УДК 004.4

МОДЕЛЬ И АРХИТЕКТУРА МНОГОУРОВНЕВОГО АНАЛИЗА СХОДСТВА ANDROID-ПРИЛОЖЕНИЙ ПО СТАТИЧЕСКИМ ПРИЗНАКАМ

В. В. Петров[0009-0004-4213-7328]

Казанский (Приволжский) федеральный университет, г. Казань, Россия

valeryvpetrov.itis@gmail.com

Аннотация

Рассмотрена задача многоуровневого анализа сходства приложений для платформы Android по статическим признакам в цифровых коллекциях мобильных приложений. В таких коллекциях встречаются дубликаты, ответвленные версии, перепакованные приложения и иные модифицированные варианты; вредоносная нагрузка рассматривается как возможный частный случай модификации, а не как синоним перепаковки.

Формализована функция сходства приложений по статическим признакам, построена статическая модель приложения и предложена архитектура анализа, разделяющая предварительный отбор кандидатов, углубленное сопоставление, интерпретацию результата и слой формирования заключения. Показано, что значимая информация о близости приложений содержится не только в байткоде classes.dex, но и в манифесте AndroidManifest.xml, ресурсах, APK-внутренних метаданных и библиотечных зависимостях. Численная оценка сходства вычисляется только при успешном построении статических моделей сравниваемых приложений; в противном случае фиксируется отдельный служебный технический статус с нормализованной причиной отказа.

На локальном пилотном наборе из пяти основных пар и двух граничных случаев наблюдалось, что явный учет библиотечных зависимостей и раздельная фиксация технических ограничений прототипа позволяют получить более интерпретируемый результат, однако эти данные следует рассматривать как предварительные и не дающие оснований для окончательной валидации архитектуры на больших коллекциях.

Ключевые слова: *приложение для платформы Android, статический анализ, анализ сходства программ, поиск модифицированных вариантов, перепакованные приложения, библиотечная зависимость, интерпретация результата, цифровая коллекция приложений*.

ВВЕДЕНИЕ

Задача поиска модифицированных вариантов приложений для платформы Android представляет практический и исследовательский интерес при работе с цифровыми коллекциями мобильных приложений. К таким вариантам относятся:

- дубликаты, т. е. копии без существенных содержательных изменений;
- ответвленные версии, сохраняющие общее происхождение, но развивающиеся по самостоятельной линии;
- перепакованные приложения, в которых исходный программный пакет модифицируется путем добавления, удаления или замены отдельных компонентов;
- иные модифицированные варианты, для которых вредоносная нагрузка является возможным частным случаем модификации, а не обязательным признаком перепаковки.

Для практики разработки, сопровождения, контроля качества и анализа безопасности требуется не только устанавливать факт сходства таких приложений, но и выявлять, какие именно признаки обусловливают их близость или различие. В сценариях перепаковки и привнесения постороннего кода к легитимному приложению добавляется сторонняя функциональность, изменяются ресурсы, расширяется набор разрешений или подключаются новые библиотеки [1]. В этих случаях аналитически значимы не только ответ на вопрос о наличии сходства, но и структура интерпретации результата: связано ли сходство с собственным кодом приложения, библиотечными включениями, изменениями манифеста либо с ресурсным слоем. Для тематики электронных библиотек это соответствует сценарию, когда в цифровом фонде APK-артефактов (Android Package Kit Artifact) требуется найти близкие версии одного и того же приложения, выделить перепакованные экземпляры и объяснить,

какие именно статические признаки послужили основанием для такого сопоставления.

Для тематики электронных библиотек это соответствует сценарию, когда в цифровом фонде APK-артефактов требуется найти близкие версии одного и того же приложения, выделить перепакованные экземпляры и объяснить, какие именно статические признаки послужили основанием для такого сопоставления. В этом случае нужны не только ранжированный поиск по коллекции, но и воспроизводимая интерпретация причин сходства.

Во многих прикладных системах поиска модифицированных вариантов анализ сводится к вычислению одной численной оценки сходства для пары приложений. Такая схема обладает ограниченной пригодностью для реального анализа. Во-первых, при росте коллекции приложений полный попарный перебор быстро становится вычислительно дорогим. Во-вторых, единый итоговый показатель скрывает структуру результата: аналитик не видит, связано ли сходство с собственным кодом приложения, повторным использованием библиотек, со сходством манифеста или ресурсным слоем. В-третьих, при неуспешном построении статической модели приложения технический исход может быть ошибочно интерпретирован как содержательно низкое сходство.

Переход к текущей постановке опирается на ранее опубликованные работы автора. В работе [2] был анонсирован базовый режим численной оценки сходства приложений для платформы Android. В [3, 4] этот подход был развит до метода, основанного на анализе байткода и сравнении графов потока управления. Эти работы служат воспроизводимой отправной точкой байткод-ориентированного этапа исследования сходства Android-приложений. Однако текущий этап разработки метода показывает, что их недостаточно рассматривать как окончательную форму анализа сходства. Приложение для платформы Android представляет собой более сложный объект, чем только classes.dex: в практически значимых сценариях сигналы сходства содержатся также в AndroidManifest.xml, ресурсах, APK-внутренних метаданных и библиотечных зависимостях.

В связи с этим актуальной становится задача перехода от сравнения, опирающегося только на байткод, к многоуровневому анализу сходства по статическим признакам.

В настоящей статье рассмотрены формализация этой задачи, архитектура соответствующего анализа и требования к экспериментальному контуру, который не смешивает предварительный отбор кандидатов, углубленное сопоставление, интерпретацию результата и служебные технические статусы вычислительного контура.

К числу основных результатов работы относятся следующие положения. Во-первых, формализована функция сходства приложений по статическим признакам. Во-вторых, введена статическая модель приложения как рабочий объект многоуровневого анализа. В-третьих, предложена архитектура многоуровневого контура, в которой предварительный отбор, углубленное сопоставление, интерпретация результата и слой формирования заключения рассматриваются как различные уровни обработки. В-четвертых, уточнены требования к экспериментальному контуру проверки такой архитектуры и приведены предварительные результаты пилотного экспериментального цикла без обобщающих утверждений о завершенной валидации метода.

По сравнению с работами [2–4] в настоящей работе добавляется три конкретных элемента. Во-первых, байткод-ориентированное сравнение расширено до многоуровневой статической модели, включающей манифест, ресурсы, APK-внутренние метаданные и библиотечные зависимости. Во-вторых, архитектура анализа явно разделяет предварительный отбор кандидатов и углубленное сопоставление, что позволяет рассматривать задачу не только как попарное сравнение, но и как поиск по доверенной коллекции. В-третьих, в экспериментальный контур введена отдельная фиксация служебных технических статусов и граничных случаев, что позволяет не смешивать ограничения текущего прототипа с содержательным результатом сравнения.

МЕТОДОЛОГИЧЕСКИЕ ОСНОВАНИЯ

Общую теоретическую рамку для анализа сходства программ задают работы по сходству программ и программным отпечаткам, где сходство рассматривается как результат извлечения признаков и сравнения представлений программных объектов [5]. Для приложений платформы Android эта линия получила развитие в исследованиях, ориентированных на статический

анализ байткода, перепаковку приложений и обнаружение вредоносных модификаций [1, 6, 7].

С одной стороны, имеются работы, в которых сходство выводится из сравнения программных структур и кода. К ней относятся, в частности, подходы на основе статического анализа байткода, сравнения сигнатур методов и графов потока управления [4, 6]. Эти методы ценны тем, что опираются на содержательные программные признаки и позволяют строить численную оценку сходства. Однако для практически значимых сценариев они часто оказываются либо вычислительно дорогими, либо недостаточно устойчивыми к обфускации, структурным преобразованиям и влиянию общих библиотечных компонентов.

С другой стороны, развиваются подходы, ориентированные на предварительный отбор кандидатов или использование альтернативных статических описаний приложения. Так, в работе [8] сравнение приложений построено по ресурсному слою, что показывает самостоятельную диагностическую ценность не только кода, но и ресурсов приложения. Для прикладных сценариев проверки коллекций приложений для платформы Android развивались и более масштабируемые линии анализа, где предварительный отбор близких приложений выполняется отдельно от последующего углубленного разбора (см., например, [9]).

Значимый исследовательский шаг связан с переходом от одного итогового показателя сходства к интерпретируемому анализу. В работе [10] показано, что практически полезный инструмент должен не только возвращать итоговое значение сходства, но и указывать структуру различий: совпадающие методы, новые методы, удаленные методы и другие типы изменений. Эта линия имеет особое значение для задач экспертной проверки, где требуется не только ранжирование кандидатов, но и интерпретация причин их близости.

Отдельный класс работ посвящен перепаковке и привнесению постороннего кода в приложения платформы Android. Для этих сценариев принципиальным является то, что сходство может проявляться не только на уровне кода, но и на уровне ресурсов, компонентов манифеста, разрешений и библиотечных включений [1]. Тем самым исследования в этой области усиливают аргумент против узкой постановки, опирающейся только на

байткод, и поддерживают переход к многослойной статической модели приложения.

Еще одна значимая линия связана с обработкой влияния общих библиотечных компонентов. Под этим влиянием далее понимается ситуация, в которой совпадающий библиотечный код искусственно увеличивает итоговую оценку сходства или скрывает различия между собственным кодом приложений. Работы по детекции библиотек в Android-приложениях показывают, что сторонние библиотечные компоненты широко распространены, существенно влияют на результаты статического анализа и требуют отдельного выявления для задач безопасности, обнаружения перепаковки и последующего сравнения приложений [11–13]. Следовательно, модуль учета библиотек не должен быть скрытой эвристикой внутри итогового показателя; его необходимо рассматривать как отдельный воспроизводимый компонент анализа.

Наконец, обзор литературы выявил и проблему сопоставимости результатов. Несмотря на большое число публикаций по обнаружению перепакованных приложений, результаты многих исследований плохо сопоставимы из-за закрытых наборов данных, слабой воспроизводимости и различия в уровне детализации эталонных разметок [9]. Это означает, что новая система анализа сходства должна проектироваться одновременно с явным контуром экспериментальной проверки, который различает основной набор валидных пар и набор граничных случаев.

Таким образом, отмеченные выше исследования поддерживают несколько принципиальных выводов. Во-первых, сходство не должно сводиться к одному числу без интерпретации структуры результата. Во-вторых, практически применимая система должна разделять предварительный и углубленный анализ. В-третьих, приложение необходимо описывать не только через код, но и через иные статические признаки. В-четвертых, влияние общих библиотечных компонентов и служебные технические статусы вычислительного контура должны учитываться явно, а не скрываться внутри единой метрики.

ФОРМАЛИЗАЦИЯ ЗАДАЧИ

Пусть Ω – множество приложений для платформы Android, рассматриваемых в задаче статического анализа. Требуется задать функцию сходства

$$S: \Omega \times \Omega \to [0, 1],$$

где $S(A, B)$ определяет нормированную оценку сходства приложений A и B по выбранному набору статических признаков.

В этой постановке равенство $S(A, B) = 1$ означает идентичность приложений A и B в рамках рассматриваемых статических характеристик, а $S(A, B) = 0$ – отсутствие общих значимых статических характеристик в рамках принятой формализации. Эти значения не следует трактовать как абсолютную идентичность или полное отсутствие общих элементов вне границ рассматриваемой модели сходства.

Для реализации такой функции введем оператор M построения статической модели приложения. Для каждого приложения $X \in \Omega$ оператор M либо строит статическую модель $M(X)$, либо в текущем прототипе возвращает служебный технический статус FAIL, означающий невозможность корректно построить модель средствами текущего вычислительного контура. Статическая модель приложения в настоящей работе понимается как структурированное описание приложения, включающее выбранные статические признаки и отношения между ними. Такая модель может строиться по одному слою или согласованной комбинации нескольких слоев: байткоду, компонентам и свойствам AndroidManifest.xml, ресурсам, APK-внутренним метаданным, библиотечным зависимостям или по их сочетанию. Внешние магазинные атрибуты и внешние пользовательские метки в данную модель не входят.

В настоящей работе численная оценка сходства вычисляется только тогда, когда модели для обоих приложений построены успешно. В этом случае используется оператор сравнения Φ и выполняется соотношение

$$S(A, B) = \Phi(M(A), M(B)), \qquad 0 \le \Phi(M(A), M(B)) \le 1.$$

Если оператор M не может корректно построить модель хотя бы для одного приложения пары, значение $S(A, B)$ не вычисляется. Вместо него

фиксируются служебный технический статус FAIL и нормализованная причина отказа. В текущем прототипе различаются по меньшей мере следующие причины: input_model_failed для входного приложения, fast_signature_failed для неуспешного построения быстрой сигнатуры и candidate_model_missing для отсутствия корректной модели кандидата в репозитории доверенной коллекции. Это различение необходимо: отсутствие сходства и невозможность корректно выполнить анализ относятся к разным типам исходов работы системы. Первый исход является результатом сравнения приложений, второй — техническим отказом вычислительного контура. Поэтому они должны фиксироваться разными статусами и интерпретироваться отдельно.

Исследовательские задачи в такой постановке состоят в следующем:

- определить, какие статические модели приложения являются содержательно значимыми для анализа сходства;
- построить нормированную оценку сходства приложений по их статическим признакам;
- развести содержательные исходы анализа и служебные технические статусы текущего вычислительного контура;
- задать воспроизводимый экспериментальный контур для проверки предварительного отбора, углубленного сопоставления и качества интерпретации.

Функции практически применимой системы анализа при этом включают:

- предварительный отбор кандидатов для сокращения пространства сравнения;
- углубленное сопоставление ограниченного набора наиболее релевантных пар;
- интерпретацию результата в терминах использованных моделей и влияния общих библиотечных компонентов;
- формирование заключения для эксперта на основе оценки сходства и структурированной интерпретации.

Тем самым система анализа сходства рассматривается не как единичный вычислитель итогового показателя, а как многоуровневый контур обработки. Значимым следствием этой постановки является отказ от предположения, что один и тот же слой признаков должен одновременно обеспечивать и масштабируемый поиск по большой коллекции, и глубокий содержательный разбор каждой пары.

АРХИТЕКТУРА И АЛГОРИТМ

Предлагаемая архитектура базируется на том, что практически применимая система анализа сходства должна быть составной и многоуровневой. Сначала целесообразно показать сам ход анализа входного приложения относительно доверенной коллекции. Именно эту процессную сторону архитектуры отражает рис. 1.

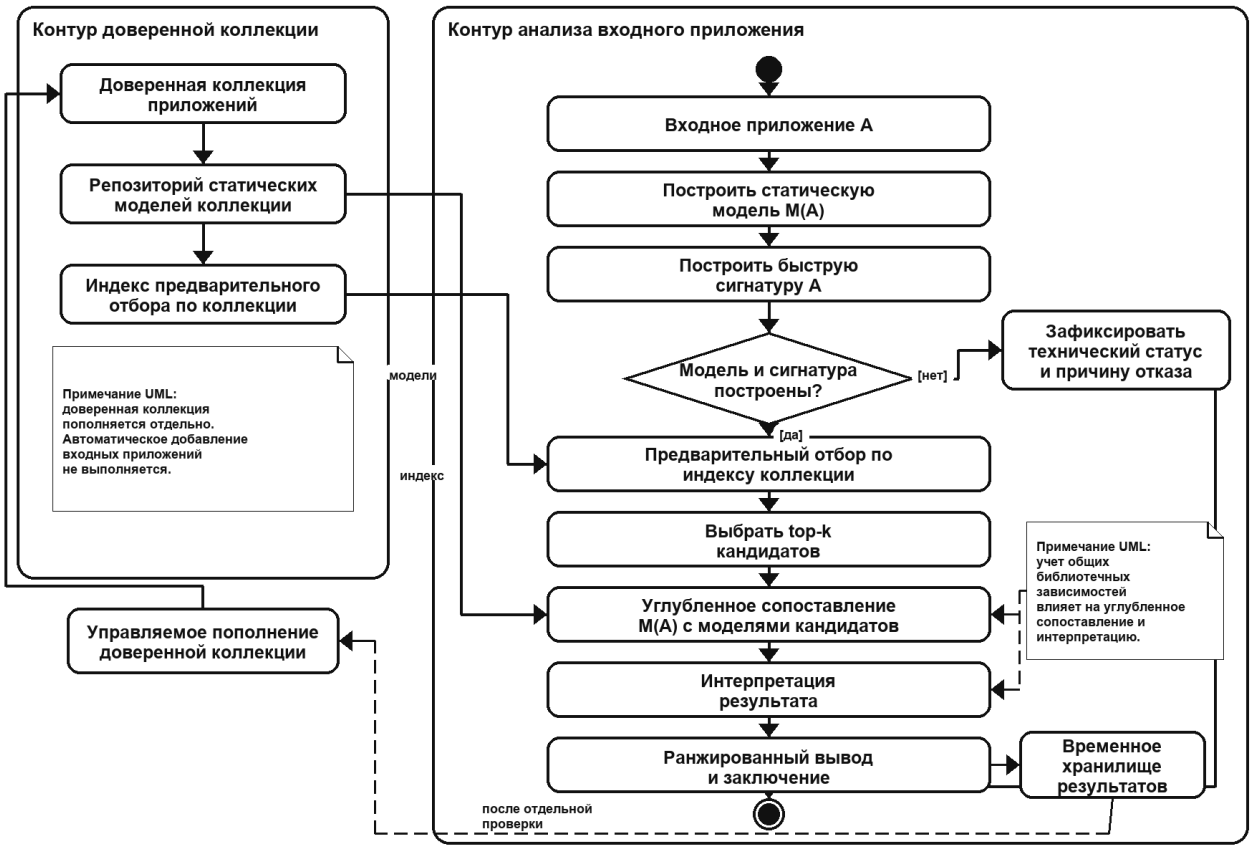


Рис. 1. UML-диаграмма деятельности двухэтапного анализа входного приложения относительно доверенной коллекции приложений для платформы Android.

На рис. 1 представлены два связанных контура: контур доверенной коллекции и контур анализа входного приложения. В первом контуре хранятся ранее построенные статические модели коллекции и индекс предварительного отбора, обеспечивающие накопление знаний системы. Во втором контуре для входного приложения *A* строятся статическая модель и быстрая сигнатура, после этого по индексу коллекции отбираются кандидаты для углубленного сопоставления. Таким образом диаграмма деятельности описывает не полный попарный перебор по коллекции, а управляемый переход от быстрого отбора к детальному сопоставлению ограниченного числа кандидатов. Отдельная техническая ветвь в этой схеме отражает служебный статус текущего вычислительного контура, а не самостоятельный содержательный результат сравнения. Входное приложение не включается автоматически в доверенную коллекцию: допускаются лишь временное хранение результатов и отдельное управляемое пополнение основной базы. Из этой процессной схемы следуют основные архитектурные принципы системы многоуровневого анализа сходства.

1. Многоуровневая статическая модель приложения: приложение для платформы Android не должно моделироваться только через код classes.dex. Для задач клонирования, перепаковки и анализа вредоносных модификаций значимы также компоненты и атрибуты AndroidManifest.xml, статические ресурсы, метаданные сборки и библиотечные зависимости.

2. Явное разделение предварительного и углубленного анализа: полный углубленный разбор всех пар приложений в коллекции плохо масштабируется, поэтому предварительный слой должен отбирать кандидатов по сравнительно дешевым признакам, а углубленный слой — запускаться только на ограниченном наборе пар.

3. Обязательная интерпретация результата: итог анализа не должен ограничиваться единственным числом. Необходимо возвращать и нормированную оценку сходства, и структурированную интерпретацию, показывающую, в каких слоях приложения обнаружено сходство или различие.

4. Явный учет влияния общих библиотечных компонентов: повторное использование библиотек может искусственно завышать оценку сходства между независимыми приложениями и скрывать различия между основной логикой

приложения и внешними зависимостями. Поэтому влияние библиотечного слоя должно быть трассируемым и интерпретируемым.

5. Разведение содержательных исходов сравнения и технических ограничений вычислительного контура: система должна различать собственно результат сравнения и служебный статус, возникающий при невозможности корректно построить модель или извлечь признаки средствами текущего прототипа.

6. Хранение и повторное использование статических моделей: практически применимая архитектура должна содержать доверенную коллекцию ранее обработанных приложений, репозиторий их статических моделей и индекс предварительного отбора, чтобы не извлекать все признаки заново для уже известных объектов при каждом новом сравнении.

7. Правильное место слоя формирования заключения: прикладной слой может использовать оценку сходства и материалы интерпретации для подготовки заключения, но не должен смешиваться с ядром анализа сходства.

Далее, после констатации этих принципов, покажем, как они раскладываются на составные подсистемы. Такое структурное представление приведено на рис. 2.

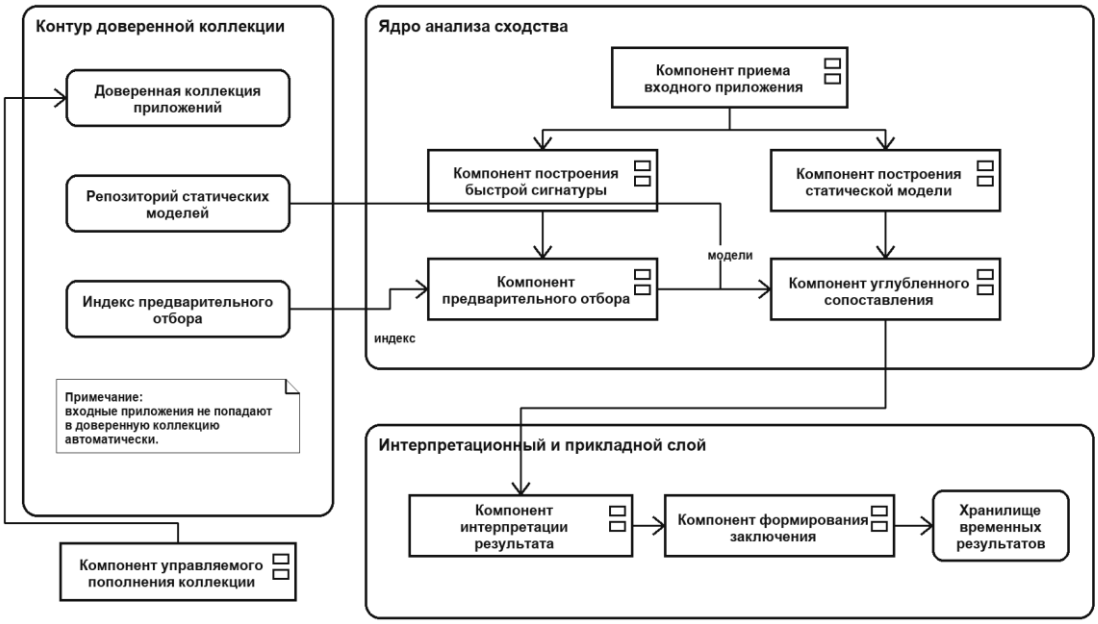


Рис. 2. UML-диаграмма компонентов системы многоуровневого анализа сходства приложений для платформы Android.

На рис. 2 ядро анализа разделено на подсистемы приема входного приложения, построения статической модели, формирования быстрой сигнатуры, предварительного отбора кандидатов, углубленного сопоставления, интерпретации результата и формирования заключения. Отдельно выделены доверенная коллекция приложений, репозиторий статических моделей, индекс предварительного отбора, временное хранилище результатов и управляемое пополнение доверенной коллекции. В настоящей работе доверенная коллекция понимается как курируемый оператором набор APK-артефактов и заранее построенных моделей, пригодных для повторного использования в поисковом контуре. Такое представление позволяет развести процессный уровень и уровень составных частей системы: рис. 1 отвечает на вопрос о ходе анализа, а рис. 2 – на вопрос о составе подсистем и их зависимостях.

Основные этапы многоуровневого анализа можно свести к следующим положениям.

1. Предварительный отбор: по быстрой сигнатуре входного приложения и индексу доверенной коллекции формируется упорядоченный список кандидатов.

2. Углубленное сопоставление: для входного приложения и отобранных кандидатов вычисляется нормированная оценка сходства на уровне статических моделей.

3. Интерпретация результата: по данным углубленного сопоставления фиксируются источники сходства и различия, включая вклад библиотечного слоя.

4. Формирование заключения: на основе оценки сходства и структурированной интерпретации готовится заключение для эксперта.

Формализованное описание двухэтапного контура, согласованное с обозначенными уровнями, дано в Алгоритме 1. Вход: входное приложение a, индекс коллекции I, репозиторий статических моделей R, оператор моделирования M, ограничение k. Выход: множество кандидатов K, оценки сходства S, структурированные интерпретации E, заключения L, служебные технические статусы F и нормализованные причины отказа G.

Алгоритм 1. Двухэтапный контур многоуровневого анализа сходства приложений для платформы Android

```
ANALYZE_SIMILARITY(a, I, R, M, k)
1   X[a] ← BuildStaticModel(a, M)
2   if X[a] = NIL
3       then return FAIL, input_model_failed
4   Z[a] ← BuildFastSignature(a)
5   if Z[a] = NIL
6       then return FAIL, fast_signature_failed
7   ▷ Предварительный отбор по доверенной коллекции
8   q ← SearchIndex(Z[a], I)
9   K ← SelectTopCandidates(q, k)
10  for each b ∈ K
11      do Y[b] ← LoadCollectionModel(R, b)
12         if Y[b] = NIL
13             then F[b] ← FAIL
14                  G[b] ← candidate_model_missing
15             else S[a, b] ← ComputeSimilarity(X[a], Y[b])
16                  E[a, b] ← BuildInterpretation(X[a],Y[b],S[a,b])
17                  L[a, b] ← BuildConclusion(S[a, b], E[a, b])
18  return K, S, E, L, F, G
```

ЭКСПЕРИМЕНТАЛЬНЫЙ КОНТУР

Экспериментальный контур нужен не для демонстрации одной итоговой цифры, а для раздельной проверки различных слоев предлагаемой архитектуры. Поэтому отдельно рассматриваются предварительный отбор кандидатов, углубленное сопоставление, интерпретация результата и устойчивость вычислительного контура. Такое разбиение позволяет проследить, на каком именно этапе возникает ограничение или, наоборот, наблюдается улучшение результата. Все результаты этого раздела следует рассматривать как пилотные наблюдения на локальном наборе данных, а не как завершенную валидацию метода на больших коллекциях.

В текущем пилотном цикле экспериментальный контур был разделен на основной и граничный. В качестве основного пилотного набора использовался локальный набор пар dataset-v2-core. Под этим обозначением понимается набор из пяти пар APK-файлов с зафиксированным происхождением, доступными локальными артефактами, анализируемым байткодом и успешным прохождением исходного байткод-ориентированного режима без специальных обходов. В его состав входят три пары с общим происхождением (P01–P03) и две контрольные пары без общего происхождения (P05, P06).

Отдельно от основного набора рассматривался граничный контур устойчивости. В него включались случаи, которые не следует использовать ни для калибровки рабочего порога, ни для итоговой оценки качества на основном наборе. Пары P11 и P12 были нужны для разведения двух разных ограничений: риска чрезмерного подавления полезного сигнала на шаге библиотечной редукции и риска нестабильного построения модели на паре с общим происхождением.

До прямого сопоставления базового и улучшенного режимов была восстановлена воспроизводимость исходного байткод-ориентированного контура: после исправлений совместимости удалось повторить пять исторических значений сходства на локальном пилотном материале. Для оценки разделимости классов на основном наборе использовалась разность между минимальной оценкой сходства в группе пар с общим происхождением и максимальной оценкой в контрольной группе без общего происхождения. Чем больше эта разность, тем лучше режим разделяет похожие и непохожие пары.

На основном наборе оба режима завершили анализ всех пяти пар без служебного технического статуса. Ключевые результаты этого сопоставления приведены ниже.

1. Основной пилотный набор: восстановлены пять исторических значений, а dataset-v2-core включает пять пар APK-файлов: три пары с общим происхождением и две контрольные пары без общего происхождения. Этого достаточно для сопоставления базового и улучшенного режимов на пилотном материале, но недостаточно для статистически полной валидации.

2. Разделение классов: в базовом режиме минимальная оценка в группе пар с общим происхождением оказалась ниже максимальной оценки в контрольной группе на 0.041804, тогда как в улучшенном режиме она превысила максимальную оценку контрольной группы на 0.135935. На данном пилотном наборе это наблюдение указывает на более отчетливое разделение похожих и непохожих пар, но еще не доказывает превосходства режима на широком корпусе.

3. Порог 0.15: при этом пороге базовый режим дал три верно распознанные близкие пары, один ложный сигнал сходства и один верно распознанный контрольный случай, тогда как улучшенный режим сохранил все

три верно распознанные близкие пары и устранил ложный сигнал сходства. Покрытие слоя интерпретации одновременно выросло с 4/5 до 5/5. В текущей работе порог 0.15 используется как иллюстративная рабочая граница пилотного прогона, а не как окончательно валидированная граница решения.

4. Граничные случаи P11 и P12: для P11 после библиотечной редукции значение сходства снизилось с 0.2 до 0, что указывает на риск чрезмерного подавления полезного сигнала. Для P12 в стандартной конфигурации фиксировался статус FAIL, тогда как в последовательной конфигурации оценка изменилась с 0.80543 до 0.276018. Это показывает, что в граничных случаях текущий прототип пока не полностью разделяет ограничения метода и ограничения реализации.

ОБСУЖДЕНИЕ

Предлагаемая постановка полезна не только для задач анализа приложений для платформы Android как таковых, но и для более общего контекста работы с большими цифровыми коллекциями приложений. В тематике электронных библиотек существенна не столько рекомендация в узком пользовательском смысле, сколько задача поиска, сопоставления и интерпретации сходства между элементами цифровой коллекции. В этом отношении предлагаемый контур анализа сходства может рассматриваться как инструмент интеллектуального поиска по коллекции приложений, где ранжирование кандидатов образует первый этап работы, а исследовательская ценность возникает на уровне интерпретации результата и воспроизводимости оснований для вывода.

С методической точки зрения необходимо удерживать границы текущих утверждений. Настоящая работа не доказывает масштабируемость анализа на больших рыночных коллекциях, не вводит окончательно валидированный многоуровневый метод и не утверждает завершенность слоя формирования заключения. Ее вклад состоит в последовательном расширении предшествующей байткод-ориентированной линии наших работ: формализуется функция сходства приложений по статическим признакам, вводится статическая модель приложения как рабочий объект анализа, а экспериментальная проверка выносится в самостоятельный контур, в котором

технические ограничения текущего прототипа фиксируются отдельно от содержательного результата сравнения.

Научная новизна настоящей работы определяется четырьмя положениями. Во-первых, функция сходства приложений формулируется для многоуровневого набора статических признаков, а не только для байткодного слоя. Во-вторых, статическая модель приложения вводится как явный рабочий объект анализа, включающий манифест, ресурсы, APK-внутренние метаданные и библиотечные зависимости. В-третьих, архитектура анализа строится как двухконтурная схема работы с доверенной коллекцией, где предварительный отбор и углубленное сопоставление разведены по ролям. В-четвертых, экспериментальный контур отделяет пилотный основной набор от граничных случаев и не смешивает ограничения текущего прототипа с содержательным результатом сравнения.

ЗАКЛЮЧЕНИЕ

Сформулирована задача многоуровневого анализа сходства приложений для платформы Android по статическим признакам, введена статическая модель приложения как рабочий объект сравнения, предложена архитектура соответствующего анализа и уточнен экспериментальный контур его проверки. Показано, что переход от базового режима, ориентированного только на итоговый показатель и только на байткод, к многоуровневому контуру обусловлен природой самого объекта исследования: приложение для платформы Android содержит значимые сигналы сходства не только в коде, но и в манифесте, ресурсах, метаданных и библиотечных зависимостях.

Предложенная архитектура исходит из разделения предварительного и углубленного анализа, обязательной интерпретации результата, явного учета влияния общих библиотечных компонентов и отдельной фиксации технических ограничений вычислительного контура. Отдельно подчеркнем, что слой формирования заключения может быть естественной прикладной надстройкой над слоем интерпретации, но не должен подменять собой ядро анализа сходства.

Результаты пилотного экспериментального цикла согласуются с гипотезой о полезности явного учета влияния общих библиотечных компонентов и

раздельной фиксации технических ограничений текущего прототипа, однако не дают оснований для обобщающих утверждений о завершенной валидации архитектуры на больших коллекциях. Дальнейшая работа связана с реализацией и сравнительной оценкой отдельных слоев многоуровневого контура, расширением многоуровневой статической модели приложения, повышением устойчивости вычислительного контура и построением воспроизводимого экспериментального контура на основных и граничных наборах данных.

СПИСОК ЛИТЕРАТУРЫ

1. *Li L. et al.* Understanding Android App Piggybacking: A Systematic Study of Malicious Code Grafting // IEEE Transactions on Information Forensics and Security. 2017. Vol. 12, No. 6. P. 1269–1284. https://doi.org/10.1109/TIFS.2017.2656460

2. *Петров В.В.* Система автоматизации численной оценки сходства Android-приложений // Научный сервис в сети Интернет: труды XXV Всероссийской научной конференции (18–21 сентября 2023 г., онлайн). М.: ИПМ им. М.В. Келдыша, 2023. С. 283–297. https://doi.org/10.20948/abrau-2023-33

3. *Петров В.В.* Система автоматизации численной оценки сходства Android-приложений // Электронные библиотеки. 2024. Т. 27, № 3. С. 336–365. https://doi.org/10.26907/1562-5419-2024-27-3-336-365

4. *Petrov V.V.* Automated System for Numerical Similarity Evaluation of Android Applications // Automatic Documentation and Mathematical Linguistics. 2024. Vol. 58 (Suppl. 3). P. 131–142. https://doi.org/10.3103/S0005105525700207

5. *Cesare S., Xiang Y.* Software Similarity and Classification. London: Springer, 2012. 88 p. https://doi.org/10.1007/978-1-4471-2909-7

6. *Desnos A.* Android: Static Analysis Using Similarity Distance // Proc. of the 45th Hawaii International Conference on System Sciences. 2012. P. 5394–5403. https://doi.org/10.1109/HICSS.2012.114

7. *Rastogi V., Chen Y., Jiang X.* DroidChameleon: Evaluating Android Anti-Malware Against Transformation Attacks // Proc. of the 8th ACM SIGSAC. 2013. P. 329–334. https://doi.org/10.1145/2484313.2484355

8. *Zhauniarovich Y. et al.* FSquaDRA: Fast Detection of Repackaged Applications // Data and Applications Security and Privacy XXVIII. 2014. P. 130–145. https://doi.org/10.1007/978-3-662-43936-4_9

9. *Li L., Bissyande T. F., Klein J.* Rebooting Research on Detecting Repackaged Android Apps // IEEE Transactions on Software Engineering. 2021. Vol. 47, No. 4. P. 676–693. https://doi.org/10.1109/TSE.2019.2901679

10. *Li L., Bissyande T. F., Klein J.* SimiDroid: Identifying and Explaining Similarities in Android Apps // 2017 IEEE Trustcom/BigDataSE/ICESS. 2017. P. 136–143. https://doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.230

11. *Backes M., Bugiel S., Derr E.* Reliable Third-Party Library Detection in Android and Its Security Applications // Proc. of the ACM Conf. on Computer and Comm. Security. 2016. P. 356–367. https://doi.org/10.1145/2976749.2978333

12. *Li M. et al.* LibD: Scalable and Precise Third-Party Library Detection in Android Markets // Proc. of the 39th Int. Conf. on Software Engineering. 2017. P. 335–346. https://doi.org/10.1109/ICSE.2017.38

13. *Huang J. et al.* Scalably Detecting Third-Party Android Libraries With Two-Stage Bloom Filtering // IEEE Transactions on Software Engineering. 2023. Vol. 49, No. 4. P. 2272–2284. https://doi.org/10.1109/TSE.2022.3215628

MODEL AND ARCHITECTURE OF MULTI-LEVEL SIMILARITY ANALYSIS OF ANDROID APPLICATIONS BASED ON STATIC FEATURES

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

Kazan Federal University, Kazan, Russia

valeryvpetrov.itis@gmail.com

Abstract

The paper addresses the problem of multi-level similarity analysis of Android applications based on static features in digital application collections. Such collections may contain duplicates, forks, repackaged builds, and other modified variants; malicious payloads are treated as a special case of modification rather than as a synonym of repackaging. The paper formulates a similarity function for Android applications, introduces a static application model as the working object of comparison, and presents a multi-level pipeline that separates candidate screening, in-depth pairwise analysis, result interpretation, and a decision layer. Meaningful

similarity signals are sought not only in classes.dex bytecode, but also in AndroidManifest.xml, resources, APK-internal metadata, and library dependencies. A numerical similarity score is computed only when static models are built successfully; otherwise the pipeline records a dedicated technical failure status together with a normalized failure reason. Preliminary evidence is reported on a local pilot set of five core pairs and two boundary cases. These results indicate that explicit handling of shared library code may improve interpretability, but they do not yet constitute a full validation of the proposed architecture on large collections.

Keywords: *Android applications, static analysis, program similarity analysis, search for modified variants, repackaged applications, library dependencies, result interpretation, digital collections of applications.*

REFERENCES

1. *Li L. et al.* Understanding Android App Piggybacking: A Systematic Study of Malicious Code Grafting // IEEE Transactions on Information Forensics and Security. 2017. Vol. 12, No. 6. P. 1269–1284. https://doi.org/10.1109/TIFS.2017.2656460

2. *Petrov V.V.* System of Automated Numerical Similarity Evaluation of Android Applications // Nauchnyi servis v seti Internet: Trudy XXV Vserossiiskoi nauchnoi konferentsii. 2023. P. 283–297. https://doi.org/10.20948/abrau-2023-33

3. *Petrov V.V.* System of Automated Numerical Similarity Evaluation of Android Applications // Russian Digital Libraries Journal. 2024. Vol. 27, No. 3. P. 336–365. https://doi.org/10.26907/1562-5419-2024-27-3-336-365

4. *Petrov V.V.* Automated System for Numerical Similarity Evaluation of Android Applications // Automatic Documentation and Mathematical Linguistics. 2024. Vol. 58 (Suppl. 3). P. 131–142. https://doi.org/10.3103/S0005105525700207

5. *Cesare S., Xiang Y.* Software Similarity and Classification. London, Springer, 2012. 88 p. https://doi.org/10.1007/978-1-4471-2909-7

6. *Desnos A.* Android: Static Analysis Using Similarity Distance // Proc. of the 45th Hawaii International Conference on System Sciences. 2012. P. 5394–5403. https://doi.org/10.1109/HICSS.2012.114

7. *Rastogi V., Chen Y., Jiang X.* DroidChameleon: Evaluating Android Anti-Malware Against Transformation Attacks // Proc. of the 8th ACM SIGSAC. 2013. P. 329–334. https://doi.org/10.1145/2484313.2484355

8. *Zhauniarovich Y. et al.* FSquaDRA: Fast Detection of Repackaged Applications // Data and Applications Security and Privacy XXVIII. 2014. P. 130–145. https://doi.org/10.1007/978-3-662-43936-4_9

9. *Li L., Bissyande T. F., Klein J.* Rebooting Research on Detecting Repackaged Android Apps // IEEE Transactions on Software Engineering. 2021. Vol. 47, No. 4. P. 676–693. https://doi.org/10.1109/TSE.2019.2901679

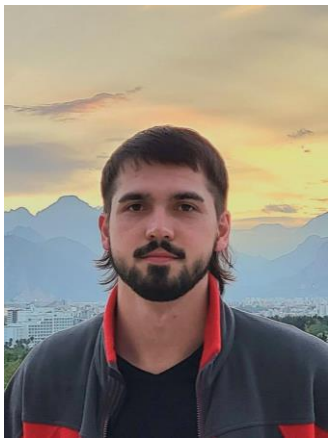
10. *Li L., Bissyande T. F., Klein J.* SimiDroid: Identifying and Explaining Similarities in Android Apps // 2017 IEEE Trustcom/BigDataSE/ICESS. 2017. P. 136–143. https://doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.230

11. *Backes M., Bugiel S., Derr E.* Reliable Third-Party Library Detection in Android and Its Security Applications // Proc. of the ACM Conf. on Computer and Comm. Security. 2016. P. 356–367. https://doi.org/10.1145/2976749.2978333

12. *Li M., Wang W., Wang P. et al.* LibD: Scalable and Precise Third-Party Library Detection in Android Markets // Proc. of the 39th International Conference on Software Engineering. 2017. P. 335–346. https://doi.org/10.1109/ICSE.2017.38

13. *Huang J., Zhang Y., Tan H. et al.* Scalably Detecting Third-Party Android Libraries With Two-Stage Bloom Filtering // IEEE Transactions on Software Engineering. 2023. Vol. 49, No. 4. P. 2272–2284. https://doi.org/10.1109/TSE.2022.3215628

СВЕДЕНИЯ ОБ АВТОРЕ



ПЕТРОВ Валерий Владимирович – магистр программной инженерии, аспирант Института информационных технологий и интеллектуальных систем Казанского федерального университета. Научные интересы: анализ сходства приложений для платформы Android, статический анализ программ, интерпретируемые методы сравнения программных артефактов, воспроизводимые исследовательские контуры.

Valery Vladimirovich PETROV – Master of Software Engineering, postgraduate student at the Institute of Information Technology and Intelligent Systems, Kazan Federal University. Research interests: Android application similarity, static program analysis, interpretable comparison of software artifacts, reproducible research workflows**.**

email: valeryvpetrov.itis@gmail.com

ORCID: 0009-0004-4213-7328

Материал поступил в редакцию 20 марта 2026 года