Министерство образования и науки Российской Федерации
Санкт-Петербургский государственный политехнический университет
—
Институт компьютерных наук и кибербезопасности
Высшая школа кибербезопасности

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

«Модификация приложения»

по дисциплине «Безопасность операционных систем»

Выполнил
студент гр. 5131001/10302 Куковякина Д. А.

<подпись>

Проверил
асс. преподавателя Гололобов Н. В.

<подпись>

Санкт-Петербург
2025

СОДЕРЖАНИЕ

1 Формулировка задания

Восстановить алгоритм работы и модифицировать код предоставленного приложения так, чтобы осуществлялся перехват/сохранение/вывод каждой пары логин + пароль, с которыми успешно осуществлен вход.

В отчете привести:

− процесс восстановления работы программы;

− блок-схему работы программы;

− права, запрашиваемые приложением, метаданные и т. д.

Условие: не использовать Frida, выполнять модификацию через smali-код.

2 Теоретические сведения

APK (Android Package Kit) — это формат файла пакета приложений для операционной системы Android. Он используется для распространения и установки мобильных приложений.

APK-файл представляет собой ZIP-архив, содержащий все необходимые компоненты для работы приложения. Его основные составляющие:

— AndroidManifest.xml – файл манифеста, содержащий информацию о приложении, такую как его название, версия, разрешения и компоненты;

— classes.dex – скомпилированный код приложения в формате Dalvik Executable;

— resources.arsc – скомпилированные ресурсы приложения такие как строки, размеры;

— res – директория, содержащая ресурсы, не скомпилированные в resources.arsc, такие как изображения и звуки;

— META-INF – директория, содержащая служебную информацию о содержимом .apk файла для установки (информация о подписи, версиях);

— assets – опциональная директория, в которой содержатся дополнительные файлы для приложения, такие как шрифты, текстовые файлы, рисунки;

— lib – опциональная директория, в которой содержатся скомпилированные нативные библиотеки под различные архитектуры процессоров.

Код Smali — это форма промежуточного кода, используемая виртуальной машиной в операционной системе Android. Он создается путем декомпиляции

файлов DEX (Dalvik Executable), которые являются исполняемыми файлами приложений Android.

Файл Smali обычно соответствует одному классу Java. Он состоит из следующих разделов:

— .class – определяет тип класса (например, public, private, abstract), имя

класса указывается в формате Lpackage/name/ObjectName — где L показывает, что это объект, package/name/ — это пакет, в котором находится объект, ObjectName — это имя объекта;

— .super – указывает родительский класс;

— .source – указывает имя исходного файла Java, из которого был получен код Smali (необязательно);

— .field – определяет поля класса, включая их тип.

— .method – определяет методы класса, включая их сигнатуру и тело.

Вызов методов в коде происходит через ключевое слово invoke, в фигурных скобках указываются аргументы, передаваемые методу. После аргументов через запятую пишется сигнатура метода. Она имеет следующий вид: Lpackage/name/ObjectName;->MethodName(III)Z, где Lpackage/name/ObjectName; – ранее описанный формат имени класса. MethodName — это имя метода. (III)Z является сигнатурой метода. III — параметры (в данном случае 3 целых числа), а Z — тип возврата (bool).

Если метод возвращает любое значение, кроме Void, то после вызова метода всегда указывается инструкция move-result для примитивных типов или move-result-object для ссылочных типов (объекты и массивы).

Регистры – основной способ хранения данных в DEX. Количество используемых виртуальных регистров указывается для каждого методе с помощью инструкций .registers или .local. В первом случае это показывает общее

количество регистров в методе вместе с аргументами, во втором случае – только количество локальных переменных.

В Smali существует два основных типа регистров:

— локальные регистры (v-регистры), которые используются для хранения локальных переменных и временных значений внутри метода;

— регистры параметров (p-регистры), которые используются для хранения параметров, передаваемых в метод, p0 всегда содержит ссылку на текущий объект (this) для нестатических методов.

Все регистры имеют размер 32 бита, поэтому для хранения 64-битных значений (например, long и double) используются два последовательных регистра.

3 Результаты работы

3.1 Анализ apk-файла

Для анализа apk-файла использовались следующие утилиты:

— APKTool — инструмент для декомпиляции и сборки APK-файлов. С его помощью можно получить исходный код и ресурсы приложения.

— Android Studio — среда разработки для Android-приложений. В ней можно открывать и анализировать apk-файлы.

— Jadx — это декомпилятор Java-кода, который позволяет преобразовать скомпилированные файлы APK или DEX (байт-код Android) в читаемый Java-код.

Сперва проанализируем файл манифеста. Из него можно получить информацию о том, какие разрешения запрашивает приложение. В данном случае запрашивается разрешение на доступ к вибрации устройства (android.permission.VIBRATE).

В папке META-INF содержатся файлы с расширением .version, которые используются для хранения информации о версии сборки и версиях компонентов приложения. Также там содержится набор правил для ProGuard — инструмента

для обфускации и оптимизации кода в Android-приложениях. Эти правила указывают, какие классы, методы, поля и конструкторы должны быть сохранены (не обфусцированы и не удалены).

Также при помощи jadx была получена информация о подписи apk-файла. Информация представлена на рисунке 1.

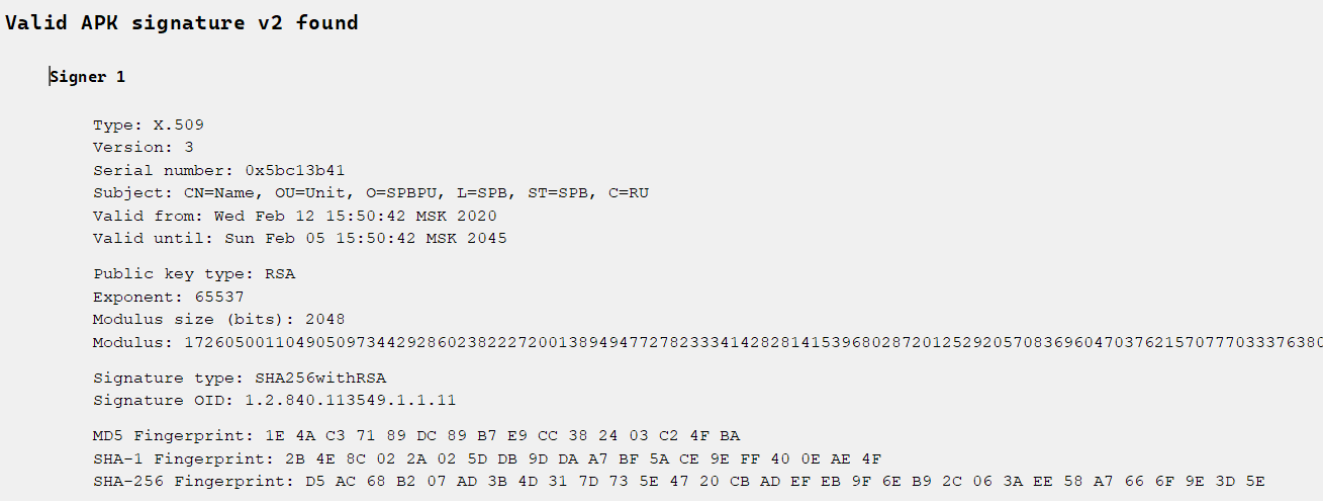

Рисунок 1 – Подпись apk-файла

Далее при помощи jadx получим java-код для анализа логики работы приложения.

Точкой входа является функция OnCreate в MainActivity. В ней сперва идет получение объектов кнопок и текстовых полей. После чего осуществляется проверка целостности как самого файла, так и его подписи.

Метод k проверяет целостность APK-файла, сравнивая контрольную сумму (CRC) с ожидаемым значением. Данный метод:

— получает имя файла из ресурса с идентификатором *d81658c978c10bfdc2,* он содержит строку classes.dex;

— открывает APK-файл приложения как ZIP-архив;

— извлекает файл из архива по раннее полученному имени;

— вычисляет CRC для извлеченного файла;

— кодирует CRC в Base64;

— сравнивает полученную Base64-кодированную CRC с ожидаемой CRC, хранящейся в ресурсе с идентификатором *a798a293d36cb77282*.

Метод 1 проверяет подпись приложения, сравнивая хэш SHA-256 сертификата подписи с ожидаемым значением. Ресурс с идентификатором. *a798a293d36cb77283* содержит ожидаемый хэш сертификата подписи в формате Base64.

Затем идет определение триггеров, срабатывающих при нажатии на кнопку или изменении текста. Основными являются обработчики нажатия на кнопки SignIn и SignUp.

В них обоих сперва идет получение введённых логина и пароля. Далее идет проверка имени пользователя (длина от 3 до 10 символов и использование только символов a-z и 0–9) и проверка пароля (длина от 6 до 30 символов). Также реализована проверка наличия данных, что оба поля не пусты.

Далее логика отличается. В случае регистрации происходит попытка добавить пользователя в базу данных. В случае же аутентификации проверяется, существует ли пользователь с таким именем и паролем. Если проверка успешна, создаётся Intent для перехода на экран NotesListActivity, и туда передаётся имя пользователя.

NotesListActivity реализует активность для отображения и управления списком заметок. Основные функции включают создание, поиск, сортировку и удаление заметок, а также настройку внешнего вида. При создании новой заметки управление передается в NoteActivity.

NoteActivity предназначен для работы с самими заметками. Он позволяет создавать, редактировать, удалять заметки. Кроме того, он обрабатывает Intent с действием ACTION_SEND, что позволяет получать текст из других приложений и создавать на его основе заметки.

На рисунке 2 представлена блок-схема алгоритма создания приложения и аутентификации пользователя.

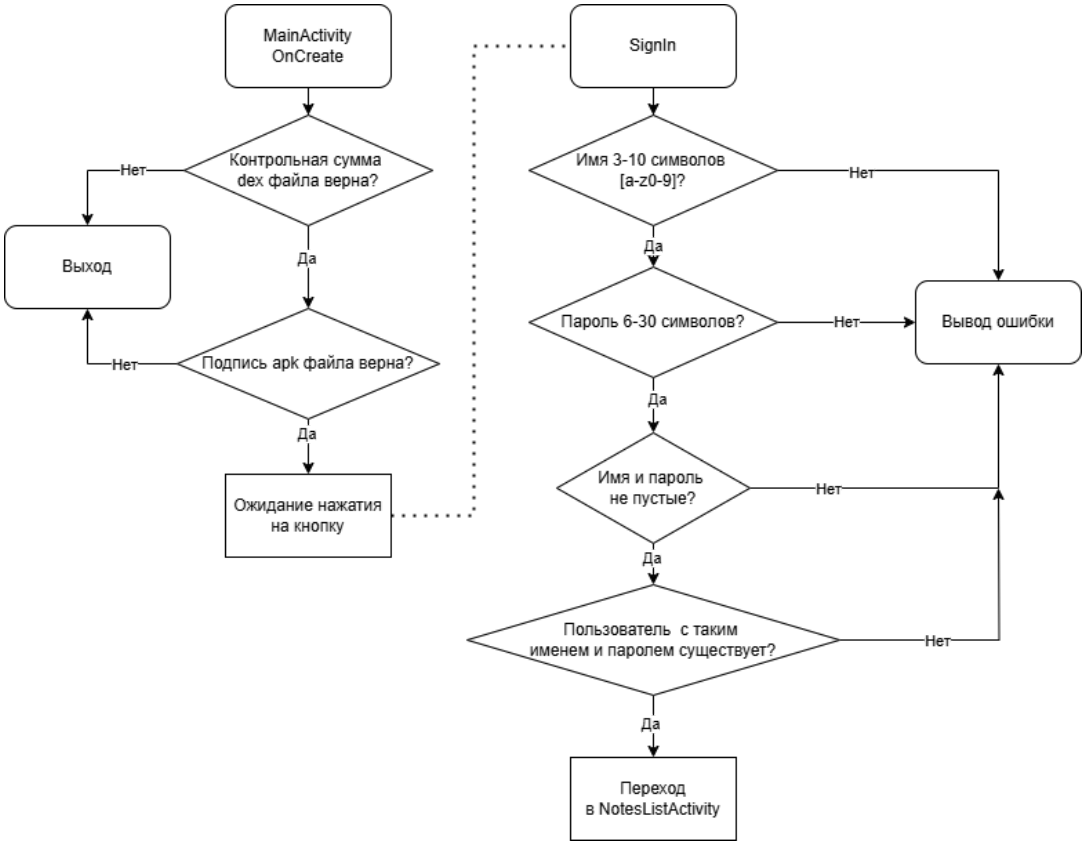

Рисунок 2 – Блок-схема

3.2 Реализация перехвата

При помощи apktool получим smali-код.

Первым делом необходимо обойти проверку целостности приложения и подписи. Найдем нужный участок в файле MainActivity:

```
invoke-virtual {p0}, Lcom/ibks/notes/MainActivity;->k()Z
move-result p1
if-eqz p1, :cond_64
invoke-virtual {p0}, Lcom/ibks/notes/MainActivity;->l()Z
move-result p1
if-nez p1, :cond_86
```

Первое условие if-eqz проверяет, равен ли регистр p1, хранящий результат выполнения метода k, 0. Если равен – сразу переходит к выполнению кода внутри условия, то есть закрытию программы. Иначе переходит ко второму условию if-nez, которое проверяет, что вернул метод l. Если вернул не 0 – переходит к

дальнейшему выполнению программы. Для обхода проверки удалим данное условие и весь вложенный в него код.

Далее модифицируем обработчик нажатия копки входа так, чтобы при успешной аутентификации данные выводились в логи.

Для этого первым делом создадим два дополнительных локальных регистра внутри метода обработчика, в которые сохраним результат получения логина и пароля из текстовых полей.

```
iget-object p1, p1, Lcom/ibks/notes/MainActivity;->n:Landroid/widget/EditText;
invoke-virtual {p1}, Landroid/widget/EditText;->getText()Landroid/text/Editable;
move-result-object p1
invoke-virtual {p1}, Ljava/lang/Object;->toString()Ljava/lang/String;
move-result-object p1
move-object v3, p1
…
invoke-virtual {v0}, Ljava/lang/Object;->toString()Ljava/lang/String;
move-result-object v0
move-object v4, v0
```

Создадим отдельный метод logMsg, который будет выводить информацию. Исходный код представлен в приложении А. Разберем его построчно:

— `method public static logMsg(Ljava/lang/String;Ljava/lang/String;)V`

–объявляет публичный статический метод, принимающий на вход две строки;

— `new-instance v1, Ljava/lang/StringBuilder` – создает новый экземпляр класса StringBuilder и сохраняет его в регистре v1;

— `invoke-direct {v1}, Ljava/lang/StringBuilder;-><init>()V` – вызывает конструктор StringBuilder для инициализации;

— `const-string v0, "Login: "` – загружает строковую константу "Login: " в регистр v0;

— `invoke-virtual {v1, v0}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder` – вызывает метод append объекта StringBuilder и добавляет строку v0 к v1;

— `invoke-virtual {v1, p0}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder` – добавляет первый аргумент метода (p0) к StringBuilder, после чего проделывает то же самое с паролем;

— `invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String` – вызывает метод toString, чтобы получить построенную строку, и сохраняет результат в регистре v2;

— `invoke-static {v0, v2}, Landroid/util/Log;->i(Ljava/lang/String;Ljava/lang/String;)I` – вызывает статический метод i класса Log для вывода информационного сообщения в лог Android. Первый аргумент (v0) — тег лога ("CATCHED"), второй аргумент (v2) — сообщение лога (построенная строка).

Сразу после успешной аутентификации создается объект intent. Добавим вывод информации прямо перед его созданием, передав в качестве параметров упомянутые ранее регистры.

```
if-nez v0, :cond_78
invoke-static {v3, v4}, Lcom/ibks/notes/MainActivity;->logMsg(Ljava/lang/String;Ljava/lang/String;)V
new-instance v0, Landroid/content/Intent;
```

Теперь необходимо собрать измененный smali-код в новый apk файл. Для этого снова воспользуемся утилитой apktool.

После сборки новый apk необходимо подписать.

Сперва нужно создать закрытый ключ. Для этого воспользуемся утилитой keytool. Это инструмент командной строки, предоставляемый Java Development Kit (JDK). Он используется для создания и управления хранилищами ключей

(keystores), которые содержат криптографические ключи и сертификаты. Команда для создания ключа:

```
keytool -genkeypair -v -keystore my-release-key.keystore -keyalg RSA -keysize 2048 -validity 10000 -alias my-key-alias
```

После этого можно подписать apk при помощи apksigner. Apksigner – это инструмент командной строки, предоставляемый Android SDK. Он используется для подписи APK-файлов и пришел на смену устаревшему инструменту jarsigner. Команда для подписи:

```
apksigner sign --ks my-release-key.keystore --ks-key-alias my-key-alias owl_notes_patch.apk
```

3.3 Тестирование

Убедимся, что все работает. Для этого загрузим новый apk файл в Android Studio.

После этого запустим приложение и зарегистрируем пользователя dasha с паролем 123456. Попробуем войти (рисунок 3) и увидим сообщение в логах с собранными данными (рисунок 4).

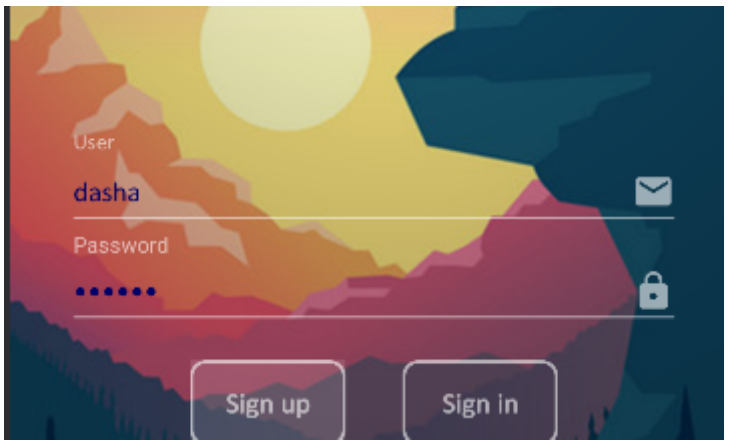

Рисунок 3 – Аутентификация

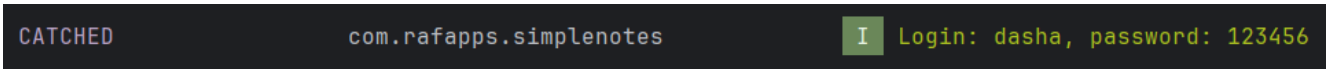

Рисунок 4 – Перехваченные данные

4 Выводы

В ходе выполнения данной лабораторной работы был проведен реверс-инжиниринг Android-приложения. Путем декомпиляции APK-файла и анализа smali-кода была восстановлена логика работы приложения и добавлена функциональность перехвата учетных данных.

Данное приложение использовало несколько методов защиты: проверка контрольной суммы приложения, проверка хэша подписи приложения, а также обфускация при помощи ProGuard, который заменяет имена всех внутренних классов, методов и полей на одно-двухбуквенные сочетания, что затрудняет понимание кода. Так как удалось модифицировать код, можно сделать вывод, что данных способов было недостаточно. Возможно, более стойким приложение сделало бы использование шифрования строк и более сильной обфускации, обнаружение отладчика и эмулятора.

ПРИЛОЖЕНИЕ А

Листинг метода «logMsg»

```
.method public static logMsg(Ljava/lang/String;Ljava/lang/String;)V
    .registers 5

    .prologue
    new-instance v1, Ljava/lang/StringBuilder;

    invoke-direct {v1}, Ljava/lang/StringBuilder;-><init>()V

    const-string v0, "Login: "

                                          invoke-virtual          {v1,          v0},
Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

                                          invoke-virtual          {v1,          p0},
Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    const-string v0, ", password: "

                                          invoke-virtual          {v1,          v0},
Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

                                          invoke-virtual          {v1,          p1},
Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

    move-result-object v2

    const-string v0, "CATCHED"

    .line 10
                                           invoke-static          {v0,          v2},
Landroid/util/Log;->i(Ljava/lang/String;Ljava/lang/String;)I

    .line 13
    return-void
.end method
```