Система автоматизации численной оценки сходства Android-приложений

В.В.Петров[1][0009-0004-4213-7328]

[1]*Институт информационных технологий и интеллектуальных систем Казанского федерального университета (КФУ)*

Аннотация. Работа посвящена проектированию и разработке системы автоматизации численной оценки сходства Android-приложений. Задача оценки сходства приведена к задаче оценки сходства множеств графов потока управления, построенных на основе кода из classes.dex файлов. Значение сходства вычисляется на основе матрицы сходства. Графы потока управления сравниваются при помощи алгоритмов расстояния редактирования графов и расстояния Левенштейна. В работе были сформулированы критерии сходства Android-приложений, исследованы формы их представления, виды моделей, виды методов построения моделей, виды оценки сходства программ и существующие решения. Разработан прототип системы и вспомогательные инструменты, выполнена оптимизация программы инструментами параллельного программирования. Проведены эксперименты и сделан вывод о способности программы выявлять сходства между Android-приложениями.

Ключевые слова: сходство Android-приложений, сходство программ, матрица сходства, расстояние редактирования графов потока управления, визуализация матрицы сходства, граф потока управления.

Automated system for numerical similarity evaluation of Android applications

V.V.Petrov[1][0009-0004-4213-7328]

[1]*Institute of Information Technologies and Intelligent Systems Kazan Federal University (KFU)*

Abstract. This work is devoted to the design and development of a system for automating the numerical assessment of the similarity of Android applications. The problem of similarity assessment is reduced to the problem of similarity assessment of sets of control flow graphs built on the basis of code from classes.dex files. The similarity value is calculated based on the similarity matrix. Control flow graphs are compared using the graph edit distance and Levenshtein distance algorithms. The paper formulated similarity criteria for Android applications, investigated the forms of their representation, types of

models, types of methods for building models, types of assessment of similarity of programs and existing solutions. A prototype of the system and auxiliary tools have been developed, the program has been optimised using parallel programming tools. Experiments were carried out and a conclusion was made about the ability of the program to identify similarities between Android applications.

Keywords: Android application similarity, program similarity, similarity matrix, control flow graph edit distance, similarity matrix visualisation, control flow graph.

1. Введение

Сходство программ оценивается во многих задачах: обнаружение дублированного или клонированного кода, проверка авторских прав или патентов, повышение утилизации кодовой базы и др. Для решения каждой из приведенных задач необходимо определить, что значит «сходство» и как его измерить.

Опытный программист способен оценить сходство программ по ключевым критериям: наименование классов, функций, переменных, поток управления, использование литералов и др. В таком случае, часть критериев может быть определена и оценена субъективно и отличаться от сравнения к сравнению не только одного программиста, но и нескольких. В таких условиях возникает необходимость в формализации критериев сравнения сходства и способов их вычисления.

Если необходимо оценить сходство между множеством программ, то количество попарных сравнений равно $\frac{□!}{2*(□-2)!}$, где n - количество программ. Для задачи нахождения плагиата возможен поиск среди миллионов программ. При этом программы могут состоять из тысяч файлов, каждый из которых может иметь тысячи строк кода. Выполнить сравнение вручную может потребовать годы. В то время как размер программ и их количество растет каждый день. Все эти условия формируют потребность в автоматизации сравнения сходства программ.

Решение проблемы автоматизации численной оценки сходства программ имеет практическую значимость. Согласно исследованию [1], от 5 до 13% Android приложений, распространяемых на 6 Android площадках, являются клонами. В исследовании [2] показано, что как минимум 141 приложение было клонировано. Также известны случаи [3, 4] плагиата мобильных приложений. Данные проблемы актуальны для индустрии приложений для Android ОС. Приложения для Android ОС распространяются в виде .apk и .aab артефактов [5]. Артефакты могут быть скачаны с различных площадок: Google Play Store, Huawei App Gallery и др. При этом площадки могут быть как официальными, так и нет. Так как перед установкой приложения необходимо скачать на устройство артефакт,

возникает ситуация, когда злоумышленник получает прямой доступ к артефакту и способен выполнить MATE-атаку [6]. После этого злоумышленник может опубликовать измененное приложение в Интернет под видом исходного и украсть пользовательские данные, ухудшить имидж компании и др.

В рамках данной работы проектируется и разрабатывается система автоматизации численной оценки сходства Android-приложений. Для моделирования Android-приложения используется матрица графов потока управления его функций. Сходство графов потока управления оценивается при помощи вычисления расстояния редактирования графов. Вычисления итогового значения сходства выполняется с использованием технологий параллельных вычислений. Разработаны инструменты для автоматической и ручной оценки сходства, визуализации матрицы сходства и графов потока управления.

Статья имеет следующую структуру. В разделе 2 описывается постановка задачи сравнения Android-приложений и вводятся обозначения. В разделе 3 представлен сравнительный анализ форм представления приложений и обоснование выбора. В разделе 4 проведен сравнительный видов моделей приложений и обоснование выбора. В разделе 5 описан сравнительный методов построения моделей приложений и обоснование выбора. В разделе 6 представлен сравнительный методов оценки сходства моделей приложений и обоснование выбора. В разделе 7 описан алгоритм численной оценки сходства Android-приложений. В разделе 8 представлены вспомогательные инструменты для выполнения экспертной оценки сходства приложений. В разделе 9 описаны эксперименты и приведен анализ результатов.

2. Постановка задачи

Даны два Android-приложения P_1 и P_2. Необходимо вычислить $sim(P_1, P_2)$ — степень сходства приложений P_1 и P_2. Причем:

- P_1 и P_2 могут иметь одну из форм представления: *исходный код, скомпилированный код, скомпилированный артефакт*;
- $sim(P_1, P_2) \in \Re$;
- $sim(P_1, P_2) \in [0; 1]$;
- $sim(P_1, P_2) = 0$ означает, что программы не имеют ни единого сходства;
- $sim(P_1, P_2) = 1$ означает, что программы абсолютно идентичны;
- $sim(P_1, P_2) = sim(P_2, P_1)$.

Для построения модели программы использовать функцию $m: P \rightarrow M$, которая выделяет характеристики программы и объединяет их в структуру данных, которая называется моделью программы. Причем:

- Характеристики программы выделяются при помощи *статического* или *динамического анализа* кода;

- $m(P_1) = M_1, m(P_2) = M_2$ являются моделями P_1 и P_2 соответственно;
- Значение $m(P)$ может быть получено только из P;
- Если P_2 является *производной* от P_1, то $M_1 = M_2$.
- P_2 называется *производной* от P_1 если над ней были выполнены преобразования, не нарушающие смысла программы. К таким преобразованиям относятся: обфускация, оптимизация и др.

3. Выбор формы представления Android-приложения

Android-приложения могут быть представлены в трех формах:

- Исходный код — в репозитории с приложением или на компьютере разработчика;
- Скомпилированный код — при промежуточной компиляции исходного кода в байт-код JVM [7] или байт-код ART [8];
- Скомпилированный артефакт — при компиляции приложения. Полученный в результате компиляции файл в формате .apk или .aab может быть установлен на устройство.

С точки зрения преобразования формы представления, исходный код содержит в себе наибольшее количество авторской информации, интеллектуальной собственности, так как исходный код создается человеком, его разработчиком. Скомпилированный код и артефакт получаются при помощи стандартных инструментов: компиляторов и др, которые преобразуют форму представления по известным алгоритмам. При этом компилятор способен удалить часть информации из исходного кода, например, комментарии, которые будет невозможно восстановить при декомпиляции кода. Поэтому количество полезной информации в исходном коде больше, чем в скомпилированном артефакте. С другой стороны, во время работы приложения выполняется скомпилированный, оптимизированный или обфусцированный, код, а не исходный.

С точки зрения доступности формы представления, скомпилированный артефакт имеет наибольшую доступность, так как может быть скачан из официального магазина приложений; любого интернет-ресурса; устройства, на котором установлено приложение. В отличие от исходного кода, доступ к которому может быть ограничен владельцем репозитория или компьютера.

В рамках данной работы в качестве сравниваемых программ выступают Android-приложения в форме скомпилированного артефакта — .apk файла. Выбор данной формы представления обосновывается ее наибольшей доступностью и фактическим использованием при работе программы.

Файл .apk файл имеет следующую структуру [9]:

- Скомпилированного Dalvik / ART байт-кода — classes.dex. Таких файлов может быть несколько;
- Манифеста приложения — AndroidManifest.xml;

- Метаинформации — META-INF;
- Ресурсов приложения — assets, res;
- Скомпилированных нативных библиотек — lib;
- Скомпилированных ресурсов — resources.arsc.

Каждый из этих файлов содержит информацию о работе приложения и может быть использован при оценке сходства. resources.arsc, META-INF, AndroidManifest.xml содержат типовую информацию о приложении и не несут особой ценности при копировании. Более того содержимое этих файлов кратно меньше, чем classes.dex, assets, res, lib. Директории assets, res содержат макеты приложения, файлы анимации, статические ресурсы и др. При реверс-инжиниринге этот код может быть использования для плагиата пользовательского интерфейса приложения. classes.dex, lib содержат скомпилирвоанный код, наибольшее количество полезной информации о работе приложения и занимают основную часть .apk файла. В lib находится скомпилированный C/C++ код. Использование нативного кода при разработке Android-приложений является узконаправленным решением для специфичных задач: отрисовка сложной графики, ресурсоемкие вычисления и др. и на практике встречается редко.

В рамках данной работы предлагается рассматривать файлы classes.dex в качестве источника информации об Android-приложении, так как они содержат скомпилированный исходный код и занимают значительную часть .apk файла.

4. Выбор модели Android-приложения

Исходно classes.dex представляет собой бинарный код для среды выполнения Android: Dalvik или ART. Для построения моделей на основе бинарного кода, инструкций, блоков инструкций, функций [10] достаточно декомпилировать .apk файл и выполнить поверхностный анализ содержимого. Такой подход позволяет добиться высокой скорости построения в сравнении с другими моделями. Однако, ни одна из них не описывает структуру приложения и является чувствительной к обфускации.

Следующие модели, помимо декомпиляции .apk, требуют дополнительных преобразований для построения, что снижает общее время работы алгоритма. Абстрактное синтаксическое дерево [11] описывает структуру программы и выполняемые инструкции, что является ее преимуществом. Основной недостаток AST заключается в том, что оно строится на основе исходного кода программы. То есть classes.dex необходимо предварительно декомпилировать до .class файлов и потом снова декомпилировать до .java, .kt файлов. Дважды выполненная декомпиляция приводит к потерям информации, что критично для AST.

Иерархия классов [12] является применимой моделью к Android-приложению, так как при его разработке используются объектно-ориентированные языки. Модель описывает различные зависимости между

классами: наследование, композиция и др. Недостатком иерархии классов является то, что она не описывает фактическую реализацию класса, то есть модель не в полной мере отражает содержимое программы.

Граф зависимости программы [13] описывает поток управления функции и поток данных, что является преимуществом среди прочих моделей. Единственным недостатком является то, что граф не содержит информации о других вызываемых функциях.

Граф вызовов [14] также может быть использован в качестве модели Android-приложения: он описывает его структуру и не зависит от используемого языка программирования. Однако, современные обфускаторы способы изменять поток управления, что влияет на чувствительность модели. Более того, граф вызовов не описывает содержимое функций.

Граф потока управления [15] в отличие от двух предыдущих моделей описывает выполняемые блоки инструкций. Также граф описывает структуру программы и имеет размерность меньше, чем у AST.

В рамках данной работы предлагается в качестве модели Android-приложения использовать граф потока управления построенный на основе скомпилированного кода файлов classes.dex. Несмотря на то, что граф потока управления также чувствителен к обфускации потока управления и требует дополнительных преобразований для построения, он является наиболее подходящим среди прочих при моделировании программ. Этот вывод подтверждается в в систематическом обзоре литературы [10] по теме сходства программ.

5. Выбор метода построения модели

Для построения графа потока управления Android-приложения могут быть использованы как статический, так и динамический анализ. При статическом подходе Android-приложения никогда не выполняется, а модель строится на основе кода из classes.dex файлов. При динамическом анализе Android-приложения запускается на виртуальном или реальном устройстве и исследуется его поведение во время выполнения.

Статический анализ эффективен, потому что он может исследовать набор всех возможных путей выполнения путем аппроксимации поведения программы. Это важно, потому что некоторые сценарии работы Android-приложений трудно вызвать динамически: выполнение задач по расписанию, работы приложения в фоне и др.

Основное преимущество динамического анализа заключается в том, что он отображается фактическое выполнение Android-приложения, а запутывания, применяемые к classes.dex, оказывают меньшее влияние на поведение программы.

В рамках данной работы предлагается строить граф потока управления Android-приложения при помощи статического анализа. Выбор

данного метода преимущественно обусловлен непоследовательной (асинхронной) природой выполнения Android-приложений. Взаимодействие с базовыми компонентами Android осуществляется по системе обратных вызовов: изменение интерфейса, обращение в сеть, работа в фоновом режиме и др., что значительно усложняет покрытие всех сценариев использовании приложения в случае использования динамического анализа.

6. Выбор метода оценки сходства моделей

Максимальный общий подграф используется для оценки сходства графов. Например, в статье [16]. Большим преимуществом данного метода является возможность контролировать размерность общего подграфа, что ускоряет его построение. Однако, данный метод имеет один значительный недостаток: наличие нескольких общих подграфов. Данная проблема особенно актуальна, при построении максимального общего подграфа для графов малого размера. Данный вывод подтверждается в исследовании [17].

Для оценки сходства графов потока управления в рамках данной работы используется оптимизированный алгоритм вычисления расстояния редактирования [18]. Данный алгоритм требует меньше памяти и времени, чем A^*вариации, способен учитывать содержимое блоков инструкций графа потока управления.

В рамках данной работы расстояние редактирования графа вычисляется по формуле [18]:

$$GED(g_1, g_2) = \min_{e_1,\cdots,e_k \in \gamma(g_1,g_2)} \sum_{i=1}^{k} c(e_i) \quad (1)$$

где $GED -$ расстояние между графами g_1 и g_2; $GED(g_1, g_2) \geq 0$;

$e_i -$ одна из операций редактирования: вставка, удаление, замена;

$e_1, e_2, \ldots, e_k \in \gamma(g_1, g_2) -$ путь редактирования длины k;

$\gamma(g_1, g_2) -$ множество путей редактирования g_1 в g_2;

$c(e_i) -$ функция стоимости операции редактирования, $c(e_i) = 1$.

Для приведения значения расстояния редактирования графа (1) в интервал [0;1] предлагается использовать следующую формулу:

$$\widehat{GED(g_1, g_2)} = \frac{GED(g_1, g_2)}{max(size(g_1), size(g_2))} \quad (2)$$

где $\widehat{GED(g_1, g_2)} -$ расстояние редактирования между графами g_1, g_2;

$\widehat{GED(g_1, g_2)} \in [0; 1]$;

$size(g) -$ размер графа $g, size(g) = |V| + |E|$.

Принятие решения о редактировании вершины графа потока управления выполняется по формуле (3). Для этого вычисляется расстояние Левенштейна [19] между инструкциями блока. Если значение sim меньше порогового, то вершина редактируется:

$$decision = sim(S_1, S_2) < threshold\ , \quad (3)$$

где $decision = \{\text{Ложь}, \text{Истина}\} —$ решение о редактировании;

$threshold -$ порог принятия решения, $threshold \in [0; 1]$;

$S_1, S_2 -$ сравниваемые последовательности;

$M = len(S_1), N = len(S_2) -$ длины последовательностей;

$sim(S_1, S_2) = max(M, N) - D(S_1, S_2).$

$$D(S_1, S_2) = D(M, N),$$

где $D -$ расстояние Левенштейна, $D \in [0; max(M, N)]$;

7. Описание алгоритма

Алгоритм автоматизации численной оценки сходства Android-приложений можно разделить на основные этапы:

1. Подготовка;
2. Построение моделей (графов потока управления) первого и второго Android-приложения;
3. Построение матрицы сходства на основе расстояния редактирования графов потока управления;
4. Нахождение пар наиболее схожих элементов, вычисление итогового значения сходства.

Для ускорения работы алгоритма построение матрицы сходства разработанная программа имеет механизмы параллелизации вычисления как на уровне строк матрицы, так и на уровне ее ячеек. Для этого создается конечное множество процессов, между которыми распределяется задача по вычисления строк матрицы сходства. Внутри каждого процесса создается конечное множество потоков, каждый из которых вычисляет сходство между двумя графами потока управления.

Помимо параллелизации при вычислении расстояние редактирования используется параметр, устанавливающий верхнюю границу времени вычисления. При достижении этой границы, алгоритм считает в качестве результата актуальное на момент остановки значение расстояние редактирования.

8. Вспомогательные инструменты

Функция calculate_apks_similarity.py в режиме оценки сходства Android-приложений генерирует логи, содержащие списки генерируемых графов потока управления, результаты вычисления сходства и др. Сгенерированные файлы содержат полезную для исследователя информацию представленную в текстовом виде. Однако логи не визуализируют матрицу сходства и пары наиболее схожих графов потока управления. Для повышения скорости, простоты и качества экспертного анализа результатов работы calculate_apks_similarity.py были разработаны две вспомогательные программы.

После запуска отобразится отдельное окно, на котором будет изображена вычисленная calculate_apks_similarity.py матрица сходства.

Отображаемая матрица сходства обрабатывает нажатия на ячейки. При нажатии на ячейку, программа генерирует .pdf файла в котором находятся визуализированные графы потока управления. Граф на первой странице соответствует строке матрицы, на второй — столбцу. Сгенерированный .pdf файл имеет имя dots_<номер строки>_<номер столбца>_<значение сходства>.pdf (напрмер, dots_1_23_1.0.pdf) и находится в папке visualize_m_comp. Папка visualize_m_comp находится в папке переданной аргументом --output_dir calculate_apks_similarity.py.

9. Эксперименты

На данный момент не существует стандартизированных критериев оценки сходства Android-приложений. Существует решение суда № А40-20593/2017 в котором присутствует формулировка «Схожесть наименований двух программ для ЭВМ и их основное назначение не могут служить основанием для вывода об их тождественности, так как для подобного вывода необходимы специальные знания (заключение эксперта)» [20]. Что касается научных исследований, то в некоторых статьях сравнивают собственные разработки с аналогами (например, в

статье [21] k-gramm сравнивается с Tamada). В других выполняется оценка существующих программ без сравнения аналогов (например, в статье [22] код изменяется вручную и затем вычисляется сходство между исходной и измененной программой).

Наиболее простыми сценариями проверки являются крайние значения интервала сходства: 0 — одно приложение пустое (нет кода) и любое Android-приложение (есть код); 1 — сравнение приложения с самим собой. Так как модель программы должна является ее инвариантом (не меняться при трансформациях с сохранением семантики), то в качестве еще одного эксперимента можно предложить сравнение Android-приложения до обфускации и после (обфускация не должна влиять на значение сходства, либо влиять незначительно). Ожидается, что чем больше техник обфускации будет применено к Android-приложению, тем меньшим сходством оно будет обладать при сравнении его с исходным вариантом. Однако, в систематическом обзоре литературы [10] подчеркивается, что на данный момент нет модели, которая способна быть устойчивой к любым трансформациям с сохранением семантики.

В рамках данной работы программное решения оценивается по следующим сценариям:

1. Android-приложение сравнивается с пустым .apk файлом;
2. Android-приложение сравнивается с самим собой;
3. Android-приложение и его вариант, в котором были изменены имена классов, пакетов при помощи ProGuard;
4. Android-приложение и его оптимизированный при помощи ProGuard вариант (опция -optimizationpasses 5);
5. Два разных Android-приложения.

Сгенерированные файлы, логи выполнения программ в рамках выполнения каждого эксперимента находятся в GitHub репозитории работы в папке с соответствующим номером в папке exp (например, для первого эксперимента эта папка exp/1).

Описанные ниже эксперименты выполнялись на компьютере со следующей конфигурацией:

- Наименование: MacBook Pro;
- Операционная система: macOS Monterey (версия 12.4);
- Процессор: Apple M1 Pro с 10 ядрами;
- Объем оперативной памяти: 16 ГБ.

Краткое описание результатов экспериментов представлено в Таблице 1. Значение колонки «Верно сопоставленные пары сходства» вычисляется как отношение корректно найденных пар к общему числу выбранных пар сходств и вычисляется экспертно на основе анализа графов потока управления.

№	$sim(P_1, P_2)$	Время вычисления, сек	Верно сопоставленные пары сходства	ins_block_sim_threshold	ged_timeout_sec, сек	processes_count	threads_count
1	0.0	2.3	0%	0.95	60	10	45
2	0.99	5.3	100%	0.95	60	10	45
3	0.9	3.6	100%	0.95	60	10	45
4	0.2	3.4	80%	0.95	60	10	45
5	0.19	64.4	23%	0.95	60	10	45

Таблица 1. Описание результатов экспериментов.

В первом эксперименте результат полностью соответствует ожиданиям. Так как второй .apk файл не содержал classes.dex, то построенная модель состояла из пустого множества графов потока управления. Поэтому dots_2.csv не содержит ни одной записи. Матрица сходства имеет размерность 17:0.

Во втором эксперименте матрица сходства симметрична по диагонали, что говорит о сравнении двух идентичных .apk файлов. Большинство ячеек матрицы имеют черный цвет и соответствуют конструкторам класса R (класс с идентификаторами ресурсов в Android).

В отличии от матрицы сходства из эксперимента 2 в эксперименте 3 пары графов не образуют диагональ. Дело в том, что имена методов при обфускации .apk изменяются, как и порядок их обхода при формировании списка графов.

Во четвертом эксперименте после выполнения оптимизации .apk файла, количество классов сократилось с 17 до 5. Помимо уже рассмотренных в эксперименте 3 случаев, когда переименование снижает сходство, в этом примере есть один новый сценарий с переносом кода и удалением соответствующего класса. Значение сходства 0.2 показывает, что модификации графа потока управления оказывают сильное влияние на итоговое значение сходства. Несмотря на низкое значение сходство программа верно сопоставила 4 из 5 графов потока управления. При использовании --ins_block_sim_threshold 0.5 сходство повышается до 0.29 (+0.09). Однако, понижение значения порога эквивалентности блоков инструкций повышает количество схожих между собой графов потока управления, что может привести к формированию некорректных пар сходства.

В пятом эксперименте значение сходства 0.19 показывает, что два принципиально разных Android-приложения незначительное сходство. Пары сходства, сформированные программой, в большинстве не имеют схожей структуры. Пары со сходством приближенным к 1 представляют собой графы потоков простых конструкторов классов и не несут полезной информации о сходстве всего приложения.

10. Заключение

Работа посвящена проектированию и разработке системы автоматизации численной оценки сходства Android-приложений. Были выполнены следующие задач включая: проведен анализ форм представления программ, видов моделей программ, методов построения моделей программ, видов сходства программ. Предложены модель представления Android-приложений, методы ее построения и алгоритм оценки сходства моделей. Спроектирован и разработана программный модуль для построения модели программы и численной оценки сходства.

Задача численной оценка сходства Android-приложений имеет ряд пракитических применений. Напрмер, оценка может использоваться платформами, где публикуются приложения. Например, Google Play Store, RuStore, Huawei AppGallery и др. для поиска приложений плагиатов. Функционал поиска плагинов позволит площадкам оповещать разработчиков-авторов о краже их интеллектуальной собственности, оповещать пользователей площадок о подозрительности приложений с высокой степенью сходства, запрещать публиковать приложения и др. Также численная оценка сходства Android-приложений может быть использована для оценки работы обфускаторов, например, ProGuard, R8 и др., при оценке кражи интеллектуальной собственности в судебных делах и прочих сценариях.

Разработанная система является прототипов решения задачи численной оценка сходства Android-приложений. Выполненные в работе эксперименты показывают состоятельность разработанного решения, показывают. Среди преимуществ: высокая скорость работы, репрезентативность модели с точки зрения структуры программы, инструменты для экспертного анализа. Среди недостатков: чувствительность к обфускации. Для более точной оценки эффективности работы системы необходимо провести дополнительные эксперименты.

Дальнейшие развитие исследования предполагает: повысить точность вычисления расстояния редактирования графов. Несмотря на то, что программа корректно формирует пары схожих графов потока управления, значения их сходства не равно 1.0. Также необходимо решить вопрос с учетом оценки графов малой мощности.

Текущая версия приложения размещена в открытом доступе в репозитории GitHub и доступна по ссылке: https://github.com/valeryvpetrov-dev/android-apps-similarity.

Литература

1. Zhou W., Zhou Y., Jiang X., Ning P. Detecting repackaged smartphone applications in third-party android marketplaces / Zhou W., Zhou Y., Jiang X., Ning P. // Second ACM conference on Data and Application Security and Privacy. — 2012. — P. 317-326. — doi: 10.1145/2133601.2133640.
2. Crussell J., Gibler C., Chen H. Attack of the clones: Detecting cloned applications on android markets / Crussell J., Gibler C., Chen H. // European Symposium on Research in Computer Security. — 2012. — P. 37-54. — doi: 10.1007/978-3-642-33167-1_3.
3. Market Shocker! Iron Soldiers XDA Beta Published by Alleged Thief // Android Headline . — https://www.androidheadlines.com/2011/01/market-shocker-iron-soldiers-xda-beta-published-by-alleged-thief.html.
4. Fake Mobile Apps Steal Facebook Credentials, Cryptocurrency-Related Keys // TREND MICRO. — https://www.trendmicro.com/en_us/research/22/e/fake-mobile-apps-steal-facebook-credentials--crypto-related-keys.html.
5. Android App Bundle frequently asked question // Android developers. — https://developer.android.com/guide/app-bundle/faq.
6. Akhunzada A., Sookhak M., Anuar N.B., Gani A., Ahmed E., Shiraz M., Furnell S., Hayat A., Khan M.K. Man-At-The-End attacks: Analysis, taxonomy, human aspects, motivation and future directionsAttack of the clones: Detecting cloned applications on android markets / Akhunzada A., Sookhak M., Anuar N.B., Gani A., Ahmed E., Shiraz M., Furnell S., Hayat A., Khan M.K. // Journal of Network and Computer Applications. — 2015. — № 48. — P. 44-57. — doi: 10.1016/j.jnca.2014.10.009.
7. The Java® Virtual Machine Specification // Oracle. — https://docs.oracle.com/javase/specs/jvms/se7/html/.
8. Android Runtime (ART) and Dalvik // Android Open Source Project. — https://source.android.com/docs/core/runtime/.
9. Ratazzi E.P. Understanding and improving security of the Android operating system : PhD dissertation / Ratazzi E.P. ; Syracuse University. — , 2016. — https://surface.syr.edu/etd/592/. — https://www.researchgate.net/publication/316793316_Understanding_and_Improving_Security_of_the_Android_Operating_System.
10. Cesare S., Xiang Y. Software similarity and classification / Cesare S., Xiang Y. — 1. — : Springer London, 2012 — 88 p. — 10.1007/978-1-4471-2909-7.

11. Jones J. Abstract Syntax Tree Implementation Idioms / Jones J. // Proceedings of the 10th conference on pattern languages of programs (plop2003). — 2003. — P. 26. — https://hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf.
12. Heck A.J.P. OOP: Class Hierarchy / Heck A.J.P. // Persoonlijke pagina's van FNWI-medewerkers Personal pages of Science staff. — https://staff.fnwi.uva.nl/a.j.p.heck/Courses/JAVAcourse/ch3/s1.html.
13. Ferrante J., Ottenstein K.J., Warren J.D. The program dependence graph and its use in optimization / Ferrante J., Ottenstein K.J., Warren J.D. // ACM Transactions on Programming Languages and Systems (TOPLAS). — 1987. — № 9(3). — P. 319-349. — doi: 10.1145/24039.24041.
14. Callahan D., Carle A., Hall M.W., Kennedy K. Constructing the procedure call multigraph / Callahan D., Carle A., Hall M.W., Kennedy K. // IEEE Transactions on Software Engineering. — 1990. — № 16(4). — P. 483-487. — doi: 10.1109/32.54302.
15. Allen F.E. Control flow analysis / Allen F.E. // ACM Sigplan Notices. — 1970. — № 5(7). — P. 1-19. — doi: 10.1145/800028.808479.
16. Kruegel C., Kirda E., Mutz D., Robertson W., Vigna G. Polymorphic worm detection using structural information of executables / Kruegel C., Kirda E., Mutz D., Robertson W., Vigna G. // Recent Advances in Intrusion Detection: 8th International Symposium. — 2006. — № 8. — P. 207-226. — doi: 10.1007/11663812.
17. Marcelli A., Quer S., Squillero G. The maximum common subgraph problem: A portfolio approach / Marcelli A., Quer S., Squillero G. // arXiv preprint. — 2019. — https://www.researchgate.net/publication/335258488_The_Maximum_Common_Subgraph_Problem_A_Portfolio_Approach.
18. Abu-Aisheh Z., Raveaux R., Ramel J.Y., Martineau P. An exact graph edit distance algorithm for solving pattern recognition problems / Abu-Aisheh Z., Raveaux R., Ramel J.Y., Martineau P. // 4th International Conference on Pattern Recognition Applications and Methods. — 2015. — № 1. — doi: 10.5220/0005209202710278.
19. Левенштейн, В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов / В. И. Левенштейн // Доклады Академий Наук СССР. — 1965. — № 163.4. — С. 845-848. — https://www.mathnet.ru/links/ebbbb75259f2fb388db92a54ec642b7d/dan31411.pdf.
20. Критерии сходства программ // ООО «АйТи-Лекс». — http://www.it-lex.ru/legal-cases/skhodstvo-programm/.
21. Myles G., Collberg C. K-gram based software birthmarks / Myles G., Collberg C. // Proceedings of the 2005 ACM symposium on Applied computing. — 2005. — P. 314-318. — doi: 10.1145/1066677.1066753.

22. Liu C., Chen C., Han J., Yu P.S. GPLAG: detection of software plagiarism by program dependence graph analysis / Liu C., Chen C., Han J., Yu P.S. // Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. — 2006. — P. 872-881. — doi: 10.1145/1150402.1150522.