Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Высшая школа кибербезопасности и защиты информации

Работа допущена к защите
Директор Высшей школы
кибербезопасности и защиты
информации, д.т.н., проф.
_____ Д.П. Зегжда
«___»_____2020 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ДИПЛОМНАЯ РАБОТА

ЗАЩИТА МОБИЛЬНЫХ УСТРОЙСТВ ПОД УПРАВЛЕНИЕМ ANDROID ОТ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ИСПОЛЬЗОВАНИЕМ ИСКУССТВЕННОЙ НЕЙРОННОЙ СЕТИ

по направлению подготовки (специальности)
10.05.03 Информационная безопасность

Направленность (профиль)
10.05.03 Информационная безопасность автоматизированных систем

Выполнил
студент гр. 3651003/50801 Корольков А.А.

Руководитель,
доцент ВШКиЗИ ИПММ, Павленко Е.Ю.
к.т.н.

Санкт-Петербург

2020

РЕФЕРАТ

На 91 с., 26 рисунков, 5 таблиц.

КЛЮЧЕВЫЕ СЛОВА: ANDROID, СИСТЕМА ОБНАРУЖЕНИЯ ВПО, ВРЕДОНОСНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, СТАТИЧЕСКИЙ АНАЛИЗ, НЕЙРОННЫЕ СЕТИ, ГЛУБОКОЕ ОБУЧЕНИЕ, АППАРАТНОЕ УСКОРЕНИЕ НЕЙРОННЫХ ВЫЧИСЛЕНИЙ

Тема выпускной квалификационной работы: «Защита мобильных устройств под управлением Android от вредоносного программного обеспечения с использованием искусственной нейронной сети».

Данная работа посвящена улучшению безопасности устройств под управлением Android путем разработки средств обнаружения вредоносного программного обеспечения с использованием искусственной нейронной сети. Задачи, которые решались в ходе исследования:

1. Провести анализ методов обнаружения ВПО для ОС Android
2. Исследовать особенности программной архитектуры ОС Android
3. Разработать подход к обнаружению ВПО с применением искусственной нейронной сети
4. Разработать программный макет, реализующий предлагаемый подход
5. Определить оптимальные параметры работы и оценить эффективность разработанного макета.

В данной дипломной работе представлено исследование существующих проблем информационной безопасности приложений в операционной системе Android, различных подходов к обнаружению вредоносного ПО. Произведено изучение возможностей аппаратного ускорения вычисления нейронных сетей на мобильных устройствах на платформе Android с помощью графического и сигнального процессоров, рассмотрены и протестированы на различных устройствах разные реализации ускорения вычислений. Предложена методика к обнаружению Android ВПО с применением искусственных нейронных сетей,

оценена точность и эффективность обнаружения вредоносного программного обеспечения разработанным программным прототипом.

THE ABSTRACT

91 pages, 26 figures, 5 tables.

KEY WORDS: ANDROID, MALWARE DETECTION SYSTEM, MALICIOUS SOFTWARE, STATIC ANALYSIS, NEURAL NETWORKS, DEEP LEARNING, HARDWARE ACCELERATION OF NEURAL COMPUTING

The subject of the graduate qualification work is "Protection of mobile devices running Android from malicious software using an artificial neural network".

This work is devoted to improving the security of devices running Android by developing tools for detecting malicious software using an artificial neural network. Tasks that were solved in the course of the study:

1. To analyse the methods of detection of malware for Android OS
2. Explore the features of the Android OS software architecture
3. Develop an approach to detecting malicious software using an artificial neural network
4. Develop a program layout that implements the proposed approach
5. Determine the optimal performance parameters and evaluate the effectiveness of the developed layout.

This paper presents a research of existing problems of information security of applications on the Android operating system, various approaches to malware detection. A research was made on the possibilities of hardware acceleration of computing neural networks on mobile devices on the Android platform with the help of graphics and signal processors. Different implementations of computing acceleration are considered and tested on various devices. A method for detecting Android malware using artificial neural networks is offered, and the accuracy and effectiveness of malware detection by the developed software prototype is assessed.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные мобильные устройства на платформе Android за последние 10 лет получили широчайшее распространение. Благодаря своей открытости и универсальности сегодня на этой системе можно найти огромное множество различных устройств: мобильные телефоны, часы, электронные книги, планшеты, ноутбуки, автомобильные мультимедиа системы и телевизоры. Операционная система Android и приложения для них тоже быстро развиваются. Пользователи хранят постоянно растущий объем конфиденциальных и чувствительных данных на портативных устройствах. Это привело к появлению все большего и большего числа вредоносного ПО. В последние пару лет вопрос информационной защиты мобильных, а в частности устройств на базе операционной системы Android стал крайне актуальным. Так как, не смотря на активное внедрение средств защиты в мобильную ОС, за прошедший 2019 год продуктами компании «Лаборатория Касперского» было обнаружено[1]:

6. 3 503 952 вредоносных установочных пакета.
7. 69 777 новых мобильных троянских приложения, маскирующихся под банковские приложения.
8. 68 362 новых мобильных троянских приложения, связанных с вымоганием денег у пользователей.

Также за 2019 год участились атаки на личные данные пользователей и случаи обнаружения троянских приложения на самых популярных площадках приложений, что свидетельствует о наличии недостатков в существующих подходах к обнаружению вредоносного ПО, встроенных в площадки по распространению приложений. Кроме этого, стоит отметить, что не все существующие методы и идеи, подходящие для обнаружения вредоносного ПО для настольных компьютеров уместны, эффективны и применимы для мобильных устройств. В основном методы обнаружения можно разделить на 3 типа. Статический анализ выполняется без запуска приложения, до его установки. Статический анализатор собирает информацию из кода и ресурсов приложения и основываясь на полученной информации делает вывод о вредоносности. Динамический анализ,

напротив, собирает информацию о приложении уже после его запуска. Это порождает необходимость использования реального устройства или его эмулирования. Это порождает некоторые накладные расходы ресурсов, но позволяет точнее отслеживать поведение приложения и делать выводы о его вредоносности в тех случаях, когда статический анализ не смог обнаружить вредоносную направленность приложения. Соответственно, анализирующие средства, использующие оба метода, называются гибридными.

В последние несколько лет большое развитие получили методы решения задач, основанные на искусственных нейронных сетях. Эти методы могут применяться для очень широкого спектра задач: распознавание объектов, голоса, улучшение фото/видео, аппроксимация данных, классификация данных [2]. Нейронные сети обладают высокой эффективностью решения задач, которой не могут добиться другие методы машинного обучения. В связи с этим нейронные сети стали применяться в области информационной безопасности, в том числе и для классификации программного обеспечения с целью выявления ВПО.

В свою очередь из-за широкой области применения нейронных сетей и необходимости решения подходящих для нее задач непосредственно на мобильных устройствах в течении последних трех лет [3] стали часто появляться аппаратные решения, позволяющие ускорить и сделать более эффективным выполнение нейронных сетей на мобильных устройствах. В связи со всем вышеизложенными факторами становится актуальным применение искусственных нейронных сетей для обнаружения ВПО на мобильных устройствах, как альтернатива другим средствам анализа мобильного ПО.

Целью данной дипломной работы является выявление вредоносных приложений на мобильных устройствах под управлением Android с использованием искусственной нейронной сети.

Для достижения данной цели необходимо решить следующие ***задачи***:

1. Провести анализ методов обнаружения ВПО для ОС Android
2. Исследовать особенности программной архитектуры ОС Android

3. Разработать подход к обнаружению ВПО с применением искусственной нейронной сети

4. Разработать программный макет, реализующий предлагаемый подход

5. Определить оптимальные параметры работы и оценить эффективность разработанного макета.

1 МЕХАНИЗМЫ БЕЗОПАСНОСТИ ANDROID, ЕЕ ПРОБЛЕМЫ И УГРОЗЫ

В данном разделе представлена общая архитектура Android приложений, особенности и механизмы системы безопасности системы Android, классификация угроз для мобильных устройств, основные разновидности ВПО, а также проблемы безопасности Android.

1.1 Архитектура Андроид приложений

В 2007 году корпорация Google совместно с другими компаниями основала организацию Open Handset Alliance, чтобы разработывать открытую операционную систему в рамках проекта AOSP (англ. Android Open Source Project), которая получила название Android.[4] Особенностью системы является ее открытость. Код операционной системы доступен всем и бесплатен для использования в устройствах любых производителей. Это помогло производителям делать различные мобильные устройства из разных ценовых сегментов, что дополнительно ускоряло распространение и развитие данной ОС.

Платформа Android представляет собой программный стек для различных слоев, включающий в себя операционную систему, программное обеспечения промежуточного слоя, а также основные пользовательские приложения. [5]

Архитектуру Android обычно делят на 4 основных уровня (рисунок 1.1):

6. Уровень ядра Linux
7. Уровень библиотек и среды выполнения
8. Уровень фреймворка Java API
9. Пользовательские и системные приложения

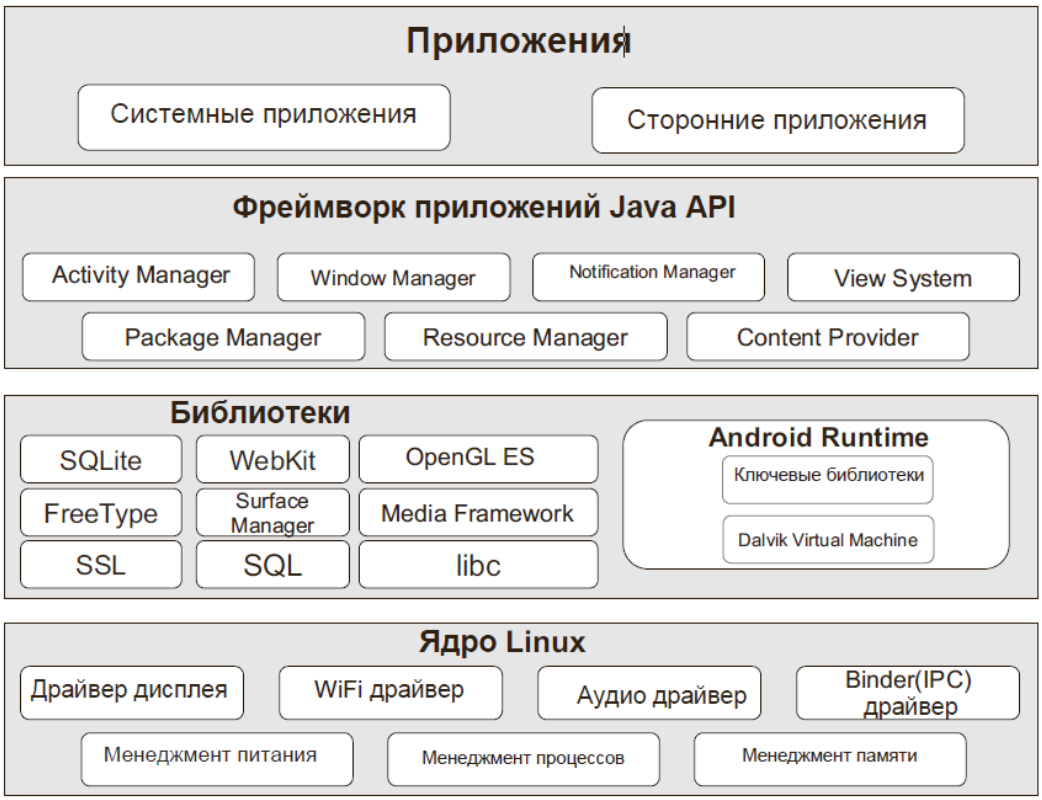

Рисунок 1.1 – Архитектура приложений в ОС Android.

1.1.1 Слой приложений

Слой приложений расположен в верхней части программного стека Android. Он включают в себя как предустановленные в конкретной реализации Android OS приложения, так и сторонние приложения, разработанные сторонними разработчиками приложений. Примерами таких приложений являются браузер, менеджер контактов и приложения электронной почты.

1.1.2 Фреймворк приложений:

– Фреймворк приложений - это набор служб, которые в совокупности формируют среду, в которой выполняются и управляются приложения Android. службы предоставляются приложениям в виде классов Java. Разработчикам приложений разрешается использовать эти службы в своих приложениях. фреймворк приложений включает в себя следующие основные службы [6]: менеджер активности, контент-провайдеры, менеджер ресурсов, менеджер уведомлений и

система показа интерфейсов.

– Менеджер активностей: менеджер активностей управляет и контролирует все аспекты жизненного цикла приложения и стека его активностей. Эта служба взаимодействует с общими процессами, выполняемыми в системе.

– Контент-провайдеры: контент-провайдеры управляют доступом к структурированному набору данных. Они инкапсулируют данные и обеспечивают механизмы для определения безопасности данных. Поставщики контента — это стандартный интерфейс, который соединяет данные в одном процессе с кодом, запущенным в другом процессе. Другими словами, эта служба позволяет приложениям публиковать и обмениваться данными с другими приложениями.

– Диспетчер ресурсов: эта служба предоставляет доступ к встроенным ресурсам кроме кода, таким как строки, настройки цвета и макеты пользовательского интерфейса из приложений. Эта служба позволяет поддерживать ресурсы приложений независимо друг от друга.

– Диспетчер уведомлений: диспетчер уведомлений позволяет приложениям отображать предупреждения и уведомления для пользователя. С помощью этой службы приложения могут уведомлять пользователя о событиях, происходящих в фоновом режиме.

– Система показа интерфейсов: эта служба представляет собой расширяемый набор видов и форм, используемых для создания пользовательских интерфейсов приложений.

1.1.3 Среда Выполнения Android:

В этом слое находится ключевой компонент, называемый Dalvik Virtual Machine (DVM), который представляет собой виртуальную машину Java (JVM), специально разработанную и оптимизированную для Android. Dalvik VM контролирует основные возможности системы Linux, такие как многопоточность,

многозадачность и управление памятью, что является неотъемлемой частью языка Java.

Dalvik VM дает возможность приложениям работать как процесс непосредственно на ядре Linux и в пределах своей собственной виртуальной машины (Sandbox). Поскольку Dalvik использует JVM, он предоставляет пользователям набор библиотек и API для разработки Android-приложений, преимущественно использующих язык программирования Java. Стандартная среда разработки Java включает в себя широкий спектр классов, которые содержатся в основных библиотеках среды выполнения Java. Во последних версиях системы среда выполнения Dalvik была заменена на среду выполнения ART. ART использует опережающую компиляцию кода, которая может улучшить производительность приложений. Каждый процесс, запускаемы в системе Android выполняется на собственном экземпляре среды выполнения Android Runtime.

1.1.4 Библиотеки:

Библиотеки Android были разработаны поверх ядра Linux. Этот уровень позволяет устройству обрабатывать различные типы данных. Он предоставляет различные библиотеки, нужные для нормального функционирования операционной системы Android.Эти библиотеки написаны на языке C или C++ и были разработаны для конкретного аппаратного обеспечения.

Примеры некоторых важных собственных библиотек включают в себя движок веб-браузера с открытым исходным кодом WebKit, используемый для отображения HTML-контента, библиотеку libc, движок баз данных SQLite, используемый для хранения данных, OpenGL, используемый для отображения 2D-или 3D-графического содержимого на экране, медиа-фреймворк, используемый для предоставления различных медиа-кодеков, и библиотеки SSL для обеспечения интернет-безопасности.

1.1.5 Ядро:

Ядро Linux — это фундаментальный слой всей системы. Этот слой настраивается специально для встраивания в решения с ограниченными ресурсами. Вся ОС Android построена поверх ядра Linux с некоторыми дальнейшими архитектурными изменениями, сделанными Google. Этот раздел действует как уровень абстракции между аппаратным и другими программными уровнями. Ядро Linux обеспечивает базовую функциональность системы, такую как управление процессами, управление памятью и управление устройствами. Ядро Linux также предоставляет множество драйверов устройств, которые облегчают задачу при взаимодействии Android с периферийными устройствами. Использование уже существующего ядра системы упростило и ускорило разработку Android.

Также важным слоем, часто относимым к ядру, является слой аппаратных абстракций (англ. Hardware Abstraction Layer, HAL). Данный слой представляет посредника между аппаратным обеспечением и высокоуровневым Java API. Это модульный набор библиотек для конкретных аппаратных компонентов таких как камера, модули связи и др.

1.2 Структура пакета Android приложения

Приложения для Android, расширяющие функциональные возможности устройств, написаны в основном на языке программирования Java. В этом подразделе рассматривается структура приложений Android и его основные четыре компонента.

1.2.1 .apk пакет приложения.

Приложение для Android содержит несколько файлов и папок, упакованных в один пакет с расширением apk. Он используется для распространения и установки прикладного программного обеспечения и промежуточного программного обеспечения на операционную систему Google Android. На рис. 1.2 показана структура пакета приложений для Android. Некоторые конкретные компоненты в файлах приложений играют важную роль. Например, каталог META-

INF включает Manifest.MF, содержит криптографическую подпись и дает возможность проверить все содержимое пакета на валидность.

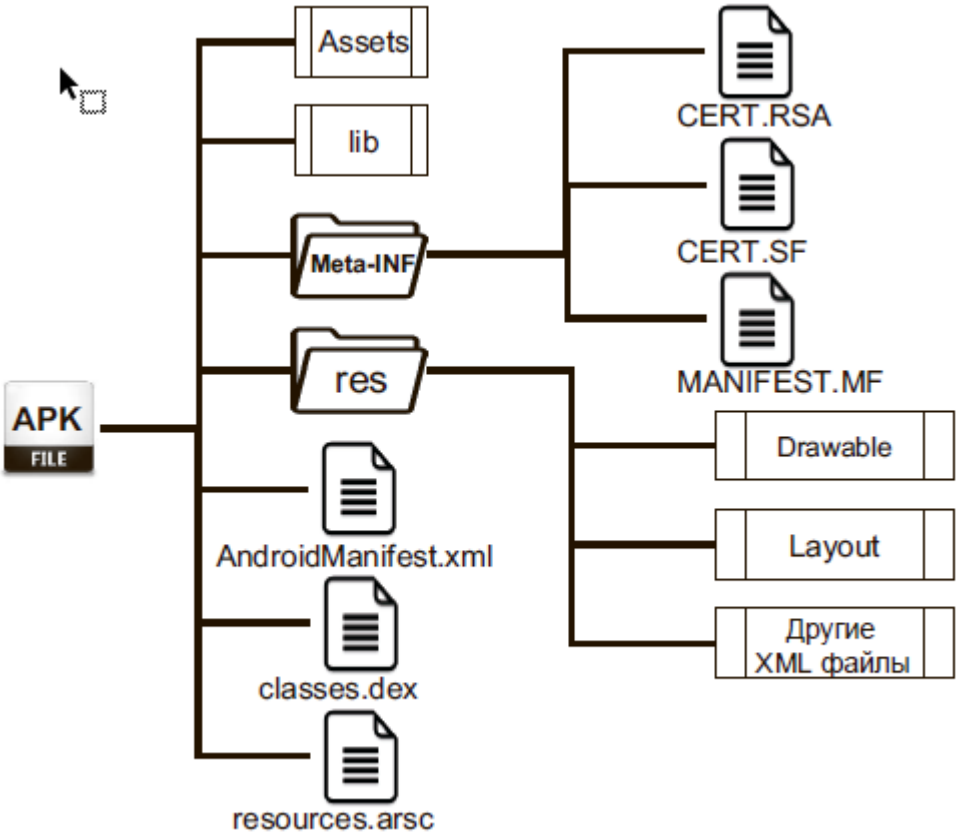

Рисунок 1.2 – Структура установочного файла приложения

1.2.2 Компоненты Приложения

Существует четыре различных типа компонентов приложения [20]. Каждый компонент служит определенной цели и имеет определенный жизненный цикл, который определяет, как компонент создается и уничтожается. На рисунке 1.3 показаны компоненты приложения для Android и связанные с ними взаимодействия.

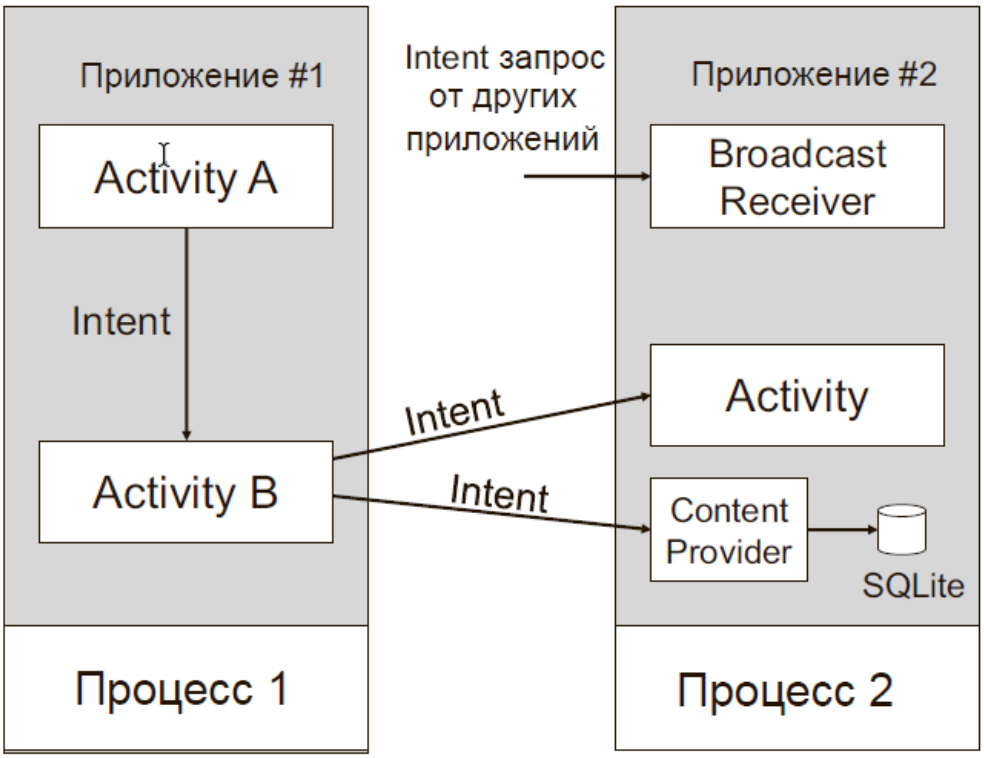

Рисунок 1.3 – Процесс взаимодействия между приложениями

Activities: Activity — это индивидуальный экран пользовательского интерфейса в приложении для Android. Android activity - это область, где могут быть размещены визуальные элементы, называемые представлениями (Views) (также известные как виджеты), и пользователь может выполнять различные действия, взаимодействуя с ними.

Службы: службы — это способ поддержания работы в фоновом режиме в Android. Службы используются для выполнения частей обработки вашего приложения в фоновом режиме. Службы обычно используются для процессов, которые занимают длительный период времени, таких как воспроизведение музыки, загрузка данных или выгрузка фотографий.

Контент-провайдер: поставщик контента (англ. Content Provider) — это компонент для управления набором данных. Контент-провайдеры в Android предоставляют гибкий способ сделать данные доступными для всех приложений. Простой пример контент-провайдера - приложение Contacts Manager. Вы можете получить контакты в нескольких приложениях, таких как приложение SMS, приложение номеронабирателя и т. д.

Широковещание: широковещательные приемники (англ. Broadcast receivers) — это один из компонентов приложения Android, который используется для приема сообщений, транслируемых системой Android или другими приложениями Android. Образцами вещания, инициируемого системой, являются сигнал о низком заряде батареи, изменение состояния сети, запуск телефона и фотография, снятая с камеры.

1.3 Механизмы и архитектура Безопасности Android

Android — это современная мобильная платформа, которая была разработана, чтобы быть открытой и бесплатной. Приложения для Android используют современное аппаратное и программное обеспечение, а также локальные и служебные данные, передаваемые через систему. Чтобы защитить их, платформа должна предложить прикладную среду, которая обеспечивает безопасность пользователей, данных, приложений, устройств и сети. Обеспечение безопасности открытой платформы требует надежной архитектуры безопасности и строгих программ безопасности. В этом разделе мы обсуждаем механизмы, используемые Android для обеспечения безопасности среды приложений [7].

1.3.1 Фреймворк разрешений для Android

Базовое приложение для Android не имеет никаких разрешений, связанных с ним по умолчанию, чтобы получить доступ к ресурсам. Перед установкой приложения текущая версия ОС Android отображает все необходимые разрешения приложения. Запрашиваемые разрешения предназначены для обеспечения соблюдения ограничений на ресурсы устройств, таких как интернет-соединения, SMS, хранилище, камера и т. д. Ознакомившись с этими разрешениями, пользователь может принять их или отказаться от них, установив приложение только в том случае, если они его примут [7]. Существует четыре класса разрешений: обычный, опасный, подпись и подпись или система. В этом подразделе рассматриваются все типы разрешений для Android.

Нормальное разрешение: разрешение с низким уровнем риска, которое предоставляет запрашивающим приложениям доступ к изолированным функциям уровня приложения с минимальным риском для других приложений, системы или пользователя.

Опасно: разрешение повышенной опасности, которое дает запрашивающему приложению доступ к личным пользовательским данным или контроль над устройством, что может негативно повлиять на пользователя.

Подпись: разрешение, которое система предоставляет только в том случае, если запрашивающее приложение подписано тем же сертификатом, что и приложение, объявившее это разрешение.

Подпись или система: разрешение, которое система предоставляет только приложениям, находящимся в образе системы Android или подписанным тем же сертификатом, что и приложение, объявившее это разрешение.

1.3.2 Применение Песочницы

Песочница приложений, также называемая контейнеризацией приложений, представляет собой подход к разработке программного обеспечения и управлению мобильными приложениями (Mobile Application Management-MAM), который ограничивает среду, в которой может выполняться определенный код. приложения Android работают в песочнице, изолированной области системы, которая не имеет доступа к остальным ресурсам системы, если только разрешения на доступ не предоставляются пользователем явно при установке приложения. Для защиты данных приложения от несанкционированного доступа ядро Android реализует дискреционный контроль доступа Linux (DAC) для управления и предохранения ресурсов устройства от неправильного использования.

Также приложение должно содержать сертификат PKI, подписанный ключом разработчика. Подпись приложения — это подтверждение доверия между Google и сторонними разработчиками для обеспечения целостности приложения и репутации разработчика. Помещаясь в изолированную песочницу, приложе-

нию присваивается уникальный UID. Помимо UID, процессу может быть присвоен один или несколько идентификаторов группы (gid). Если сертификат приложения A совпадает с уже установленным приложением B, система Android назначает один и тот же UID (т. е. песочницу) приложениям A и B, позволяя им совместно использовать свои личные файлы и разрешения, определенные манифестом. Этот непреднамеренный общий доступ может быть использован авторами вредоносных программ. Поэтому, разработчикам рекомендуется хранить свои сертификаты в тайне, чтобы избежать их неправомерного использования.

1.3.3 Межкомпонентная коммуникация (Inter-Component Communication - ICC)

Хотя каждое приложение выполняется в выделенной песочнице, Android позволяет приложениям взаимодействовать друг с другом через четко определенный механизм Межкомпонентной связи (ICC) или связыватель (binder). Программное обеспечение Android обеспечивает передачу ICC между компонентами приложения. Binder или ICC заботится о переносе выполнения запроса от инициатора запроса к целевому процессу прозрачно для приложений. Приложения могут вызывать компоненты или службы других приложений как сервис [7].

1.3.4 Защита магазина приложений Google Play

Для обеспечение безопасности пользователей в магазин приложений Google был внедрен ряд защитных мер. Один из таких Google Play Защита

Google Play Защита помогает обеспечить безопасность устройства, а именно [8]:

– предварительно проверяет все приложения, которые вы скачиваете в Play Маркете;

– сканирует ваше устройство на наличие потенциально опасных приложений из сторонних источников (вредоносного ПО);

– предупреждает вас об угрозах и удаляет известные опасные приложения с устройства;

– сообщает вам о приложениях, которые скрывают или искажают важную информацию, тем самым нарушая Правила в отношении нежелательного ПО;

– отправляет вам оповещения о нарушениях конфиденциальности, если приложения получают доступ к вашим персональным данным, воспользовавшись разрешениями пользователя, и тем самым нарушают Правила для разработчиков.

1.4 Классификация угроз мобильных устройств

Существуют различные типы угроз для мобильных устройств, которые могут плохо повлиять на мобильные устройства, такие как вирусы и шпионские программы, которые могут заразить персональные компьютеры (ПК). Эти угрозы можно разделить на четыре широкие категории: уровень приложений, веб-уровень, сетевой уровень и физический уровень (см. рисунок 1. 4).

1.4.1 Угрозы на уровне приложений

Угрозы на уровне приложений основаны на приложениях, которые являются основной функционала каждого мобильного устройства. Эти угрозы являются наиболее широко обсуждаемыми угрозами в литературе, которая представляет угрозы прикладного уровня как наиболее широко распространенную угрозу. Поскольку приложения, работающие на этих мобильных устройствах, доступны на сторонних магазинах приложений, ясно, что они могут быть целевыми направлениями для нарушений безопасности мобильных устройств . Вредоносные программы — это приложения для Android, которые выполняют вредоносные действия, могут внедрять вредоносный код в мобильное устройство, которые отправляют нежелательные сообщения и позволяют противнику удаленно управлять устройством.

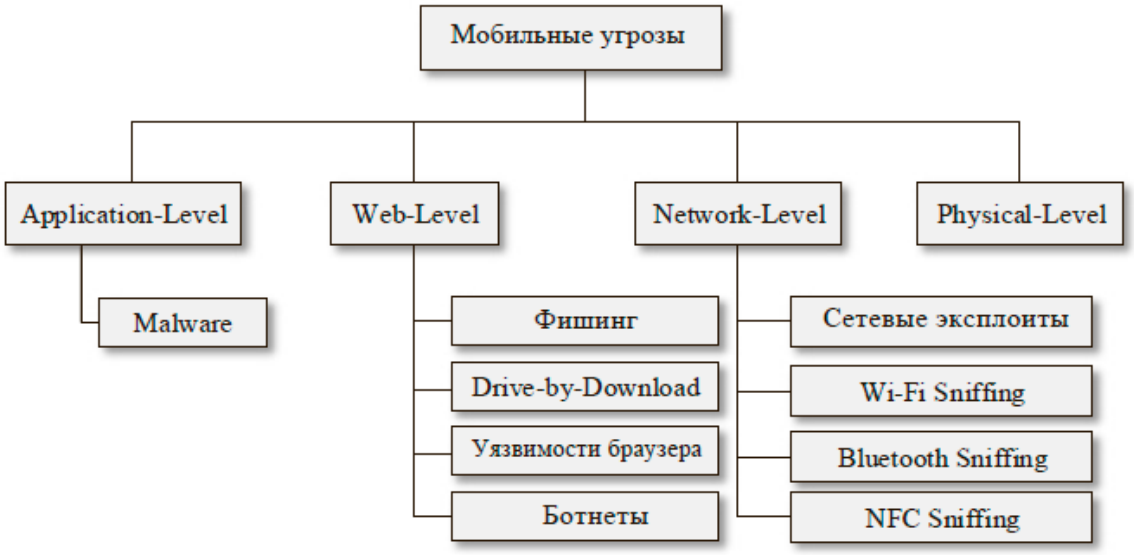

Рисунок 1.4 –Общая классификация мобильных угроз

Вредоносная программа (Malware). Это специально разработано для повреждения устройств, на которых они выполняются, или сети, по которой он взаимодействует. Вредоносное ПО в основном устанавливается на устройства жертв для выполнения незаконных действий без ведома владельца. Диапазон вредоносных программ варьируется; это может быть просто всплывающая реклама или настолько опасно, что программа вызывает повреждения ОС или устройства. Кража конфиденциальных учетных данных владельца и заражение новых уязвимых устройств являются основными целями вредоносных программ. Наиболее распространенными вредоносными программами являются крадущие доступ к финансам, шифрующие данные и рекламные вредоносные программы.

1.4.2 Веб-Угроз

Угрозы безопасности и конфиденциальности для мобильных устройств от веб-сайтов не редки. Наиболее опасными угрозами на веб-уровне являются фишинговые мошенничества, скачиваемые файлы и эксплойты браузера.

Фишинговые мошенничества являются ключевой угрозой на веб-уровне, которая использует электронную почту или другие приложения социальных сетей для отправки ссылок пользователю на фишинговый сайт, предназначенный для обмана пользователей и кражи конфиденциальной информации, такой как

учетные данные пользователя. Фишинг является одной из семи основных угроз безопасности, выявленных Лабораторией Касперского.

Drive-by-Download. Это скачанный файл, который относится к потенциально вредоносным приложениям, который устанавливается на устройство пользователя без разрешения. Пользователь может даже не знать, что программное обеспечение было установлено. Drive-by-downloads — это разновидность вредоносных программ, обычно обнаруживаемых на скомпрометированных веб-страницах. Просто посещая веб-страницу файл начинает загружаться и затем устанавливается в фоновом режиме на компьютере или мобильном устройстве без предупреждения пользователя.

Уязвимостей браузера. Это вредоносный код, который использует часть программного обеспечения или уязвимостей ОС для нарушения безопасности браузера. Эксплойты браузера выполняют эти вредоносные действия без уведомления владельца устройства.

Ботнет — это сеть подключенных к интернету зараженных устройств (ботов) под контролем ботмастера (киберпреступника) для выполнения киберпреступной деятельности без ведома владельца устройства. Существует два типа ботнетов: традиционные ботнеты и мобильные ботнеты. Эта работа посвящена мобильным (Android) ботнетам, но также будут рассмотрены общие теоретические сведенья о ботнетах.

1.4.3 Угрозы Сетевого Уровня

Угрозы сетевого уровня могут возникать из-за подключения мобильных устройств к сотовым/мобильным сетям, локальным беспроводным сетям или ближней полевой связи (NFC). Сетевые эксплойты, Wi-Fi sniffing, Bluetooth и NFC являются основными типами сетевых угроз.

Сетевые эксплойты. Сетевые эксплойты используют недостатки в мобильной операционной системе или другом программном обеспечении, которое

работает в локальных или сотовых сетях. После подключения они могут перехватывать подключения и данные и находить способ внедрения вредоносного программного обеспечения на телефоны пользователей без их ведома.

Wi-Fi sniffing захватывает данные, когда они перемещаются между устройством и точкой доступа Wi-Fi. Большинство приложений Android не используют надлежащие меры безопасности при отправке незашифрованных данных по сети. Киберпреступник может легко прочитать данные. Публичные места, такие как кафе, рестораны и книжные магазины могут не иметь защищенного соединения и вполне вероятно, что любой человек может перехватить ваши пакеты.

Bluetooth. Люди, которые оставляют BT всегда включенным, оставляют себя уязвимыми для связи со сторонними устройствами. Такие атаки как Blue jacking могут привести к попыткам фишинга и распространению вредоносных программ или вирусов.

Коммуникации ближнего поля (NFC). NFC позволяет обмениваться изображениями, приложениями и другими данными между двумя устройствами без предварительного сопряжения. Для этой цели оба устройства используют функцию, которую Google называет Android Beam.

1.4.4 Угрозы Физического Уровня

Угрозы физического уровня более важны, чем другие упомянутые угрозы. Поскольку мобильные устройства являются небольшими, портативными и ценными, это делает их физическую безопасность более важной. Кража и неуместное использование устройств являются общей проблемой среди пользователей этих устройств. Эти устройства ценны не только потому, что их перепродают на нелегальном рынке, но и, что более важно, потому что они содержат конфиденциальные и личные данные. Большинство пользователей мобильных устройств используют свой телефон для банковских операций, социальных коммуникаций и многого другого, в то время как конечные пользователи всегда подключены к этим счетам, что делает украденный более уязвимым для преступной

деятельности. Кроме того, потерянное или украденное устройство может быть использовано для получения доступа к секретным данным, хранимым на них.

1.5 Основные разновидности ANDOID ВПО

В этом разделе рассмотрены основные разновидности ВПО на платформе Android:

Трояны

Трояны маскируются под доброкачественные приложения, но они выполняют вредные действия без согласия или ведома пользователей. Трояны крадут конфиденциальную информацию пользователя, такую как пароли, контакты и тд. До второго квартала 2012 года большинство вариантов androidсвирусов принадлежало различным семействам SMS-троянцев. SMS-троянские приложения способны отправлять SMS на номера премиум-класса без ведома и / или согласия пользователя, что влечет за собой финансовые потери для владельца.

Также в связи с увеличением числа людей, использующих возможности мобильного банкинга, авторы вредоносных программ нацелились на двухфакторную аутентификацию банков. После захвата имени пользователя и пароля целевых учетных записей, трояны Zitmo и Spitmo отслеживают и крадут номера карт и перехватывают подтверждения о транзакциях, тем самым проводя операции с картой незаметно для пользователя.

Бэкдор

Бэкдор позволяет другим вредоносным программам бесшумно проникать в систему, облегчая им обход обычных процедур безопасности. Бэкдор может использовать root-эксплойты, чтобы получить привилегию суперпользователя и скрыться от антивирусных сканеров. Ряд root-эксплойтов, таких как rage-against-The-cage, rageagainst-thecage и gingerbreak [9], получают полный контроль над устройством. Basebridge [10], KMin [10], Obad [11] являются заметным примером известных бэкдоров.

Червь

Червь-приложение может создавать точные или похожие копии самого себя и распространять их через сеть и/или съемные носители. Например, черви Bluetooth могут использовать функции bluetooth и отправлять копии на сопряженные устройства. Android.Obad.OS [11] пример bluetooth- червя.

Ботнеты

Ботнет-приложения компрометируют устройство для создания бота, так что устройство управляется удаленным сервером, называемым Bot-master, с помощью ряда команд. Сеть таких ботов называется ботнет. Команды могут быть как простыми, как отправка личной информации на удаленный сервер, так и сложными, приводящими к атакам типа "отказ в обслуживании". Бот также может включать команды для автоматической загрузки вредоносных полезных нагрузок. Geinimi [10], Anserverbot [10], Beanbot [10] - это известные ботнеты для Android.

Spyware

Шпионское ПО само по себе может быть не вредоносной утилитой, но имеет скрытые функции для мониторинга контактов, сообщений, местоположения, банковских карт и т. д. это приводит к нежелательным последствиям. Он может отправлять собранную информацию на удаленный сервер. Nickyspy [10], GPSSpy [12] являются известными примерами шпионских приложений.

За прошлый год в полтора раза — с 40 386 уникальных атакованных пользователей в 2018 году до 67 500 в 2019 году — участились случаи атак на личные данные пользователей мобильных устройств. Речь идет не о классическом шпионском ПО, а о полноценном ПО для слежки за конкретными пользователями. [1] [13]

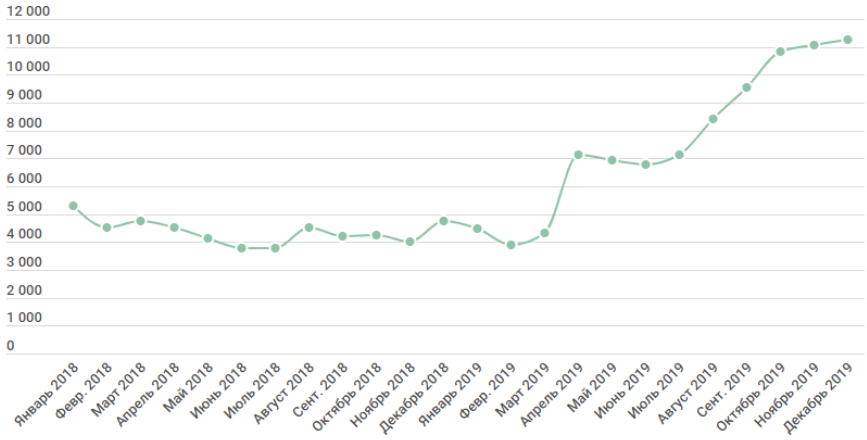

Рисунок 1.5–Количество уникальных пользователей, атакованных stalkerware, 2018–2019 гг.

Агрессивная реклама

Android предоставляет очень точную и удобную службу месторасположения устройства. Некоторые рекламные партнерские сети злоупотребляют такими услугами определения местоположения и отправляют персонализированную рекламу на пользовательское устройство для получения дохода. Агрессивное рекламное ПО может создавать ярлыки на главном экране, изменять закладки пользователя, изменять настройки поисковой системы по умолчанию и присылать ненужные уведомления, что мешает использованию устройства. Plankton [3] является примером такого ВПО.

Вымогатели

Программы-вымогатели могут заблокировать пользовательское устройство, чтобы сделать его недоступным до тех пор, пока некоторая сумма выкупа не будет выплачена злоумышленикам. Например, FakeDefender.B [14] маскируйся под антивирус Avast! отображает поддельные предупреждения о вредоносных программах, чтобы уговорить пользователя установить фальшивую вредоносную программу. Далее он блокирует устройство и требует выкуп, чтобы разблокировать устройство.

1.6 Проблемы и угрозы безопасности Android

Безопасность Android основана на механизме, основанном на разрешениях, который регулирует доступ сторонних приложений Android к критическим ресурсам на устройстве Android. Такой механизм, основанный на разрешениях, широко критикуется разработчиками, маркетологами и конечными пользователями за его грубый контроль разрешений приложений и неэффективное управление разрешениями. Например, пользователи могут либо принимать все разрешительные запросы от приложения для его установки, либо не устанавливать его. Однако в новых версиях Android этот подход был частично изменен, позволяя выдавать разрешения выборочно и непосредственно перед использованием запрашиваемого ресурса. Так же в Android 10 Q появилась возможность ограничивать доступ к особо конфиденциальным данным в фооновой активности приложения, таким как местоположение, запись звуков и т.д. Но не всегда система безопасности, основанная на разрешениях, оказывается эффективной и удобной. В этом разделе будут рассмотрены основные проблемы безопасности Android, которые приводят к утечке пользовательской информации и ставят под угрозу конфиденциальность пользователя.

1.6.1 Утечка информации

В текущем дизайне архитектуры Android приложения ограничены в доступе к ресурсам или другим приложениям, если это не разрешено пользователями. Пользователи должны предоставить все запросы на доступ к ресурсам перед установкой и использованием приложения. Утечка информации происходит, когда пользователи предоставляют ресурсы без каких-либо ограничений со стороны ОС. Однако механизм контроля разрешений Android оказался неэффективным для защиты конфиденциальности пользователей и ресурсов от вредоносных приложений. Вместе с более чем 1,4 миллиона доступных приложений в Google Play и немалым количеством приложений из разных сторонних рынков, значительное количество вредоносных приложений доступно пользователям Android

для установки. Однако при установке нового приложения только небольшая часть пользователей обращает внимание на запрашиваемый разрешение на ресурс, так как они, как правило, спешат пропустить экраны запросов на получение разрешений, чтобы быстрее получить возможность использовать приложение. Лишь небольшая часть (3%) пользователей проявляет осторожность и дает правильные ответы на вопросы о предоставлении разрешений. Кроме того, текущие предупреждения о разрешениях Android не помогают большинству пользователей принимать правильные решения в области безопасности [15]. Подход" обвинения пользователей " превратился в неспособность обезопасить пользователей Android.

Как указывается в [15,16], причины неэффективности нынешней системы контроля разрешений заключаются в следующем:

– Неопытные пользователи не осознают, что запросы ресурсов неуместны и поставят под угрозу их конфиденциальность,

– Пользователи хотят использовать приложение и могут быть вынуждены обменять свою конфиденциальность на использование приложения.

1.6.2 Повышение привилегий

Атаки с повышением привилегий или разрешений были использованы путем использования общедоступных уязвимостей ядра Android для получения повышенного доступа к ресурсам, которые обычно защищены от приложения или пользователя. Этот тип атаки может привести к несанкционированным действиям со стороны приложений с большим количеством привилегий, чем предполагалось, что приводит к утечке многих конфиденциальных данных. Экспортированные компоненты Android могут быть использованы для получения доступа к критическим разрешениям [17].

1.6.3 Атака типа "отказ в обслуживании" (DoS)

Растущее число пользователей смартфонов и преобладание мобильных устройств (телефонов, планшетов), подключенных к Интернету, может стать платформой для роста DoS-атак. Поскольку большинство смартфонов не оснащены теми же средствами защиты, что и ПК (т. е. антивирусными программами), то вредоносные приложения используют их в качестве подходящей платформы для DoS-атак. Использование ресурсов мобильных устройств в корыстных целях– частая цель злоумышленников [18].

1.6.4 Скрытый замысел

Угроза скрытого замысла — это атака на стороне клиента. В этой атаке пользователи устанавливают набор приложений, разработанных одним и тем же разработчиком и одним и тем же сертификатом, и предоставляют различные типы разрешений, включая конфиденциальные и нечувствительные. После установки приложений эти приложения могут воспользоваться общим UID и получить доступ ко всем своим разрешениям и ресурсам [19].

1.6.5 Выполнение нативного кода

Android позволяет выполнять машинный код через библиотеки, реализованные в C / C++ с помощью Native Development Kit (NDK). Несмотря на то, что собственный код выполняется вне виртуальной машины Dalvik, он изолирован с помощью комбинации user-id/group-id(ы). Тем не менее, собственный код имеет потенциал для повышения привилегий путем использования уязвимостей платформы [20],[9] в недавнем прошлом было продемонстрировано довольно много вредоносных атак [21].

1.6.6 Переупаковка приложений

Переупаковка — это одна из самых важных и распространенных проблем безопасности ОС Android. Переупаковка — это процесс разборки/декомпиляции

apk файла с использованием методов обратного инжиниринга и добавления (инъекции) вредоносного кода в основной исходный код. Методы переупаковки, которые можно использовать на платформе Android, позволяют замаскировать вредоносный код под обычное приложение. Трудно отличить переупакованный вредоносный код от обычного приложения, потому что переупакованное приложение обычно функционирует так же, как и нормальное. Этапы переупаковки заключаются в следующем [22, 23]:

Распаковка: распаковка файлов APK с помощью доступных инструментов, таких как apktool, который является инструментом, основанным на обратном инжиниринге.

Декомпиляция: декомпиляция исходного кода Java с использованием JAD и извлечение исходного кода классов Java.

Инъекция кода: инъекция кода и добавление ресурсов в основной исходный код с помощью среды разработки Java.

Переупаковка: восстановление файлов с помощью apktool и подписание сгенерированных файлов с помощью jarsigner. Geimini и KungFu являются примерами троянов, которые основаны на переупаковке APK. Эти троянские программы можно интегрировать во многие валидные приложения для Android.

1.6.7 Техники обфускации вредоносного ПО

Авторы вредоносных программ запутывают полезные нагрузки ВПО, чтобы помешать или усложнить обнаружение антивирусами. Методы скрытия, такие как шифрование кода, перестановка ключей (key permutations), динамическая загрузка, выполнение машинного кода, остаются предметом интереса для решений защиты от вредоносных программ на основе сигнатур.

Методы обфускации реализуются для одной или нескольких из следующих целей.

– Защитить собственный алгоритм от конкурентов, сделав реверс-инжиниринг затруднительным.

– Защита цифровых прав на использование мультимедийных ресурсов для снижения уровня пиратства.

– Обфускация приложений может делать их более сжатыми и компактными, а, следовательно, может делать их более быстрыми в работе.

– Чтобы скрыть уже известное вредоносное ПО от антивирусных средств, чтобы распространить и заразить больше устройств.

– Чтобы предотвратить или, по крайней мере, задержать человека-аналитика и/или автоматические механизмы анализа от выяснения истинного мотива нового ВПО.

Байт-код Dalvik поддается обратному инжинирингу благодаря наличию типобезопасной информации, такой как типы классов/методов, определений, переменных, регистров, литеральных строк и инструкций. Методы преобразования кода могут быть легко реализованы на байт-коде dalvik, оптимизируя его с помощью инструмента защиты кода, такого как Proguard [24]. Proguard — это инструмент оптимизации для удаления неиспользуемых классов, методов и полей. Осмысленные имена классов / методов / полей / локальных переменных заменяются неразборчивым кодом, чтобы усложнить реверс-инжиниринг.

Dex-guard [25] — это инструмент защиты кода Android. Он может быть использован для реализации методов запутывания кода, таких как шифрование классов, слияние методов, шифрование строк, искажение потока управления и т. д. чтобы защитить приложение от реверс-инжиниринга. Методы преобразования кода также могут быть использованы для противодействия подходам обнаружения вредоносных программ [26]. Faruki и др. [27] предложили автоматизированную структуру преобразования байт-кодов dalvik для генерации неизвестных вариантов уже известных вредоносных программ с различными технологиями запутывания байт-кодов. Кроме того, они также оценили неизвестные образцы вредоносных программ на основе лучших коммерческих методов защиты от вредоносных программ и статического анализа. Авторы сообщили, что даже тривиальные методы трансформации могут обмануть существующие коммерческие антивирусные программы.

1.7 Способы проникновения и сокрытия ВПО

Ниже мы рассмотрим различные методы преобразования кода, используемые для запутывания существующих известных вредоносных программ и создания огромного количества неизвестных сигнатур вредоносных программ. Также преобразование кода также может быть реализовано, чтобы помешать инструментам дизассемблирования.

1. Вставка бесполезного кода и изменение порядка операций: вставка ненужного кода или кода отсутствия операций (nop) — это хорошо известный метод, который изменяет размер исполняемого файла и обходит обнаружение через базу данных сигнатур антивирусных программ. Вставка нежелательного кода сохраняет смысл исходного приложения. Однако она изменяет последовательность кодов операций, чтобы изменить сигнатуру вредоносного приложения. Код операции может быть переупорядочен с помощью инструкций goto между функциями и варьировать потоки управления, сохраняя исходную семантику процесса работы. Эти методы могут быть использованы для уклонения от решений обнаружения на основе сигнатур или кодов операций [26].

2. Переименования пакетов, классов или методов: android-приложение идентифицируется его уникальным именем пакета. Байт-код Dalvik, будучи типобезопасным, сохраняет имена классов и методов. Многие антивирусные программы используют тривиальные сигнатуры, такие как имена пакетов, классов или методов известных вредоносных программ, как сигнатуры обнаружения. Такие простые преобразования могут быть использованы для уклонения от обнаружения сигнатур, основанных на защите от вредоносных программ [28].

3. Изменение потока управления: некоторые антивирусные программы используют семантические сигнатуры, такие как поток управления и/или анализ потока данных, чтобы обнаружить варианты вредоносных программ, используя простые методы преобразования. Поток управления программой может быть изменен с помощью инструкций goto или путем вставки и вызова нежелательных

методов. Хотя такие методы и тривиальны, они обходят некоторые решения для защиты от вредоносных программ [28].

4. Шифрование строк: буквенные строки, такие как сообщения, URL-адреса и shell-команды, показывают многое о приложении. Чтобы предотвратить такой анализ, строки обычного текста могут быть зашифрованы и сделаны нечитаемыми. Кроме того, каждый раз, когда выполняется шифрование строк, различные методы шифрования (или ключи) затрудняют автоматизацию процесса расшифровки. В этом случае литеральные строки могут быть доступны только во время выполнения кода. Следовательно, ВПО уклоняется от методов статического анализа.

5. Шифрование классов: важная информация, такая как проверка лицензий на продукт, загрузка платного контента и DRM защита, может быть скрыта путем шифрования всех классов с использованием вышеуказанной конфиденциальной информации [25].

6. Шифрование ресурсов: содержимое папки ресурсов, активов и собственных библиотек может быть изменено на нечитаемое, следовательно, они должны быть расшифрованы непосредственно во время выполнения [28].

7. Использование API рефлексии: методы статического анализа ищут API в вредоносных приложениях, отображающие вредоносное поведение. Пользовательские приложения допускают java-рефлексию, позволяющую создавать программные экземпляры классов и / или вызывать методы с использованием литеральных строк. Однако литеральные строки могут быть зашифрованы, что затрудняет автоматический поиск API рефлексии. Такие методы могут легко обходят подходы статического анализа.

1.8 Улучшения безопасности в новый версиях Android

С учетом проблем безопасности, уязвимостей и сообщений о вредоносных атаках AOSP выпускает исправления, обновления, улучшения и обновления.

Здесь рассмотрены заметные исправления безопасности и функции, включенные в последующие версии ОС Android:

1) Android устранил переполнение буфера стека и целочисленное переполнение в ОС версии 1.5. В версии 2.3 Android исправил уязвимости формата строк и добавил аппаратную поддержку No eXecute (NX) для пресечения выполнения кода в стеке и куче.

2) в Android 4.0 была добавлена случайность размещения адресного пространства (ASLR), чтобы предотвратить атаки, связанные с return-to-libc и памятью.

3) Информация может быть украдена через подключение устройства к ПК, используя отладочный мост Андроид (драйвер ADB). Хотя ADB разработан как инструмент отладки, он позволяет устанавливать/отлаживать/удалять приложения, читать системные разделы и т. д. даже если устройство заблокировано, но подключено к персональному компьютеру (ПК). Для предотвращения такого несанкционированного доступа, Андроид 4.2.2 проверяет подключение к ADB с использованием ключей RSA. Если adb-соединение поступает на устройство, на его экране появляется запрос на получение согласия пользователя. Таким образом, если устройство заблокировано, злоумышленник не сможет получить контроль.

4) чтобы вредоносная программа не могла скрыто отправлять SMS-сообщения на платные номера, В Android 4.2 была введена дополнительная функция уведомления, позволяющая запрашивать разрешение пользователя непосредственно перед тем, как приложение отправит SMS-сообщение.

5) Android в версии 4.2 (API version 17) позволил создавать несколько пользователей (MU), чтобы позволить нескольким пользователям получить доступ к общему устройству, такому как планшет. Для каждого пользователя назначается отдельная учетная запись, выбранные пользователем приложения, пользовательские настройки, личные файлы и личные данные пользователя. Эта возможность позволяет нескольким пользователям совместно использовать одно устройство.

В сценарии MU основной учетной записью является владелец устройства. Используя настройки устройства, владелец может создать дополнительные MUs. Кроме исходного пользователя, другие созданные пользователи MU не могут создавать, изменять или удалять пользователей устройства MU.

6) Android 4.3 удалил утилиты setuid()/setgid () , поскольку они были уязвимы для root-эксплойтов.

7) Android 4.3 экспериментировал с SELinux, чтобы обеспечить повышенную безопасность. Android 4.4 представил SELinux с принудительным режимом для многих корневых процессов. SELinux ввела политику мандатного контроля доступа (Mandatory Access Control - MAC) вместо традиционного дискреционного контроля доступа (DAC). В DAC владелец ресурса решает, какие другие заинтересованные субъекты могут получить к нему доступ, тогда как в MAC система (а не пользователи) разрешает субъекту доступ к определенному ресурсу. Таким образом, MAC имеет потенциал предотвратить вредоносную активность(ы), даже если корневой доступ устройства скомпрометирован. Таким образом, MAC существенно снижает эффективность атак на повышение привилегий на уровне ядра.

2 НЕЙРОННЫЕ СЕТИ В МОБИЛЬНЫХ УСТОЙСТВАХ ANDROID

В данном разделе исследуются возможности мобильных устройств по запуску и выполнению искусственных нейронных сетей, рассматривается вопрос аппаратного ускорения на уровне системы Android, на уровне мобильных систем на чипе и на уровне реализации ускорения производителя чипов Qualcomm. Также в данном разделе приведены результаты и выводы тестирования выполнения нейронных сетей на различных устройствах различных поколений одного производителя.

Искусственная нейронная сеть (ИНС) — математическая модель, а также ее программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма. Нейронная сеть представляет собой структуру, состоящую из искусственных нейронов, определенным образом связанных друг с другом и внешней средой с помощью связей, каждая из которых имеет определённый коэффициент, на который умножается поступающее через него значение (эти коэффициенты называют весами). Нейронные сети могут реализоваться как программно, так и аппаратно (нейрочипы, нейрокомпьютеры).

В процессе функционирования нейронная сеть осуществляется преобразование данных, конкретный вид которого определяется весами межнейронных связей, видом активационной функции нейронов, архитектурой и конфигурацией сети

Нейронные сети представляют собой модели, основанные на машинном обучении, т.е. приобретают необходимые свойства в процессе обучения, который заключается в итеративной подстройке весов сети по некоторому правилу, называемому алгоритмом обучения.

Как видно на рисунке справа, у нейрона есть n входов Xi, у каждого из которого есть вес Wi, на который умножается сигнал, проходящий по связи. По-

сле этого взвешенные сигналы Xi·Wi направляются в сумматор, который агрегирует все сигналы во взвешенную сумму. Эту сумму также называют net. Таким образом, net=$\sum_{i=1}^{i=n} w_i \cdot x_i = w^T \cdot x_i$

Далее нейрон должен как обработать net и сформировать адекватный выходной сигнал. Для этих целей используют функцию активации, которая преобразует взвешенную сумму в какое-то число, которое и будет являться выходом нейрона. Функция активации обозначается ϕ(net). Таким образом, выходов искусственного нейрона является ϕ(net). Например, одной из самых популярных функций активации является сигмоида, представленная в формуле (2.1):

$$\phi(net) = \frac{1}{1-e^{-net'}} \quad (2.1)$$

На рисунке 2.1 можно увидеть схему описанного искусственного нейрона.

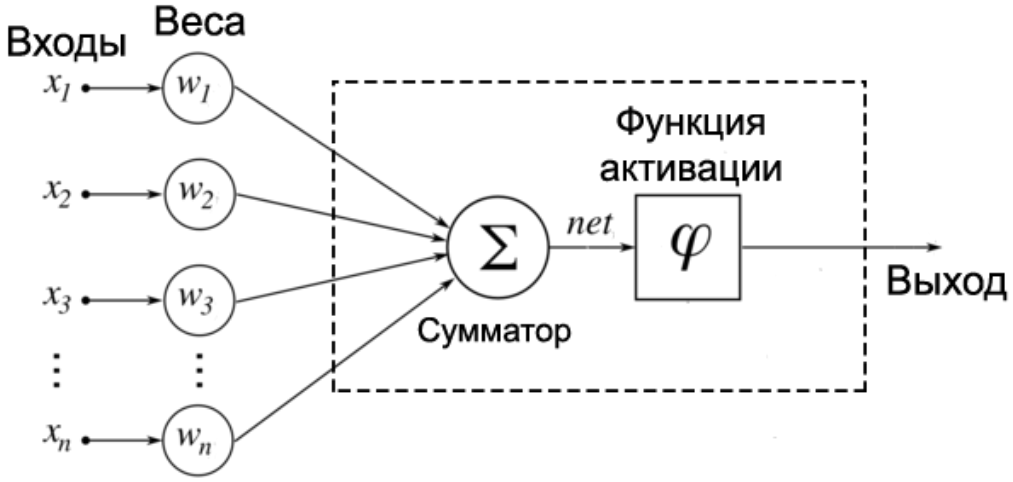

Рисунок 2.1 – схема искусственного нейрона.

Структура простейшей нейронной сети представлена на рисунке 2.2. Зелёным цветом обозначены нейроны входного слоя, голубым — нейроны скрытого слоя, жёлтым — нейрон(ы) выходного слоя.

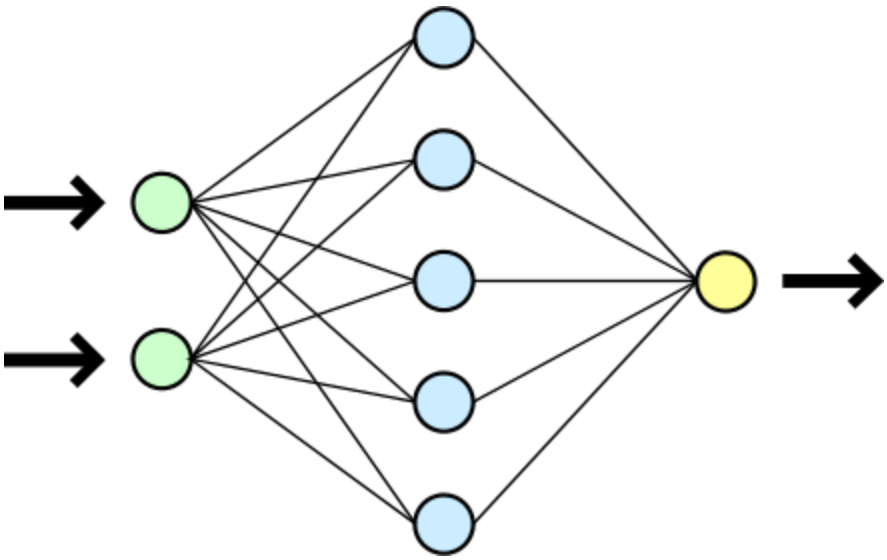

Рисунок 2.2. – схема простейшей нейронной сети

Нейроны входного слоя получают данные извне и после их обработки передают сигналы через синапсы нейронам следующего слоя. Нейроны второго (скрытого) слоя обрабатывают полученные сигналы и передают их нейронам выходного слоя. Такая, простейшая ИНС способна к обучению и может находить простые взаимосвязи в данных. В ней может быть несколько скрытых слоев нейронов, перемежаемых слоями, которые выполняют сложные логические преобразования. Каждый последующий слой сети ищет взаимосвязи в предыдущем. Такие ИНС способны к глубокому (глубинному) обучению.

В настоящее время различные модели глубокого обучения можно найти практически на любом мобильном устройстве. Наиболее популярными задачами являются различные задачи компьютерного зрения, такие как классификация изображений, улучшение изображения, увеличение разрешения изображения, моделирование размытие боке, отслеживание объектов в кадре, оптическое распознавание символов, распознавание лиц, дополненная реальность и т. д. Другая важная группа задач, выполняемых на мобильных устройствах, связана с различными проблемами обработки естественного языка, такими как перевод, завершение предложений, анализ предложений [29, 30, 31], голосовые помощники и интерактивные чат-боты. Так же глубокое обучение часто применяется для распознавания активностей пользователя, контроля сна и распознавания жестов.

2.1 Средства аппаратного ускорения нейронных сетей в мобильных устройствах

Хотя первые потребительские компьютеры были в основном оснащены одним автономным процессором, вскоре стало ясно, что его вычислительная производительность слишком ограничена для ряда мультимедийных приложений. Это привело к созданию специальных сопроцессоров обработки сигналов (DSP), работающих параллельно с основным процессором. Их архитектура была оптимизирована для задач обработки множества сигналов. Первые DSP имели довольно ограниченные возможности из-за ограниченного набора инструкций и ограниченной памяти. Они были популярны для приложений, связанных с компьютерной графикой, декодированием звука и видео, в качестве математических сопроцессоров и ускорителей для различных программ редактирования фотографий и даже для запуска первых моделей глубокого обучения для распознавания текста , разработанных в 1989 году [32]. Задача классификации рукописных цифр достигла в то время высоких скоростей (до 12 изображений в секунду) благодаря эффективным векторным и матричным вычислениям. Такие результаты возникли из-за сильно распараллеливаемой архитектуры DSP.

В начале 1990-х годов DSP начали появляться в мобильных телефонах. Сначала они использовались только для кодирования и сжатия речи, а также для некоторой обработки радиосигналов. Позже, с интеграцией камер и многих мультимедийных функций, таких как воспроизведение музыки и видео на мобильных устройствах, интегрированные DSP стали широко использоваться для обработки изображений, видео и звука. В отличие от того, что произошло с настольными компьютерами, DSP не были вытеснены здесь процессорами и графическими процессорами, потому что они часто предлагали превосходную производительность при более низком энергопотреблении, что так важно для портативных устройств. В последние годы вычислительная мощность мобильных DSP и других компонентов SoC резко возросла, и теперь, дополненные графическими

процессорами, NPU и выделенными ядрами для нейронных вычислений, они позволяют эффективно проводить операции с нейронными сетями.

2.2 Реализация аппаратного ускорения в Android OS

Хотя существует ряд проприетарных SDK для доступа к DSP, GPU или NPU на различных мобильных платформах, это на самом деле не решает проблему использования аппаратного ускорения для запуска алгоритмов глубокого обучения на мобильных телефонах, поскольку все эти SDK предоставляют доступ только к некоторым конкретным чипсетам и дополнительно несовместимы друг с другом. Чтобы решить эту проблему, Google представила унифицированный Android Neural Networks API (NNAPI), который представляет собой Android C API, предназначенный для выполнения вычислительно интенсивных машинных и глубоких операций обучения на мобильных устройствах. Основываясь на требованиях приложения и аппаратном обеспечении устройства, среда выполнения нейронных сетей Android может эффективно распределять вычислительную нагрузку между доступными процессорами на устройстве, включая выделенные микросхемы нейронных сетей, графические процессоры и DSP. NNAPI доступен на всех устройствах под управлением Android 8.1 (уровень API 27) или выше, но для доступа к аппаратному обеспечению устройства по-прежнему требуется специализированный драйвер поставщика. Для устройств, которые не имеют этого драйвера, среда выполнения NNAPI использует оптимизированный код для выполнения запросов на ЦП.

С широким использованием операционной системы Android на эту платформу был перенесен ряд популярных фреймворков глубокого обучения, включая Torch , Deeplearning4j , TensorFlow (Mobile , Lite), Caffe, Caffe2, MXNet, NNabla и др. В настоящее время наиболее часто используются из них Tensorflow Lite .

TensorFlow Lite [33] был представлен в конце 2017 года в качестве преемника TF Mobile. По данным Google, он обеспечивает лучшую производительность и меньший размер двоичного файла благодаря оптимизированным ядрам, предварительно объединенным активациям нейронов и меньшему количеству зависимостей. Аналогично TF Mobile, общая предварительно обученная модель TensorFlow теоретически может быть преобразована в формат tflite и позже используется для вывода на платформах Android или iOS.

В последних выпусках TensorFlow Lite предоставляет API для переноса выполнения подграфов нейронной сети в внешние библиотеки (называемые делегатами) [34]. Учитывая модель работы нейронной сети, TFLite сначала проверяет, какие операторы в модели могут быть выполнены с предоставленным делегатом. Затем TFLite разбивает Граф на несколько подграфов, заменяя поддерживаемые делегатом подграфы виртуальными "узлами делегата" [35]. С этого момента делегат отвечает за выполнение всех подграфов в соответствующих узлах. Неподдерживаемые операторы по умолчанию вычисляются процессором, хотя это может значительно увеличить время получения результата, поскольку существует накладные расходы для передачи результатов из подграфа. Такими делегатами могут выступать как графический процессор, так и DSP.

2.3 Реализация аппаратного ускорения в мобильных SOC

По сравнению с простыми статистическими методами, ранее применявшимися на смартфонах, модели глубокого обучения требовали огромных вычислительных ресурсов, и поэтому их запуск на процессорах Arm был практически невозможен как с точки зрения производительности, так и с точки зрения энергоэффективности. Первые попытки ускорения моделей искусственного интеллекта на мобильных графических процессорах и DSP были предприняты в 2015 году компаниями Qualcomm, Arm [36] и другими производителями SoC, хотя вначале в основном за счет адаптации моделей глубокого обучения к существу-

ющему оборудованию. Специализированный AI чип начал использоваться в мобильных SoC с выпуском Snapdragon 820/835 с серией Hexagon V6 68x DSP, оптимизированной для нейронных сетей, Huawei Kirin 970 с выделенным блоком NPU, разработанным Cambricon, Samsung Exynos 8895 с отдельным блоком обработки зрения и другие.

Можно выделить четыре поколения мобильных SoC на основе их производительности ИИ, возможностей и даты выпуска:

Поколение 1: все устаревшие чипсеты, которые не могут обеспечить ускорение AI через операционную систему Android, но все же могут быть использованы для ускорения вывода машинного обучения с помощью специальных SDK или библиотек на основе GPU. К этой категории относятся все чипы Qualcomm SoC с Hexagon 682 DSP и ниже, а также большинство чипсетов от HiSilicon, Samsung и MediaTek. Тем не менее, этот подход приводит к заметному снижению производительности на многих SoC с графическими процессорами старого поколения.

Поколение 2: мобильные SoC, поддерживающие Android NNAPI и выпущенные после 2017 года. Они могут обеспечить ускорение только для одного типа моделей (float или quantized) и являются типичными для производительности искусственного интеллекта в 2018 году.

Среди чипов производства Qualcomm к таким относятся данные SOC:

Qualcomm: Snapdragon 845 (Hex. 685 + Adreno 630);

Snapdragon 710 (Hexagon 685);

Snapdragon 670 (Hexagon 685);

А также чипы других производителей.

Поколение 3: мобильные SoC, поддерживающие Android NNAPI и выпущенные после 2018 года. Они обеспечивают аппаратное ускорение для всех типов моделей, а их производительность ИИ типична для соответствующего сегмента SoC в 2019 году.

К этим чипам (среди чипов Qualcomm) относятся более новые модели высшего и среднего сегмента:

Qualcomm: Snapdragon 855+ (Hex. 690 + Adreno 640);
Snapdragon 855 (Hex. 690 + Adreno 640);
Snapdragon 730 (Hex. 688 + Adreno 618);
Snapdragon 675 (Hex. 685 + Adreno 612);
Snapdragon 665 (Hex. 686 + Adreno 610);

Поколение 4: недавно представленные чипсеты с ускорителями ИИ следующего поколения. В настоящее время к этой категории относятся только HiSilicon Kirin 990, HiSilicon Kirin 810 и Unisoc Tiger T710 SoC и новый Qualcomm Snapdragon 865.

Далее будет подробнее рассмотрена одна из реализаций аппаратного ускорения вычислений. Для более детального рассмотрения была выбрана платформа Qualcomm Snapdragon, так как она являет одной из самых распространенных на данный момент. Первый NNAPI драйвер для запуска квантованных нейронных сетей на Hexagon DSPs был представлен Qualcomm в Android O, хотя в то время он не использовался ни в каких коммерческих устройствах и впервые были только более поздние OnePlus 6 и Xiaomi Mi8 со следующей версией Android. В Android P эти драйверы получили дополнительную поддержку для запуска моделей с весами float типа на графическом процессоре Adreno.

2.4 Реализация аппаратного ускорения в SOC Qualcomm

Компания Qualcomm впервые обратилась к проблеме аппаратного ускорения AI(Artificial intelligence) вычислений на устройстве в Snapdragon 820 в мае 2015 года, а также анонсировала свой проприетарный Snapdragon Neural Processing Engine (SNPE) SDK в мае 2016 года, который предлагает ускорение времени выполнения для всех компонентов процессинга Snapdragon. Он предназначен для того, чтобы разработчики могли запускать свои собственные пользовательские модели нейронных сетей на различных устройствах с поддержкой Qualcomm. SDK изначально поддерживался на 17 мобильных процессорах

Snapdragon, начиная с чипов высшего уровня (Snapdragon 845, 835, 820), среднего уроовня (Snapdragon 710, 670, 660, 652, 650, 653, 636, 632, 630, 626 и 625) , а также младшего уровня (Snapdragon 450, 439, 429).

Первый NNAPI драйвер для запуска квантованных нейронных сетей на Hexagon DSPs был представлен Qualcomm в Android O, хотя в то время он не использовался ни в каких коммерческих устройствах и впервые появился только позже в OnePlus 6 и Xiaomi Mi8 со следующей версией Android. В Android P эти драйверы получили дополнительную поддержку для запуска моделей с весами float типа на графическом процессоре Adreno.

Во всех SoC Qualcomm, поддерживающих Android NNAPI, графический процессор Adreno используется для моделей глубокого обучения с числами с плавающей точкой (float), в то время как Hexagon DSP отвечает за квантованный вывод (int8). Следует отметить, что хотя чипы Hexagon 68x/69x все еще представляются как DSP, их архитектура была оптимизирована для вычислений глубокого обучения, и они включают в себя выделенный AI чип,например ускоритель тензорных операций, таким образом, не сильно отличаясь от NPU и TPU, предлагаемых другими производителями. Единственным серьезным недостатком Hexagon DSP является отсутствие поддержки моделей с плавающей точкой (как и в Google Pixel TPU, MediaTek APU 1.0 и Exynos NPU), поэтому последние делегируются графическим процессорам Adreno.

В конце 2018 года компания Qualcomm анонсировала свой флагманский SoC-Snapdragon 855, содержащий восемь пользовательских процессорных ядер Kryo 485 (три кластера функционируют с разной частотой, ядра производные от Cortex-A76), графический процессор Adreno 640 и Hexagon 690 DSP. По сравнению с Hexagon 685, используемым в SDM845, новый DSP получил 1024-битный SIMD (single instruction, multiple data) с двойным числом конвейеров и дополнительным тензорным блоком-ускорителем. Его графический процессор также был обновлен по сравнению с предыдущим поколением, получив в два раза больше ALUs и ожидаемое увеличение производительности на 20% по сравнению с Adreno 630.

2.5 Тестирование реализаций выполнения нейронных сетей

Для тестирования различных реализаций аппаратного ускорения были выбраны устройства оснащенные SOC Snapdragon: Xiaomi A1, mi8 и mi9.

Они основаны на чипах Snapdragon 625, 845 и 855 соответственно. Эти 3 чипсета соответствуют с первого по третье поколениям по AI производительности.

Все 3 устройства поддерживают NNAPI и оснащены системами Android OS >9.0 версии. Тесты производительности проводились в программе AI Benchmark версии 4.0.1.

Первым было протестировано устройство Xiaomi A1. Устройство оснащено довольно устаревшим чипсетом Snapdragon 625 с устаревшим GPU Adreno 506 и Hexagon DSP. Оно поддерживает GPU ускорение операций, но не поддерживает выполнение операций на Hexagon 546 DSP. По умолчанию программа выбрала вариант использования NNAPI без дополнительного использования GPU делигата. Устройство набрало результаты, показанные на рисунке ниже:

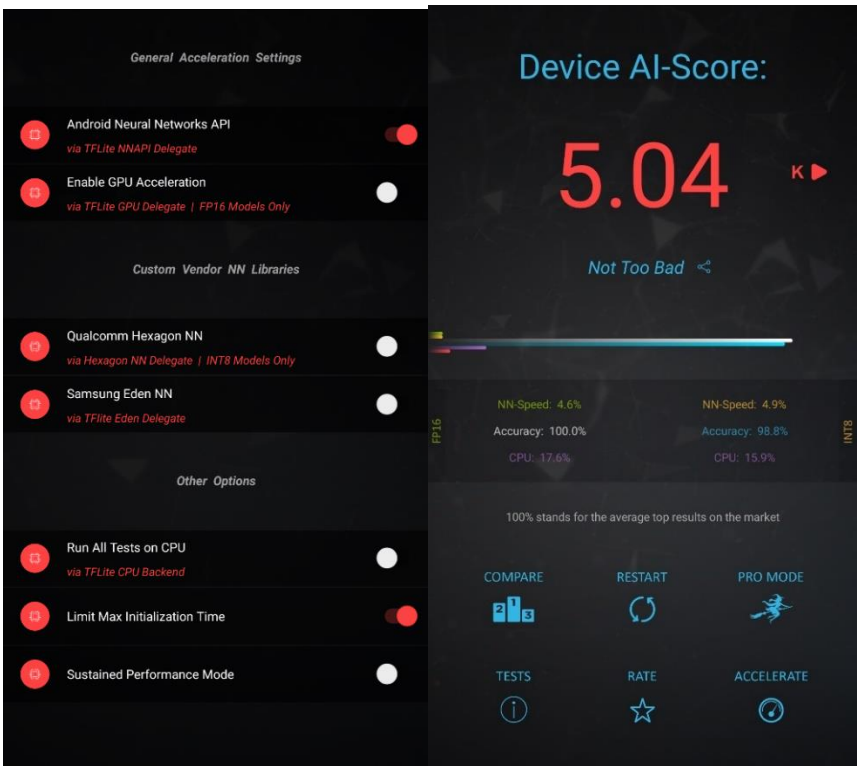

Рисунок 2.3 –Настройки и результаты теста устройства

Результаты теста показали, что выполнение нейронных сетей проходило силами только ядрами процессора, это можно подтвердить, принудительно включив выполнение нейронных сетей на процессорных ядрах. Тесты сетей выполнились Большой точностью, это связанно с тем, что при выполнении нейросети на процессоре не происходит уменьшения размерности весов сети и вычисления могут происходить с большой точностью, но с ухудшением производительности, в том случае, если GPU может выполнять необходимые операции. При принудительном использовании только мощностей процессора с отключением NNAPI получилось добиться чуть лучших результатов, что можно объяснить наличием накладных расходов ресурсов при использовании NNAPI.

Следующий тест (Рисунок 2.4) был проведен с принудительным ускорением мощностями GPU. Но графический процессор в данном SoC оказался слишком слаб. Параллельное выполнение просчета квантинизированной FP16 нейросети на ядрах GPU оказалось медленней (по причине малой производительности) и менее точным (из-за уменьшения размера весов сети до Float16).

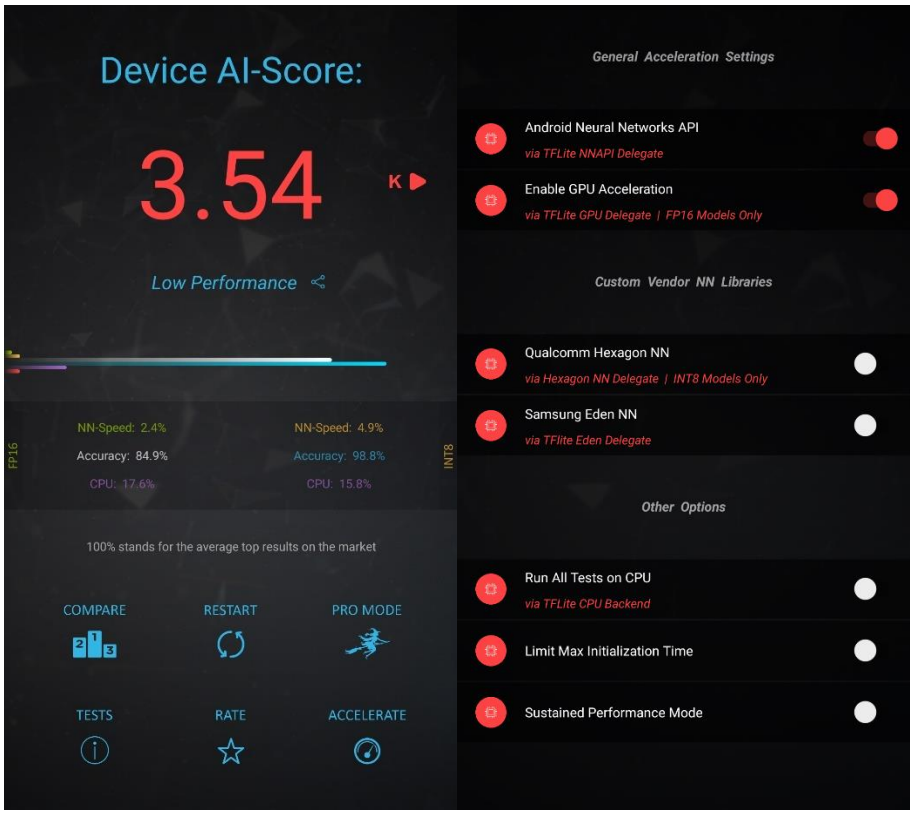

Рисунок 2.4 – Использование аппаратного ускорения GPU для FP16 сетей

Следующим устройством для теста был Xiaomi mi9 с процессором Snapdragon 855, GPU Adreno 640 и DSP Hexagon 690. Данное устройство поддерживает NNAPI и поддерживает ускорение нейронных вычислений любого типа с помощью GPU и DSP, это устройство относится к третьему поколению SoC. Была проведена серия тестов, в ходе которых было выяснено, что для данного устройства оптимальным вариантом по сочетанию скорости выполнения и точности тестов выступает использование и GPU и DSP для всех поддерживаемых ими типов сетей, однако принудительное использование сразу обоих ускорителей делает невозможным использование NNAPI, что увеличивает накладные расходы. Использование лишь процессорных мощностей увеличивает точность прохождения тестов, но уменьшает производительность в более чем 4 раза, что можно наблюдать на рисунках 2.3-2.7 (17 условных единиц, против 80.1 для FP16 сетей и 22 против 99.1 для INT8 сетей). В другом тесте использование NNAPI исключало возможность использовать оба типа ускорителей через принудительные настройки теста, принудительные средства ускорения были выключены, и в таком случае нагрузку по аппаратным ресурсам распределял NNAPI. По результатам теста можно сделать предположение, что NNAPI эффективно применяет DSP ускорение INT8 вычислений, а для вычислений с точностью FP16 можно предположить две гипотезы.

8. Для FP16 был задействован GPU, что видно по результатам точность прохождения теста, сравнимом с первым тестом. Но задействование NNAPI повлекло накладные расходы, и сказалось на скорости выполнения (35,1 условных единиц, против 80.1 для FP16 сетей и 80,6 против 99.1 для INT8 сетей) В большей степени использование NNAPI сказалось на скорость выполнения FP16 сетей с GPU ускорением

9. Так как чипсет устройства относится к третьему поколению ускорения AI, NNAPI принимает решение проводить вычисления с точностью FP16 на DSP. Производительность DSP в операциях с точностью FP16 оказывается меньше, чем у GPU Adreno, но NNAPI принимает решение использовать DSP по причине более низкого энергопотребления и тепловыделения.

Для подтверждения гипотез был проведен еще один тест, отображенный на рисунке 2.6. В нем было задано использование NNAPI совместно с принудительным использованием GPU вычислений для FP16 сетей. Результаты показали большую производительность FP16 в сравнении с предыдущим тестом NNAPI, без принудительных ускорений на GPU. Но всё же результат FP16 оказался хуже использования GPU без NNAPI, что подтверждает существование накладных расходов при использовании NNAPI. Таким образом можно сделать вывод и правильности второй гипотезы: NNAPI автоматически выбирает использование полного DSP ускорения вычислений.

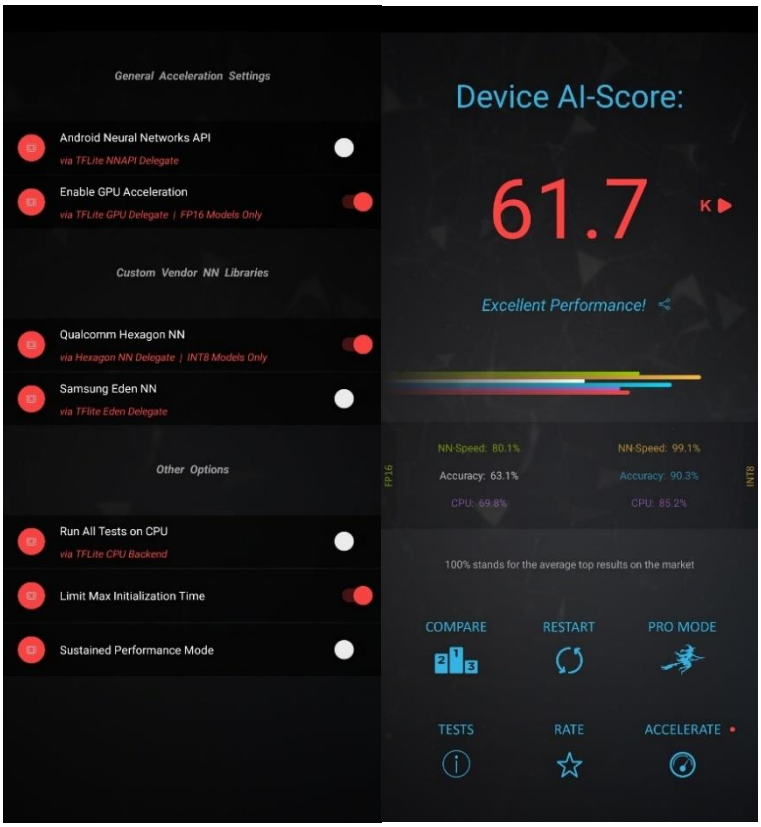

Рисунок 2.5 – Результаты теста mi9 с использованием обоих типов аппаратного ускорения, но без NNAPI.

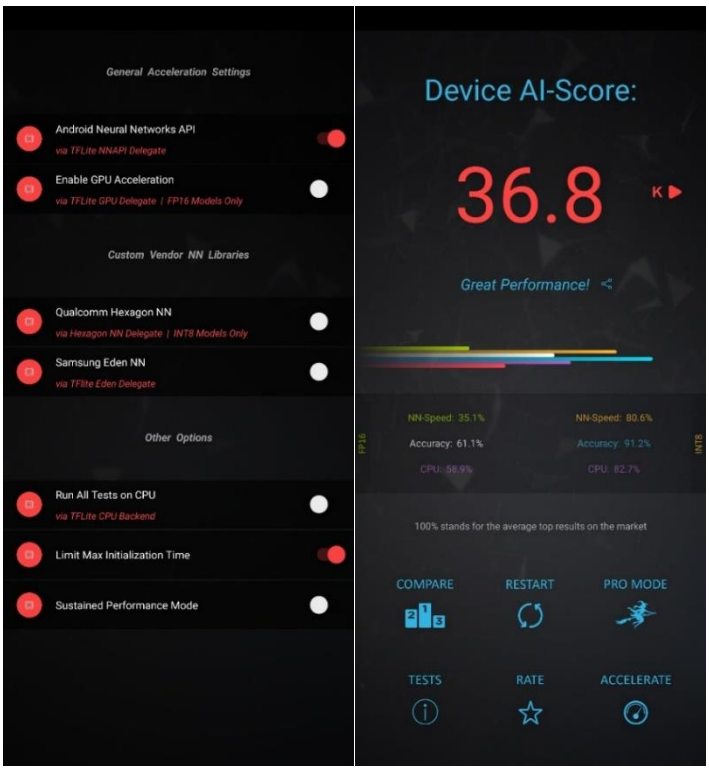

Рисунок 2.6– Результаты теста Mi9 с включением только NNAPI и автоматическим использованием аппаратных ресурсов средствами NNAPI

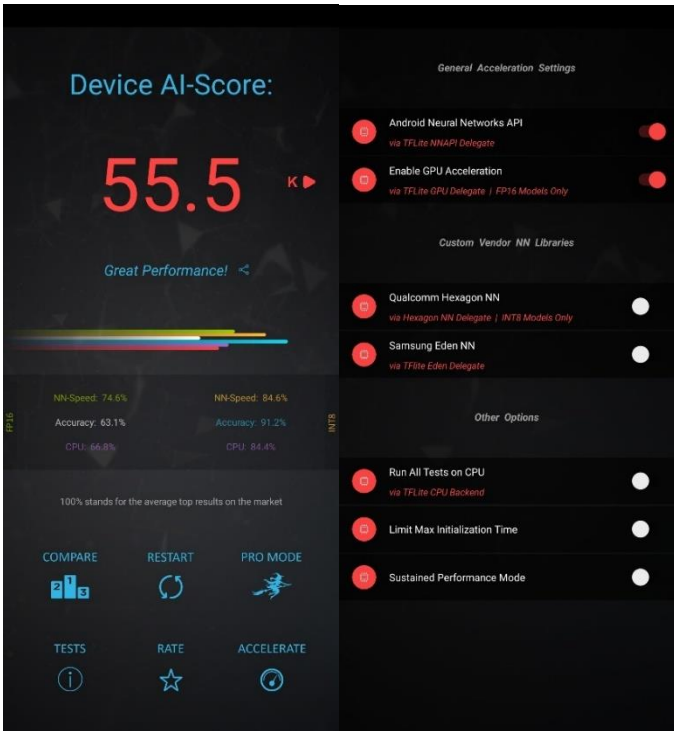

Рисунок 2.7 – Результаты теста mi9 с использованием NNAPI и принудительным дополнительным использованием ускорения на GPU.

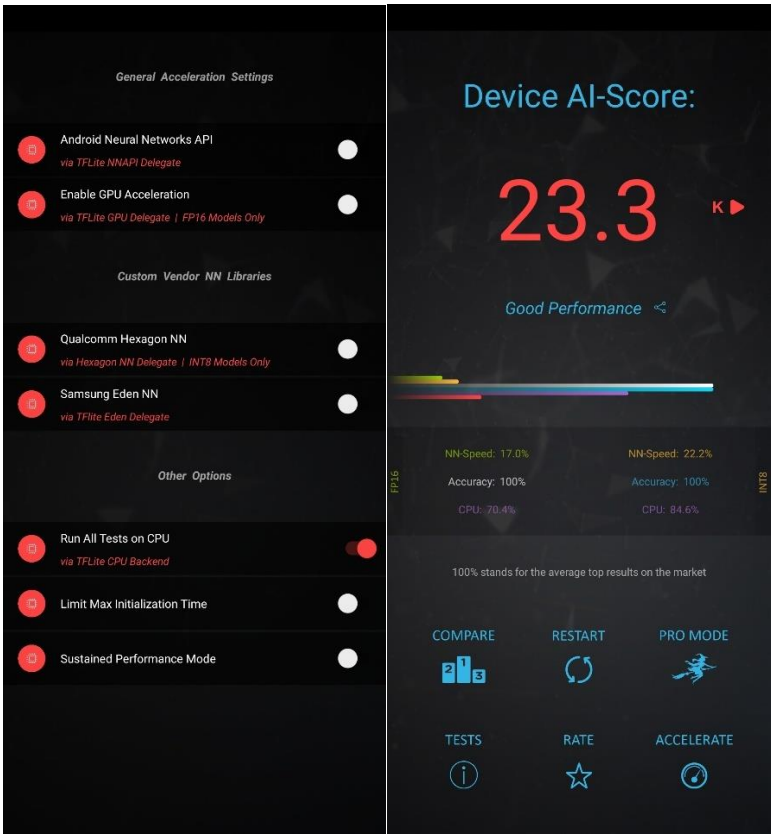

Рисунок 2.8 –Тест mi9 с принудительным использованием только мощностей процессора без использования NNAPI

Последним устройством для теста был Xiaomi mi8 с процессором Snapdragon 845, GPU Adreno 630 и DSP Hexagon 685. Данное устройство поддерживает NNAPI и ускорение нейронных вычислений с помощью GPU и DSP.было проведено несколько тестов, в ходе которых было выявлено значительное отставание от mi9. Различия в производительность при использовании только ядер процессора оставляет от 50 до 300 процентов(11 единиц против 17 для FP16 и 8.3 против 22.2 для int8), а при использовании обоих аппаратных ускорений составляет от 28 до 226 процентов (62.2 единиц против 80.1 для FP16 и 43.8 против 99.1 для int8). Эти результаты подтверждают данные производителя о различиях в производительность процессоров, DSP и GPU.

В целом работа аппаратного ускорения на mi8 (snapdragon 845) работает образом, аналогичным с mi9 (snapdragon 855).

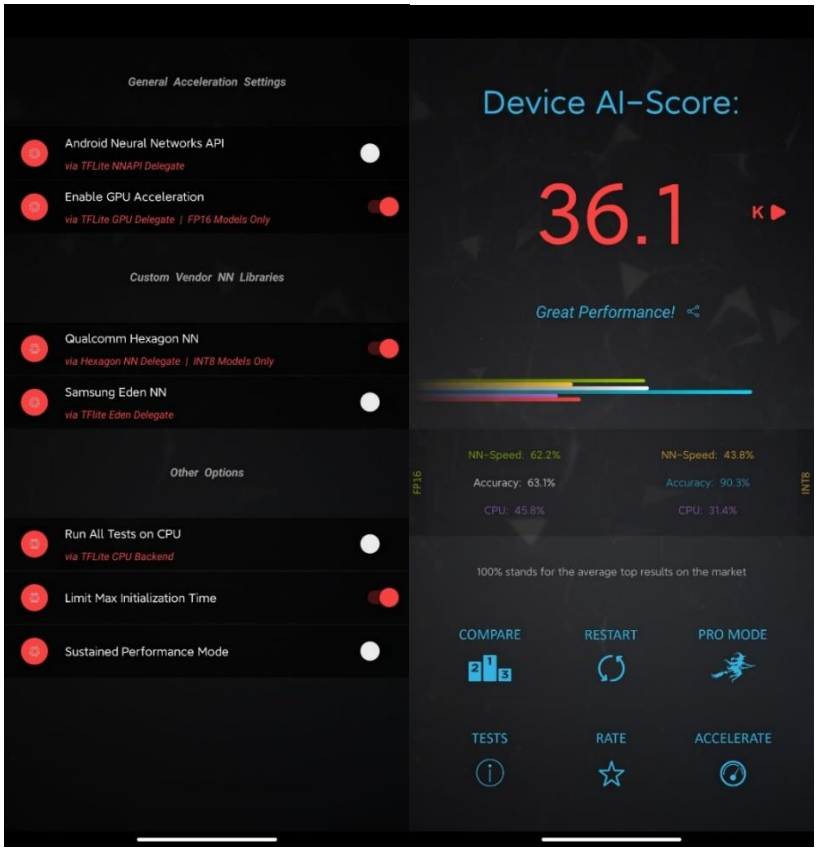

Рисунок 2.9 –Тест mi8 с аппаратными ускорениями GPU и DSP без использования NNAPI

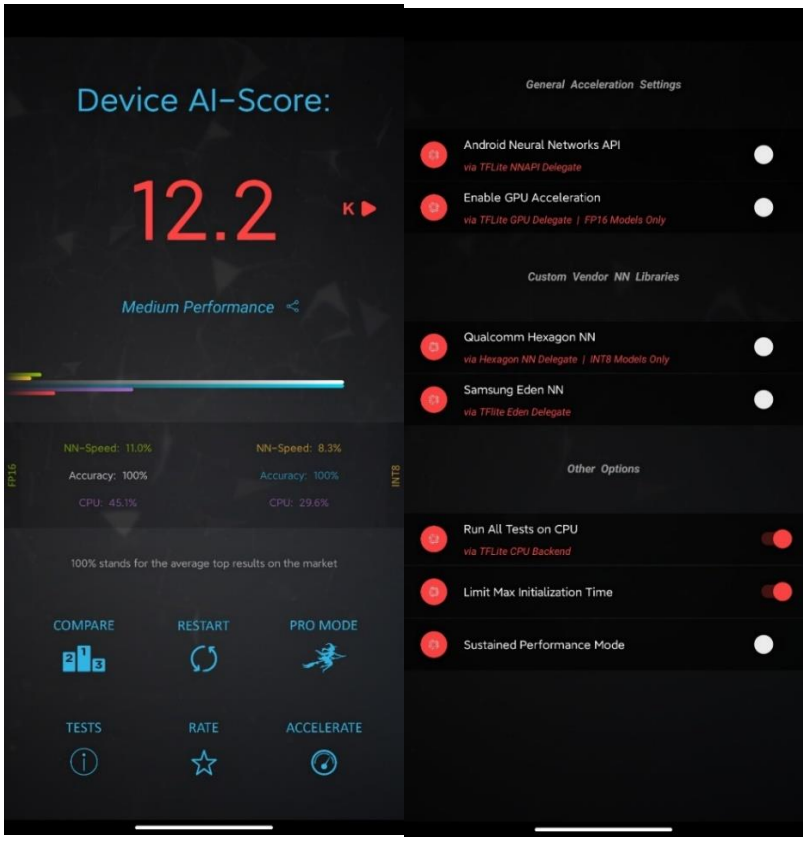

Рисунок 2.10 – Тест mi8 без использования аппаратных ускорений и без NNAPI

По результатам тестов можно составить несколько выводов:

1. Для старых устройств лучше всего подходит выполнение нейросетей силами процессорных ядер.

2. Для современных устройств задействование аппаратного ускорения многократно повышает производительность нейронной сети.

3. NNAPI позволяет самостоятельно и автоматически задействовать доступное аппаратное ускорение.

4. NNAPI имеет накладные расходы, отражающиеся на результатах, но это компенсируется универсальностью API, простотой разработки и оптимальным по энергопотреблению использованием ресурсов устройства.

5. Если тип сети и точность операций, и целевое устройство известны заранее, может быть более эффективно применение проприетарных и отдельных делегатов для аппаратного ускорения, без использования NNAPI.

При разработке мобильных приложений, работающих с нейронными сетями и глубоким обучением, обязательно нужно учитывать особенности целевой платформы, чтобы вычисления могли проводиться с наименьшей необходимой точностью и наибольшей энергоэффективностью. Энергоэффективность становится особа важна, если идет речь о часто используемом сервисе, таком как разблокировка устройства по лицу или фоновая активация цифрового помощника голосовой командой владельца. По заверению производителя чипсетов MediaTek, если взять в качестве примера распознавание лиц, то их APU для аппаратного ускорения вычислений может работать до 20 раз быстрее и снизить энергопотребление в 55 раз по сравнению с процессором.

Также хочется отметить, что на скорость работы с нейронными сетями влияют задержки на инициализацию, которые зависят от реализаций аппаратных ускорителей. Тестированная программа предоставляет данные по этим задержкам и потреблению памяти и учитывает в подсчете конечного результата теста. Также стоит учитывать энергопотребление и нагрев устройства, так как DSP потребляет значительно меньше электроэнергии, чем GPU или ядра процессора.

Ставя перед собой цель разработки нейронной сети для обнаружения ВПО на Android устройствах стоит учитывать вышеописанные особенности мобильной платформы. Оптимальным вариантом будет создание и реализация нейронной сети с использованием специального NNAPI и задействованием DSP ускорения и др. возможностей энергоэффективного ускорения вычислений, так как обнаружение ВПО должно проводиться в фоновом режиме, и при таком сценарии остро встает вопрос энергопотребления разрабатываемого средства. Использование аппаратного ускорения можно добиться, ограничив размер весов сети до INT8, или применив квантование, уменьшая точность уже обученной сети.

3 ПОДХОДЫ, ИСПОЛЬЗУЮЩИЕСЯ В СРЕДСТВАХ ДЛЯ ВЫЯВЛЕНИЯ ВПО

В данном разделе кратко рассматриваются различные подходы к анализу ВПО, архитектуры и примеры существующих решений, предлагается и описывается гибридный подход для анализа приложений.

3.1 Подходы к анализу приложений

Мобильные вредоносные программы требуют подробного анализа. Одним из критических моментов мобильных телефонов является то, что они представляют собой систему, постоянно получающую события, которая позволяет вредоносным программам реагировать на приходящие SMS, установку положений и т. д., повышая сложность автоматизированных методов анализа вредоносных программ. Кроме того, приложения могут использовать сервисы и действия и интегрировать различные языки программирования (например, Java и c++) в одном приложении. Основываясь на способах классификации приложений, анализ может быть статическим, динамическим и гибридным, объединяющий статический и динамический анализ. Каждый из этих анализов имеет свои сильные и слабые стороны.

Статический анализ сканирует части приложения, фактически не выполняя их. Этот метод включает в себя сигнатурный анализ, проверку разрешений приложения и анализ его отдельных компонентов. Анализ на основе сигнатур ищет особенности и выделяет отличительные знаки для идентификации конкретных вредоносных программ. Следовательно, он не может распознать вариацию или неопознанную вредоносную программу. Анализ на основе разрешений распознает запросы приложения для распознавания вредоносных программ. Методы, основанные на анализе компонент, декомпилируют приложение для анализа и проверки байт-кодов значимых компонентов (активити, сервисы и т.д.). Основными недостатками статического анализа являются отсутствие реального выполнения и условий, которые могут возникнуть только во время выполнения

программы. Кроме того, существуют проблемы в случае обфускации кода или динамической загрузки кода.

Разновидности статического подхода

1. Обнаружение вредоносных программ на основе сигнатур: существующие коммерческие антивирусные программы используют подходы к обнаружению вредоносных программ на основе сигнатур. Он извлекает интересные синтаксические или семантические паттерны, особенности [82] и создает уникальную сигнатуру, соответствующую этой конкретной вредоносной программе. Методы, основанные на сигнатурах, не справляются с неизвестными вариантами уже существующих или еще неизвестных вредоносных программ.

2. Анализ на основе разрешений: запрос разрешения на доступ к конфиденциальным ресурсам является центральным элементом модели безопасности Android. Ни одно приложение по умолчанию не имеет никаких разрешений, которые могут повлиять на безопасность пользователя. Обнаружение опасного запроса разрешения не является достаточным для объявления вредоносного приложения, но тем не менее, сопоставление запрошенных разрешений и используемых разрешений является важным методом идентификации. Sanz Borja et al. [37] применяли теги <uses-permission> и <uses-features>, присутствующие в AndroidManifest.xml для обнаружения вредоносных приложений. Авторы использовали алгоритмы машинного обучения Naive Bayes, Random Forest, J48 и Bayes-Net для набора данных из 249 вредоносных программ и 357 доброкачественных приложений. В работе [38] авторы сопоставили запрошенные и используемые разрешения из манифеста и соответствующего им API в байт-коде dalvik. Сопоставленные атрибуты были использованы с алгоритмами машинного обучения для 125 249 вредоносных программ и набора данных доброкачественных приложений.

3. Анализ байт-кода Dalvik: байт-код Dalvik является семантически богатым, содержащим информацию о типах, таких как классы, методы и инструк-

ции. Информация о типе может быть использована для проверки поведения приложения. Детальный анализ, основанный на контроле и потоке данных, дает представление об опасных функциях таких как утечка конфиденциальности.

4. Анализ потока управления байт-кодом определяет возможные пути, которые может пройти приложение во время выполнения. Байт-код Dalvik содержит элементы jump, branch, которые изменяют порядок выполнения. Для облегчения дальнейшего анализа генерируется внутрипроцедурный (т. е. внутри одного метода) или межпроцедурный (т. е. охватывающий все методы) граф потока управления байт-кода (CFG).

5. Taint-анализ еще один тип метода анализа потока данных для идентификации помеченных переменных, содержащих конфиденциальную информацию. Например, taint анализ может выявить утечку конфиденциальности, которая может быть использована для кражи конфиденциальной информации пользователей приложений [39].

Методы статического анализа и обнаружения быстры, но они не справляются с зашифрованными или полиморфными ВПО. Метод динамического анализа включает выполнение приложения либо на виртуальной машине, либо на физическом устройстве. Во время выполнения ПО за поведением приложения наблюдают и могут вмешиваться в его выполнение. Динамический анализ ближе в анализу реального поведения программы, чем статический анализ. Пути выполнения кода во время анализа, являются подмножеством всех доступных путей. Основной целью анализа является достижение высокого уровня покрытия кода, поскольку каждое возможное событие должно быть активировано для наблюдения за любым возможным вредоносным поведением [31]. Основные недостатки динамического анализа заключаются в том, что динамический анализ требует значительных ресурсов по сравнению со статическим, что препятствует его распределению на сотовых телефонах. Кроме того, динамический анализ малоэффективен при низком охвате кода. В последнее время вредоносное ПО пытается распознать выполнение на эмуляторе и другие платформы динамического

анализа и воздерживаются от демонстрации их полезной нагрузки. Следовательно, некоторые динамические анализаторы уязвимы для уклонения ВПО от анализа.

1. обнаружение аномалий на основе профилей приложений: вредоносные приложения могут чрезмерно использовать аппаратные ресурсы. Ряд параметров, таких как использование процессора, статистика использования памяти, сетевой трафик, расход батареи и системные вызовы для нормальных и вредоносных приложений, собираются в системе Android. Методы автоматического анализа наряду с методами машинного обучения используются для различения аномального поведения

2. Обнаружение вредоносного поведения: конкретные подозрительные действия, такие как утечка конфиденциальных данных, отправка SMS/электронных писем, голосовые вызовы без согласия пользователя, могут быть точно обнаружены путем мониторинга конкретных объектов, представляющих интерес.

Третий тип анализа называется гибридным анализом, который объединяет статический и динамический анализ. Благодаря комбинации статического и динамического анализа, из анализа можно извлечь больше функций. Использование соответствующих функций для анализа даст гораздо лучший результат. Гибридный анализ может быть использован для преодоления недостатков статического и динамического анализа, таких как сложность анализа при методах запутывания кода и ресурсоемкая «песочница» динамического анализа. Хотя все методы обычно используются при анализе и обнаружении вредоносных программ, эти методы также применимы в качестве основы для анализа и обнаружения мобильных ботнетов. Следовательно, гибридный анализ является оптимальным выбором при обнаружении ботнетов.

3.1.1 Архитектуры анализирующих решений

Анализ вредоносных программ может быть развернут в нескольких местах, как на конечном устройстве, так и с применением облачных технологий.

Анализ на устройстве

Детальный анализ очень ограничен на мобильном устройстве по сравнению с защитой от вредоносных программ на стационарных компьютерах или ноутбуках. Решением могут стать упрощенный анализ на основе анализа компонентов и разрешений приложения. Ниже приведены некоторые ограничения по защите устройств от вредоносных программ.

– Антивирусные приложения работают как обычное приложение без каких-либо специальных привилегий. Оно попадает под изоляцию процессов предусмотренную в Android и не может напрямую сканировать память других приложений и читать их файлы.

– Android позволяет выполнять фоновую активность сервисам и службам. Но при нехватке аппаратных ресурсов такие сервисы могут быть выгружены. Также на выгрузку фоновых активностей могут повлиять другие приложения, в том числе и ВПО.

– Без получения root прав антивирусное ПО не может следить за системными вызовами, проводить мониторинг файловой системы и сети

– Без root прав ПО не сможет самостоятельно удалить ВПО и потребуется вмешательство пользователя.

Распределенная архитектура

Первичный анализ может быть выполнен на устройстве, детальный и вычислительно дорогостоящий анализ может быть выполнен на удаленном сервере. В случае обнаружения аномалий параметры использования ресурсов собираются на стороне клиента и отправляются обратно на удаленный сервер для детального анализа. Результаты могут быть окончательно отправлены обратно в устройство [40], [41]. Однако необходимость постоянного соединения с сетью накладывают ограничения и дополнительные расходы. В случае недоступности сетевых ресурсов анализ на конечном устройстве может защитить пользователя. [42].

На основе облачной архитектуры

Некоторые методы анализа требуют сбора и анализа большого количество информации. Такие автоматизированные решения для глубокого анализа требуют много вычислительной мощности и памяти. Из-за этого они обычно разворачиваются вне устройства [43].

3.2 Сравнение существующих средств анализа

Многие инструменты были разработаны для извлечения различных динамических и статических функций из файлов пакета приложений для Android (APK). В исследовании [44] была представлена Kirin, службу безопасности для Android, которая предоставляет правила безопасности для приложений во время установки. В другой работе [45] было предложено машинное обучение с использованием моделей байесовской классификации для обнаружения мобильных вредоносных программ наряду со статическим анализом кода с использованием функций, таких как разрешения, вызовы API и системные команды Linux.

Между тем, динамический анализ не проверяет исходный код, а выполняет приложения, отслеживает и регистрирует каждую соответствующую операцию выполнения. В исследовании [40] было предложено Andromaly, приложение для обнаружения вредоносных программ на базе Android, с использованием машинного обучения. Он был протестирован с использованием 88 параметров для описания поведения приложения, такого как использование процессора и памяти, энергопотребление, загрузка сети и количество запущенных процессов. Хотя этот подход дает хороший результат, он не способен обнаружить вредоносное ПО, которое избегает системных вызовов с правами root.

Существует множество работ, описывающих применение **статических методов** обнаружения ВПО, ниже приведены несколько из них.

Исследователь W. Li [46] реализовал систему идентификации вредоносных программ, использующую метод глубокого обучения, который использует как опасные вызовы API, так и рискованные комбинации разрешений в качестве

функций для построения модели DBN, которая может автоматически распознавать вредоносные программы. Результаты их тестов показали, что модель получила точность 90%.

Yi Zhang [47] разработал DeepClassifyDroid на базе CNN. Предлагаемая система выполняет статический анализ для достижения пяти различных характеристик. Система, протестированная на наборе данных, содержит в общей сложности 10 770 приложений, в том числе 5546 вредоносных и 5224 доброкачественных. Результаты показали, что точность подхода составила 97,4%.

Ниже приведены несколько работ по **динамическому анализу** для выявления ВПО.

H. Liang [48] разработал методы обработки естественного языка для анализа вредоносных программ Android. Они разработали модель, которая рассматривает последовательности системных вызовов как тексты и рассматривает функцию обнаружения вредоносных программ как извлечение темы текста. Предлагаемый подход был протестирован на наборе данных из 14 231 приложения. Результаты показали, что точность была 93,16%.

S. Hou и соавторы [49] предложил автоматическую систему обнаружения вредоносных программ для Android Deep4MalDroid. Представление извлеченных системных вызовов Linux в виде графов и группы автокодировщиков (SAEs) была применена для сканирования общих шаблонов вредоносных программ и, таким образом, для определения новых неизвестных вредоносных приложений. Экспериментальные результаты на наборе данных из 3000 приложений показали, что наилучшая достигнутая точность составила 93,68%.

Ниже рассматриваются системы **гибридного анализа.**

Z. Yuan разработал Droid-Sec [50], который является первой попыткой применить глубокое обучения в области обнаружения вредоносного ПО для Android. Авторы используют многочисленные функции для создания моделей сетей DBN, которые способны отличить вредоносные программы от доброкачественных. При обучении модели на выборке из 250 вредоносных и 250 доверенных приложений результат обнаружения оказался на уровне 96%.

Z. Yuan и его команда разработали Droid-Detector [51] на базе DBN. Предлагаемый подход был протестирован на большом несбалансированном наборе данных, включающем 20000 образцов доброкачественных и вредоносных программ. Результаты показали хорошую производительность DBN с точностью 96,76%.

R. Vinayakumar и его команда [52] предложили использовать Long Short-Term Memory сети (LSTM), которые представляет собой особый вид рекуррентной нейронной сети. Авторы извлекли динамические и статические характеристики и использовали набор данных из 1738 приложений для оценки производительности модели, которая достигла 93,9% в динамическом анализе и 97,5% в статическом анализе.

Но, к сожалению, исходные коды рассмотренных решений не были выложены в открытый доступ.

3.3 Применение систем глубокого обучения для динамического анализа

Разработчики системы DL-Droid [53], представили систему глубокого обучения для обнаружения вредоносных приложений для Android с помощью динамического анализа с использованием генерации входных данных с учетом предыдущих состояний. В исследовании [53] представлены эксперименты, выполненные с более чем 30 000 приложениями (доброкачественными и вредоносными) на реальных устройствах. Кроме того, исследователями были также проведены эксперименты по сравнению производительности обнаружения и покрытия кода совместно с методом генерации пользовательского ввода с подходом без сохранения состояния с использованием системы глубокого обучения. Для создания входных данных для обучения модели динамического анализа разработчики использовали DynaLog [54].

DynaLog предназначен для автоматического установки большого количества приложений для Android, их последовательного запуска с помощью эмулятора (виртуального устройства Android "AVD") или реального телефона, а также

регистрации и извлечения нескольких динамических функций (т. е. вызовов API, действий/событий). С динамическим анализом Android-приложений, генерация тестового пользовательского ввода необходима для того, чтобы обеспечить достаточное покрытие кода, чтобы вызвать вредоносное поведение. DynaLog способен использовать различные методы генерации тестовых входных данных: без учета состояния (случайный ввод) (с помощью разработчиков Monkey tool), и с отслеживанием состояния (с помощью DroidBot), и гибридный (который сочетает в себе инструменты генерации входных данных без состояния и с сохранением состояния Alzaylaee). Фактически, большинство существующих платформ динамического анализа для обнаружения вредоносных программ на Android используют подход без сохранения состояния (основанный на инструменте Monkey tool). Разработчики DL Droid использовали только случайную и генерацию входных данных с учетом состояний.

3.3.1 Сравнение глубоко обучения с другими методами машинного обучения.

В DL Droid используется бинарный классификатор, написанный с использованием H2O, который в настоящее время поддерживает только многослойный персептрон (MLP). Для оценки эффективности различных классификаторов в системе подсчитывается матрица ошибок и сравнивается производительность между DL Droid и семью популярными подходами машинного обучения. Классификаторы, выбранные для сравнения: метод опорных векторов (SVM Linear), машина опорных векторов с ядерными функциями (SVM RBF), Наивный байесовский классификатор (NB), простой логистический классификатор (SL), частичные деревья решений (PART), случайный лес (RF) и дерево решений J48.

TP –вероятность верно положительного результата

TN–вероятность верно отрицательного

FP–вероятность ложно положительного результата

FN–вероятность ложно отрицательного результата

Точность задается следующим выражением:P = T P / (T P + F P)

Для сравнения с методами машинного обучения исследователями была выбрана модель персептрона с тремя слоями по 200 нейронов в каждом, как наиболее результативная. Результаты сравнения можно увидеть в таблице 3.1.

Таблица 3.1- Результаты сравнения методов машинного обучения.

	TP	**TN**	**FP**	**FN**	**P**
Naive Bayes	0.62	0.855	0.145	0.38	0.765
SL	0.761	0.933	0.067	0.239	0.87
SVM Linear	0.758	0.938	0.062	0.242	0.872
SVM RBF	0.758	0.944	0.056	0.242	0.876
J48	0.855	0.954	0.046	0.145	0.917
PART	0.861	0.955	0.045	0.139	0.92
RF	0.88	0.971	0.029	0.12	0.938
DL(200,200,200)	0.9776	0.9086	0.0914	0.0224	0.9482

3.4 Концепция предлагаемого к разработке средства

Учитывая все вышеописанные методы и подходы к анализу был разработан концепт системы обнаружения вредоносного ПО на платформе андроид.

Система представляет из себя систему гибридного анализа с распределённой архитектурой и с использованием глубокого обучения. Схему предложенного решения можно наблюдать на рисунке 3.1.

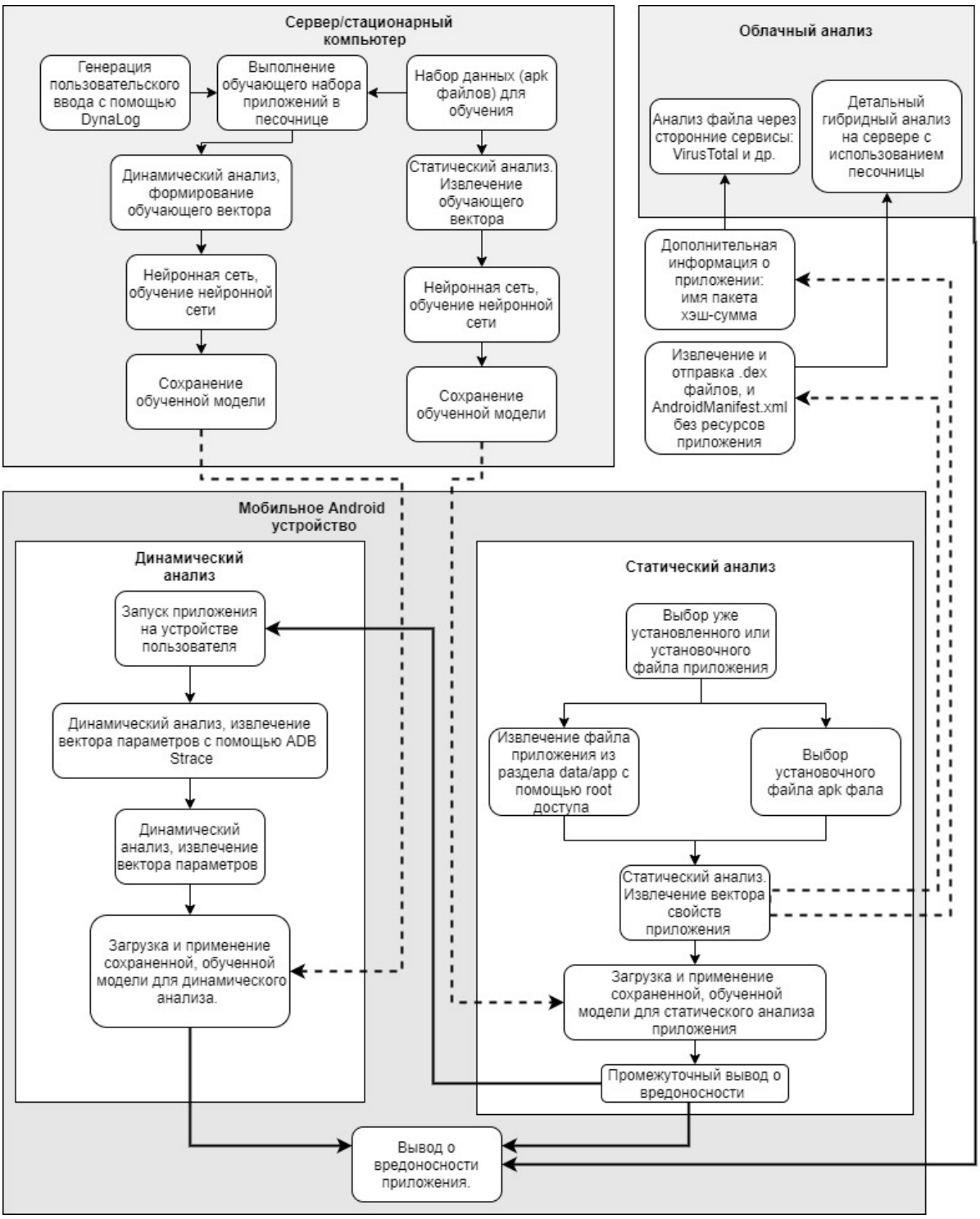

Рисунок 3.1. –схема предложенного решения

Архитектурно решение состоит из трех основных частей:

1. Компьютер или сервер для подготовки и обучения моделей
2. Мобильное устройств, на котором происходит основной анализ приложений
3. Удаленный сервер и сторонние интернет-ресурсы по анализу приложений

3.4.1 Предварительная подготовка моделей

Первая часть анализирующей системы выполняется на сервере или компьютера разработчика. В начале производится сбор наборов установочных файлов для извлечения данных для обучения искусственных нейронных сетей. Далее происходит статический и динамический анализ приложений.

Во время статического анализа из пакета приложения извлекаются такие параметры, как:

1. Использованные разрешения
2. Наличие обращения в БД
3. Наличие обфускации
4. Список использованных вызовов API
5. Список строковых данных, найденных в коде приложения
6. Распределение вызовов API по категориям.

Из извлеченных данных формируется вектор, состоящий из булевых и числовых показателей. Полученный массив векторов поступает в вход в нейронную сеть для ее обучения и сохранения в виде готовой и обученной модели.

В процессе динамического анализа происходит запуск ПО на эмуляторе Android устройства. В процессе работы приложения генерируется список задействованных API вызовов и использованных функциях. Для увеличения покрытия кода тестируемых приложений происходит генерация пользовательского ввода. Для данной цели подходит подход, использованный разработчиками DL Droid, описанный в предыдущем подразделе. После запуска набора приложений и сбора всех данных списки последовательностей API и использованных возможностей поступают на вход во вторую нейронную сеть, для ее обучения.

На выходе из первого этапа анализа имеются две обученных и экспортированных нейронных сети, готовых к использованию на мобильном устройстве.

3.4.2 Применение обученных искусственных нейронных сетей на мобильном устройстве.

По окончанию первого этапа полученные модели попродаются на мобильное устройство пользователя. Далее по выбору пользователя, либо автоматически при установке нового приложения запускается процесс анализа приложения. Также пользователь может выбрать уже установленное приложение, так как после установки изначальный установочный файл приложения по остается храниться в разделе /data и при наличии root прав есть возможность получить к нему доступ для анализа. Затем происходит статический анализ приложения: из установочного файла извлекаются параметры, описанные в пункте 3.4.1, и подаются на вход в ранее обученную нейронную сеть для классификации. В случае обнаружения ВПО происходит уведомление пользователя, для совершения действий по удалению или отказу от установки приложения. В случае не обнаружения ВПО происходит вторая стадия анализа – динамический анализ. В ходе динамического анализа происходит запуск приложения на устройстве, сбор и логирование использованных API с помощью средств ADB Strace [55] и последующий анализ цепочки вызовов с помощью второй обученной нейронной сети. Полученные результаты обоих анализов в дальнейшем будут отражаться в выводе о вредоносности приложения.

3.4.3 Облачный анализ приложения

Также во время статического анализа извлекается метаинформация о пакете приложения, такая как SHA-256, что позволяет с малыми затратами трафика автоматически обратиться в сетевым анализаторам, таким как VirusTotal и при необходимости дополнительно отправить файл classes.dex, содержащий код приложения без дополнительны ресурсов (таких как различные медиа файлы, данные и т.д.) для его детального анализа.

Данная архитектура предлагаемого решения позволяет использовать преимущества сразу многих подходов:

1. Первичное использование статического анализа позволит обнаруживать подавляющее большинство ВПО и отбрасывать дальнейшую необходимость в динамическом анализе.

2. Применение дополнительного динамического анализа позволит обезопасить устройство пользователя в случае использования ВПО методов сокрытия и противодействия обнаружению статическим анализатором

3. При наличии доступа в интернет появляется возможность быстрого первичного анализа приложения

4. Комбинация различных источников информации о вредоносности предложения дает возможность составить максимально точную оценку приложения.

5. Подход с использованием нейронных сетей дает преимущество в качестве обнаружения. А наличие аппаратных возможностей по ускорению вычислений нейронных сетей делает этот подход быстрым и энергоэффективным на современных мобильных устройствах.

6. Использование описанных подходов дает возможность системе обнаружения ВПО работать автономно (без доступа в сеть интернет и без доступа к полноценному компьютеру для проведения анализа), при этом при возможности или необходимости может совершать облачный анализ.

4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ МАКЕТА СИСТЕМЫ, РЕАЛИЗУЮЩЕЙ ПРЕДЛАГАЕМЫЙ ПОДХОД

В данном разделе описывается реализация разработанного подхода для обнаружения вредоносных Android-приложений.

Для всех дальнейших операций с установочными файлами приложений используется пакет Androguard[56]. Для подготовки данных, построения и обучения модели используются пакеты numpy[57], sklearn[58], tensorflow[60], keras[59].

Для практической реализации была выбрана часть ранее предложенного решения, так как она является основной в гибридной системе обнаружения ВПО. Реализация статического анализатора отображает и раскрывает основные особенности предлагаемого подхода выполнения нейронных сетей на мобильном устройстве Android. Смеху разработанной системы можно наблюдать на рисунке 4.1.

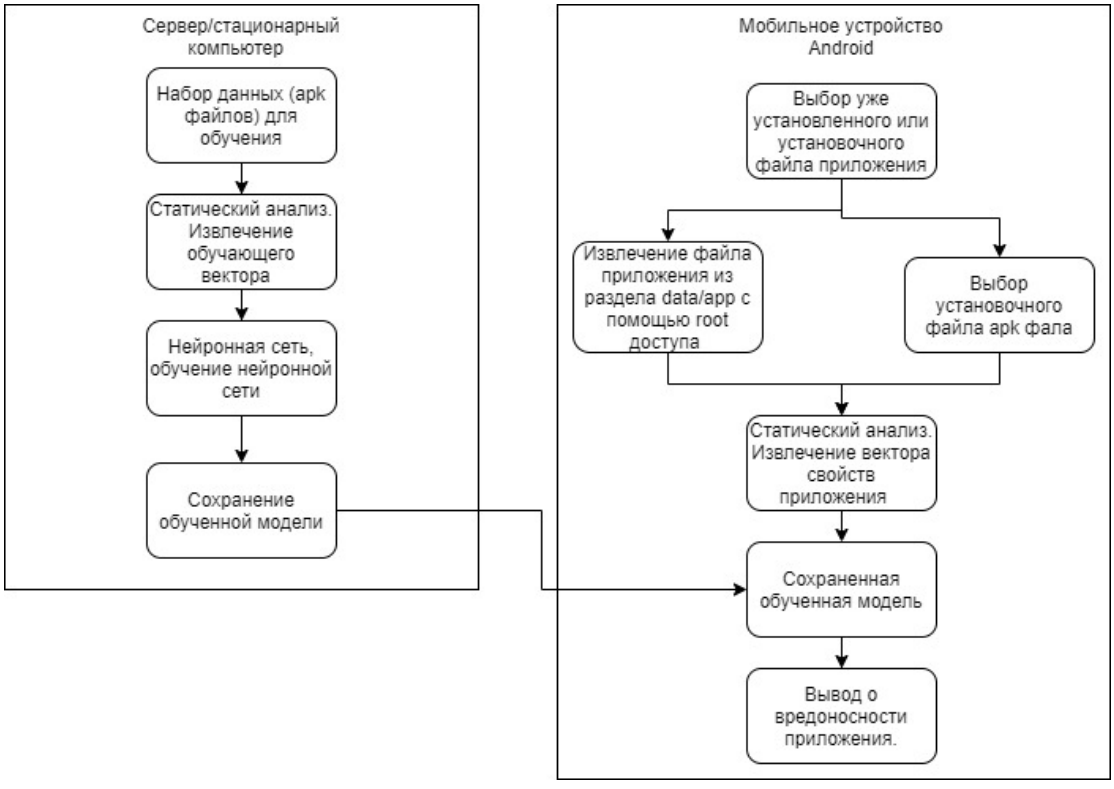

Рисунок 4.1 – Схема разработанной системы статического анализа приложений.

4.1 Подготовка данных для анализа

Важным условие для успешного обучения нейронной сети является правильно составленная и большая выборка данных.

4.1.1 Используемая выборка

Для бучения сети был составлен набор вредоносных и легитимных приложений. В качестве набора легитимных приложений был использована коллекция случайных приложений, найденная в открытом доступе. [61]. Коллекция представляет из себя архив, содержащий более 7100 легитимных установочных файлов приложений случайных категорий, общим объемом 154 Гб. Также был организован сбор 200 легитимных популярных приложений с интернет-ресурса ApkPure [62].

К сожалению, подавляющее большинство источников вредоносных приложений используемых для схожих целей в других работах стали недоступны или были убраны из открытого доступа. Для составления вредоносной выборки использовалось несколько источников:

- 298 образцов с проекта Android Malware Samples [63]
- 1929 образцов различных семейств андроид ботнетов из Android Botnet dataset[64]
- Более 250 образцов из коллекции проекта AndroidMalware_2019 [65]
- Более 200 образцов из коллекции проекта AndroidMalware_2020 [66]
- Более 100 образцов из коллекции проекта Android-Malwares [67]
- Более 2000 образцов из коллекции проекта Drebin

Таким образом выборка оказалась недостаточно сбалансированной (около 4000 вредоносных образцов и более, чем 7100 легитимных), что было исправлено использованием только части легитимных приложений. Общее количество образцов может оказаться недостаточным, но при неудовлетворительных результатах обучения сети можно нарастить выборку вредоносного ПО искусственным путем.

4.1.2 Извлечение параметров из приложения

Из открытых проектов, таких как Drebin[68] и RevealDroid [69] были взяты, адаптированы и использованы списки вызовов API, соотношение API вызова и категории действия, список специфичных разрешений для вредоносных приложений, список специальных распространенных системных команд.

Для статического анализа из приложений были извлечены следующая информация

1. Использованные разрешения. Были составлены списки разрешений, относящихся к категориям Special, Dangerous, Signature длинной в 283 элемента. Данные списки разрешений были преобразованы в вектора содержащие булевы значения.

2. Наличие обращений в БД было взято средствами пакета Androguard функцией get_regex_strings() осуществляющей поиск в текстовых строках приложения с помощью регулярных выражений. Данное регулярное выражение приведено ниже:

```
    DB_REGEX                    =                    '((SELECT|se-
lect)\s[\w\*\)\(\,\s]+\s(FROM|from)?\s[\w]+)|          '          \
           '((UPDATE|update)\s[\w]+\s(SET|set)\s[\w\,\'\=]+)|  '  \
           '((INSERT|in-
sert)\s(INTO|into)\s[\d\w]+[\s\w\d\)\(\,]*\s(VALUES|val-
ues)\s\([\d\w\'\,\)]+)|                    '                    \
           '((DELETE|delete)\s(FROM|from)\s[\d\w\'\=]+)'
```

3. Наличие обфускации ппрверялось средствами функции is_ascii_obfuscation() из пакета Androguard

4. Список использованных в приложении вызовов API сравнивается со списком часто встречаемых и значимых в ВПО API, такими как printStackTrace(), Cipher(),getSimCountryIso() и другими.

5. Список строковых данных, найденных в коде приложения, сравнивается со списком строк из проекта RevealDroid.

6. Распределение вызовов API по категориям определяется поиском API по специальному словарю из более чем 9000 API сигнатур, содержащих используемый имя класса, имя метода и его категорию загружаемой из текстового файла. Пример данных из этого файла приведен ниже:

```
UNIQUE_IDENTIFIER % com.android.internal.telephony.sip.SipPhoneBase % getDeviceSvn
LOCATION_INFORMATION % com.android.server.location.GeocoderProxy % getFromLocation
```

Списки найденных API, строковых значений и найденные API по категориям были просуммированы по количеству одинаковых элементов, таким образом формируя вектор, состоящий из чисел. Данные после извлечения из образцов сохраняются в CSV формате (рисунок 4.2). В дальнейшем данный вектор будет конкатенирован с вектором, содержащим информацию о разрешениях, наличие обращений в БД, обфускации и поступит на вход в нейронную сеть.

	A	B	C	D	E	F	G
1	APP_Name	folder	android.permission.BIND_WALLPAPER	android.p	android.p	android.p	android.p
2	0029.apk	viruses/	0	0	0	0	0
3	021d55c415ff951c8	viruses/	0	0	0	0	0
4	03122ade6371753	viruses/	0	0	0	0	0
5	03eef67a161fc253	viruses/	0	0	0	0	0
6	0a8298d77996ec1	viruses/	0	0	0	0	0
7	0F182524C0FE8FF9	viruses/	0	0	0	0	0

Рисунок 4.2 – Формат файла с набором данных

Таким образом на вход в нейронную сеть поступает вектор, содержащий в себе 538 чисел. По меркам методов глубоко обучения — это довольно маленький объем входных данных, сопоставимый с изображением размера около 20 на 20 точек. Но такая небольшая размерность извлекаемого вектора может быть полезна, если идёт речь о применениях метода на мобильных устройствах с ограниченными вычислительными мощностями и питанием.

4.1.3 Построение искусственной нейронной сети.

В данной работе в качестве программного пакета для построения нейронных сетей был выбран пакет Keras, исполняющийся поверх более низкоуровневого пакета разработки нейронный сетей TensorFlow.

Сети прямого распространения (Feedforward neural network — искусственные нейронные сети, в которых сигнал распространяется строго от входного слоя к выходному. В обратном направлении сигнал не распространяется. Проблема нейронной сети прямого распространения в том, что при большом количестве слоев и нейронов возникает громадное количество связей и параметров, которые возникают по причине того, что каждый нейрон слоя нейронной сети прямого распространения связан со всеми нейронами предыдущего и последующего слоёв, такие слои ещё называют полносвязными слоями (англ. fully connected layer). Алгоритм обратного распространения ошибки (англ. backpropagation) – алгоритм обучения таких сетей работает крайне медленно вследствие такого большого количества связей.

Принцип функционирования алгоритма обратного распространения ошибки основан на вычислении градиента функции потери (англ. loss function), которая вычисляется на основе выходного значения сети и ранее заданного значения в наборе данных. Вычислив градиент, алгоритм определяет, как надо подкорректировать значения весов и смещений нейросети, чтобы достичь минимального значения функции потери, фактически вычисляя минимум функции потерь.

В разработанной системе для решения бинарной классификация используется многослойный перцептрон. Он относится к многослойным сетям прямого распространения. Для построения сети использовался пакет Keras, который выполняться поверх TensorFlow. Была составлена базовая модель класса `Sequential` (нейронной сети задается последовательным добавлением слоев) состоящая из трех полносвязных слоев. Первые два слоя имеют по 538 нейронов, что соответствует размерности входящего вектора. В последнем полносвязном слое (Dense) используется всего один нейрон, так как в данном случае ставить задача бинарной классификации, выходное значение нейрона равно 1– приложение относится к ВПО, 0 – приложение легитимно.

```
model = keras.Sequential([
keras.layers.Dense(538,…),
keras.layers.Dense(538, …),
```

```
keras.layers.Dense(1, activation='sigmoid')
])
```

В результате построения программного макета имеется выборка из более чем 4 тысяч Android-приложений, представленных в виде csv данных, а также модель нейронной сети. Для последующего использования нейросети в качестве классификатора необходимо её обучить и произвести оценку, а до этого ещё правильно подобрать гиперпараметры нейросети.

4.2 Оценка эффективности разработанного макета

В данном разделе производится подбор оптимальных значений гиперпараметров построенной в предыдущем разделе модели нейронной сети, а также производится оценка её эффективности.

Для проведения экспериментов используется компьютер со следующими характеристиками (таблица 4.1)

Таблица 4.1- Характеристики тестового стенда

Характеристика	**Значение**
CPU	AMD Ryzen 2600 (6-ядерный, 12-потоковый)
RAM	DDR4 16 Гб
GPU	AMD Radeon RX580 (8 Гб)
Диск	SSD 256 Гб
ОС	Windows 10

К сожалению, пакет TensorFlow не поддерживает выполнение нейронных сетей на видеокартах без технологии CUDA и все вычисления происходили на процессоре.

Для оценки эффективности модели используется показатель точности (англ. accuracy), представленный в формуле (4.1):

$$Accuracy=\frac{TP+TN}{Total} \qquad (4.1)$$

где TP (англ. true positive) – количество верно классифицированных вредоносных Android-приложений, TN (англ. true negative) – легитимных, определенных как вредоносные, а $Total$ – общее количество приложений в тестовой выборке.

4.2.1 Поиск оптимальных параметров работы нейросети

Все гиперпараметры сети можно разделить на параметры слоев нейронной сети и параметры алгоритма обучения.

К гиперпараметрам алгоритма обучения в первую очередь относят следующие параметры:

– Функция потери (loss) – позволяет оценить отклонение значений, получаемых нейросетью от требуемых;

– Количество эпох (epochs) – количество полных прохождений элементов выборки через алгоритм обучения;

– Размер партии (batch_size) – количество элементов выборки, после прохождения которых через происходит корректирование параметров нейросети, т.е. выполнение алгоритма обратного распространения ошибки;

– Оптимизатор алгоритма обучения (optimizer) и его параметры

К параметрам слоя относятся:

– Количество нейронов (units)

– Функция активации (activation)

– Размер выходных данный (input_shape).

Для параметров слоев существует ряд общих рекомендаций:

– В качестве размера входного слоя рекомендуется выбирать число с множителем двойкой в некоторой степени.

– Количество нейронов в полносвязном слое также стоит выбирать кратным двум. Чаще всего используется от 256 до 4096 нейронов в слое.

Дополнительно нужно учитывать, что существует возможность столкнуться с проблемой переобучения.

Переобучение — это результат чрезмерной подгонки параметров модели к зависимостям, содержащимся в обучающем множестве. Если происходит переобучение, то модель не приобретает способности к обобщению — возможности распространять обнаруженные на обучающем множестве зависимости и закономерности на новые данные. Пример переобучения можно наблюдать на рисунке 4.3. Один из способов избежать переобучения– использовать dropout слои.

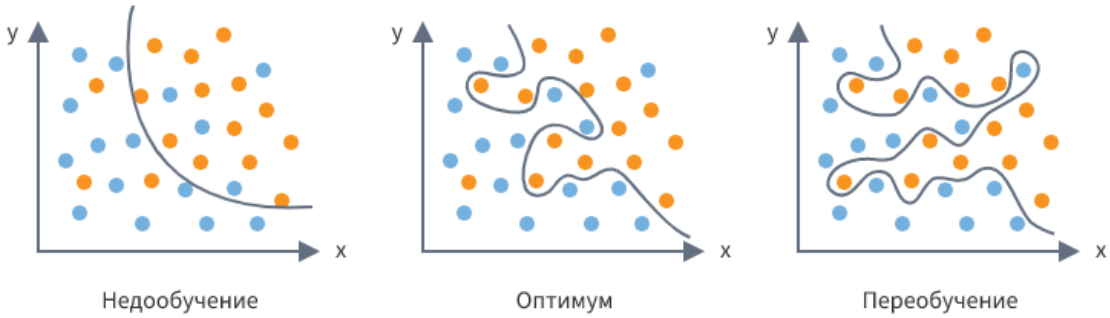

Рисунок 4.3 – Пример переобученной и недообученой сети

Dropout – слой, реализующий операцию исключения для предыдущего слоя нейросети. Этот слой представляет реализацию одноименного метода регуляризации нейросети, направленного на предотвращение переобучения сети. Некоторая часть входных нейроном отбрасывается из дальнейших вычислений. Таким образом в каждую эпоху создаются несколько отличные по-разному обученных копий исходной нейросети и их результаты усредняются. Такой приём улучшает эффективность обучения и качество результата.

В результате поиска оптимальных параметров работы, были выбраны следующие значения гиперпараметров, представленные в таблице 4.2:

Таблица 4.2- Значения гиперпараметров

Слой модели	**Гиперпараметр и его значение**
keras.layers.Dense	input_shape=(538,)
	activation='relu'
	units=768
keras.layers.Dropout	rate=0.5

Таблица 4.2- Значения гиперпараметров (продолжение)

keras.layers.Dense	units=768
	activation=sigmoid
keras.layers.Dropout	rate=0.5
keras.layers.Dense	units=1
	activation=’sigmoid’

Были использованы следующие параметры обучения:

– optimizer='adam',
– loss='binary_crossentropy',

Функция потерь binary_crossentropy – вычисляет перекрестную энтропийную потерю между истинными метками и предсказанными метками. Использовать эту кросс-энтропийную потерю, рекомендуется [70], когда есть только два класса меток (предполагается, что они равны 0 и 1). Для каждого прогноза дается результат в одно значение с плавающей запятой. Преимущество бинарной кросс-энтропии перед своей предшественницей, квадратичной функцией, состоит в том, что она лишена такого недостатка, как замедление при обучении (англ. learning slow-down).

Оптимизация 'adam' — это стохастический метод градиентного спуска, основанный на адаптивной оценке моментов первого и второго порядка. Согласно статье [71], метод "вычислительно эффективен, имеет мало требований к памяти, инвариантен к диагональному масштабированию градиентов и хорошо подходит для задач, которые являются большими с точки зрения данных/параметров".

Так как выборка данных оказалось сравнительно небольшой, было принято решение использовать метод кроссвалидации. Метод K-Fold Cross Validation делит данные на k подмножеств. Теперь этап обучения повторяется k раз, так что каждый раз одно из k подмножеств используется в качестве тестового набора / набора проверки, а другие k-1 подмножества объединяются для форми-

рования обучающего набора. По средней величене ошибок и точности предсказания получается общая эффективность модели. Каждая точка данных попадает в набор проверки ровно один раз и попадает в обучающий наборк N-1 раз. На рисунках ниже можно наблюдать результаты изменения функции потерь (4.4) и точности (4.5) от количества эпох при использовании кроссвалидации спараметром k=10. На графике сведены 10 результатов с различными тестовыми и обучающими выборками. Средняя точность обучения составила 97.9818856716156% (+- 0.4719756021202134%), а значение функции потерь: 0.060996432602405545. Средняя точность тестирования составила 95.41015386581421% (+- 1.720939047042017%), а значение функции потерь: 0.17589301019906997.

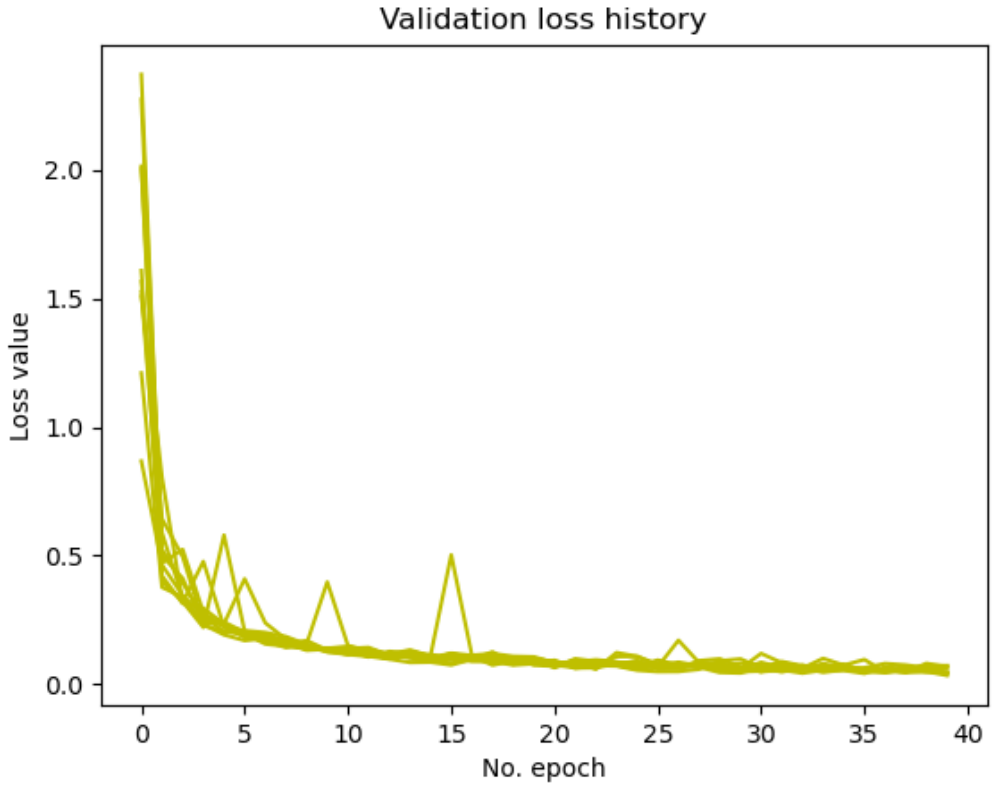

Рисунок 4.4– график изменения функции потерь от количества эпох при кросс=валидации

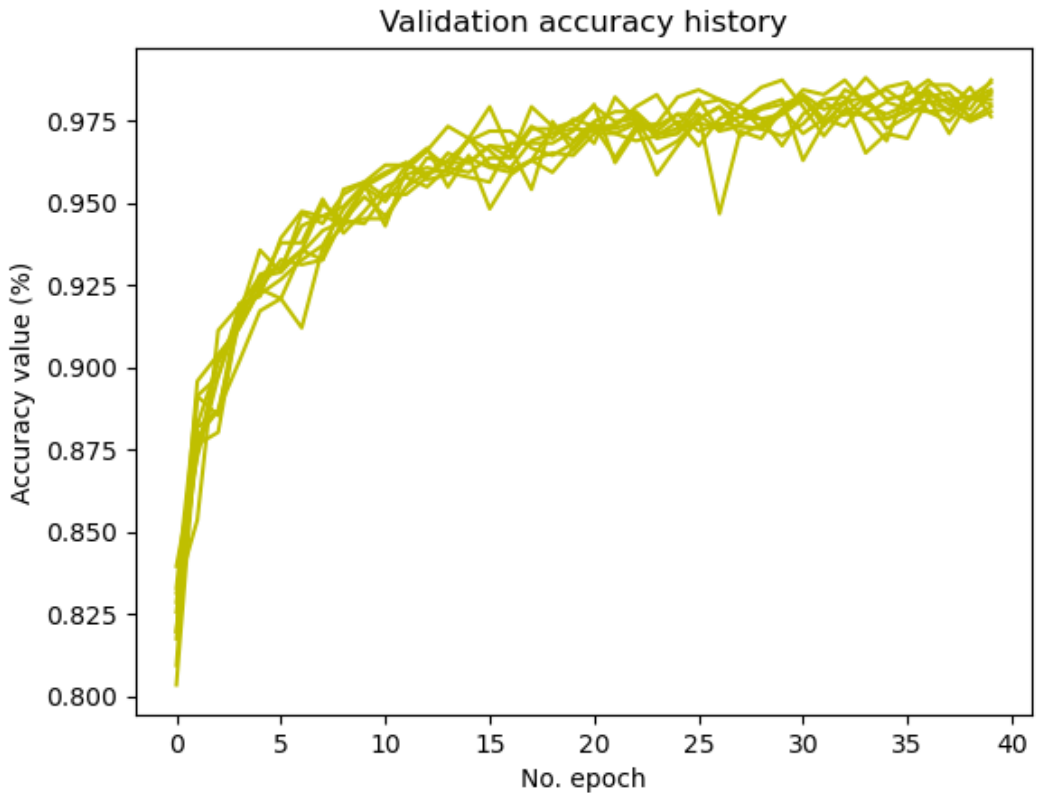

Рисунок 4.5– график изменения точности обучения от количества эпох при кросс-валидации

Дополнительно для ускорения процесса обучения было решено использовать простую нормализацию параметров, отличных от булевых. Каждый числовой параметр был поделён на его максимально значение, увеличенное на единицу. Таким образом диапазон значений был приведен к диапазону [0,1). Полученные вектора для нормализации были сохранены в отдельные файлы, для возможности дальнейшей загрузки для нормализации данных при выполнении сети на мобильном устройстве. Так как в момент выполнения на устройстве для выбранных приложений вектор для нормализации будет отличаться, а для корректной работы сети необходимо использовать одинаковую нормализацию для тренировочных и тестовых данных. На рисунке 4.6 и 4.7 можно увидеть результаты использования нормализации. Скорости обучения сети и уменьшения функции потерь значительно выросла начиная с первой эпохи, также улучшились итоговые показатели обучения на 40 эпох. Средняя точность обучения составила 99.68950271606445% (+- 00.19780436116652975%), а значение функции потерь: 0.00867054876871407. Средняя точность тестирования составила

95.74525237083435% (+- 2.0956835824661195%), а значение функции потерь: 0.21153920702636242.

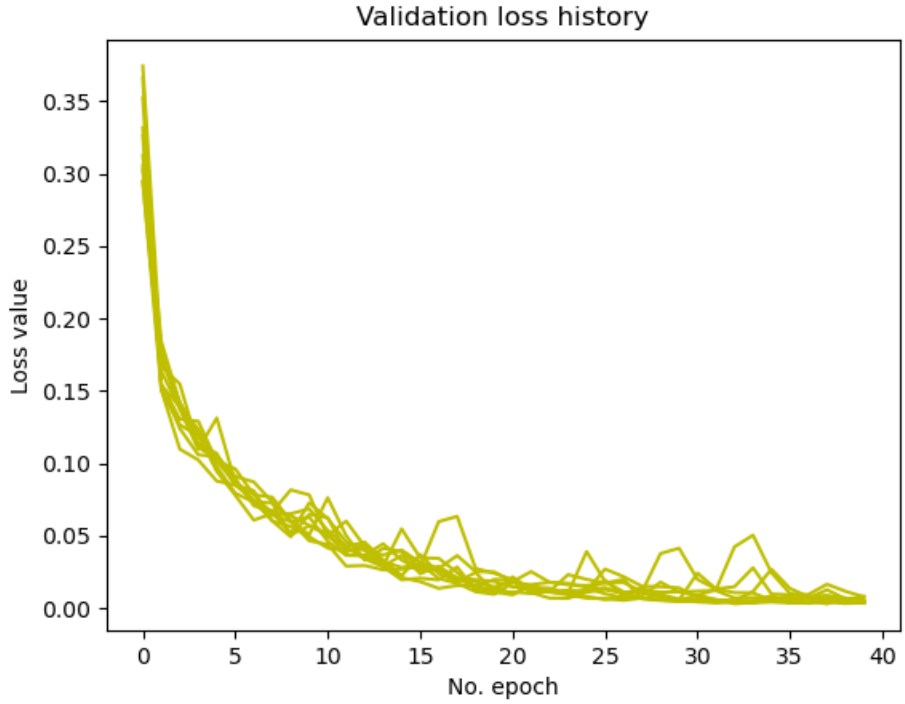

Рисунок 4.6 – график изменения функции потерь от количества эпох при использовании нормализации

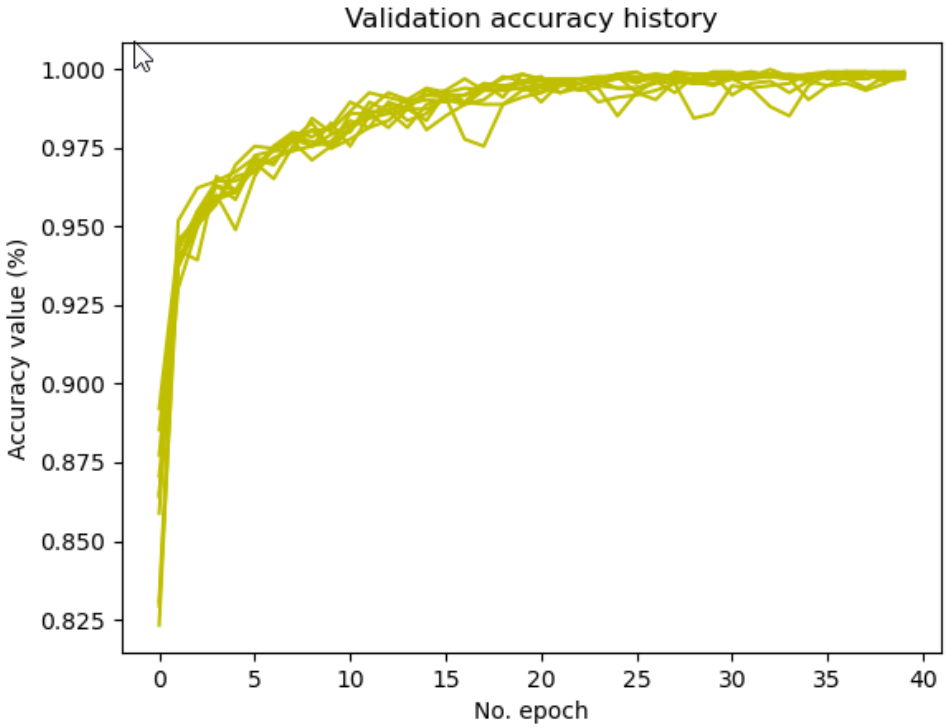

Рисунок 4.7– график изменения точности обучения от количества эпох при использовании нормализации

Для дальнейших тестов была отобрана отдельная, независимая выборка из вредоносных и легитимных приложений. Она была собрана с целью проведения более точных результатов теста, так как часто недостаточно поделить на тестовую и обучающую выборки изначальный массив данных, ведь нейросеть максимально подстроится под выборку и потеряет работоспособность на реальных данных. [72] В новую тестовую выборку вошли 121 легитимное и 162 вредоносных приложения.

Далее были получены данные по эффективности нейронной сети на новой тестовой выборке данных с отключенными включенным Dropout слоем. Результаты можно наблюдать на рисунках 4.8. и 4.9 (график слева– с отключенным Dropout, график справа с включённым).

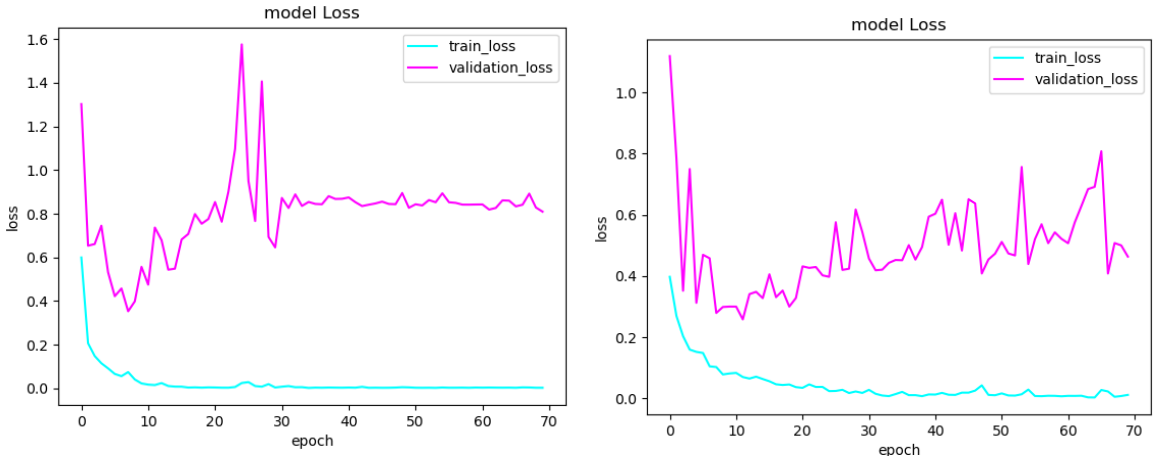

Рисунок 4.8 – графики изменения функции потерь от количества эпох при использовании новой тестовой выборки. (график слева– с отключенным Dropout, график справа с включённым).

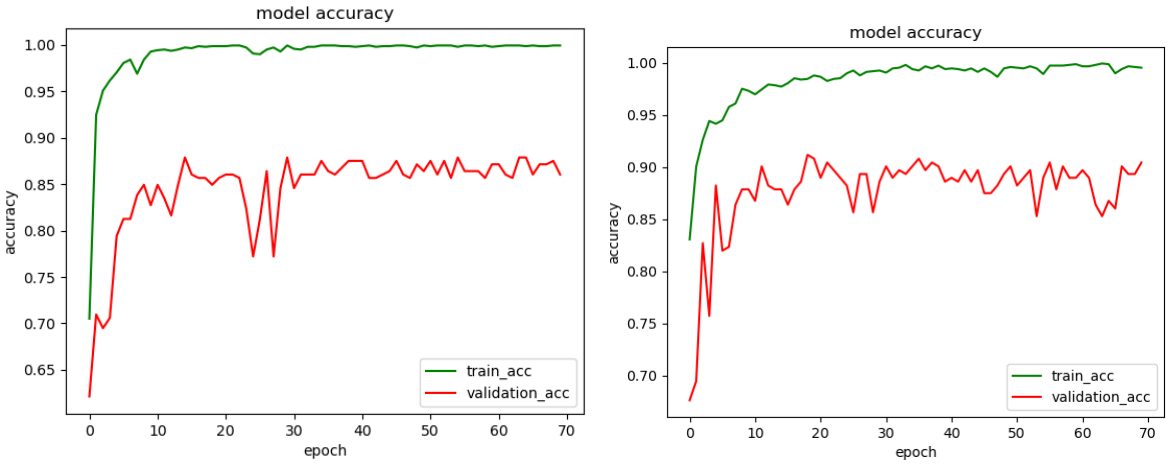

Рисунок 4.9 – графики изменения функции потерь от количества эпох при использовании новой тестовой выборки. (график слева– с отключенным Dropout, график справа с включённым).

По данным графикам можно наблюдать, что использование dropout слоя негативно повлияло на стабильность тестирования модели, но позволило добиться лучших показателей точности на новой тестовой выборке. Результаты, полученные при использовании слоев Dropout и отдельной, новой тестовой(валидирующей) выборки приведены в таблице 4.3.

Таблица 4.3

	Точность, %	**Значение функции потери**
Обучающий набор	99.529	0.0120
Тестовый набор	95.705	0.2312
Новая тестовая, валидирующая выборка	91.441	0.4634

4.2.2 Результаты оценки эффективности

Было проведено тестирование разработанной модели с оптимально подобранными параметрами условиях на валидирующей выборке. По результатам была составлена таблица 4.4 содержащая количество ошибок первого и второго рода при классификации приложений.

Таблица 4.4

	Предсказание легитимного приложения	Предсказание вредоносного приложения
Легитимное приложение	TN=115	FP=5
Вредоносное приложение	FN=19	TP=133

Относительно большое количество ошибок первого рода (FN) может свидетельствовать о недостаточном объеме вредоносных приложении в выборке.

Также было проведено тестирование заранее обученной сети на мобильном устройстве Xiaomi Mi 8 на базе чипа Snapdragon 845 (рисунок 4.10). Анализ приложения на мобильном устройстве занимает в среднем на 40 процентов дольше, в сравнении с стационарным компьютером. Причем почти все время анализа уходит на формирование вектора статическим анализатором. В среднем формирование вектора для предсказания вредоносности приложения на мобильном устройстве занимает от 3 до 30 секунд, на работу нейронной сети уходит меньше секунды.

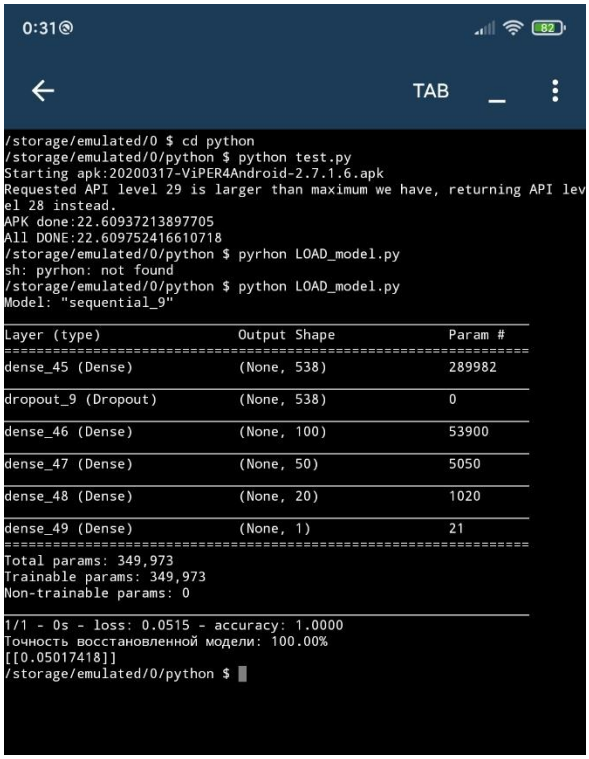

Рисунок 4.10 – Результаты тестирования легитимного приложение.

Стоит отметить, что в данной работе используется довольно маленькая выборка приложений, и для получения более хороших результатов на валидирующей выборке необходимо значительно увеличить набор данных, используемых при обучении. Увеличение обучающей выборке приведет к улучшению способности модели к обобщению, тем самым снизив количество ошибок первого и второго рода.

ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе представлены результаты разработки метода выявления вредоносного программного обеспечения для ОС Android на основе статического анализа с использованием методов глубокого обучения. В результате проведенных исследований и экспериментов все поставленные задачи были выполнены.

Рассмотрена общая архитектура устройства Android приложений, механизмы обеспечения безопасности мобильных устройств, основные виды ВПО, улучшения в системе безопасности, добавленные в последних версиях ОС Android. С учетом всех рассмотренных факторов сформулированы основные проблемы и угрозы безопасности мобильных устройств.

Исследованы возможности и способы применения нейронных сетей на мобильных устройствах, возможности по аппаратному ускорению их выполнения на уровне системы и производителей систем на чипе. Отдельно, более детально рассмотрена реализация аппаратного ускорения нейронных вычислений на чипах производства Qualcomm. На устройствах с разными поколениями чипов данного производителя были проведены тесты и сделаны выводы по эффективности использованных методов программно-аппаратного ускорения выполнения нейронных вычислений.

В третьей главе описаны существующие подходы к обнаружению ВПО, сравниваются существующие средства анализа ПО, рассмотрена возможность применения искусственных нейронных сетей для задачи обнаружения ПО, описана предлагаемая система анализа с гибридным подходом к обнаружению вредоносного ПО.

Реализован программный макет предлагаемого подхода к выявлению вредоносных Android-приложений, описан этап подготовки выборки для обучения нейронной сети и моделирования нейронной сети посредством программного пакета Keras.

Приведены гиперпараметры нейронной сети, результаты поиска оптимальных параметров обучения. В результате экспериментальной оценки эффективности разработанного макета была достигнута пиковая точность классификации равная 91.44%. Такой результат свидетельствует о работоспособности предложенного подхода и целесообразности его применения для обеспечения защиты пользователей мобильных устройств на базе ОС Android.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Мобильная вирусология 2019 [Электронный ресурс] // Securelist – URL: https://securelist.ru/mobile-malware-evolution-2019/95602/ – (дата обращения: 10.06.2020).

2. Круг П. Г. Нейронные сети и нейрокомпьютеры. – 2002. [Электронный ресурс] // Цифровой репозиторий DSpace ИВТ СО РАН – URL: http://elib.ict.nsc.ru/jspui/handle/ICT/956 – (дата обращения: 10.06.2020).

3. Huawei Kirin 970 – первая в мире однокристальная платформа с собственным нейроморфным процессором [Электронный ресурс] Itc.ua // – URL: https://itc.ua/news/huawei-kirin-970-pervaya-v-mire-odnokristalnaya-platforma-s-sobstvennyim-neyromorfnyim-protsessorom/ – (дата обращения: 10.06.2020).

4. Android – платформа для всех. [Электронный ресурс] // Android.com – URL: https://www.android.com/intl/ru_ru/everyone/enabling-opportunity/ – (дата обращения: 10.06.2020).

5. Голощапов А. Л. Google Android: программирование для мобильных устройств -2-е издание. – БХВ-Петербург, 2012. – С.17-22

6. Elenkov N. Android security internals: An in-depth guide to Android's security architecture. – No Starch Press, 2014.

7. Faruki P. et al. Android security: a survey of issues, malware penetration, and defenses //IEEE communications surveys & tutorials. – 2014. – Т. 17. – №. 2. – С. 998-1022.

8. Как Google Play Защита борется с вредоносными приложениями [Электронный ресурс] // Справка Google Play – URL: https://support.google.com/googleplay/answer/2812853?hl=ru – (дата обращения: 10.06.2020).

9. GingerBreak [Электронный ресурс] Xda-developers// – URL: https://forum.xda-developers.com/showthread.php?t=1044765 – (дата обращения: 10.06.2020).

10. Zhou Y., Jiang X. Dissecting android malware: Characterization and evolution //2012 IEEE symposium on security and privacy. – IEEE, 2012. – С. 95-109.

11. Chebyshev V., Unuchek R. Mobile malware evolution: 2013 //Kaspersky Lab ZAO's SecureList. – 2014. – Т. 24.

12. Zheng M., Sun M., Lui J. C. S. Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware //2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. – IEEE, 2013. – С. 163-171.

13. Самые распространенные мобильные угрозы в 2019 году [Электронный ресурс] // Блог Касперского – URL: https://www.kaspersky.ru/blog/mobile-virusology-2019/26401/ – (дата обращения: 10.06.2020).

14. Fakedefender.B [Электронный ресурс] // Android Fake sAntivirus – URL: http://contagiominidump.blogspot.in/2013/11/fakedefenderb-androidfake-antivirus.html – (дата обращения: 10.06.2020).

15. Felt A. P. et al. Android permissions: User attention, comprehension, and behavior //Proceedings of the eighth symposium on usable privacy and security. – 2012. – С. 1-14.

16. Felt A. P. et al. Android permissions demystified //Proceedings of the 18th ACM conference on Computer and communications security. – 2011. – С. 627-638.

17. Davi L. et al. Privilege escalation attacks on android //international conference on Information security. – Springer, Berlin, Heidelberg, 2010. – С. 346-360.

18. Wi-Fi Direct Flaw Exposes Android Devices to DoS Attacks [Электронный ресурс] // Security week – URL: https://www.securityweek.com/wi-fi-direct-flaw-exposes-android-devices-dos-attacks – (дата обращения: 10.06.2020).

19. Marforio C., Francillon A., Capkun S. Application collusion attack on the permission-based security model and its implications for modern smartphone systems. – ETH Zurich, 2011.

20. z4root.md [Электронный ресурс] //android-development-codex – URL: https://github.com/bibanon/android-developmentcodex/blob/master/General/ Rooting/z4root.md– (дата обращения: 10.06.2020).

21. Zhou Y., Jiang X. Dissecting android malware: Characterization and evolution //2012 IEEE symposium on security and privacy. – IEEE, 2012. – С. 95-109.

22. Huang H. et al. A framework for evaluating mobile app repackaging detection algorithms //International Conference on Trust and Trustworthy Computing. – Springer, Berlin, Heidelberg, 2013. – С. 169-186.

23. Zhou W. et al. Detecting repackaged smartphone applications in third-party android marketplaces //Proceedings of the second ACM conference on Data and Application Security and Privacy. – 2012. – С. 317-326.

24. ProGuard: Оптимизатор с открытым исходным кодом для Java и Kotlin [Электронный ресурс] // GuardSquare – URL: https://www.guardsquare.com/en/products/proguard – (дата обращения: 10.06.2020).

25. DexGuard: безопасность приложений Android [Электронный ресурс] // GuardSquare – URL: https://www.guardsquare.com/en/products/dexguard – (дата обращения: 10.06.2020).

26. Rastogi V., Chen Y., Jiang X. Droidchameleon: evaluating android anti-malware against transformation attacks //Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. – 2013. – С. 329-334.

27. Faruki P. et al. Evaluation of android anti-malware techniques against dalvik bytecode obfuscation //2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications. – IEEE, 2014. – С. 414-421.

28. Zheng M., Lee P. P. C., Lui J. C. S. ADAM: an automatic and extensible platform to stress test android anti-virus systems //International conference on detection of intrusions and malware, and vulnerability assessment. – Springer, Berlin, Heidelberg, 2012. – С. 82-101.

29. Severyn A., Moschitti A. Twitter sentiment analysis with deep convolutional neural networks //Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. – 2015. – С. 959-962.

30. Socher R. et al. Recursive deep models for semantic compositionality over a sentiment treebank //Proceedings of the 2013 conference on empirical methods in natural language processing. – 2013. – С. 1631-1642.

31. Sigtia S., Dixon S. Improved music feature learning with deep neural networks //2014 IEEE international conference on acoustics, speech and signal processing (ICASSP). – IEEE, 2014. – С. 6959-6963.

32. Krishnamoorthi R. Quantizing deep convolutional networks for efficient inference: A whitepaper //arXiv preprint arXiv:1806.08342. – 2018.

33. TensorFlow-Lite [Электронный ресурс] // TensorFlow.org – URL: https://www.tensorflow.org/lite – (дата обращения: 10.06.2020).

34. TensorFlow Lite delegates [Электронный ресурс] // TensorFlow.org – URL: https://www.tensorflow.org/lite/performance/delegates – (дата обращения: 10.06.2020).

35. Lee J. et al. On-device neural net inference with mobile gpus //arXiv preprint arXiv:1907.01989. – 2019.

36. Smile to the camera, it’s OpenCL! [Электронный ресурс] //ARM.com – URL: https://community.arm.com/developer/tools-software/graphics/b/blog/posts/smile-to-the-camera-it-s-opencl – (дата обращения: 10.06.2020).

37. Sanz B. et al. Puma: Permission usage to detect malware in android //International Joint Conference CISIS’12-ICEUTE 12-SOCO 12 Special Sessions. – Springer, Berlin, Heidelberg, 2013. – С. 289-298.

38. Huang C. Y., Tsai Y. T., Hsu C. H. Performance evaluation on permission-based detection for android malware //Advances in Intelligent Systems and Applications-Volume 2. – Springer, Berlin, Heidelberg, 2013. – С. 111-120.

39. Kim J. et al. ScanDal: Static analyzer for detecting privacy leaks in android applications //MoST. – 2012. – Т. 12. – №. 110. – С. 1.

40. Shabtai A. et al. “Andromaly”: a behavioral malware detection framework for android devices //Journal of Intelligent Information Systems. – 2012. – Т. 38. – №. 1. – С. 161-190.

41. Burguera I., Zurutuza U., Nadjm-Tehrani S. Crowdroid: behavior-based malware detection system for android //Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. – 2011. – С. 15-26.

42. Damopoulos D., Kambourakis G., Portokalidis G. The best of both worlds: a framework for the synergistic operation of host and cloud anomaly-based IDS for smartphones //Proceedings of the seventh european workshop on system security. – 2014. – С. 1-6.

43. McClurg J. R. Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets by Yajin Zhou, et al. (NDSS, 2012). – 2012.

44. Enck W., Ongtang M., McDaniel P. On lightweight mobile phone application certification //Proceedings of the 16th ACM conference on Computer and communications security. – 2009. – С. 235-245.

45. Yerima S. Y. et al. A new android malware detection approach using bayesian classification //2013 IEEE 27th international conference on advanced information networking and applications (AINA). – IEEE, 2013. – С. 121-128.

46. Li W. et al. An android malware detection approach using weight-adjusted deep learning //2018 International Conference on Computing, Networking and Communications (ICNC). – IEEE, 2018. – С. 437-441.

47. Zhang Y., Yang Y., Wang X. A novel android malware detection approach based on convolutional neural network //Proceedings of the 2nd International Conference on Cryptography, Security and Privacy. – 2018. – С. 144-149.

48. Liang H., Song Y., Xiao D. An end-to-end model for android malware detection //2017 IEEE International Conference on Intelligence and Security Informatics (ISI). – IEEE, 2017. – С. 140-142.

49. Hou S. et al. Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs //2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW). – IEEE, 2016. – С. 104-111.

50. Yuan Z. et al. Droid-sec: deep learning in android malware detection //Proceedings of the 2014 ACM conference on SIGCOMM. – 2014. – С. 371-372.

51. Yuan Z., Lu Y., Xue Y. Droiddetector: android malware characterization and detection using deep learning //Tsinghua Science and Technology. – 2016. – Т. 21. – №. 1. – С. 114-123.

52. Vinayakumar R. et al. Detecting Android malware using long short-term memory (LSTM) //Journal of Intelligent & Fuzzy Systems. – 2018. – Т. 34. – №. 3. – С. 1277-1288.

53. Alzaylaee M. K., Yerima S. Y., Sezer S. DL-Droid: Deep learning based android malware detection using real devices //Computers & Security. – 2020. – Т. 89. – С. 101663.

54. Alzaylaee M. K., Yerima S. Y., Sezer S. DynaLog: An automated dynamic analysis framework for characterizing android applications //2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security). – IEEE, 2016. – С. 1-8.

55. Using Strace [Электронный ресурс] AOSP// – URL: https://source.android.com/devices/tech/debug/strace – (дата обращения: 10.06.2020).

56. Androguard [Электронный ресурс] // github.com – URL: https://github.com/androguard/androguard – (дата обращения: 10.06.2020).

57. NumPy [Электронный ресурс] // numpy.org. – URL: http://www.numpy.org/. – (дата обращения: 10.06.2020).

58. Machine Learning in Python [Электронный ресурс] // scikit-learn.org – URL: https://scikit-learn.org/stable/ – (дата обращения: 10.06.2020).

59. Keras: The Python Deep Learning library [Электронный ресурс] // keras.io. – URL: https://keras.io/. – (дата обращения: 10.06.2020).

60. TensorFlow [Электронный ресурс] // tensorflow.org. – URL: https://www.tensorflow.org/. – (дата обращения: 10.06.2020).

61. Random .APK Collection (February 2018) [Электронный ресурс] // Internet Archive – URL: https://archive.org/details/2018-02-random-apk-collection – (дата обращения: 10.06.2020).

62. ApkPure Apps [Электронный ресурс] // ApkPure.com – URL: https://apkpure.com/app?sort=rating – (дата обращения: 10.06.2020).

63. Collection of android malware samples [Электронный ресурс] // github.com – URL: https://github.com/ashishb/android-malware– (дата обращения: 10.06.2020).

64. Android Botnet dataset [Электронный ресурс] // unb.ca – URL: https://www.unb.ca/cic/datasets/android-botnet.html – (дата обращения: 10.06.2020).

65. Popular Android threats in 2019 [Электронный ресурс] // github.com – URL: https://github.com/sk3ptre/AndroidMalware_2019 – (дата обращения: 10.06.2020).

66. Popular Android threats in 2020 [Электронный ресурс] // github.com – URL: https://github.com/sk3ptre/AndroidMalware_2020 – (дата обращения: 10.06.2020).

67. Collection of android malware samples [Электронный ресурс] // github.com – URL: https://github.com/hxp2k6/Android-Malwares – (дата обращения: 10.06.2020).

68. Drebin - NDSS 2014 Re-implementation [Электронный ресурс] // github.com – URL: https://github.com/MLDroid/drebin – (дата обращения: 10.06.2020).

69. RevealDroid [Электронный ресурс] // bitbucket.org – URL: https://bitbucket.org/joshuaga/revealdroid.git – (дата обращения: 10.06.2020).

70. Probabilistic losses [Электронный ресурс] // Keras API reference – URL: https://keras.io/api/losses/probabilistic_losses/#binary_crossentropy-function – (дата обращения: 10.06.2020).

71. Kingma D. P., Ba J. Adam: A method for stochastic optimization //arXiv preprint arXiv:1412.6980. – 2014.

72. Нейронные сети, «вредные» советы [Электронный ресурс] // habr.com – URL: https://habr.com/ru/post/211610/] – (дата обращения: 10.06.2020).