Министерство образования и науки Российской Федерации

Санкт-Петербургский Политехнический Университет Петра Великого

Институт прикладной математики и механики

Кафедра «Информационная безопасность компьютерных систем»

**ОТЧЕТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ**

|  |  |
| --- | --- |
| Тема: | «Поиск уязвимостей в Android приложениях» |
| Специальность: | 10.05.03 «Информационная безопасность автоматизированных систем» |

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.Ю. Павленко

Выполнил

Студент гр. 33609/3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А.Корольков

Санкт-Петербург

2018

СОДЕРЖАНИЕ

Введение……………………………………………………………………………………………4

1. Разновидности уязвимостей в системе Android………………………………………………6

1.1. Уязвимости ядра Linux и его модулей………………………………………………..6

1.2. Уязвимости модификаций и компонентов производителей устройств…………….6

1.3. Уязвимости модулей в машинных кодах……………………………………………..6

1.4. Уязвимости механизмов межкомпонентного взаимодействия…………………...…7

1.5. Уязвимости в самих приложениях………………………………………………….....7

1.6. Уязвимости во встроенных сервисах и библиотеках ……………………………..…8

1.7. Интернет-источники……………………………………………………………………8

1.8. Уязвимости аппаратуры и связанных с ней модулей и протоколов……………...…8

2. Устройство Android приложения………………………………………………………………9

2.1. Архитектурные уровни Android……………………………………………………….9

2.2. Манифест Android приложения………………………………………………………11

2.2.1. Объявление компонентов………………………………...…………………...12

2.2.2. Объявление возможностей компонентов……………………………...…….12

2.2.3. Объявление требований приложения………………………………….…….14

3. Способы поиска уязвимостей в приложениях Android………………………………….…..14

3.1 Статический анализ………………………………………………………………….…14

3.2. Динамические и смешанные инструменты анализа…………………………………15

4. Изучения основных популярных уязвимостей Android……………………………………..17

4.1. XSS в приложении…………………………………………………………….…….…17

4.2. UXSS и file://...................................................................................................................17

4.3. SQLITE-INJ в приложении………………………………………………………....…18

4.4. SPOOFING……………………………………………………………………………..18

4.5. Совместное использование Java и C/C++. Уязвимости в нативном коде……….....19

4.6. Уязвимости межпроцессного взаимодействия с намерениями (Intent)…………....19

4.7. Уязвимости сериализации в Java………………………………………………….….21

4.8. Уязвимости приложений, связанные с возможностью фаззинга…………………...23

5. Исследование приложения на наличие уязвимостей………………………………………..24

5.1. Обнаружения и эксплуатации уязвимости Intent Sniffing…………………...……...24

5.2. Обнаружение и эксплуатация экспортированных Android Activities………….…..26

5.3. Обнаружение программных закладок (Backdoors)………………………………….27

5.4. Обнаружение и эксплуатация экспортированных Android Content Providers……28

5.5. Вывод и обобщение по рассмотренным уязвимостях…………………………..….28

6. Концепт разрабатываемого инструмента для поиска уязвимостей………………………..31

Заключение……………………………………………………………………………………….32

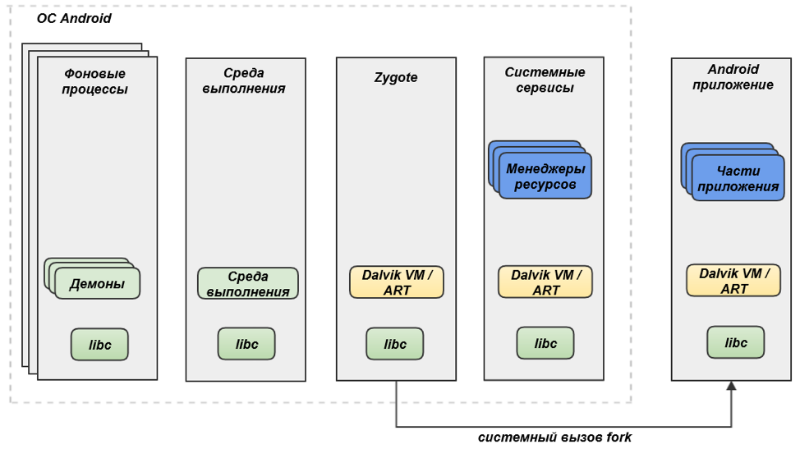
Список источников………………………………………………………………………………33

# ВВЕДЕНИЕ

Родительским процессом для всех приложений в ОС Android является процесс Zygote. Данный процесс представляет собой каркас Android-приложения, в котором уже загружены все необходимые библиотеки окружения Android, но отсутствует код самого приложения. Запуск приложения Android с точки зрения операционной системы происходит следующим образом:

1. Вначале происходит системный вызов fork для создания потомка от процесса Zygote (см. рис. 1).
2. В этом новом процессе открывается файл запускаемого приложения (системный вызов open).
3. Происходит чтение информации о файлах классов (classes.dex) и ресурсов из файла приложения. Происходит открытие сокетов для IPC.
4. Выполняется системный вызов mmap для отображения файлов приложения в память.
5. Среда выполнения производит настройку необходимого окружения и выполняет приложение (интерпретирует байт-код Dalvik или передает управление функциям в исполняемом коде в случае ART ).

На уровне ядра ОС Android каждое приложение является отдельным процессом с уникальными значениями user/group ID, которые даются ему при установке. Таким образом, каждая программа работает в своей песочнице. Начиная с версии 4.3 в дополнение к этой политике безопасности добавилось использование компонента мандатного контроля доступа SELinux.



*Рис. 1. Запуск нового приложения в ОС Android*

По умолчанию приложение ограничено в своих возможностях и не может сделать ничего, чтобы негативно повлиять на другое приложение или пользователя: ни прочитать пользовательские данные, ни модифицировать системные приложения; отсутствует доступ к сети. Для получения доступа к различным ресурсам приложение обращается к сервисам ОС Android. Разрешения на доступ задаются статически в файле манифеста приложения и выдаются приложению во время его работы по мере необходимости. Система запрашивает у пользователя согласие на выдачу этих разрешений во время установки или во время обновления приложения. Ответственность за выдачу доступа приложению лежит на пользователе, он самостоятельно решает, какому приложению давать разрешения на определенные действия, а какому не давать, — во время его установки. Использование такого подхода, при котором все разрешения выдаются разом при установке приложения, привело к тому, что пользователи стали раздавать полномочия приложениям, не задумываясь о последствиях. В свою очередь, приложения стали запрашивать все больше разрешений по мере расширения их функциональности. В Android 6 Marshmallow данная система заменена на другую: приложение запрашивает доступ к ресурсам у пользователя во время его работы, а пользователь может либо разрешить доступ, либо запретить его.

# 1. РАЗНОВИДНОСТИ УЯЗВИМОСТЕЙ В СИСТЕМЕ ANDROID

1.1. УЯЗВИМОСТИ ЯДРА LINUX И ЕГО МОДУЛЕЙ

К данной категории проблем относятся уязвимости, которые присущи всем ОС, основанным на той же версии ядра Linux, что и ОС Android. Эксплойты, использующие уязвимости в ядре, могут получить данные пользователя или права администратора системы. Повысив привилегии процесса до прав администратора системы, вредоносная программа может отключить систему безопасности Android, вставить в существующие программы вредоносный код и установить руткит. К тому же производители различных устройств добавляют в ядро модули для своих устройств; в этих модулях также могут быть уязвимости.

1.2. УЯЗВИМОСТИ МОДИФИКАЦИЙ И КОМПОНЕНТОВ ПРОИЗВОДИТЕЛЕЙ УСТРОЙСТВ

Производители различных мобильных устройств выпускают свои модифицированные прошивки Android. Эти прошивки могут содержать различные приложения и сервисы, разработанные производителем устройства, которые чаще всего нельзя удалить. Анализ этих сервисов, приведенный в статьях [[2]](https://habr.com/company/pt/blog/332904/#17), показывает, что в них содержится от 65 до 85% уязвимостей, обнаруженных во всей системе. К тому же стоит отметить, что уязвимости, которые были обнаружены и исправлены в основной ветке Android, могут долгое время оставаться в ветках производителей устройств .

1.3. УЯЗВИМОСТИ МОДУЛЕЙ В МАШИННЫХ КОДАХ

Android-приложения поддерживают запуск машинного кода через интерфейс JNI. Это порождает еще одну категорию уязвимостей, связанную с широко известными уязвимостями утечек памяти в низкоуровневых языках (например, в С и С++ ) [3]. Поскольку на уровне процессов ОС Android нет никаких ограничений, кроме накладываемых ядром Linux, сторонние библиотеки в машинных кодах могут использовать разрешения, выданные всему приложению, для совершения вредоносной активности. Также модули приложения в машинных кодах используются авторами вредоносных приложений, чтобы обойти инструменты анализа и мониторинга уровня Android. Эти модули также могут использовать техники противодействия анализу, используемые в обычных программах.

1.4. УЯЗВИМОСТИ МЕХАНИЗМОВ МЕЖКОМПОНЕНТНОГО ВЗАИМОДЕЙСТВИЯ

К данной категории относятся уязвимости, связанные с взаимодействием между различными компонентами приложений. Так как на уровне операционной системы приложение ограничено песочницей процесса, ему необходим механизм доступа к компонентам ОС Android для взаимодействия с ними. Это происходит через устройство /dev/Binder и различные другие сервисы ОС Android. Как уже говорилось выше, параметры этого доступа задаются в файле манифеста, в виде XML-файла с разрешениями. Анализ, приведенный в статьях [4,5], указывает на недочеты этой системы ограничений и показывает пути их обхода. Так, например, приложение может воспользоваться правами доступа другого приложения и получить с помощью него данные через ICC. Также могут быть уязвимости, связанные со сторонними библиотеками. Сторонние библиотеки, используемые в приложении, получают тот же набор ограничений, что и само приложение. Поэтому сторонние библиотеки могут использовать разрешения, выданные всему приложению, для совершения вредоносной активности. Приложения к тому же могут перехватывать системные события, пересылаемые через широковещательный запрос, и сохранять информацию о входящих звонках и СМС.

1.5. УЯЗВИМОСТИ В САМИХ ПРИЛОЖЕНИЯХ

Каждое приложение сохраняет какие-то данные о пользователе. Эти данные должны быть защищены должным образом, чтобы к ним не могли получить доступ другие приложения, — но такая защита предусмотрена не всегда. Например, Skype старых версий приложения сохранял базу данных контактов в открытом виде на диске. Таким образом, контакты можно было прочитать любым другим приложением, у которого есть доступ к диску [6]. Также приложения могут использовать криптографические библиотеки с ошибками [7] или же какие-то собственные реализации. К тому же не все приложения производят хорошую аутентификацию и авторизацию пользователя. Кроме этого, приложения могут позволять SQL-инъекции и подвержены атакам XSS. Также стоит отметить, что большинство разрабатываемых приложений написаны на Java без использования какой-либо защиты для бинарного кода, а байт-код Java, как известно, легко поддается дизассемблированию и анализу.

1.6. УЯЗВИМОСТИ ВО ВСТРОЕННЫХ СЕРВИСАХ И БИБЛИОТЕКАХ

Стандартный набор библиотек и сервисов, работающих в Android, также содержит уязвимости. Например, недавно была обнаружена уязвимость Stagefright в библиотеке для отображения видео в MMS-сообщениях, которой были подвержены все версии Android, начиная с 2.2 [8]. Позже была обнаружена уязвимость в компоненте MediaServer, которой подвержены все версии Android c 2.3 до 5.1 [9]. В статье [10] показаны уязвимости рантайма Dalvik: запустив большое количество процессов в системе, можно добиться, что последующий процесс запустится с правами администратора.

1.7. ИНТЕРНЕТ-ИСТОЧНИКИ

Android-приложения распространяются через широкое количество источников помимо официального магазина приложений. Поскольку Android-приложения написаны в основном на Java, то они легко поддаются обратной разработке и переупаковке с использованием вредоносного кода. Кроме того, песочницу анализа приложений Bouncer, используемую в официальном каталоге, легко обойти[11]. Поэтому и в самом официальном магазине содержится большое количество вредоносных программ. Кроме этого, Android поддерживает удаленную установку приложений через Google Play на устройства, связанные с Google-аккаунтом. Таким образом, если взломать учетную запись Google для устройства, можно установить из Google Play вредоносное приложение, которое туда предварительно загрузил злоумышленник. При этом на экране мобильного телефона не требуется каких-либо подтверждений этих действий, поскольку они запрашиваются в окне браузера и приложение устанавливается на телефон в фоновом режиме при получении доступа к Интернету. Также к этой категории уязвимостей относится использование социальной инженерии, когда для продолжения работы предлагают установить приложение из неавторизованного источника .

1.8. УЯЗВИМОСТИ АППАРАТУРЫ И СВЯЗАННЫХ С НЕЙ МОДУЛЕЙ И ПРОТОКОЛОВ

Мобильные устройства, работающие под управлением ОС Android, имеют широкий набор аппаратуры для взаимодействия с внешним миром. Соответствующие уязвимости можно эксплуатировать при непосредственной близости к устройству или при наличии физического доступа к устройству.

Примерами таких атак служат атака типа «отказ в обслуживании» на технологию Wi-Fi Direct , кража данных кредитных карт с помощью NFC, исполнение удаленного кода через Bluetooth , установка вредоносного приложения без ведома пользователя через adb с помощью механизма бэкапов , а так же уязвимости, с помощью которых можно повысить привилегии приложения, используя ошибки в реализации протокола adb.

# 2. УСТРОЙСТВО ANDROID ПРИЛОЖЕНИЯ

Перед тем как мы начнем понимать структуру систем защиты Android, мы должны рассмотреть работу программ на устройстве. Приложение работает в изолированной среде на устройстве. Каждое приложение имеет свои собственные разрешения и область рабочей памяти. Различные приложения могут взаимодействовать между собой. Приложение может запросить систему для выделения ресурсов только через предписания.

Практически все Android-приложения запускаются в своих собственных Linux-процессах. Процесс создается тогда, когда программе нужно выполнить какой-то код, и дальше уже остается запущенным до тех пор, пока приложение используется. После завершения процесса система чистит свою память для ее эффективного использования другими программами.

Все приложения для Android скомпилированы в файл Android Package (APK). APK и устанавливается на устройство.

Формат APK является вариантом формата JAR.

Файл APK обычно содержит две папки:

* META-INF
* res

со следующими файлами:

* AndroidManifest.xml
* classes.dex
* resources.arsc

AndroidManifest.xml и classes.dex - два основных файла, которые будем рассматривать.

# 2.1. АРХИТЕКТУРНЫЕ УРОВНИ ANDROID

Операционная система Android имеет три весьма различных и сильно отделённых друг от друга уровня:

1. В основе лежит модифицированная и урезанная версия Linux.
2. Над уровнем Linux находится уровень инфраструктуры приложения, содержащий [виртуальную машину Dalvik](http://en.wikipedia.org/wiki/Dalvik_virtual_machine), веб-браузер, базу данных SQLite, некие инфраструктурные решения и Java API.
3. И, наконец, уровень написанных в Google Android-приложений. Вообще говоря, они являются расширением уровня инфраструктуры, поскольку разработчик может использовать эти приложения или их части как строительные блоки для собственных разработок.

Для разработки приложений под Android предоставляется четыре основных строительных блока, которые используются в различных сочетаниях.

Четыре типа компонентов:

Операции

Операция (Activity) представляет собой один экран с пользовательским интерфейсом. Например, в приложении для работы с электронной почтой одна операция может служить для отображения списка новых сообщений, другая – для составления сообщения и третья операция – для чтения сообщений. Несмотря на то что операции совместно формируют связное взаимодействие пользователя с приложением по работе с электронной почтой, каждая из них не зависит от других операций. Любые из этих операций могут быть запущены другим приложением (если это позволяет приложение по работе с электронной почтой). Например, приложение для камеры может запустить операцию в приложении по работе с электронной почтой, которая составляет новое сообщение, чтобы пользователь мог отослать фотографию.

Службы

Служба (Service) представляет собой компонент, который работает в фоновом режиме и выполняет длительные операции, связанные с работой удаленных процессов. Служба не имеет пользовательского интерфейса. Например, она может воспроизводить музыку в фоновом режиме, пока пользователь работает в другом приложении, или же она может получать данные по сети, не блокируя взаимодействие пользователя с операцией. Служба может быть запущена другим компонентом, который затем будут взаимодействовать с ней, – например операцией.

Поставщики контента

Поставщик контента (Content provider*)* управляет общим набором данных приложения. Данные можно хранить в файловой системе, базе данных SQLite, в Интернете или любом другом постоянном месте хранения, к которому у вашего приложения имеется доступ. Посредством поставщика контента другие приложения могут запрашивать или даже изменять данные (если поставщик контента позволяет делать это). Например, в системе Android есть поставщик контента, который управляет информацией контактов пользователя. Любое приложение, получившее соответствующие разрешения, может запросить часть этого поставщика контента, для чтения и записи сведений об определенном человеке.

Поставщики контента также используются для чтения и записи данных, доступ к которым внешним компонентам приложение не предоставляет. Например, в образце приложения Note Pad с помощью поставщика контента выполняется сохранение заметок.

Поставщик контента относится к подклассу класса ContentProvider. Он должен реализовывать стандартный набор API-интерфейсов, с помощью которых другие приложения будут выполнять транзакции.

Приемники широковещательных сообщений

Приемник широковещательных сообщений (Broadcast receiver) представляет собой компонент, который реагирует на объявления, распространяемые по всей системе. Многие из этих объявлений рассылает система — например объявление о том, что экран выключился, аккумулятор разряжен или был сделан фотоснимок. Объявления также могут рассылаться приложениями, — например, чтобы сообщить другим приложениям о том, что какие-то данные были загружены на устройство и теперь готовы для использования. Несмотря на то, что приемники широковещательных сообщений не имеют пользовательского интерфейса, они могут создавать уведомления в строке состояния, чтобы предупредить пользователя о событии "рассылка объявления". Однако чаще всего они являются просто "шлюзом" для других компонентов и предназначены для выполнения минимального объема работы. Например, они могут инициировать выполнение службой определенных действий при возникновении события.

Приемник широковещательных сообщений относится к подклассу класса BroadcastReceiver , а каждое такое сообщение предоставляется как объект Intent. [1]

2.2. МАНИФЕСТ ANDROID ПРИЛОЖЕНИЯ

Для запуска компонента приложения системе Android необходимо знать, что компонент существует. Для этого она читает файл AndroidManifest.xml приложения (файл манифеста). В этом файле, который должен находиться в корневой папке приложения, должны быть объявлены все компоненты приложения.

Помимо объявления компонентов приложения, манифест служит и для других целей, среди которых:

* указание всех полномочий пользователя, которые требуются приложению, например разрешения на доступ в Интернет или на чтение контактов пользователя;
* объявление минимального уровня API, требуемого приложению, с учетом того, какие API-интерфейсы оно использует;
* объявление аппаратных и программных функций, которые нужны приложению или используются им, например камеры, службы Bluetooth или сенсорного экрана;
* указание библиотек API, с которыми необходимо связать приложение (отличные от API-интерфейсов платформы Android), например библиотеки Google Maps;
* и многое другое.

2.2.1. ОБЪЯВЛЕНИЕ КОМПОНЕНТОВ

Основная задача манифеста – это информировать систему о компонентах приложения. Например, файл манифеста может объявлять операцию следующим образом:

<?xml version="1.0" encoding="utf-8"?>

<manifest ... >

    <application android:icon="@drawable/app\_icon.png" ... >

        <activity android:name="com.example.project.ExampleActivity"

                  android:label="@string/example\_label" ... >

        </activity>

        ...

    </application>

</manifest>

Все компоненты приложения необходимо объявлять следующим образом:

* элементы <activity> для операций;
* элементы <service> для служб;
* элементы <receiver> для приемников широковещательных сообщений;
* элементы <provider> для поставщиков контента

Системе не видны операции, службы и поставщики контента, которые имеются в исходном коде, но не объявлены в манифесте, поэтому они не могут быть запущены. А вот приемники широковещательных сообщений можно либо объявить в манифесте, либо создать динамически в коде (как объекты BroadcastReceiver) и зарегистрировать в системе путем вызова registerReceiver().

2.2.2. ОБЪЯВЛЕНИЕ ВОЗМОЖНОСТЕЙ КОМПОНЕНТОВ

С помощью объекта Intent можно запускать операции, службы и приемники широковещательных сообщений. Для этого в объекте Intent следует явно указать имя целевого компонента (с помощью имени класса компонента). Однако в полной мере возможности объектов Intent раскрываются при использовании концепции неявных Intent. В неявном сообщении Intent просто описывается тип действия, которое требуется выполнить (а также, хотя это и не обязательно, дата, в которую вы бы хотели выполнить это действие). Системе же предоставляется возможности найти на устройстве компонент, который может выполнить это действие, и запустить его. При наличии нескольких компонентов, которые могут выполнить действие, описанное в сообщении Intent, пользователь выбирает, какой из них будет использоваться.

Система определяет компоненты, которые могут ответить на сообщение Intent, путем сравнения полученного сообщения Intent с фильтрами объектов Intent*,* указанными в файле манифеста других приложений, имеющихся на устройстве.

При объявлении операции в манифесте своего приложения по желанию можно указать фильтры объектов Intent, которые указывают возможности операции, с тем чтобы она могла реагировать на сообщения Intent от других приложений. Чтобы объявить фильтр Intent для своего компонента, необходимо добавить элемент &lt;intent-filter&gt; в качестве дочернего для элемента объявления компонента.

Например, если вы создали приложение для работы с электронной почтой с операцией составления нового сообщения, вы можете объявить фильтр для ответа на сообщения Intent типа "send" (для отправки нового сообщения электронной почты) следующим образом:

<manifest ... >

    ...

    <application ... >

        <activity android:name="com.example.project.ComposeEmailActivity">

            <intent-filter>

                <action android:name="android.intent.action.SEND" />

                <data android:type="\*/\*" />

                <category android:name="android.intent.category.DEFAULT" />

            </intent-filter>

        </activity>

    </application>

</manifest>

Затем, если другое приложение создаст объект Intent с действием ACTION\_SEND и передаст его в startActivity(), система сможет запустить вашу операцию, дав пользователю возможность написать и отправить сообщение электронной почты.

Подробные сведения о создании фильтров объектов Intent приведены в документе Объекты Intent и фильтры объектов Intent.

2.2.3. ОБЪЯВЛЕНИЕ ТРЕБОВАНИЙ ПРИЛОЖЕНИЯ

Существует огромное количество устройств, работающих под управлением Android, и не все они имеют одинаковые функциональные возможности. Чтобы ваше приложение не могло быть установлено на устройствах, в которых отсутствуют функции, необходимые приложению, важно четко определить профиль для типов устройств, поддерживаемых вашим приложением, указав требования к аппаратному и программному обеспечению в файле манифеста. Эти объявления по большей части носят информационный характер, система их не читает. Однако их читают внешние службы, например Google Play, с целью обеспечения фильтрации для пользователей, которые ищут приложения для своих устройств.

Например, если вашему приложению требуется камера и оно использует API-интерфейсы из Android 2.1 (уровень API 7), эти параметры следует объявить в файле манифеста в качестве требований следующим образом:

<manifest ... >

    <uses-feature android:name="android.hardware.camera.any"

                  android:required="true" />

    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />

    ...

</manifest>

Теперь ваше приложение нельзя будет установить из Google Play на устройствах, в которых нет камеры, а также на устройствах, работающих под управлением Android версии ниже 2.1.

Однако можно также объявить, что приложение использует камеру, но для его работы она не является непременно необходимой. В этом случае в приложении атрибуту required необходимо задать значение "false", а во время работы оно должно проверять, имеется ли на устройстве камера, и при необходимости отключать свои функции, которые используют камеру.

# 3. СПОСОБЫ ПОИСКА УЯЗВИМОСТЕЙ В ПРИЛОЖЕНИЯХ ANDROID

3.1 СТАТИЧЕСКИЙ АНАЛИЗ

Инструменты статического анализа можно разделить на следующие категории:

* Инструменты, извлекающие метаинформацию из манифеста приложения и предоставляющие информацию о запрашиваемых разрешениях, компонентах Activity, сервисах и зарегистрированных Broadcast receivers. Метаинформация часто используется позже в динамическом анализе для запуска функций приложения.
* Инструменты для модификации существующего байт-кода приложения с использованием техники инструментирования. Это позволяет, например, добавлять трассирование функциональности в существующие приложения.
* Инструменты, которые реализуют декомпилятор или дизассемблер байт-кода Dalvik.

Одним из наиболее популярных инструментов обратной разработки является Apktool . Он имеет возможности декодирования ресурсов приложения приблизительно в оригинальную форму, переупаковки приложений обратно в APK/JAR-файлы, отладку байт-кода smali. Для декомпилирования и компилирования в apktool байт-кода Dalvik используется другой широко известный проект smali/backsmali . Также для дизассемблирования байт-кода Dalvik часто применяется инструмент Dedexer.

Radare2 — инструмент с открытым исходным кодом для дизассемблирования, анализа, отладки и изменения бинарных файлов Android-приложения.

Один из самых разносторонних инструментов для статического анализа — Androguard . Он может дизассемблировать и декомпилировать приложение обратно в исходный код Java. Получив два APK-файла, он может посчитать коэффициент их похожести для детектирования переупакованных приложений или известных вредоносных приложений. Благодаря своей гибкости он используется в некоторых инструментах смешанного анализа.

Следует отметить, что статический анализ имеет ряд существенных ограничений, связанных с тем, что имеется лишь абстрактное представление о программе. Например, статический анализ становится намного сложнее, если к программе применены обфусцирующие преобразования. В зависимости от сложности обфускации некоторые (а может, и все) статические подходы могут стать абсолютно бесполезными. Если вызов каждого метода делается только косвенно, вряд ли удастся построить граф вызовов программы. В Android такое преобразование можно сделать, используя Java Reflection API для вызова методов и создания объектов вместо использования обычных вызовов и инструкций создания нового объекта. На рынке решений по защите исходного кода уже представлено несколько продуктов для обфускации файлов Android-приложений, которые могут свести на нет все преимущества статического анализа [12].

3.2. ДИНАМИЧЕСКИЕ И СМЕШАННЫЕ ИНСТРУМЕНТЫ АНАЛИЗА

Инструменты динамического анализа отслеживают поведение неизвестного приложения во время выполнения при запуске его в контролируемой песочнице для получения поведенческого следа. Для этого производится инструментирование песочницы на различных уровнях архитектуры (рис. 2) участками кода, которые отслеживают поведение приложения и ОС Android.



*Рис. 2. Уровни архитектуры песочницы Android*

Архитектура песочницы Android представляет из себя эмулятор Android (чаще всего QEMU), внутри которого работает ОС Android. Инструментирование производится либо на уровне эмулятора, либо на уровне ОС Android, либо и там, и там. Уровень ОС Android также делится на четыре подуровня:

* приложения,
* среда разработки приложения,
* рабочее окружение приложения и библиотеки,
* ядро и его модули.

Техники, используемые в динамическом анализе приложения:

* Отслеживание помеченных данных. Такие инструменты используются, чтобы отслеживать потенциальные утечки конфиденциальной информации.
* Мониторинг системных вызовов. Инструменты могут записывать системные вызовы с помощью инструментирования виртуальных машин, strace или модуля ядра. Это позволяет производить трассировку машинного кода.
* Трассировка методов (функций). Инструменты могут отслеживать вызовы Java-методов приложения в виртуальной машине Dalvik, вызовы функций в машинных кодах через JNI, системные вызовы и прерывания.

К инструментам смешанного анализа относятся инструменты, которые сочетают в себе техники динамического и статического анализа.

# 4. ИЗУЧЕНИЯ ОСНОВНЫХ ПОПУЛЯРНЫХ УЯЗВИМОСТЕЙ ANDROID

4.1. XSS В ПРИЛОЖЕНИИ

Межсайтовый скриптинг, или Cross site scripting, или XSS, предполагает наличие сайта, подключающего непредусмотренный код Javascript, который, в свою очередь, передается пользователям, исполняющим этот код в своих браузерах.

Например, если в приложении с помощью компонента WebView() включен javascript, то мы можем выполнить вредоносный код, который позволит получить со смарт-фона приватные данные. В качестве примера можно рассмотреть старую уязвимость в приложении Gmail под Android. Суть уязвимости заключалась в следующем. Если в приложении послать письмо с адреса "onload=window.location='http:// google.com'"0somedmn.com, то можно было увидеть страницу Google в приложении Gmail. На основе этого нетрудно написать JS-код, который будет получать e-mail адреса и посылать их нам на заготовленный сниффер.

Таким образом при нахождении XSS в приложении существует большой шанс получить данные. Однако не стоит забывать, что для этого должен использоваться компонент WebView и setjavascriptEnabled().[13]

4.2. UXSS И FILE://

Существует отдельный вид уязвимостей — UXSS (Universal Cross Site Scripting). Этот тип уязвимостей специфичен для браузеров, так как позволяет выполнять javascript-код на любой своей открытой странице. А в качестве примера рассмотрим уязвимость браузера Chrome. Суть данной уязвимости проста — использование javascript: в URL текущей вкладки.

Рассмотрим PoC: Запускаем shell на Android-устройстве

shelliSandroid:/ $ am start -n

com.android.chrome/com.google.android.

apps.chrome.SimpleChromeActivity -d

'http://www.google.ru'

И у нас запускается Chrome с google.ru в текущей вкладке. А теперь используем javascript внутри URL:

shelliSandroid:/ $ am start -n

com.android.chrome/com.google.android.

apps.chrome.SimpleChromeActivity -d

'javascript:alert(document.cookie)'

И мы увидим куки (document.cookie) сайта на текущей вкладке (в данном случае google. ru).

Так же рассмотрим другую уязвимость: Как известно, файлы приложения доступны только самому приложению. Но разработчики Chrome допустили использование протокола file://, и благодаря этому мы можем получить файлы приложения и загрузить их на карточку с правами, позволяющими любому приложению открыть их. PoC выглядит следующим образом:

shelliSandroid:/ $ am start -n

com.android.chrome/com.android.chrome.

Main -d 'file:///data/data/com.android.

chrome/app chrome/Default/Cookies'

И в /sdcard/Downloads/Cookies.bin мы увидим наши cookies файлы». Теперь любое приложение может прочитать такой файл и отправить их злоумышленнику, чтобы он смог авторизоваться на каком-нибудь веб-сервисе, используя полученные личные данные.

4.3. SQLITE-INJ В ПРИЛОЖЕНИИ

Обычно для хранения данных приложение использует или SQLite-таблицы, или XML-файлы. Чтобы найти файлы определенного приложения, нужно обратиться по адресу /data/ data/%app\_name%/.

Рассмотрим подробнее атаку на приложение с использованием SQLite injection. Найти уязвимое приложение довольно просто, так же как и с обычными SQL inj, только приложение при подстановке кавычки в уязвимое поле завершится с ошибкой, а в консоли можно увидеть подробный отчет об SQL-ошибке. Изучая исходный код приложения, не стоит забывать об опасных участках при работе с базой данных, например подстановке переменных напрямую в запрос.

4.4. SPOOFING

Рассмотрим тип уязвимостей ,направленный на спуфинг контента. Если перенаправить весь трафик Android-системы, запущенной под эмулятором, на прокси, то можно изменять пакеты, которые отправляются устройством и приходят на него.В реальных условиях спуфинг можно провести с помощью специализированных утилит вроде Ettercap. Предположим, что у нас есть приложение, которое получает данные с внешнего источника и записывает их в базу. Разработчики часто забывают об этом и могут допустить ошибку, составив запрос с уязвимостью. Таким образом, подменяя ответ сервера на нужный нам SQL inj, мы можем проводить различные операции на устройстве жертвы.

4.5. СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ JAVA И C/C++. УЯЗВИМОСТИ В НАТИВНОМ КОДЕ.

Интеграция Java-систем с обслуживающими библиотеками, написанными на C/C++, является широко распространенной практикой. Java поддерживает загрузку библиотек DLL и библиотек программного кода. Экспортированные из библиотек функции затем непосредственно могут использоваться из Java. При подобной интеграции возникает реальная вероятность того, что переполнения буфера и другие недостатки в поддерживающих библиотеках будут использованы для проведения атак. Рассмотрим Java-программу, которая поддерживает интерфейс для работы с низкоуровневыми пакетами (raw packet). Такая Java-программа может, например, проводить анализ пакетов и создавать низкоуровневые пакеты. Такие действия вполне возможны после загрузки библиотеки пакетов из Java-программы.

public class MyJavaPacketEngine extends Thread {

public MyJavaPacketEngine () {

static {System.loadLibrary("packet\_driver32");}}

Представленный выше Java-класс загружает библиотеку DLL под названием packet\_driver32.DLL. После этого вызовы могут осуществляться непосредственно к этой библиотеке. Предположим, что Java-программа позволяет задать адаптер для выполнения действий с пакетами. А теперь рассмотрим, что произойдет, если программный код внутри библиотеки DLL передает в буфер строку для привязки к адаптеру, не ограничивая при этом размер входных данных.

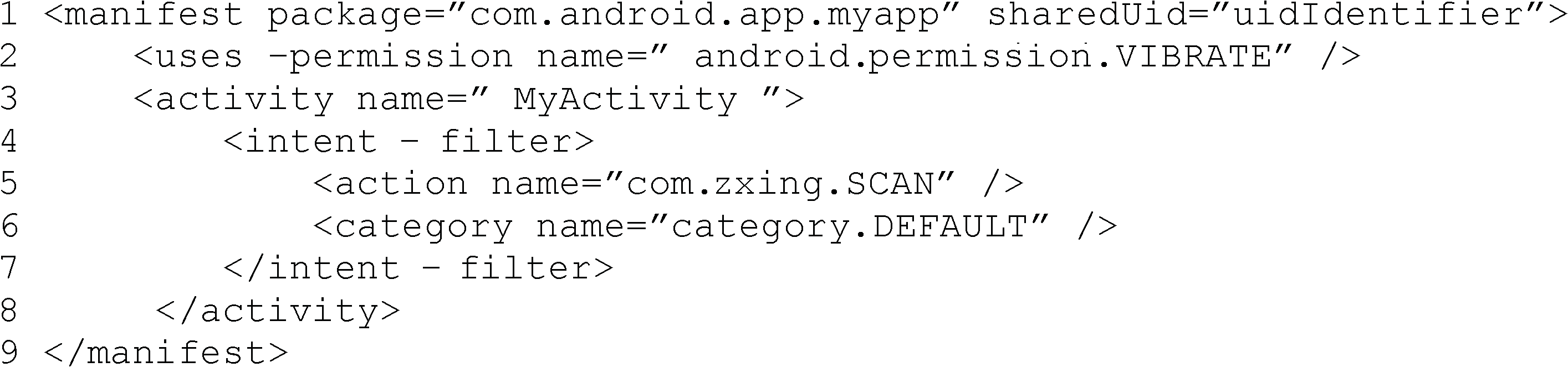
PVOID PacketOpenAdapter(LPTSTR p\_AdapterName)

{wsprintf(lpAdapter->SyimbolicLink,TE»OT«"\\\\.\\%s%s"J,DOSNAMEPREFIX,\*>p\_AdlaptteriNia«ee ) ;}

Здесь вполне вероятно может произойти переполнение буфера. Что может быть использовано для запуска вредоносного кода. Независимо от того, связано ли это каким-либо образом с Java или нет, но уязвимые места в ядре системы все равно остаются.

4.6. УЯЗВИМОСТИ МЕЖПРОЦЕССНОГО ВЗАИМОДЕЙСТВИЯ С НАМЕРЕНИЯМИ (INTENT)

Intent могут использоваться для связи между активностями и классами одного и того же приложения или для взаимодействия между приложениями. Во втором случае Intents - это фактически примитивы, передающие сообщения об обработке. Чтобы указать подмножество намерений, на которые отвечает действие, разработчики добавляют к манифесту приложения фильтр намерений, связанный с активностью. Фильтр намерений в листинге (рис. 3) указывает, что MyActivity можно вызвать, отправив Intent с действием com: zxing: SCAN; такое намерение называется неявным(Implicit) намерением, потому что оно не указывает конкретное действие, которое нужно вызвать. Неявные намерения создаются с использованием единственного конструктора параметров нового намерения (String).

**

*Рис. 3. Листинг*

Разработчики включают или отключают вызовы между приложениями своих классов активности, устанавливая значение логического экспортированного атрибута каждой активности в манифесте приложения. Поведение этого атрибута — это деталь, которая может быть источником путаницы, поскольку значение зависит от наличия другого элемента XML, фильтра Intent:

• Если объявлен фильтр намерения, а экспортированный атрибут явно не установлен в true или false, его значение по умолчанию - true, что делает активность доступной для любого приложения.

• Если фильтр намерения не объявлен, и экспортированный атрибут не установлен, по умолчанию действие доступно только через Intents, источником которого является одно и то же приложение.

Исключение из вышеприведенных правил допускается, если разработчик указывает атрибут sharedUid в файле манифеста. В этом случае другое приложение может работать в том же процессе и с тем же идентификатором пользователя Linux, что и текущее приложение. Это добавление изменяет поведение классов Activity, которые не экспортируются: их можно вызывать не только из одного приложения, но также из приложения двойника с тем же идентификатором. В листинге (рис. 3) показано использование атрибута sharedUserId.

Важно понимать, что механизм фильтра намерений не предоставляет никаких гарантий безопасности и подразумевается только как свободное связывание между классами Деятельности и намерениями; любое действие с фильтром намерения может по-прежнему отправлять явное намерение, и в этом случае фильтр намерения игнорируется. Присутствие этого атрибута, однако, изменяет поведение атрибута экспорта, связанного с безопасностью, как описано выше.

 Все приложения Android с графическим интерфейсом содержат по крайней мере одно действие, что означает, что уязвимости, связанные с классами Activity, могут повлиять на большинство приложений. Все обнаруженные нами уязвимости имеют общее сходство с информационными потоками, которые предназначены для того, чтобы позволить дочерним классам активности общаться с родительским, но вместо этого могут использоваться неавторизованными приложениями для доступа к конфиденциальной информации без явного объявления соответствующего разрешения.

4.7. УЯЗВИМОСТИ СЕРИАЛИЗАЦИИ В JAVA

Сериализация - это процесс сохранения состояния объекта в последовательность байт; десериализация это процесс восстановления объекта, из этих байт. Java Serialization API предоставляет стандартный механизм для создания сериализуемых объектов. В этой статье вы увидите как сериализовать объект, и почему сериализация иногда необходима. Вы узнаете об алгоритме сериализации используемом в Java и увидите пример, который иллюстрирует сериализованый формат объекта. В конце у вас должно сложиться чёткое представление о том, как работает алгоритм сериализации, а также каким образом представлены части объекта в сериализованном виде.

В сегодняшнем мире типичное промышленное приложение будет иметь множество компонентов и будет распространено через различные системы и сети. В Java всё представлено в виде объектов; если двум компонентам Java необходимо общаться друг с другом, то им необходим механизм для обмена данными. Есть несколько способов реализовать этот механизм. Первый способ — это разработать собственный протокол и передать объект. Это означает, что получатель должен знать протокол, используемый отправителем для воссоздания объекта, что усложняет разработку сторонних компонентов. Следовательно, должен быть универсальный и эффективный протокол передачи объектов между компонентами. Сериализация создана для этого, и компоненты Java используют этот протокол для передачи объектов.[14]

Уязвимость в Java-десериализации — давно известная и изученная проблема. С помощью этой бреши можно выполнить код внутри приложения. На платформе Android эта проблема выражена более ярко из-за особых алгоритмов обмена данными. Между приложениями отправляются «сообщения» — intents.

Если приложение получает такое «сообщение» и имеет доступ к его данным, а также хотя бы один Java-класс, загруженный с помощью десериализации, то такое приложение уязвимо.

**Использование modjoda**

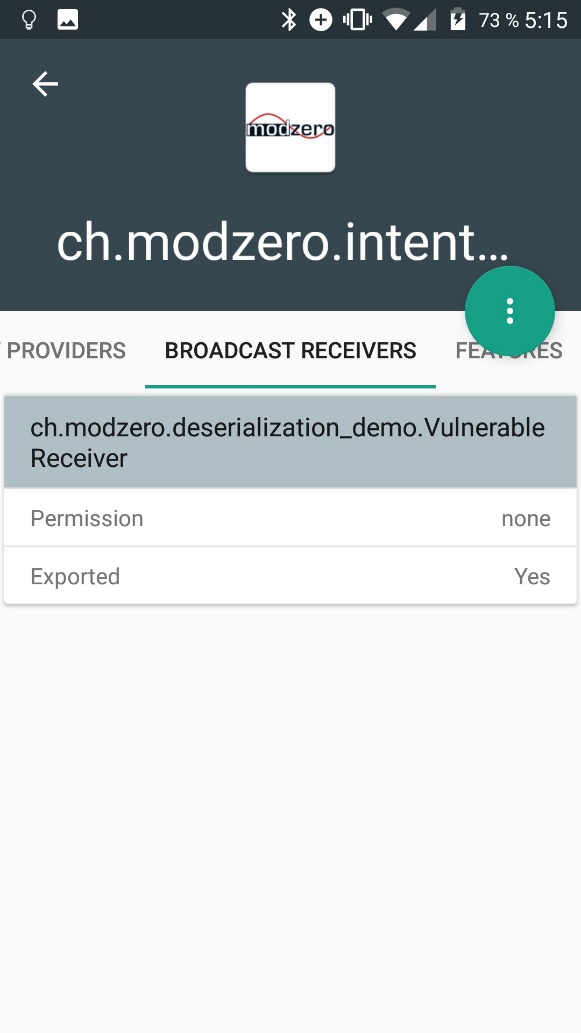
Для исследований безопасности был разработан инструмент, осуществляющий поиск данных уязвимостей. В репозитории[15] modjoda находятся два приложения: атакующее, которое отправляет intent с вредоносными данными, и уязвимое — для демонстрации работоспособности инструмента.

Для тестирования приложения необходимо выполнить несколько шагов:

1. Скомпилировать в Android Studio папку с атакующим приложением.
2. Установить полученный APK-файл на Android-устройство.
3. Убедиться, что на устройстве установлено тестируемое приложение.
4. Запустить тестируемое приложение.
5. Запустить атакующее приложение.
6. В атакующем приложении ввести название intent и нажать кнопку send.

Все полученные полезные данные появятся в атакующем приложении. Если удастся выполнить сторонний код в тестируемом приложении, оно отправит атакующему intent, и данные об этом появятся на экране.

Воспользовавшись сторонней программой мы видим экспортированный уязвимый приемник (рис. 4):

**

*Рис. 4.* Э*кспортированный уязвимый приемник:*

Используя атакующее приложение атака на десериализациюбыла успешно проведена ,о чем свидетельствует полученный ответ на приемник атакующего приложения (рис. 5, 6) и отмеченный чекбокс:

|  |  |
| --- | --- |
| https://pp.userapi.com/c846220/v846220711/86794/rfl7DuT-Dbk.jpg | https://pp.userapi.com/c846522/v846522711/884b0/lLv4vFV1Jdw.jpg |
| *Рис. 5 Приемник атакующего приложения* | *Рис. 6. Атакующее приложение* |

4.8. УЯЗВИМОСТИ ПРИЛОЖЕНИЙ, СВЯЗАННЫЕ С ВОЗМОЖНОСТЬЮ ФАЗЗИНГА

Фаззинг — техника тестирования программного обеспечения, часто автоматическая или полуавтоматическая, заключающая в передаче приложению на вход неправильных, неожиданных или случайных данных. Предметом интереса являются падения и зависания, нарушения внутренней логики и проверок в коде приложения, утечки памяти, вызванные такими данными на входе. Фаззинг является разновидностью выборочного тестирования ( random testing), часто используемого для проверки проблем безопасности в программном обеспечении и компьютерных системах.

Фаззинг часто можно классифицировать как мутационное или генерационное. Мутационный фаззинг в основном относится к тому факту, что мы берем исходный действительный ввод и применяем различные типы мутаций, прежде чем тестировать его против целевой системы. С другой стороны, генерационный фаззинг относится к процессу создания ввода с нуля с учетом конкретного формата типа ввода. Оба подхода имеют преимущества в определенных ситуациях [18]

Для примера был рассмотрен фаззер: Droid Application Fuzz Framework[16]

Droid Application Fuzz Framework (DAFF) помогает вызвать падения в Android-браузерах и PDF-просмоторщиках . DAFF состоит из встроенных фазеров и аварийного монитора. В настоящее время он поддерживает фаззинг следующих приложений:

**Browsers**

* Google Chrome - com.android.chrome
* Mozilla Firefox - org.mozilla.firefox
* Opera - com.opera.browser
* UC Browser - com.UCMobile.intl

**PDF Viewers/Readers**

* Adobe Acrobat Reader - com.adobe.reader
* Foxit PDF Reader - com.foxit.mobile.pdf.lite
* Google PDF Viewer - com.google.android.apps.pdfviewer
* WPS Office + PDF - cn.wps.moffice\_eng
* Polaris Office - com.infraware.office.link (WIP)

К сожалению, данный фаззер не смог вызвать аварийных завышений процессов на тестируемом устройстве с Android 7.1 и с последними обновлениями приложений, что может свидетельствовать о своевременном исправлении уязвимостей в приведенных программах и использовании подходов фаззинга для тестирования приложения на этапе его создания.

# 5. ИССЛЕДОВАНИЕ ПРИЛОЖЕНИЯ НА НАЛИЧИЕ УЯЗВИМОСТЕЙ

Для примера обнаружения и эксплуатации уязвимостей было взято заведомо уязвимое приложение InsecureBankv2.А также настроена среда для изучения приложения и его декомпиляции.

**Сканирование файлов APK ScanDroid'ом и основными утилитами**

Приложение ScanDroid сканирует код и определяет возможные уязвимости. Эта статья показывает использование ScanDroid с использованием кода Ruby. Этот код является прототипом для показа возможности использования ScanDroid. Успростим задачу и будем рассматривать три вида уязвимостей для приложений:

* 1. чтение/запись на локальный носитель;
  2. 2) доступ к внешнему URL;
  3. 3) создание сокет соединения.

ScanDroid выбирает вышеуказанные файлы из файла APK. Программа конвертирует код Dalvik в Java код и затем сканирует код на возможные уязвимости по ранее установленным правилам.

ScanDroid выполняет следующие операции:

* Распаковка - распаковывает файлы из APK файла
* Xml - конвертирует файлы в xml файлы, а classes в формат .smali
* dex2clas - конвертирует classes.dex в отдельные файлы . class
* Проверка - выполняет проверку кода на возможные уязвимости
* AndroidManifest - выполняет проверку Androidmanifest.xml на наличие дополнительной информации

Промежуточные файлы на каждом этапе процесса создаются для дальнейшего использования. Библиотеки ScanDroid могут быть использованы для ручного выполнения каждой стадии процесса.

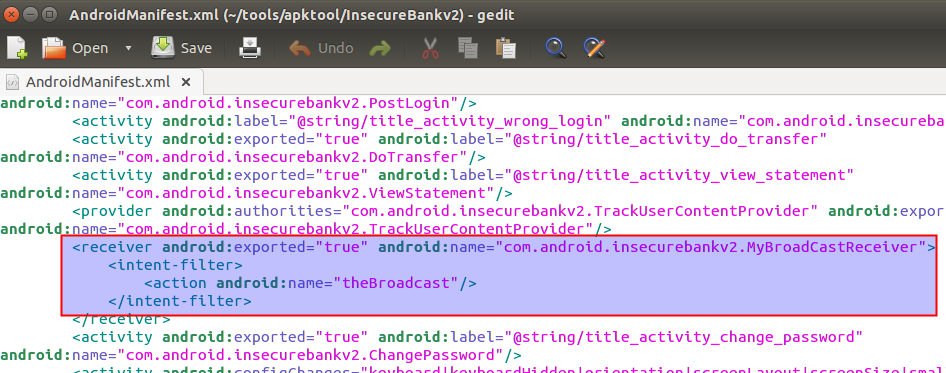
5.1. ОБНАРУЖЕНИЯ И ЭКСПЛУАТАЦИИ УЯЗВИМОСТИ INTENT SNIFFING

При помощи утилиты jadx и APKTool приложение было разобрано на его составляющие и был произведен анализ файла AndroidManifest.xml

Был найден широковещательный приемник, который был объявлен в manifest-файле приложения, он явно экспортируется. Это означает, что он доступен любому другому стороннему приложению. Любое другое приложение, в свою очередь, может послать интент, и программа может выполнить какие-то действия, если они там предусмотрены (рис. 7).

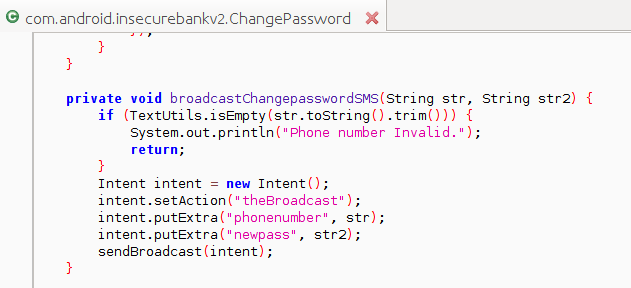
В данном случае мы видим потенциальный вектор для атаки. Если копнем глубже и посмотрим исходники, по ним станут понятно, что этот сервис позволяет, перехватить личные данные пользователя. Навряд ли разработчики предусматривали такую функцию для каких-то сторонних приложений.

Ещё один момент: когда у нас какая-то компонента использует интент-фильтры, то по умолчанию она тоже становится экспортируемой. Тут, возможно, начинающие разработчики могут споткнуться. Если интент-фильтр есть, то эта компонента становится публичной.

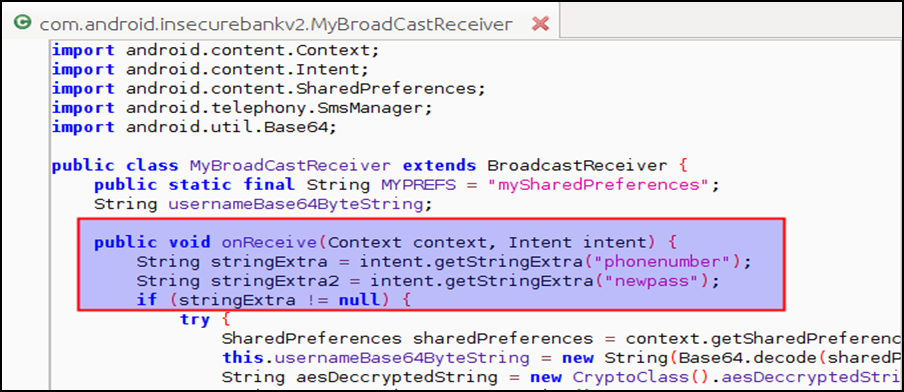
**

*Рис. 7. Экспортированный приемник*

Далее было найдено место в исходном коде программы, в котором показаны параметры, которые передаются широковещательному приемнику, объявленному в приложении, которое было показано ранее (рис. 8, 9).

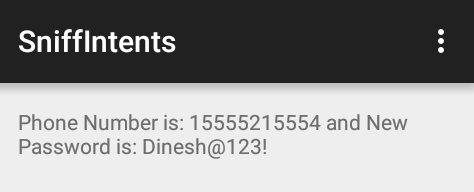
**

*Рис. 8. Участок кода приемника*

**

*Рис. 9. Действия приемника при получении сообщения*

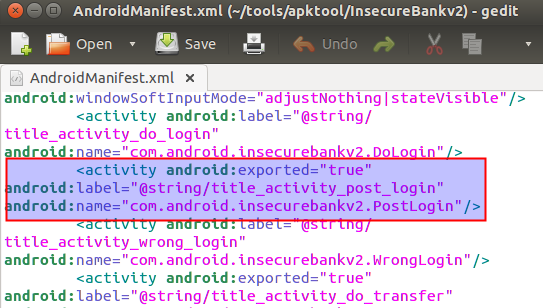
Далее при помощи приложения Sniff Intents ,предоставляемого в виде исходного кода вместе с уязвимой программой была проведена успешная попытка перехвата сообщения ,содержавшего приватную информацию (рис. 10).

**

*Рис. 10. Перехват сообщения с помощью* *Sniff Intents*

5.2. ОБНАРУЖЕНИЕ И ЭКСПЛУАТАЦИЯ ЭКСПОРТИРОВАННЫХ ANDROID ACTIVITIES

Получив исходные файлы исследуемого приложения на предыдущем этапе в манифесте приложения так же была найдена явно экспортированная ,но не защищённая активность (рис. 11).

**

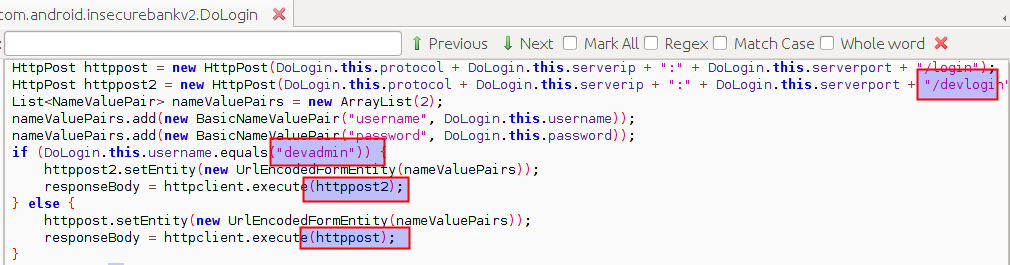
*Рис. 11.* *Экспортированная, не защищённая активность*

Далее при помощи утилиты Apk Analyzer был получен список экспортированных активностей с возможностью из запуска .При помощи утилиты незащищенная активность была успешно запущенна. (рис. 12, 13)

|  |  |
| --- | --- |
| https://pp.userapi.com/c844417/v844417768/8ed46/_ZNq7Yu5Ucc.jpg | https://pp.userapi.com/c846524/v846524768/8a1d8/AYPYdpXfFMc.jpg |
| *Рис. 12.* *Список экспортированных активностей* | *Рис. 13. Результат эксплуатации уязвимости* |

5.3. ОБНАРУЖЕНИЕ ПРОГРАММНЫХ ЗАКЛАДОК (BACKDOORS)

На следующем снимке экрана (рис. 14) показан декомпилированный код для бэкдора разработчика, который присутствует в приложении Android-InsecureBankv2, который позволяет пользователю с именем пользователя как «devadmin» достичь другой конечной точки по сравнению со всеми другими пользователями.

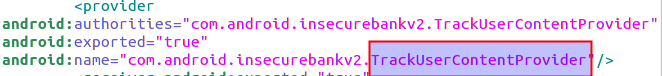
**

*Рис. 14. Декомпилированный код для бэкдора разработчика.*

Было установлено, что любой пользователь может использовать имя пользователя «devadmin» для учетной записи и войти в приложение с любым паролем независимо от срока действия пароля.

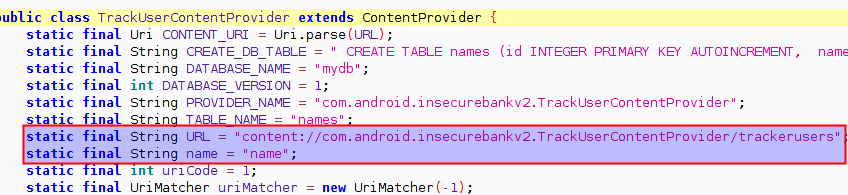
5.4. ОБНАРУЖЕНИЕ И ЭКСПЛУАТАЦИЯ ЭКСПОРТИРОВАННЫХ ANDROID CONTENT PROVIDERS

Как и в предыдущих случаях экспортированный провайдер был найден в манифесте приложения (рис. 15).

**

*Рис. 15.* *Экспортированный провайдер*

Данный провайдер отвечает за историю входа пользователей в исследуемое приложение. Проанализировав исходный код было найден участок, попадающий под уязвимость (рис. 16):

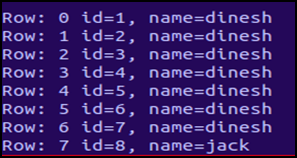
**

*Рис. 16.* *Участок, попадающий под уязвимость*

После этого была предпринята попытка получить данные из этого провайдера. На смартфоне было запущенно уязвимое приложение и была введена следующая команда терминала андроид :

content query –uri ontent://com.android.insecurebankv2.TrackUserContentProvider/trackerusers

И как итог был получена история входа пользователей в незашифрованном виде (рис. 17)

**

*Рис. 17.* *История входа пользователей в незашифрованном виде*

5.5. ВЫВОД И ОБОБЩЕНИЕ ПО РАССМОТРЕННЫМ УЯЗВИМОСТЯХ

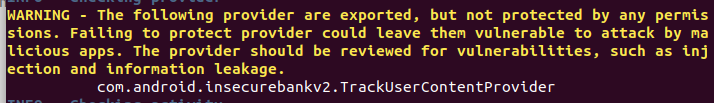
Данные уязвимости легко обнаруживаемы при статическом анализе приложения и его файлов. Для такого анализа существует много утилит ,одна и таких : QARK[17]

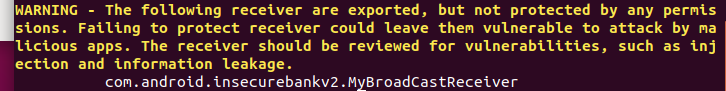
Quick Android Review Kit - этот инструмент предназначен для поиска нескольких уязвимых приложений для Android, связанных с безопасностью, либо в исходном коде, либо в пакетах APK. Этот инструмент также способен создавать «APP» и / или команды ADB с поддержкой «Proof-of-Concept», способные использовать многие обнаруженные уязвимости.

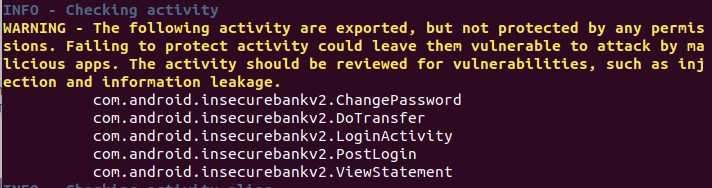
Типы уязвимостей безопасности, которые этот инструмент пытается найти:

* Непреднамеренно экспортированные компоненты.
* Неправильно защищенные экспортируемые компоненты.
* Намерения, которые являются уязвимыми для перехвата или подслушивания
* Создание общедоступных файлов.
* Действия, которые могут привести к утечке данных.
* Использование Sticky Intents.
* Небезопасно создаваемые ожидающие намерения.
* Отправка небезопасных трансляционных намерений.
* Частные ключи, встроенные в исходные файлы.
* Слабая или неправильная криптография.
* Потенциально уязвимые конфигурации WebView.
* Экспортированные активности.
* Приложения, которые позволяют создавать резервные копии.
* Приложения, которые отлаживаются.
* Приложения, поддерживающие устаревшие версии API, с известными уязвимостями.

Используя это приложение можно значительно ускорить поиск уязвимостей. На скриншотах приведен результат анализа манифеста исследуемого приложения (рис. 18).

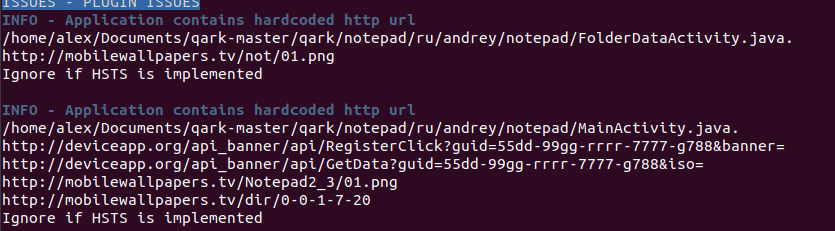
**

**

**

*Рис. 18. Результат анализа манифеста исследуемого приложения*

Так же утилита осуществлять поиск hardcoded строк в исходном коде (рис. 19):

**

*Рис. 19. Поиск hardcoded строк в исходном коде*

Так же для анализа приложений могут использоваться утилиты andro\_Bugs и SCanDroid предоставляющие схожий функционал, но так же исследующие уязвимости других типов (SSL, уязвимости WebView ,XSS, SQL):

|  |
| --- |
| **<SSL\_Security> SSL Connection Checking:**  URLs that are NOT under SSL (Total:2):  http://plus.google.com/  => Lcom/google/android/gms/common/internal/zzm;-><clinit>()V  http://www.google.com   * + Lcom/google/android/gms/internal/zzgk$zza;-><init>(Landroid/content/Context;)V …   **[Critical] <WebView><Remote Code Execution><#CVE-2013-4710#> WebView RCE Vulnerability Checking:**  Found a critical WebView "addJavascriptInterface" vulnerability. This method can be used to allow JavaScript to control the host application.  This is a powerful feature, but also presents a security risk for applications targeted to API level JELLY\_BEAN(4.2) or below, because JavaScript could use reflection to access an injected object's public fields. Use of this method in a WebView containing untrusted content could allow an attacker to manipulate the host application in unintended ways, executing Java code.  **[Warning] <WebView> WebView Local File Access Attacks Checking:**  Found "setAllowFileAccess(true)" or not set(enabled by default) in WebView. The attackers could inject malicious script into WebView and exploit the opportunity to access local resources. This can be mitigated or prevented by disabling local file system access. (It is enabled by default) permissions of the host application.  **[Warning] <WebView> WebView Potential XSS Attacks Checking:**  Found "setJavaScriptEnabled(true)" in WebView, which could exposed to potential XSS attacks.  **[Info] <Database> Android SQLite Databases Encryption (SQLite Encryption Extension (SEE)):**  This app is "NOT" using SQLite Encryption Extension (SEE) on Android (http://www.sqlite.org/android) to encrypt or decrypt databases. |

*Рис. 20. Результат анализа программы andro\_Bugs*

# 6. КОНЦЕПТ РАЗРАБАТЫВАЕМОГО ИНСТРУМЕНТА ДЛЯ ПОИСКА УЯЗВИМОСТЕЙ

Выбирая между статическим и динамическим анализом нельзя сделать однозначный выбор. В хорошем инструменте анализа приложений должны сочетаться оба подхода.

Методы статического анализа применимые при исследовании Android приложений:

* Анализ манифеста приложения для поиска незащищенных объектов и их поиск в исходном коде
* Анализ исходно кода с целью нахождения hardcoded строк
* Сигнатурный поиск уязвимостей CVE
* Поиск возможных уязвимостей XSS
* Поиск уязвимостей в WebView
* Поиск устаревших сертификатов и уязвимых версий используемых инструментов криптографии
* Анализ исходного кода и приложения для определения возможности доступа к файлам /открытия определенных файлов
* Поиск бекдоров и подозрительных данных и строк в исходном коде приложения
* Поиск известных уязвимостей в устаревших API

Методы динамического анализа, применимые при исследовании Android приложений:

* Инструментация кода для анализа поведения приложения
* Построение графов передачи привилегий между приложений
* Построение графов вызовов и взаимодействия между приложениями
* Динамический анализ API вызовов
* Использование запуска песочницы для перехвата и анализа совершаемых действий
* Применение фаззинга после получения информации и взаимодействиях приложения с файлами и проверкой покрытия кода через инструментацию
* Попытки совершения атак типа человек посередине

# ЗАКЛЮЧЕНИЕ

Проблемы безопасности мобильной ОС Android существуют на всех уровнях платформы и требуют более комплексного подхода, чем те меры защиты, которые можно наблюдать сейчас. Одна из главных проблем безопасности мобильной ОС Android — значительная фрагментация рынка, которая не позволяет организовать своевременную поставку обновлений безопасности всем устройствам.

При этом ныне существующие средства защиты, включая антивирусы и песочницы, имеют множество недостатков и не могут полностью защитить пользователя.

Но несмотря на это последнее время наблюдается ряд позитивных движений в сторону улучшения безопасности Android. В частности, компания Google запустила программу выпуска обновлений безопасности: они выходят каждый месяц и некоторые вендоры все же добавляют их в свои версии прошивки.

Существует множество инструментов для анализа уязвимостей приложений ,но большинство из них используют лишь часть способов ,тем самым они не могут давать более полную картину о безопасности исследуемого приложения.

Исходя из этого был сделан вывод о том, что важным очень важна разработка и внедрение комплексных систем обнаружения уязвимостей включающих все возможные известные и нетрадиционные методики, а также являющееся гибким в отношении появления новых уязвимостей . Пример такого средство предлагается к реализации в рамках данной НИР.

СПИСОК ИСТОЧНИКОВ

* 1. Статья «Основы создания приложений» https://developer.android.com/guide/components/fundamentals?hl=ru
  2. Wu L. et al. The impact of vendor customizations on android security // Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. ACM, 2013. P. 623—634.
  3. Sun M., Tan G. NativeGuard: Protecting android applications from third-party native libraries // Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks. ACM, 2014. P. 165—176.
  4. Enck W., Ongtang M., McDaniel P. Understanding android security // IEEE security & privacy. 2009. № 1. P. 50—57.
  5. Enck W., Ongtang M., McDaniel P. Understanding android security // IEEE security & privacy. 2009. № 1. P. 50—57.
  6. Статья «Vulnerability Details» <https://www.cvedetails.com/cve/CVE-2011-1717>
  7. Fahl S. et al. Why Eve and Mallory love Android: An analysis of Android SSL (in) security // Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012. P. 50—61.
  8. Статья «Android Stagefright contains multiple vulnerabilities» <https://kb.cert.org/vuls/id/924951>
  9. CVE-2015-3842 // cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3842
  10. Shabtai A., Mimran D., Elovici Y. Evaluation of Security Solutions for Android Systems // arXiv preprint arXiv:1502.04870. — 2015.
  11. Petsas T. et al. Rage against the virtual machine: hindering dynamic analysis of android malware // Proceedings of the Seventh European Workshop on System Security. ACM, 2014. P. 5.
  12. Сайт разработчика программы <https://dexprotector.com>
  13. Статья «A Local Cross-Site Scripting Attack against Android Phones» <https://www.infsec.cs.uni-saarland.de/projects/android-vuln/android_xss.pdf>
  14. Статья «The Java serialization algorithm revealed» <https://www.javaworld.com/article/2072752/the-java-serialization-algorithm-revealed.html>
  15. Исходники программы «Android Java Deserialization Vulnerability Tester» <https://github.com/modzero/modjoda>
  16. Исходники программы «Droid Application Fuzz Framework» <https://github.com/ajinabraham/Droid-Application-Fuzz-Framework>
  17. Исходники программы «QARK» <https://github.com/linkedin/qark>
  18. Статья «Fuzzing Android: a recipe for uncovering vulnerabilities inside system components in Android» <https://www.blackhat.com/docs/eu-15/materials/eu-15-Blanda-Fuzzing-Android-A-Recipe-For-Uncovering-Vulnerabilities-Inside-System-Components-In-Android-wp.pdf>