможно несколько улучшений межсетевого экрана уровня веб-приложения.

В данной версии дерево решений для каждого запроса строится динамически. На небольшом количестве запросов в секунду это не является проблемой, и практически никак не сказывается на производительности межсетевого экрана. Однако, для выдерживания высокой производительности по обработке запросов необходимо строить деревья решений в фоновом режиме.

В качестве улучшения самого алгоритма определения вредоносности запроса можно использовать алгоритмы градиентного бустинга, например, случайный лес. Данный алгоритм позволяет использовать ансамбль решающих деревьев для решения одной задачи. В результате, отдельные деревья, слабые по своим собственным качествам, при использовании в совокупности позволяют на порядки улучшить общее качество решения задачи [27].

Люди совершают действия в веб-приложении с некоторой задержкой, в отличие от запрограммированных компьютеров. В дополнение к этому действия пользователей обычно имеют некоторый порядок. Боты обычно запрограммированы на конкретную цель с конкретным планом выполнения, в результате их поток взаимодействия может не совпадать ни с одним пользователем веб-приложения. Кроме того, временные интервалы между этапами взаимодействия с веб-приложением могут сильно отличаться. Для обнаружения таких случаев мошенничества необходимо сохранять контекст запроса, другие запросы пользователя за сессию взаимодействия. Ориентированный граф со средним интервалом между запросами на дугах может быть построен на основе потока взаимодействия пользователя с веб-приложением. Запросы, которые не подходят под граф для конкретной страны, города или интернет-провайдера рассматриваются как подозрительные.

В технологическом плане исполнения межсетевого экрана уровня веб-приложения, при существенных задержках на этапе регистрации событий, язык программирования Kotlin может быть заменён на Go, который является компилируемым, что обеспечивает существенно большую производительность.

# 3. СПИСОК ЛИТЕРАТУРЫ

1. Fatkieva R.R. Development of metrics for attack detection on the basis of network traffic analysis // Bulletin of the Buryat State University. Mathematics, Informatics. 2013. Vol. 1. P. 81-86.
2. Endraca A., King B., Nodalo G., Maria M. Sta., Sabas I. Web Application Firewall (WAF) // International Journal of e-Education, e-Business, e-Management and e-Learning. 2013. Vol. 3, No. 6. P. 451-455.
3. Baranov P.A., Beybutov E.R. Securing information resources using web application firewalls // Business Informatics. 2015. Vol. 4(34). P. 71-78.
4. Melnikov V.G., Trifanov A.V. WAF Bypass // Interexpo GEO-Siberia. 2017. Vol. 1. P. 113-117.
5. Платонов Т.С. Обзор характерных шаблонов работы шифровальщиков и способов их обнаружения // Сборник трудов X международной научно-практической конференции молодых ученых «Программная инженерия и компьютерная техника (Майоровские чтения)». 2019. Т. 1. С. 111-113.
6. Платонов Т.С., Оголюк А.А. Межсетевые экраны уровня веб-приложения в современном мире // Сборник трудов X международной научно-практической конференции молодых ученых «Программная инженерия и компьютерная техника (Майоровские чтения)». 2019. Т. 1. С. 107-109.
7. Mukhamediev R.I., Muhamedijeva J.L., Kuchin Y.I. Taxonomy of Machine Learning Techniques and Assessment of Qualification Quality and Learnability // Cloud of Science. 2015. Vol. 2, No. 3. P. 359-378.
8. Gorbachevskay E.N., Krasnov S.S The history of the development of neural networks // Vestnik Of Volzhsky University after V.N. Tatishchev. 2015. Vol. 1(23).
9. Саядян Р.К. Исследование методов защиты информации при построении средств защиты класса «Application Firewall» // Актуальные проблемы авиации и космонавтики. 2016. T. 1/2016. C. 773-775.
10. Adler A., Mayhew M.J., Cleveland J., Atighetchi M., Greenstadt R. Using Machine Learning for Behavior-Based Access Control: Scalable Anomaly Detection on TCP Connections and HTTP Requests // IEEE Military Communications Conference. 2013. V. 1. P. 1880-1887.
11. Gomez-Boix A., Laperdrix P., Baudry B. Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale // 27th International World Wide Web Conference. 2018. Vol. 1. P. 1-10.
12. Vastel A., Laperdrix P., Rudametkin W., Rouvoy R. FP-STALKER: Tracking Browser Fingerprint Evolutions // 39th IEEE Symposium on Security and Privacy. 2018. Vol. 1. P. 728-741.
13. Wang T., Goldberg I. Improved Website Fingerprinting on Tor // WPES '13 Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society. 2013. Vol. 1. P. 201-212.
14. Eckersley P. How Unique Is Your Web Browser // PETS'10 Proceedings of the 10th international conference on Privacy enhancing technologies. 2010. Vol. 1. P. 1-19.
15. Bodal K., Foldes A. M., Gulyas G. G., Imre S. User Tracking on the Web via Cross-Browser Fingerprinting // NordSec 2011: Information Security Technology for Applications. 2011. Vol. 1. P. 31-46.
16. Fifield D., Egelman S. Fingerprinting web users through font metrics // International Conference on Financial Cryptography and Data Security. 2015. Vol. 1. P. 107-124.
17. Laperdrix P., Rudametkin W., Baudry B. Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints // 37th IEEE Symposium on Security and Privacy. 2016. Vol. 1. P. 1-18.
18. Mowery K., Shacham H. Pixel Perfect: Fingerprinting Canvas in HTML5 // Proceedings of W2SP. 2012. Vol. 1. P. 1-12.
19. Cao Y., Li S., Wijmans E. (Cross-)Browser Fingerprinting via OS and Hardware Level Features // Network and Distributed System Security Symposium. 2017. Vol. 1. P. 1-15.
20. Starov O., Nikiforakis N. XHOUND: Quantifying the Fingerprintability of Browser Extensions // 2017 IEEE Symposium on Security and Privacy (SP). 2017. Vol. 1. P. 1-16.
21. Nikiforakis N., Kapravelos A., Joosen W., Kruegel C., Piessens F., Vigna G. Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting // 2013 IEEE Symposium on Security and Privacy. 2013. Vol. 1. P. 541-555.
22. Kurtzer G.M., Sochat V., Bauer M.W. Singularity: Scientific containers for mobility of compute // PLoS One. 2017. Vol. 12(5). P. 1-20.
23. Abadi D.J., Madden S., Hachem N. Column-stores vs. row-stores: how different are they really? // In Proceedings of the ACM SIGMOD Conference on Management of Data. 2008. Vol. 1. P. 967-980.
24. Esmeir S., Markovitch S. Anytime Learning of Decision Trees // Journal of Machine Learning Research (JMLR). 2007. Vol. 8. P. 891-933.
25. Breiman L., Friedman J.H., Olshen R.A., Stone C.J. Classification and regression trees / L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone. Monterey: Wadsworth & Brooks/Cole Advanced Books & Software, 1984. 358 p.
26. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software / E. Evans. New Jeresy: Prentice Hall, 2003. 560 p.
27. Breiman L. Random Forests // Machine Learning. 2001. Vol. 45. P. 5-32.

# ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

**package** org.kwaf.kwaf.web.controllers  
  
**import** org.kwaf.kwaf.entities.Event  
**import** org.kwaf.kwaf.inputs.EventInput  
**import** org.springframework.http.server.reactive.ServerHttpRequest  
**import** org.springframework.stereotype.Controller  
**import** org.springframework.web.bind.annotation.RequestHeader  
**import** org.springframework.web.bind.annotation.RequestMapping  
**import** org.springframework.web.bind.annotation.RequestMethod  
**import** org.springframework.web.bind.annotation.ResponseBody  
**import** reactor.core.publisher.Mono  
**import** java.util.\*  
**import** java.util.function.Function  
  
@Controller  
**class** IngressController(**private val createEvent**: Function<EventInput, Event>) {  
 @RequestMapping(  
 method = [RequestMethod.**GET**, RequestMethod.**POST**, RequestMethod.**PUT**, RequestMethod.**DELETE**],  
 value = [**"/ingress/\*\*"**]  
 )  
 @ResponseBody  
 **fun** call(@RequestHeader(value = **"User-Agent"**) userAgent: String,  
 @RequestHeader(value = **"User-Id"**, required = **false**) userId: Optional<UUID>,  
 request: ServerHttpRequest): Mono<Event> {  
 **val** host = request.*remoteAddress*!!.*address*.*hostAddress* **val** input = EventInput(userId.orElse(UUID.randomUUID()), host, userAgent)  
  
 **return** Mono.just(**createEvent**.apply(input))  
 }  
}

**package** org.kwaf.kwaf.useCases.events  
  
**import** org.kwaf.kwaf.entities.Event  
**import** org.kwaf.kwaf.gateways.EventGateway  
**import** org.kwaf.kwaf.inputs.EventInput  
**import** org.kwaf.kwaf.useCases.ratings.CalculateRating  
**import** org.springframework.stereotype.Component  
**import** java.util.\*  
**import** java.util.function.Function  
  
@Component  
**class** CreateEvent(**private val eventGateway**: EventGateway,  
 **private val calculateRating**: CalculateRating) : Function<EventInput, Event> {  
 **override fun** apply(eventInput: EventInput): Event {  
 **val** event = Event(  
 UUID.randomUUID(),  
 eventInput.**userId**,  
 eventInput.**userAgent**,  
 eventInput.**host**,  
 Date(),  
 rating = **null** )  
 **val** rating = **calculateRating**.apply(event)  
 **val** eventWithRating = event.copy(rating = rating)  
  
 **return eventGateway**.save(eventWithRating)  
 }  
}

**package** org.kwaf.kwaf.useCases.ratings  
  
**import** org.kwaf.kwaf.entities.Event  
**import** org.kwaf.kwaf.gateways.EventGateway  
**import** org.kwaf.kwaf.gateways.RatingGateway  
**import** org.springframework.stereotype.Component  
**import** java.util.function.Function  
  
@Component  
**class** CalculateRating(**private val eventGateway**: EventGateway,  
 **private val ratingGateway**: RatingGateway) : Function<Event, Double> {  
 **override fun** apply(event: Event): Double {  
 **val** previousEvents = **eventGateway**.findAllByUserId(event.**userId**)  
 **return ratingGateway**.calculateRating(event, previousEvents).join()  
 }  
}

**package** org.kwaf.kwaf.gateways  
  
**import** com.google.gson.GsonBuilder  
**import** org.kwaf.kwaf.entities.Event  
**import** org.kwaf.kwaf.inputs.CalculateRatingResponse  
**import** org.springframework.core.env.Environment  
**import** org.springframework.core.env.get  
**import** org.springframework.scheduling.annotation.Async  
**import** org.springframework.stereotype.Component  
**import** java.net.URI  
**import** java.net.http.HttpClient  
**import** java.net.http.HttpRequest  
**import** java.net.http.HttpResponse  
**import** java.text.SimpleDateFormat  
**import** java.util.concurrent.CompletableFuture  
  
@Component  
**class** RatingGateway(**private val environment**: Environment) {  
 @Async  
 **fun** calculateRating(event: Event, previousEvents: Iterable<Event>): CompletableFuture<Double> {  
 **val** gson = GsonBuilder().setDateFormat(SimpleDateFormat(**"yyyy-MM-dd'T'HH:mm:ss.SSSZ"**).toPattern()).create()  
  
 **val** payload = gson.toJson(previousEvents)  
 **val** url = URI(**"${environment**[**"kwaf.rating\_url"**]**}/rating"**)  
 **val** request = HttpRequest  
 .newBuilder(url)  
 .POST(HttpRequest.BodyPublishers.ofString(payload))  
 .header(**"Content-Type"**, **"application/json;charset=UTF-8"**)  
 .build()  
 **val** client = HttpClient.newHttpClient()  
  
 **return** client  
 .sendAsync(request, HttpResponse.BodyHandlers.ofString())  
 .thenApply **{** gson.fromJson(**it**.body(), CalculateRatingResponse::**class**.*java*).**rating }** .exceptionally **{** exception **->** 0.0 **}** }  
}

**package** org.kwaf.kwaf.entities  
  
**import** java.util.\*  
**import** javax.persistence.Column  
**import** javax.persistence.Entity  
**import** javax.persistence.Id  
**import** javax.persistence.Table  
  
@Entity  
@Table(name = **"events"**)  
**data class** Event(  
 @Id  
 **val id**: UUID,  
 @Column(name = **"user\_id"**)  
 **val userId**: UUID,  
 @Column(name = **"user\_agent"**)  
 **val userAgent**: String,  
 @Column(name = **"host"**)  
 **val host**: String,  
 @Column(name = **"committed\_at"**)  
 **val committedAt**: Date,  
 @Column(name = **"rating"**)  
 **val rating**: Double?  
)

**import** json  
**from** http.server **import** \*  
**from** rating.entities.event **import** Event  
**from** rating.use\_cases.calculate\_rating **import** CalculateRatingUseCase  
**from** rating.use\_cases.regressor **import** Regressor  
  
**class** HTTPHandler(BaseHTTPRequestHandler):  
 **def** do\_POST(self):  
 self.send\_response(200)  
 self.send\_header(**'Content-Type'**, **'application/json;charset=UTF-8'**)  
 self.end\_headers()  
  
 content\_len = int(self.headers.get(**'Content-Length'**))  
 body = self.rfile.read(content\_len).decode(**"utf-8"**)  
  
 objects = json.loads(body)  
 events = list(map(**lambda** o: Event.build(o), objects))  
  
 rating = CalculateRatingUseCase(Regressor()).call(events)  
 self.wfile.write(rating.toJSON())  
  
  
server = HTTPServer((**"0.0.0.0"**, 8081), HTTPHandler)  
server.serve\_forever()

**from** typing **import** List  
**from** rating.entities.event **import** Event  
**from** rating.entities.rating **import** Rating  
**from** rating.use\_cases.regressor **import** Regressor  
  
**class** CalculateRatingUseCase:  
 **def** \_\_init\_\_(self, regressor: Regressor):  
 self.regressor = regressor  
  
 **def** call(self, events: List[Event]) -> Rating:  
 fit\_data = events[:-1]  
 to\_predict = events[-1]  
  
 x = list(map(**lambda** e: [e.user\_agent, e.host], fit\_data))  
 y = list(map(**lambda** e: e.rating / 100, fit\_data))  
 x\_test = [to\_predict.user\_agent, to\_predict.host]  
  
 **return** Rating(self.regressor.call(x, y, x\_test) \* 100)

**from** typing **import** List  
**from** sklearn **import** tree  
  
**class** Regressor:  
 **def** call(self, x: List[List[float]], y: List[float], x\_to\_test: List[float]) -> float:  
 regressor = tree.DecisionTreeRegressor(max\_depth=3)  
 regressor.fit(x, y)  
  
 **return** regressor.predict(x\_to\_test)[0]

1. <https://securelist.ru/ddos-attacks-in-q4-2018/93384/> [↑](#footnote-ref-1)
2. <https://www.ptsecurity.com/ru-ru/research/analytics/web-application-vulnerabilities-statistics-2019/> [↑](#footnote-ref-2)
3. <https://www.anti-malware.ru/compare/web-application-firewall-waf-2017> [↑](#footnote-ref-3)
4. <https://www.gartner.com/doc/3888676/magic-quadrant-web-application-firewalls> [↑](#footnote-ref-4)
5. <https://tools.ietf.org/html/rfc7230> [↑](#footnote-ref-5)
6. <https://tools.ietf.org/html/rfc8446> [↑](#footnote-ref-6)
7. <https://tools.ietf.org/html/rfc7540> [↑](#footnote-ref-7)
8. <https://tools.ietf.org/html/draft-ietf-quic-http-20> [↑](#footnote-ref-8)
9. <https://12factor.net/> [↑](#footnote-ref-9)
10. <https://linuxcontainers.org/lxc/> [↑](#footnote-ref-10)
11. <https://www.docker.com/> [↑](#footnote-ref-11)
12. <https://mobyproject.org/> [↑](#footnote-ref-12)
13. <https://github.com/rkt/rkt> [↑](#footnote-ref-13)
14. <https://www.sylabs.io/singularity/> [↑](#footnote-ref-14)
15. <https://www.opencontainers.org/> [↑](#footnote-ref-15)
16. <https://kubernetes.io/> [↑](#footnote-ref-16)
17. <https://cassandra.apache.org/> [↑](#footnote-ref-17)
18. <https://www.postgresql.org/> [↑](#footnote-ref-18)
19. <https://kotlinlang.org/> [↑](#footnote-ref-19)
20. <https://spring.io/projects/spring-boot> [↑](#footnote-ref-20)
21. <https://www.tensorflow.org/> [↑](#footnote-ref-21)
22. <https://keras.io/> [↑](#footnote-ref-22)
23. <https://scikit-learn.org/stable/> [↑](#footnote-ref-23)
24. <https://xgboost.ai/> [↑](#footnote-ref-24)
25. <https://catboost.ai/> [↑](#footnote-ref-25)
26. <https://dev.maxmind.com/geoip/> [↑](#footnote-ref-26)
27. <https://www.torproject.org/> [↑](#footnote-ref-27)
28. <https://cybermap.kaspersky.com/stats> [↑](#footnote-ref-28)