НЕДОСТАТКИ В ПРОТОКОЛЕ РАЗРЕШЕНИЙ ANDROID ПРИ ОГРАНИЧЕННОЙ ВЕРИФИКАЦИИ

Ерохин В.В.¹

Цель статьи: анализ протокола разрешения, реализованного в операционной системе Android, как наиболее популярной для смартфонов и других электронных гаджетов; рассмотреть формальную модель протокола разрешений Android и описать автоматический анализ безопасности этой модели; выявить потенциальные недостатки в протоколе разрешений.

Метод исследования: применяется формальная модель протокола разрешений Android на основе C++ с использованием Java NDK, базирующимся на реляционной логике первого порядка, с механизмом анализа, который выполняет ограниченную проверку моделей.

Полученный результат. Создана формальная модель протокола разрешений Android с использованием C++ с использованием Java NDK. Модель определила недостатки в протоколе разрешений Android, и тем самым выявила уязвимости безопасности Android. Разработанная модель разрешений протокола Android состоит из трех частей: запрос архитектуры устройства Android; запрос схемы разрешений Android; системные операции. Исправлены недостатки в ОС Android, связанные с уязвимостью настраиваемых разрешений. Представлен эксперимент, демонстрирующий осуществимость и распространенность уязвимости настраиваемых разрешений в существующих приложениях Android. Исследование реальных приложений Android подтверждает наш вывод о том, что недостатки в протоколе разрешений Android могут иметь серьезные последствия для безопасности приложений электронного гаджета, и в некоторых случаях позволяет злоумышленнику полностью обойти проверки разрешений. Проведенное исследование одной из уязвимостей показало, что она широко распространена среди многих существующих приложений Android. Большинство разработчиков не выполняют никаких дополнительных проверок, чтобы убедиться, что входящие API-интерфейсы поступают от доверенных приложений или поставщиков, предполагая, что они могут не знать об уязвимости настраиваемых разрешений, несмотря на ее потенциальную возможность нарушения безопасности. Полученный результат будет полезен для разработчиков программного обеспечения под операционные системы с разрешениями – Android, iOS и Fire OS.

Ключевые слова: вредоносное программное обеспечение, мобильная безопасность, уровень API, мобильное приложение, программирование, динамический анализ, исходный код, моделирование, операционная система.

DOI: 10.21681/2311-3456-2021-1-2-17

1. Введение

Современные мобильные устройства предоставляют основу для взаимодействия нескольких приложений друг с другом путем экспорта и вызова API. С точки зрения безопасности и конфиденциальности некоторые ресурсы, совместно используемые через API, могут считаться более важными, чем другие; например, возможность отправить текстовое сообщение более опасно, чем возможность изменить мелодию звонка в телефоне. Следовательно, очень важен механизм, который может использо-

вать разработчик для управления доступом к критическим ресурсам.

Популярные операционные системы, такие как Android, iOS и Fire OS, реализуют модель на основе разрешений для управления типами ресурсов, к которым разрешен доступ каждому приложению. В этой модели разработчик защищает критический ресурс внутри приложения, назначая явное разрешение, которое должно быть получено любым приложением, которое хочет получить доступ к ресурсу.

¹ Ерохин Виктор Викторович, доктор технических наук, доцент, профессор кафедры математических методов и бизнес-информатики Московского государственного института международных отношений (университет), Москва, Россия. E-mail: erohinv@mail.ru, ORCID 0000-0002-8754-0012

Разрешения обычно предоставляются приложению по усмотрению конечного пользователя, который принимает решение на основе предполагаемой надежности приложения.

В последние годы исследователи выявили ряд недостатков в механизмах разрешений, которые приводят к серьезным нарушениям безопасности и конфиденциальности [1, 2, 3]. Типичный способ обнаружения этих проблем – это тщательное изучение экспертами по безопасности влияния разрешений на безопасность приложений. Многие проблемы связаны с общими недостатками конструкции операционных систем и их систем разрешений, которые требуют общесистемного обоснования. Эти проблемы нелегко решить с помощью традиционных методов анализа, таких как тестирование и статический анализ, которые больше подходят для обнаружения ошибок в отдельных частях системы.

Подобно тому, как приемы формальных методов оказались практичными при оценке безопасности сетевых протоколов [4, 5], на основе нашего опыта считаем, что построение формальной модели протокола разрешений и выполнение тщательного анализа может выявить потенциальные уязвимости и возможные исправления. В статье не будет проведен анализ кода для проверки конкретного приложения на наличие уязвимостей, вместо этого будут рассмотрены моделирование и анализ протокола разрешений Android на предмет недостатков проектирования приложений. Модель безопасности приложений написана на основе C++ с использованием Java NDK, базирующимся на реляционной логике первого порядка, с механизмом анализа, который выполняет ограниченную проверку моделей.

Представленная модель безопасности приложений позволила выявить совокупность уязвимостей в протоколе, которые позволяют вредоносному приложению полностью обходить проверки разрешений. В частности, проведено исследование уязвимости настраиваемого разрешения. Чтобы подтвердить, что абстрактный сценарий атаки, выявленный в ходе анализа, действительно реалистичен, рассмотрена атака на конкретные приложения Android в разных версиях Android. В ходе исследования показано, что уязвимость, связанная с настраиваемыми разрешениями, широко распространена и что многие популярные приложения на самом деле уязвимы для этого типа атак.

2. Предпосылки и мотивация

Приложение является основной функциональной единицей в Android: на типичном устройстве постоянно работают многочисленные приложения для поддержки потребностей пользователя, такие как почто-

вый клиент, служба обмена сообщениями, приложение для навигации и многое другое. Успех Android отчасти объясняется его гибкой структурой для обмена данными между приложениями. Каждое приложение организовано в виде набора компонентов, которые экспортируют API-интерфейсы в другие приложения, что позволяет многократно использовать функциональность несколькими поставщиками проектов и программного обеспечения. Например, разработчик навигационного приложения может инкапсулировать свои функции поиска по карте в отдельный компонент и предоставить их в качестве услуги для остальной части устройства. Существует четыре типа компонентов: служба (*service*), действие (*activity*), широкоэмиттерный приемник (*broadcast receiver*) и контент-провайдер (*content provider*), каждый из которых предназначен для обеспечения функциональности ОС.

Потенциальным недостатком открытой платформы Android является повышенный риск нарушения безопасности и конфиденциальности. Некоторые компоненты обрабатывают информацию, которая считается особенно важной, и поэтому свободный доступ к этим компонентам без ограничения прав доступа может привести к нежелательным последствиям для пользователя. Например, навигационное приложение может не захотеть показывать историю поиска по карте как часть компонентного API, так как мошенническое приложение может использовать эти данные в злонамеренных целях для экстраполяции схемы передвижения пользователя [6, 7].

Android использует механизм на основе разрешений для управления взаимодействием приложений друг с другом. Прежде чем приложение сможет получить доступ к компоненту, ему должно быть предоставлено явное разрешение на это пользователем. Каждое разрешение связано с уровнем защиты, который указывает на надежность приложения, которому может быть предоставлено это разрешение. Существует несколько уровней защиты:

- 1) нормальный, то есть разрешение предоставляется каждому приложению;
- 2) опасный, предоставляется только по усмотрению пользователя устройства;
- 3) подпись (*signature*), предоставляемая только приложениям от одного разработчика;
- 4) подпись/система (*signature/system*), предоставляемая только приложениям спроектированных для определенной ОС (используется редко, поэтому не будет использован в этой статье).

Механизм выполнения отслеживает каждый вызов операции API и гарантирует, что вызывающее приложение имеет разрешение на выполнение этой операции.

Устройства с ОС Android содержат ряд встроенных разрешений для основных функций, таких как отправка текстового сообщения, включение GPS и доступ в Интернет. Кроме того, ОС Android позволяет стороннему приложению определять пользовательские разрешения и выборочно контролировать доступ к своим компонентам и приложениям. Обычно разрешения предоставляются приложению во время его установки; однако особый тип разрешений, называемый разрешениями URI (Uniform Resource Identifiers, Унифицированный Идентификатор Ресурса), может быть временно предоставлен и отменен в течение жизненного цикла приложения.

Цель протокола разрешений ОС Android – предотвратить любой несанкционированный доступ. Каждое приложение должно иметь доступ только к тем компонентам ОС, для которых ему предоставлены разрешения, и не более того. Однако обеспечение этой цели, является сложной задачей, так как особенно трудно предсказать все способы, которыми вредоносное приложение может попытаться использовать систему не по назначению. Атака может включать выполнение сложной, но непонятной последовательности операций, которые вряд ли встретятся в обычных сценариях использования ОС. Выявление таких атак требует общесистемного мышления и не может быть легко достигнуто с помощью традиционных методов анализа, таких как тестирование и статический анализ, которые больше подходят для обнаружения уязвимостей в отдельных частях ОС. Руководствуясь этой целью, далее представлен подход к анализу безопасности протокола разрешений ОС Android на основе построения формальной модели и её анализа. В этом анализе используются два основных аспекта:

1. Общесистемное динамическое обоснование. Выполняются общесистемные рассуждения при моделировании поведения ОС Android в терминах операций архитектурного уровня (таких как установка или удаление приложения), выполняемых в течение последовательности дискретных временных шагов. Такой анализ может исследовать все возможные варианты безопасности и проверить, может ли злоумышленник использовать конкретный набор разрешений для реализации реальной атаки с использованием настраиваемых разрешений.

2. Конкретизация. Результат анализа абстрактной модели используется для управления безопасностью ОС Android на уровне реализации, т.е. проверяет конкретное приложение на наличие уязвимости.

Использование этих аспектов в анализе безопасности ОС Android демонстрирует потенциальную синергию между методами анализа на основе моделей и кода.

3. Модель разрешений ОС Android

В этом разделе описывается формальная модель протокола разрешений Android с использованием C++ с использованием Java NDK. Это язык позволяет моделировать и интегрировать различные элементы ОС или приложения, имеет в своем составе инструментарий, который обеспечивает автоматический анализ для проверки утверждений и генерацию контрпримеров.

Разработанная модель разрешений протокола Android состоит из трех частей:

- 1) запрос архитектуры устройства Android, алгоритм представлена на рис. 1;
- 2) запрос схемы разрешений Android, алгоритм представлен на рис. 2;
- 3) системные операции, которые изменяют разрешения или зависят от них (рис. 3 ... рис. 6).

Оператор `sig` (рис. 1) вводит подпись, которая определяет набор элементов. Подпись может содержать одно или несколько полей, каждое из которых вводит отношение, которое отображает элементы подписи в выражение поля; например, поле `ProtectionLevel` в `Permission` (рис. 2, блок 3) – это двоичное отношение, которое сопоставляет каждый объект `Permission` с его уровнем защиты (рис. 2, блок 4). Класс `extends` (рис. 2, блок 3) создает отношения подтипов между двумя подписями; подпись `abstract` не имеет элементов, кроме принадлежащих ее расширениям, а `one sig` вводит подпись, которая содержит только один элемент.

3.1. Разрешения

Устройство Android состоит из ряда взаимодействующих приложений, каждое из которых может либо не содержать, либо содержать несколько компонентов, которые могут экспортировать службы в другие приложения. Набор приложений, работающих на устройстве, может со временем меняться по мере установки новых приложений и удаления существующих. Моделируем динамический аспект системы, в которой выполнение представлено как последовательность временных шагов, и каждый изменяемый объект связан с различным состоянием на каждом временном шаге. Для этого вводим набор полностью упорядоченных элементов как сигнатуру `Time` и добавляем ее в качестве последнего столбца отношений, которые считаются изменяемыми; например, поле `apps` использует `Time` для отслеживания установленных приложений на каждом временном шаге (рис. 1, блок 5). Библиотека `ordering` устанавливает общий порядок входной подписи (рис. 1, блок 2).

Приложение может использовать разрешения для управления доступом к своим компонентам со

Рис. 1. Запрос архитектуры устройства Android в модели протокола разрешений ОС Android

стороны других приложений. Каждый объект разрешения, показанный на рис. 2 в блоке 3, связан с именем и уровнем защиты, который может принимать одно из трех значений: *Normal*, *Dangerous* и *Signature* (в порядке возрастания критичности). Разрешения могут быть назначены приложению на двух разных уровнях. Каждый компонент может быть защищен не более чем одним разрешением (представленным полем *guard* рис. 1 в блоке 12), которое должно быть получено приложением перед тем, как получить доступ к компоненту. Кроме того, приложению может быть назначена собственная защита (рис. 1, блок 11), которая накладывается на каждый из его компонентов; когда и у приложения, и у одного из его компонентов есть защита, приоритет имеет разрешение для конкретного компонента.

Тип защиты поля как в приложении, так и в компоненте это только *PermName*. Защита не содержит информации об уровне защиты, предназначенной для компонента, к которому осуществляется доступ, что является недостатком конструкции Android и может быть использовано вредоносным приложением для несанкционированного доступа.

В дополнение к набору встроенных разрешений, доступных по умолчанию в Android, разработчик приложения может создать одно или несколько

настраиваемых разрешений для защиты компонента, специфичного для приложения (рис. 1, блок 6 и блок 7). Например, каждое устройство Android содержит встроенное разрешение под названием *android.permission.INTERNET*, определяющее, каким приложениям разрешено использовать встроенный компонент, обеспечивающий доступ в Интернет. Стороннее навигационное приложение может предоставлять свои возможности поиска по карте как услугу для других приложений и определять настраиваемое разрешение под названием *com.myapp.perm.SEARCH_MAP* для управления своим доступом.

Поставщик контента (*content provider*) – это тип компонента хранилища, содержащий одну или несколько таблиц базы данных, которые идентифицируются URI (рис. 1, блок 13). Другие типы компонентов – службы, действия и широкоэмиттерный приемник – могут обрабатываться одинаково относительно разрешений (не показаны на рис. 1). По умолчанию, получение разрешения у поставщика контента обеспечивает доступ ко всем его таблицам базы данных. Чтобы обеспечить более детальный контроль, Android предоставляет специальный тип разрешений, называемый разрешениями URI (рис. 2, блок 5), который можно использовать для предо-

Рис. 2. Запрос схемы разрешений Android в модели протокола разрешений ОС Android

ставления доступа к определенному URI внутри поставщика контента.

Далее приложение указывает свое намерение получить доступ к компоненту, включая имя связанного разрешения в качестве одного из разрешений на использование (рис. 1, блок 10). Когда приложение установлено, устройство определяет набор разрешений, которые должны быть предоставлены приложению, с помощью `usesPerms`.

3.2. Поведение системы

В модели описаны четыре типа операций, относящихся к схеме разрешений Android:

- 1) вызов компонента, который успешен только в том случае, если вызывающее приложение имеет соответствующее разрешение (рис. 3);
- 2) установка приложения, которое может изменять настраиваемые разрешения на устройстве (рис. 4);
- 3) удаление приложения, которое может изменять настраиваемые разрешения на устройстве (рис. 5);
- 4) трассировка событий (рис. 6).

Операция вызова компонента. Операция вызова компонента, вызывающего другой компонент, выра-

жается как предикат `invoke` (рис. 3, блоки 2...5), который оценивается как истинный тогда и только тогда, когда `caller` успешно вызывает `callee` между временными шагами `t` и `t1`. Предикат, в свою очередь, определяется как сочетание трех ограничений: 1) `caller`, и `callee` должны принадлежать какому-либо приложению на устройстве (рис. 3, блок 3); 2) `caller` должен иметь разрешение на доступ к `callee` (рис. 3, блок 4); 3) никакие изменения не вносятся в активные разрешения во время вызова (рис. 3, блок 5).

Предикат `canCall` определяет, что означает для `caller` возможность вызывать `callee` на временном шаге `t` (рис. 3, блок 6); то есть `caller` должен обладать разрешением, которое охраняет `callee` (операторы «+» и «in» являются соответственно операторами объединения и подмножества). При этом, `callee` может быть защищен вообще без разрешения (т.е. `guardedBy` может возвращать пустой набор), и в этом случае `canCall` тривиально удовлетворяется. А значит к компоненту без защиты может получить доступ любой другой компонент.

Рис. 3. Вызов компонента приложения Android в модели протокола разрешений ОС Android

Рис. 4. Установка приложения Android в модели протокола разрешений ОС Android

Рис. 5. Удаление приложения Android в модели протокола разрешений ОС Android

Операция установки приложения. Первое ограничение в `install` (рис. 4, блок 2) описывает предва- рительное условие для операции: `app` (приложение) не должен еще существовать на устройстве в момент времени t (рис. 4, блок 3). Следующие четыре огра- ничения описывают влияние операции на устрой- ство:

- Если `app` объявляет свои собственные раз- рещения, они добавляются на устройство, за исключением тех, которые уже существуют на устройстве в момент времени t ; функция `newCustomPerms` описывает новые разре- шения, которые необходимо добавить (рис. 4, блок 5).
- Каждое разрешение, запрашиваемое `app` в его `usesPerms`, предоставляется устройством новому приложению (рис. 4, блок 6).
- Разрешения, предоставленные другим прило- жениям на устройстве, не затрагиваются.
- `app` добавлен в набор существующих прило- жений на устройстве.

В этой модели неявно подразумевается процесс предоставления разрешения через одобрение поль- зователя; `grantPermissions` упрощенно устанавливает предоставленные разрешения для разреше- ний в `usesPerms` приложения (рис. 4, блок 6), не описывая, как принимается решение о каждом раз- решении. Этот выбор моделирования определяет не- достаточную безопасность разрешений Android. Если приложению не будут предоставлены все его разре- шения на использование, оно не будет установлено

на устройстве. Пользователь не имеет возможности выборочно предоставлять разрешения, что являет- ся основным источником проблем с удобством ис- пользования и конфиденциальностью в Android [1, 8]. Другими словами, независимо от того, как предо- ставляются разрешения, всегда каждое установлен- ное приложение будет обладать всеми разрешения- ми, которые оно запрашивает.

Операция удаления приложения. Эта операция (рис. 5) удаляет указанное приложение с устройства, а также все связанные с ним настраиваемые разре- шения. Разрешения, предоставленные любому дру- гому приложению, остаются неизменными во время операции.

Операция трассировки событий. Трассировка `fact traces` определяет поведение системы как совокупность трассировок, которые она может про- изводить (рис. 6). `fact` – это ограничение, которое выполняется для каждого удовлетворительного эк- земпляра модели. Концептуально, трассировка – это последовательность временных шагов, где между каждой парой смежных шагов, t и $t1$, выполняется одна или несколько системных операций. Такое определение трассировки исключает заикливание. В этом случае также можно добавить операцию, представляющую `noop` (команда протокола переда- чи данных, которая предписывает бездействие CPU). Учитывая это определение, удовлетворительный эк- земпляр модели, найденный анализатором, будет со- ответствовать ровно одной из возможных вариантов системы.

Рис. 6. Трассировка событий в модели протокола разрешений ОС Android

В нашей модели присутствуют и другие типы операций протокола разрешений: различные типы компонентов (помимо поставщиков контента), динамическое распределение и проверка разрешений URI и подписи приложений.

Необходимо учитывать, что `guard`, относящийся к компоненту, – это просто наименование разрешения, и поэтому его уровень защиты, по замыслу, не играет роли в определении того, следует ли `caller` вызывать `callee`. Такое проектное решение основано на одном критическом предположении: если приложение имеет разрешение на доступ к компоненту с определенным уровнем защиты, то оно должно быть авторизовано пользователем во время установки. Однако, как покажет наш анализ, это предположение неверно: вредоносное приложение может получить разрешение на компонент с высоким уровнем защиты (например, опасный), даже если авторизация предназначалась для более низкого уровня защиты (например, нормально). Далее подробно опишем такую атаку.

4. Анализ

В этом разделе описываем автоматизированный анализ, чтобы проверить, удовлетворяет ли протокол разрешений Android, указанный в нашей модели,

своей цели по предотвращению несанкционированного доступа.

Утверждение в нашей модели используется для указания свойства, которому должна удовлетворять модель. Когда предлагается проверить утверждение, анализатор исследует все возможные варианты поведения системы и находит контрпример, если таковой имеется, который соответствует нарушению утверждения. Анализ является исчерпывающим, но ограничен заданной пользователем областью действия по размеру доменов: если в области есть контрпример, анализатор гарантированно найдет его, но отсутствие контрпримера не подразумевает действительности утверждения. На практике многие недостатки системы могут быть продемонстрированы на небольшом количестве объектов [9, 10], и при желании пользователь может итеративно повторно проанализировать модель с большими объемами, чтобы получить дополнительную уверенность.

Важным свойством безопасности Android является то, что каждый вызов компонента авторизован; то есть, когда компонент вызывает другой компонент, вызываемому должно быть предоставлено разрешение, объявленное разработчиком для защиты вызываемого.

Рис. 7. Утверждения о протоколе разрешений Android

Это свойство формально указано как утверждение `NoUnauthorizedAccess` на рис. 7. Предикат `authorized` описывает, что означает для компонента `caller` быть авторизованным для вызова `callee`. Его определение основывается на двух разных типах разрешений: `grantedPerm` представляет собой разрешение, которое предоставляется `caller` во время его установки; `requiredPerm` представляет собой настраиваемое разрешение, которое было объявлено специально для защиты `callee`. Тогда считается, что `caller` имеет право вызывать `callee` только в том случае, если уровень защиты `grantPerm` равен или выше уровня `requiredPerm`.

4.1. Уязвимость настраиваемых разрешений

Анализ. При активации анализатора будет предложено проверить утверждение, после чего анализатор возвращает контрпримерную трассировку, которая демонстрирует, как недостаток конструкции в Android может привести к нарушению свойства. Анализ шести доменов и занимает до 0.4 секунды на CPU Intel Core i5-10400.

Визуализация контрпримера показана на рис. 8. В этой трассировке приложение №0 (`Application_0`) объявляет настраиваемое разрешение (`Permission_1`) для защиты своего компонента (помеченного как `victim_1`) с уровнем защиты `Signature`, что означает – только те приложения, которые используют одну и ту же подпись, могут получить к нему доступ. Отдельное вредоносное приложение `Application_1` обходит требование подписи, используя уязвимости в Android: а именно, оно позволяет нескольким приложениям определять настраиваемые разрешения с одним и тем же именем, но без четкой спецификации того, какое из них должно иметь приоритет, если у них разные уровни защиты.

Для проведения этого типа атаки `Application_1` объявляет собственное настраиваемое разрешение (`Permission_0`) с тем же именем, что и `Permission_1`, но с самым низким уровнем защиты `Normal`. Атака состоит из следующих трех операций:

- Шаг 1 (рис. 8): `Application_1` устанавливается до `Application_0`, активируя его настраиваемое разрешение (`Permission_0`) с уровнем защиты `Normal` на устройстве.
- Шаг 2 (рис. 9): `Application_0` установлено, но настраиваемое разрешение с тем же именем уже активно, поэтому разрешение (`Permission_1`) игнорируется. В результате `Application_1` продолжает иметь то же разрешение, которое было предоставлено на шаге 2.
- Шаг 3 (рис. 10): вредоносный компонент внутри `Application_1` может получить доступ к

жертве, несмотря на то, что у него нет такой же сигнатуры, как у `Application_0`.

Одним из вариантов исправления этого недостатка является запрет нескольким приложениям, которые определяют настраиваемое разрешение с тем же именем, одновременно существовать на устройстве. В нашей модели такое исправление можно выразить, добавив блок между блоками 2 и 3 на рис. 4 следующее ограничение к операции установки:

```
no p.name in (Device.
customPerms.t).name
```

Утверждение `NoUnauthorizedAccess` (рис. 7) можно проанализировать и по-другому сценарию. Сценарий начинается так же, как и сценарий на рис. 8, где вредоносное приложение (`Application_1`) определяет собственное настраиваемое разрешение с тем же именем, что и другое разрешение, но с более низким уровнем защиты. Кроме того, `Application_1` позволило установить другое вредоносное приложение (`Application_2`), использующее это разрешение. На следующем шаге приложение `Application_1` удаляется, и соответствующее настраиваемое разрешение удаляется с устройства. Однако Android не может отозвать такое же разрешение у приложений, которые его используют (а именно, `Application_1`), что приводит к зависанию разрешения. Когда приложение-жертва (`Application_0`) установлено, `Application_2` по-прежнему может получить доступ к компоненту-жертве, но с более низким уровнем защиты, который был определен `Application_1`.

Значит просто запретить установку приложений с одинаковыми разрешениями недостаточно. Операция удаления также должна быть изменена, чтобы гарантировать, что предоставленные разрешения отменяются при удалении приложения, объявляющего эти разрешения. Это можно сделать, изменив ограничение в блоке 5 на рис. 5 следующим образом:

```
all a.grantedPerms.t1 =
a.grantedPerms.t - app.
declaredPerms
```

4.2. Другие обнаруженные уязвимости

Анализ выявил два других типа уязвимостей в протоколе разрешения. Кратко их обсудим.

Ошибка разрешения URI. Вредоносное приложение может получить разрешение URI для части поставщика контента или приложения, доступ к которой у него нет. Эта уязвимость связана с другим недостатком протокола разрешений Android: предоставленные разрешения URI не аннулируются при удалении связанного поставщика контента, оставляя

Недостатки в протоколе разрешений Android при ограниченной верификации

Рис. 8. Шаг 1, несанкционированный доступ к компоненту-жертве со стороны вредоносного Application_1 через неправильное использование настраиваемого разрешения

Рис. 9. Шаг 2, несанкционированный доступ к компоненту-жертве со стороны вредоносного Application_1 через неправильное использование настраиваемого разрешения

Рис. 10. Шаг 3, несанкционированный доступ к компоненту-жертве со стороны вредоносного Application_1 через неправильное использование настраиваемого разрешения

активные разрешения, которые можно использовать для атак аналогичного типа, как в разделе 4.1. Уязвимость с разрешениями URI существует до версии Android 3.0. С версии Android 3.0 и выше разрешения URI отменяются во время удаления, что предотвращает атаку.

Неправильное делегирование. Вредоносное приложение может иметь возможность косвенно вызывать компонент, не имея на это разрешения, путем взаимодействия с третьим компонентом, у которого есть разрешение. Эта уязвимость была идентифицирована как атака повторного делегирования разрешений [1, 11, 12].

5. Эксперименты

Строгий анализ формальной модели, такой как описанная в разделе 4, может быть использован для выявления потенциальных недостатков на уровне проектирования, но сам по себе не формирует полного анализа безопасности системы. Вместо этого формальный анализ должен дополняться систематическим анализом конкретной системы, чтобы подтвердить, могут ли эти недостатки привести к реальным уязвимостям и последующим атакам.

В этом разделе представляем экспериментальное исследование, чтобы ответить на следующие два исследовательских вопроса:

1. Могут ли недостатки, выявленные в нашем формальном анализе протокола разрешений An-

droid, вызвать настоящую атаку с серьезными последствиями для безопасности?

2. Насколько уязвимы реальные приложения Android для атак, вызванных рассмотренными недостатками в протоколе разрешений Android?

В частности, мы сосредоточимся на уязвимости пользовательского разрешения, т.к. она ранее не изучалась в литературе. Уязвимость разрешения URI опущена, поскольку она существует только в устаревшей версии Android, а неправильный поток делегирования уже изучался в [1, 11]. Для решения проблемы, поставленной в вопросе 1, были разработаны приложения, которые представляют вредоносное поведение как на рис. 8 ... рис. 10, и наблюдалось – можно ли обойти требование разрешения. Для решения проблемы, поставленной в вопросе 2, были проведены исследования 45 реальных приложений Android и количественно измерили распространенность уязвимости безопасности из-за недостатков, обнаруженных в протоколе разрешений Android.

5.1. Демонстрация атаки

Чтобы проверить выполнимость модели на рис. 8 ... рис. 10, было разработано приложение-скелет телефонной записной книги, которое соответствует приложению-жертве в трассировке (Application_0 на рис. 8 ... рис. 10). На рис. 11 частично показан файл манифеста (manifest) Android для этого приложения (строки 1-8). Файл манифеста содержит, среди прочего, объявления об использовании и настраиваемых

```

1 <permission android:name="com.example. PhoneBook_READ"
2   android:protectionLevel="signature" />
3 <application android:label=" PhoneBook">
4   <provider android:name=".PhoneBookProvider"
5     android:authorities=". PhoneBookProvider"
6     android:readPermission="com.example.PhoneBook_READ"
7   </provider>
8 </application>
9 <permission android:name="com.example. PhoneBook_READ"
10   android:protectionLevel="normal" />
11 <uses-permission android:name="com.example. PhoneBook_READ" />
12 <application android:label="PhoneApp">
13   <activity android:name=".PhoneActivity" android:label="PhoneApp" >
14     <intent-filter>
15       <action android:name="MAIN" />
16       <category android:name="LAUNCHER" />
17     </intent-filter>
18   </activity>
19 </application>

```

Рис. 11. Фрагменты приложений, спроектированные для модели атаки, показанной на рис. 8 ... рис. 10.

разрешениях для приложения. Он определяет настраиваемое разрешение с именем `PhoneBook_READ` (рис. 11) с уровнем защиты подписи (рис. 11, строки 1–2). Затем это разрешение указывается как средство защиты (рис. 11, строка 6) для защиты доступа к компоненту `PhoneBookProvider` (рис. 11, строки 3–6), в котором хранится содержимое телефонной записной книги.

Как заявлено в манифесте (рис. 11), приложение `PhoneBook` не предоставляет доступ к своим данным какому-либо другому приложению. Таким образом, ожидается, что только приложениям, которые явно запрашивают разрешение `PhoneBook_READ` (рис. 11) и подписаны такой же подписью, будет разрешено читать содержимое телефонной записной книги.

Затем разработали приложение, которое имеет вредоносное поведение (рис. 11, строки 9–19). Это часть реализации файла манифеста для `PhoneApp` (соответствует `Application_1` на рис. 8 ... рис. 10). Подобно приложению телефонной записной книги, оно объявляет разрешение `PhoneBook_READ`, хотя и с более низким уровнем защиты, нормальным (`normal`). Он также включает элемент `uses-permission`, чтобы объявить, что ему требуется самообъявленное настраиваемое разрешение (рис. 11, строка 11). Компонент `PhoneActivity`, который представляет собой вредоносный компонент, затем просто отправляет запрос компоненту `PhoneBookProvider`.

Два приложения были подписаны разными ключами, чтобы отразить реальный сценарий, в котором они были бы от разных разработчиков. Затем эти приложения были установлены и выполнены, согласно рис. 8 ... рис. 10, в двух версиях Android SDK – 4.0 и 9.0 – под эмулятором Genymotion (www.genymotion.com). Повторили эксперименты с разными комбинациями уровней защиты для `PhoneBook` и `PhoneApp`. Во всех случаях наблюдалось, что `PhoneApp` успешно смог получить доступ к содержимому телефонной записной книги, что подтвердило возможность атаки.

Приложение содержит уязвимость настраиваемых разрешений, если:

- 1) оно определяет настраиваемое разрешение, используемое для защиты API компонента;
- 2) оно не реализует дополнительную динамическую проверку, чтобы гарантировать, что вызывающее приложение имеет право доступа к API.

Для проверки этих двух условий было сделано следующее:

- декомпиляция анализируемого файла пакета Android;
- извлечение из этого файла его манифеста;

- связывание файла манифеста с байт-кодом соответствующего приложения для выполнения проверок:

А. Разрешение: проверяется файл манифеста на предмет любого объявления настраиваемых разрешений и того, действительно ли эти разрешения используются для защиты компонентов.

Б. Динамическое принуждение. Существует программный, но ограниченный метод защиты приложения от атаки с настраиваемым разрешением. Если ему известен белый список доверенных вызывающих приложений, он может реализовать динамическую проверку для отклонения вызовов от неизвестных приложений. Однако может оказаться невозможным создать такой список для открытого приложения, которое предназначено для взаимодействия со многими приложениями. Далее анализируется байт-код на наличие этой дополнительной проверки, выполняя поиск встроенных функций Android, например, таких как `getCallingUid`, которая возвращает информацию о вызывающем абоненте.

Согласно нашим экспериментальным данным около 61% компонентов Android, защищенных настраиваемыми разрешениями, относятся к типу `Service` или `Broadcast Receiver`. Это важно, потому что отсутствие видимого пользовательского интерфейса в этих типах компонентов способствует возможности скрытой атаки повторного делегирования разрешений [1, 11, 13]. Более 85% настраиваемых разрешений определены на уровнях сигнатур или опасных уровней защиты, которые регулируют доступ к критически важным API.

Результаты экспериментов показывают, что настраиваемые разрешения широко используются в реальных приложениях Android для защиты критически важных API. Большинство разработчиков не выполняют никаких дополнительных проверок, чтобы убедиться, что входящие API-интерфейсы поступают от доверенных приложений или поставщиков, предполагая, что они могут не знать об уязвимости настраиваемых разрешений, несмотря на ее потенциальную возможность нарушения безопасности [14, 15, 16].

6. Заключение

В статье была представлена формальная модель протокола разрешений Android и выполнен автоматизированный анализ, который выявил ряд недостатков в протоколе, вызывающих серьезные уязвимости безопасности. Проведено исследование одной из уязвимостей и показано, что она широко распространена среди многих существующих приложений Android.

Недостаточная спецификация протокола разреше-

ний Android была существенной; это позволило избежать определения аспектов поведения, которые не были ясны в документации Android, и привело к обнаружению уязвимостей, которые не учитывали другие исследователи протокола разрешений Android.

Построив точную модель механизма разрешений и подвергнув ее исчерпывающему анализу, разработчик устройства может обнаружить потенциальные уязвимости вместо того, чтобы полагаться исключительно на ручную проверку экспертов по безопасности.

Литература

1. Egelman, S., Felt, A. P., & Wagner, D. Choice architecture and smartphone privacy: There's a price for that // The economics of information security and privacy. 2013. pp. 211-236. DOI:10.1007/978-3-642-39498-0_10.
2. Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M., & Rajarajan, M. Android security: A survey of issues, malware penetration, and defenses // IEEE Communications Surveys and Tutorials. 2015. 17(2). pp. 998-1022. DOI:10.1109/COMST.2014.2386139.
3. Gao, J., Li, L., Kong, P., Bissyande, T. F., & Klein, J. Understanding the evolution of android app vulnerabilities // IEEE Transactions on Reliability. 2019. pp. 1-19. DOI:10.1109/TR.2019.2956690.
4. Demissie, B. F., Ceccato, M., & Shar, L. K. Security analysis of permission re-delegation vulnerabilities in android apps // Empirical Software Engineering. 2020. 25(6). pp. 5084-5136. DOI:10.1007/s10664-020-09879-8.
5. Thiagarajan, J., Akash, A., & Murugan, B. Improved real-time permission based malware detection and clustering approach using model independent pruning. // IET Information Security. 2020. 14(5). pp. 531-541. DOI:10.1049/iet-ifs.2019.0418.
6. Scalas, M., Maiorca, D., Mercaldo, F., Visaggio, C. A., Martinelli, F., & Giacinto, G. On the effectiveness of system API-related information for android ransomware detection // Computers and Security. 2019. 86. pp. 168-182. DOI:10.1016/j.cose.2019.06.004.
7. Alazab, M., Alazab, M., Shalaginov, A., Mesleh, A., & Awajan, A. Intelligent mobile malware detection using permission requests and API calls // Future Generation Computer Systems. 2020. 107. pp. 509-521. DOI:10.1016/j.future.2020.02.002.
8. Moore, S. R., Ge, H., Li, N., & Proctor, R. W. Cybersecurity for android applications: Permissions in android 5 and 6 // International Journal of Human-Computer Interaction. 2019. 35(7). pp. 630-640. DOI:10.1080/10447318.2018.1489580.
9. Yang, X., Lo, D., Li, L., Xia, X., Bissyandé, T. F., & Klein, J. Characterizing malicious android apps by mining topic-specific data flow signatures // Information and Software Technology. 2017. 90. pp. 27-39. DOI:10.1016/j.infsof.2017.04.007.
10. Xiao, J., Chen, S., He, Q., Feng, Z., & Xue, X. An android application risk evaluation framework based on minimum permission set identification // Journal of Systems and Software. 2020. 163. pp. 1-43. DOI:10.1016/j.jss.2020.110533.
11. Doğru, I. A., & Önder, M. AppPerm analyzer: Malware detection system based on android permissions and permission groups // International Journal of Software Engineering and Knowledge Engineering. 2020. 30(3). pp. 427-450. DOI:10.1142/S0218194020500175.
12. De Lorenzo, A., Martinelli, F., Medvet, E., Mercaldo, F., & Santone, A. Visualizing the outcome of dynamic analysis of android malware with VizMal // Journal of Information Security and Applications. 2020. 50. DOI:10.1016/j.jisa.2019.102423.
13. Iadarola, G., Martinelli, F., Mercaldo, F., & Santone, A. Call graph and model checking for fine-grained android malicious behaviour detection // Applied Sciences (Switzerland). 2020. 10(22). pp. 1-20. DOI:10.3390/app10227975.
14. Sharmeen, S., Huda, S., Abawajy, J., & Hassan, M. M. An adaptive framework against android privilege escalation threats using deep learning and semi-supervised approaches // Applied Soft Computing Journal. 2020. 89. pp. 1-20. DOI:10.1016/j.asoc.2020.106089.
15. Nguyen-Vu, L., Ahn, J., & Jung, S. Android fragmentation in malware detection // Computers and Security. 2019. 87. pp. 1-10. DOI:10.1016/j.cose.2019.101573.
16. Bagheri, H., Sadeghi, A., Garcia, J., & Malek, S. COVERT: Compositional analysis of android inter-app permission leakage. // IEEE Transactions on Software Engineering. 2015. 41(9). pp. 866-886. DOI:10.1109/TSE.2015.2419611.

FLAWS IN THE ANDROID PERMISSION PROTOCOL WITH LIMITED VERIFICATION

Erokhin V.V.²

Purpose of the article: analysis of the resolution protocol implemented in the Android operating system as the most popular for smartphones and other electronic gadgets; consider a formal model of the Android permission protocol and describe the automatic security analysis of this model; identify potential flaws in the permitting protocol.

Research method: A formal model of the Android permission protocol based on C++ using the Java NDK based on first-order relational logic is considered, with an analysis engine that performs limited model validation.

Result. Created a formal model of Android permission protocol using C++ using Java NDK. The model identified flaws in the Android permission protocol, and thus exposed Android security vulnerabilities. The developed Android protocol permission model consists of three parts: an Android device architecture query; Android permission scheme request; system operations. Fixed flaws in Android OS related to custom permissions vulnerability. An experiment is presented to demonstrate the feasibility and prevalence of custom permissions vulnerability in existing Android applications. Examination of real Android applications supports our finding that flaws in the Android permission protocol can have serious security implications for electronic gadget applications, and in some cases allows an attacker to completely bypass permission checks. A study of one of the vulnerabilities showed that it is widespread among many existing Android applications. Most developers do not perform any additional validation to ensure that inbound APIs come from trusted applications or vendors, assuming they may not be aware of a custom permissions vulnerability despite its potential for security breaches. The result will be useful for software developers for operating systems with permissions - Android, iOS and Fire OS.

Keywords: malware, mobile security, API level, mobile applications, programming, dynamic analysis, source code, modeling, operating system.

References

1. Egelman, S., Felt, A. P., & Wagner, D. Choice architecture and smartphone privacy: There's a price for that // The economics of information security and privacy. 2013. pp. 211-236. DOI:10.1007/978-3-642-39498-0_10.
2. Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M., & Rajarajan, M. Android security: A survey of issues, malware penetration, and defenses // IEEE Communications Surveys and Tutorials. 2015. 17(2). pp. 998-1022. DOI:10.1109/COMST.2014.2386139.
3. Gao, J., Li, L., Kong, P., Bissyande, T. F., & Klein, J. Understanding the evolution of android app vulnerabilities // IEEE Transactions on Reliability. 2019. pp. 1-19. DOI:10.1109/TR.2019.2956690.
4. Demissie, B. F., Ceccato, M., & Shar, L. K. Security analysis of permission re-delegation vulnerabilities in android apps // Empirical Software Engineering. 2020. 25(6). pp. 5084-5136. DOI:10.1007/s10664-020-09879-8.
5. Thiyagarajan, J., Akash, A., & Murugan, B. Improved real-time permission based malware detection and clustering approach using model independent pruning. // IET Information Security. 2020. 14(5). pp. 531-541. DOI:10.1049/iet-ifs.2019.0418.
6. Scalas, M., Maiorca, D., Mercaldo, F., Visaggio, C. A., Martinelli, F., & Giacinto, G. On the effectiveness of system API-related information for android ransomware detection // Computers and Security. 2019. 86. pp. 168-182. DOI:10.1016/j.cose.2019.06.004.
7. Alazab, M., Alazab, M., Shalaginov, A., Mesleh, A., & Awajan, A. Intelligent mobile malware detection using permission requests and API calls // Future Generation Computer Systems. 2020. 107. pp. 509-521. DOI:10.1016/j.future.2020.02.002.
8. Moore, S. R., Ge, H., Li, N., & Proctor, R. W. Cybersecurity for android applications: Permissions in android 5 and 6 // International Journal of Human-Computer Interaction. 2019. 35(7). pp. 630-640. DOI:10.1080/10447318.2018.1489580.
9. Yang, X., Lo, D., Li, L., Xia, X., Bissyandé, T. F., & Klein, J. Characterizing malicious android apps by mining topic-specific data flow signatures // Information and Software Technology. 2017. 90. pp. 27-39. DOI:10.1016/j.infsof.2017.04.007.
10. Xiao, J., Chen, S., He, Q., Feng, Z., & Xue, X. An android application risk evaluation framework based on minimum permission set identification // Journal of Systems and Software. 2020. 163. pp. 1-43. DOI:10.1016/j.jss.2020.110533.

2 Viktor Erokhin, Dr.Sc. (in Tech), Associate Professor, Professor at the Department of Mathematical Methods and Business Informatics, Moscow State Institute of International Relations (University), , Moscow, Russia. E-mail: erohinv@mail.ru, ORCID 0000-0002-8754-0012

11. Doğru, I. A., & Önder, M. AppPerm analyzer: Malware detection system based on android permissions and permission groups // International Journal of Software Engineering and Knowledge Engineering. 2020. 30(3). pp. 427-450. DOI:10.1142/S0218194020500175.
12. De Lorenzo, A., Martinelli, F., Medvet, E., Mercaldo, F., & Santone, A. Visualizing the outcome of dynamic analysis of android malware with VizMal // Journal of Information Security and Applications. 2020. 50. DOI:10.1016/j.jisa.2019.102423.
13. Iadarola, G., Martinelli, F., Mercaldo, F., & Santone, A. Call graph and model checking for fine-grained android malicious behaviour detection // Applied Sciences (Switzerland). 2020. 10(22). pp. 1-20. DOI:10.3390/app10227975.
14. Sharmeen, S., Huda, S., Abawajy, J., & Hassan, M. M. An adaptive framework against android privilege escalation threats using deep learning and semi-supervised approaches // Applied Soft Computing Journal. 2020. 89. pp. 1-20. DOI:10.1016/j.asoc.2020.106089.
15. Nguyen-Vu, L., Ahn, J., & Jung, S. Android fragmentation in malware detection // Computers and Security. 2019. 87. pp. 1-10. DOI:10.1016/j.cose.2019.101573.
16. Bagheri, H., Sadeghi, A., Garcia, J., & Malek, S. COVERT: Compositional analysis of android inter-app permission leakage. // IEEE Transactions on Software Engineering. 2015. 41(9). pp. 866-886. DOI:10.1109/TSE.2015.2419611.