Основы reverse-engineering и модификации приложений Android

Строение APK, инструменты для работы с ними, Smali как язык ассемблера.

APK: строение

APK – ZIP архив, содержащий:

- AndroidManifest.xml
- classes.dex
- resources.arsc
- Нативные библиотеки (*.so)
- Ресурсы
- Ассеты

APK: выравнивание данных

Некоторые данные в APK пакуются без сжатия и должны быть выровнены внутри APK по 4 байтам:

- resources.arsc
- *.so
- *.png
- *.ogg
- *.wma
- *.wav

APK: ресурсы

В APK хранятся ресурсы в двух видах:

- Упакованные в resources.arsc (из каталогов values*/)
- Не упакованные (все прочие виды)

APK: XML

XML-файлы в APK хранятся виде Binary XML, для их чтения и модификации их требуется преобразовать в текстовый вид.

APK: подпись

APK подписываются.

Виды подписи:

• v1 – jar signature - пофайловая

Можно модифицировать содержимое APK после подписывания, не трогая подпись и подписанные файлы.

• v2/v3 – Full APK Signature - подпись целого архива

Нельзя модифицировать APK после подписывания.

APK: подпись v1 (jar signature)

Хранится в виде файлов в ZIP в директории META-INF/:

- MANIFEST.MF – хэши подписываемых файлов
- CERT.SF – метаданные и хэши подписываемых файлов
- CERT.RSA - сертификат и подпись хэша файла CERT.SF

Позволяет модифицировать подписанный APK не трогая сами файлы подписи и подписанные ею файлы.

APK: подпись v2/v3 (Full APK Signature)

- Хранится в блоке подписи APK, расположенном непосредственно перед центральным каталогом ZIP
- Не представлена в виде файлов в ZIP
- Не позволяет модифицировать APK даже не модифицируя файлы в нём

APK: Split APK

- Split APK позволяют отделить ресурсы (языковые, графические, нативные библиотеки) от основного APK
- Устанавливаются только подходящие конкретному устройству Split APK вместо установки большого APK со всеми разрешениями, языками и архитектурами нативных библиотек
- Все Split APK должны быть подписаны одним сертификатом
- Все Split APK должны быть одной версии (versionCode)
- Требуется устанавливать/обновлять все Split APK одной сессией даже если изменился лишь один из них

Инструменты

- apktool
- zipalign
- apksigner
- Android Studio
- java2smali
- Developer Assistant
- jadx
- Jadx Class Decompiler

Инструменты: apktool

- Комбайн для разборки и повторной сборки APK до редактируемых стандартными утилитами форматов
- Даёт на выходе неподписанный и невыровненный APK
- Для установки APK потребуется его выровнять через zipalign и подписать

Инструменты: zipalign

- Утилита для выравнивания несжатых данных в ZIP по заданному значению в байтах
- Нужна при сборке APK, выполняется перед apksigner

Инструменты: apksigner

- Утилита для подписывания APK файлов
- Выполняется после zipalign

Инструменты: Android Studio

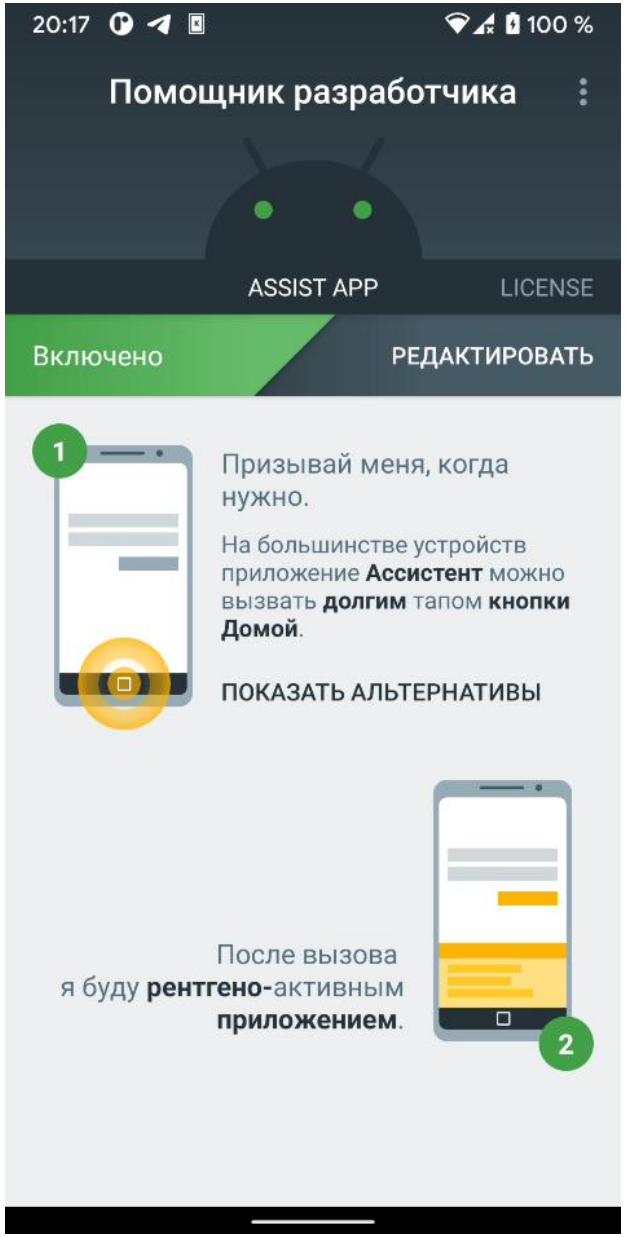
- Официальная IDE (от JetBrains) для разработки приложений под Android
- Позволяет удобно писать код патчей и редактировать ресурсы
- Closed Source

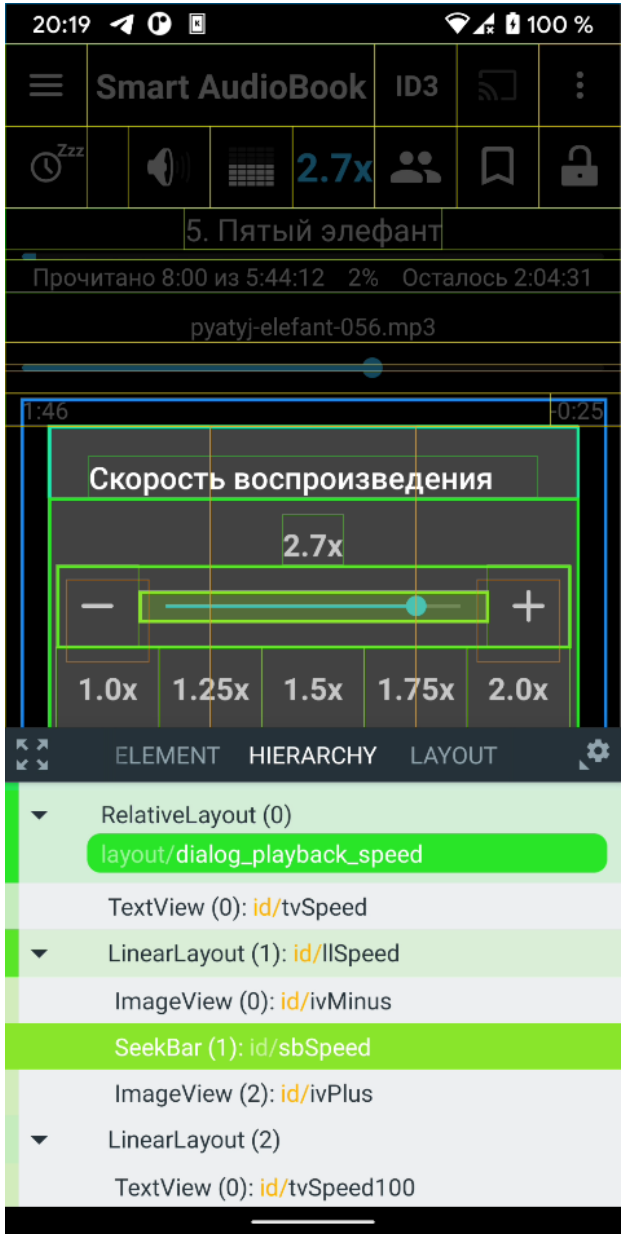
Инструменты: java2smali

- Плагин к Android Studio
- Требует, чтобы проект собирался
- Позволяет одним кликом получить из исходного кода Java/Kotlin код Smali
- Очень удобно для компиляции кода патчей.

Инструменты: Developer Assistant

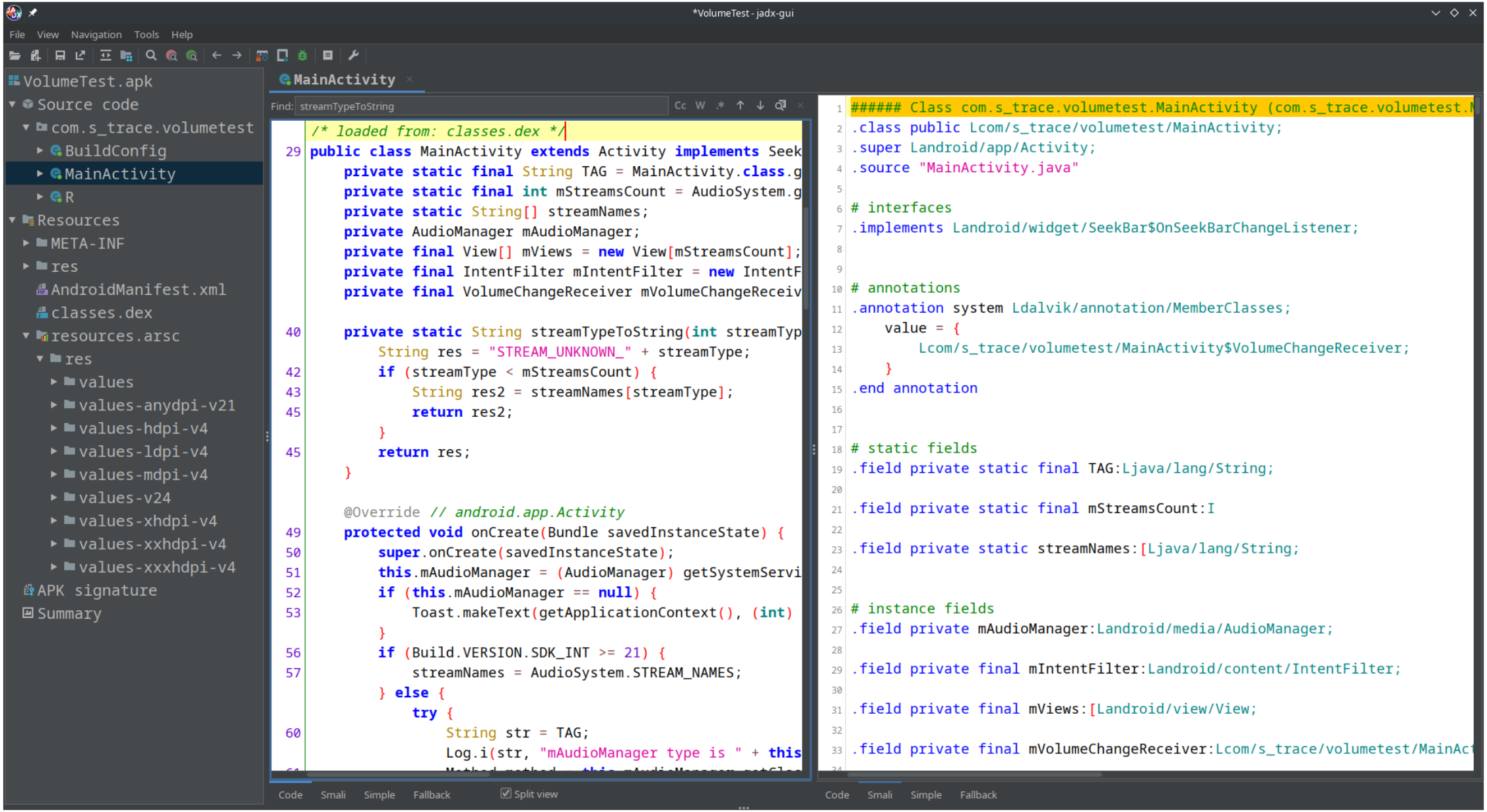
- Утилита для Android
- Является ассистентом
- Позволяет буквально потыкать интерфейс другого приложения пальчиком и узнать много полезного о нём (ID элементов, иерархия View и так далее)
- Очень удобно на начальной стадии реверсинга, когда об исследуемом приложении ничего не известно
- Closed Source


20:17
100 %
Помощник разработчика
ASSIST APP
LICENSE
Включено
РЕДАКТИРОВАТЬ
1
Призывай меня, когда нужно.
На большинстве устройств приложение Ассистент можно вызвать долгим тапом кнопки Домой.
ПОКАЗАТЬ АЛЬТЕРНАТИВЫ
После вызова я буду рентгено-активным приложением.
2


20:19
100 %
Smart AudioBook
ID3
2.7x
5. Пятый элефант
Прочитано 8:00 из 5:44:12 2% Осталось 2:04:31
pyatyj-elefant-056.mp3
1:46
-0:25
Скорость воспроизведения
2.7x
1.0x
1.25x
1.5x
1.75x
2.0x
ELEMENT
HIERARCHY
LAYOUT
RelativeLayout (0)
layout/dialog_playback_speed
TextView (0): id/tvSpeed
LinearLayout (1): id/llSpeed
ImageView (0): id/ivMinus
SeekBar (1): id/sbSpeed
ImageView (2): id/ivPlus
LinearLayout (2)
TextView (0): id/tvSpeed100

Инструменты: jadx

- Декомпилятор APK/DEX/SMALI/CLASS файлов
- Декомпилирует DEX/CLASS/SMALI до исходного кода Java
- Оптимизирован для декомпиляции APK
- Понимает метаданные Kotlin
- Показывает имена ресурсов вместо числовых ID
- Можно переименовывать классы, поля и методы
- Показывает ресурсы
- Можно экспортировать результат код в проект Gradle


*VolumeTest - jadx-gui
File View Navigation Tools Help
VolumeTest.apk
Source code
com.s_trace.volumetest
BuildConfig
MainActivity
R
Resources
META-INF
res
AndroidManifest.xml
classes.dex
resources.arsc
res
values
values-anydpi-v21
values-hdpi-v4
values-ldpi-v4
values-mdpi-v4
values-v24
values-xhdpi-v4
values-xxhdpi-v4
values-xxxhdpi-v4
APK signature
Summary
MainActivity
Find: streamTypeToString
/* loaded from: classes.dex */
public class MainActivity extends Activity implements Seek
private static final String TAG = MainActivity.class.g
private static final int mStreamsCount = AudioSystem.g
private static String[] streamNames;
private AudioManager mAudioManager;
private final View[] mViews = new View[mStreamsCount];
private final IntentFilter mIntentFilter = new IntentF
private final VolumeChangeReceiver mVolumeChangeReceiv
private static String streamTypeToString(int streamTyp
String res = "STREAM_UNKNOWN_" + streamType;
if (streamType < mStreamsCount) {
String res2 = streamNames[streamType];
return res2;
}
return res;
}
@Override // android.app.Activity
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
this.mAudioManager = (AudioManager) getSystemServi
if (this.mAudioManager == null) {
Toast.makeText(getApplicationContext(), (int)
}
if (Build.VERSION.SDK_INT >= 21) {
streamNames = AudioSystem.STREAM_NAMES;
} else {
try {
String str = TAG;
Log.i(str, "mAudioManager type is " + this
Code Smali Simple Fallback
Split view
Class com.s_trace.volumetest.MainActivity (com.s_trace.volumetest.M
.class public Lcom/s_trace/volumetest/MainActivity;
.super Landroid/app/Activity;
.source "MainActivity.java"
interfaces
.implements Landroid/widget/SeekBar$OnSeekBarChangeListener;
annotations
.annotation system Ldalvik/annotation/MemberClasses;
value = {
Lcom/s_trace/volumetest/MainActivity$VolumeChangeReceiver;
}
.end annotation
static fields
.field private static final TAG:Ljava/lang/String;
.field private static final mStreamsCount:I
.field private static streamNames:[Ljava/lang/String;
instance fields
.field private mAudioManager:Landroid/media/AudioManager;
.field private final mIntentFilter:Landroid/content/IntentFilter;
.field private final mViews:[Landroid/view/View;
.field private final mVolumeChangeReceiver:Lcom/s_trace/volumetest/MainAct
Code Smali Simple Fallback

Инструменты: Jadx Class Decompiler

- Плагин к Android Studio
- Требует, чтобы проект собирался
- Позволяет открывать Jadx на файле APK из Android Studio
- Преобразует исходник в Bytecode, Smali, ASM и Groovy
- Умеет показывать diff между старой и новой версией
- Собирается стать платным
- Closed Source

Smali

- Smali – дизассемблированный байт-код Dalvik/ART VM
- Дизассемблер - утилита Baksmali, ассемблер - утилита Smali
- Один файл Smali - один класс
- Вложенные классы - отдельные файлы Smali

Smali: пример smali-кода

```
.method private static streamTypeToString(I)Ljava/lang/String;
    .registers 3
    .param p0, "streamType"    # I

    .line 41
    new-instance v0, Ljava/lang/StringBuilder;
    invoke-direct {v0}, Ljava/lang/StringBuilder;-><init>()V
    const-string v1, "STREAM_UNKNOWN_"
    invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
    invoke-virtual {v0, p0}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;
    invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
    move-result-object v0

    .line 42
    .local v0, "res":Ljava/lang/String;
    sget v1, Lcom/s_trace/volumetest/MainActivity;->mStreamsCount:I
    if-ge p0, v1, :cond_19

    .line 43
    sget-object v1, Lcom/s_trace/volumetest/MainActivity;->streamNames:[Ljava/lang/String;
    aget-object v0, v1, p0

    .line 45
    :cond_19
    return-object v0

.end method
```

Smali: исходный код примера

```
private static String streamTypeToString(int streamType) {
    String res = "STREAM_UNKNOWN_" + streamType;
    if (streamType < mStreamsCount) {
        return streamNames[streamType];
    }
    return res;
}
```

Smali: регистры

Dalvik/ART VM – регистровая:

- Доступно 65535 регистров
- Некоторые опкоды принимают 255 или 15 первых регистров
- VM хранит тип значения в регистре
- Регистры локальны для метода, глобальных регистров нет
- Параметры передаются в последних регистрах
- Возвращённое значение явно помещается в любой из первых 255 регистров отдельным опкодом
- Каждый метод определяет число используемых регистров

Smali: регистры vX и pX

- Регистры нумеруются с нуля
- Регистры vX (v0, v1...) – основной список регистров
- Регистры pX (p0, p1...) – регистры параметров
- При этом именование pX – лишь для удобства, на уровне VM нет разницы между pX и vX
- Например, если в методе указано .locals 5 – регистр v5 будет регистром p0, и к нему можно обратиться по любому из этих имён

Smali: invoke-kind

Для вызова различных видов методов используются разные опкоды:

- invoke-direct - конструкторы и private-методы. В p0 - инстанс
- invoke-virtual – protected и public-методы. В p0 - инстанс
- invoke-super – методы родительского класса. В p0 - инстанс
- invoke-static – static-методы. В p0 – первый параметр
- invoke-*-range – методы более чем с 5 параметрами, параметры должны быть в регистрах идущих подряд. Позволяет вызывать методы на регистрах более v15

Smali: конструкторы

- Методы <clinit> и <init> являются специальными методами класса
- <clinit> выполняется в статическом контексте и служит для инициализации static полей класса
- <init> (их может быть несколько, и конкретный метод выбирается по сигнатуре) - конструктор класса.
- После создания объекта в регистре (new-instance - выделение памяти под объект) его обязательно нужно инициализировать вызовом конструктора (<init>)

Smali: именование классов

- В начале имени класса идёт ‘L’, разделителем служит ‘/’, а в конце идёт ‘;’
- Например, класс this.is.class в Smali будет именоваться как Lthis/is/class; и будет лежать в файле classes/this/is/class.smali если APK с таким классом был разобран при помощи Apktool.

Smali: ограничения

- В Smali можно передать непосредственно в метод не более пяти регистров (long и double занимают два идущих подряд регистра). Если нужно больше – придётся использовать invoke-*-range, но параметры должны идти подряд
- В Smali нельзя обратиться из вложенного класса к родительскому, поэтому во вложенном классе компилятор создаёт синтетическое поле содержащее ссылку на инстанс родительского класса и инициализирует его в конструкторе вложенного класса
- В Smali нельзя обратиться из вложенного класса к полю родительского класса, поэтому компилятор создаёт синтетические геттеры и сеттеры в родительском классе для этого

DEX: Ограничения

- В DEX не может быть более 65535 методов
- В APK может быть несколько DEX (multi-dex)
- Если в конкретном DEX не хватило места для вставки класса или метода – можно добавить или перенести этот класс в другой DEX, так как все классы из всех DEX загружаются в единую Dalvik/ART VM

Модификация: первые шаги

- Разобрать APK через apktool
- Инициализировать GIT в проекте, закоммитить всё
- Собрать проект через apktool
- Выровнять собранный APK через zipalign
- Подписать выровненный APK и все Split APK через apksigner
- Установить все APK (пересобранный и его Split APK) через adb install-multiple
- Проверить как работает приложение – если всё OK – двигаться дальше

Модификация: возможные проблемы

- APK не разбирается, ошибка декодирования ресурсов – использовать ключ —no-res если не нужно модифицировать ресурсы
- APK не собирается, ошибка кодирования ресурсов - использовать ключ —no-res при разборке, если не нужно модифицировать ресурсы
- APK не устанавливается, ошибка UPDATE_INCOMPATIBLE – нужно удалить оригинальное приложение перед установкой модифицированного
- APK не работает после пересборки – возможно, в этом APK или в других APK от этого же разраоботчика есть проверка собственной подписи или кросс-проверка подписей, можно попробовать удалить всё остальное от этого разработчика или попытаться обойти эту проверку в коде приложения.

Smali: количество регистров

- Зачастую можно расширить количество регистров в методе путём редактирования директивы .locals или .registers в его начале
- Однако, следует помнить, что не в каждую инструкцию можно передать регистры с номерами выше v15, и может получиться так, что при добавлении нового регистра последний регистр параметров выйдет за границу этого диапазона – в этом случае могут быть проблемы
- Поэтому лучше не добавлять сложный код в существующий метод, а вместо этого создать новый метод и в существующий метод добавить его вызов (при этом меньше риск ошибки)

Smali: static – это удобно

- Хотя в регистре p0 обычно хранится ссылка на инстанс класса, но иногда она теряется, так как регистр p0 может быть использован как регистр общего назначения, если к нему нет обращений далее в коде метода
- Вызывать static методы в SMALI удобнее чем другие, так как не требуется иметь ссылку на инстанс класса
- Поэтому при патчинге лучше добавлять именно static методы – их можно вызвать откуда угодно и в любой момент

Smali: личная библиотека методов

В процессе патчинга постепенно формируется библиотека методов которые используются каждый раз (например, если надо вывести в лог значение регистра, или его значение и backtrace вызова) – их можно сделать статическими и вынести в отдельный класс, и в дальнейшем просто копировать этот класс в проект и вызывать его методы.

Smali: static <-> {direct,virtual}

- Набор регистров при вызове метода не зависит от его invoke-kind, и поэтому можно заменить вызов invoke-direct или invoke-virtual вызовом invoke-static если static метод принимает первым параметром инстанс этого класса (direct и virtual методы также принимают инстанс первым параметром)
- Эта особенность позволяет выполнять статический перехват методов путём автоматического патчинга .smali файлов при помощи grep и perl/sed, что может быть полезным к примеру для обмана самопроверки подписи APK

Jadx: проверка корректности кода

После редактирования кода можно декомпилировать его .smali файл через jadx-gui и посмотреть, соответствуют ли изменения декомпилированного кода ожиданиям, если нет – нужно проверить прежде всего правильно ли регистры используются в коде, и не были ли случайно перезаписаны регистры pX (возможно, через схему именования vX).

logcat: наблюдение за ошибками

- В процессе установки APK происходит верификация классов, в ходе которой в logcat выводятся сообщения о найденных ошибках
- При этом, даже с ошибками верификации APK всё же установится, и даже будет работать (но только до первого обращения к классу с ошибкой)
- Но при падении будет просто сообщение о том, что verifier rejected class, без подробностей какой именно метод в классе не прошёл проверку и какая именно проверка не прошла
- Всегда полезно смотреть в logcat во время установки модифицированного APK

apktool: ошибки в ресурсах

- Иногда apktool не может корректно распаковать или запаковать обратно распакованные ресурсы, поэтому если их модификация не требуется – можно обойтись без неё (попутно сэкономив время), передав apktool параметр --no-res: apktool d --no-res file.apk
- Но при исследовании APK всё равно полезно иметь распакованные ресурсы, чтобы grep’ать по всему проекту их идентификаторы, поэтому лучше держать два проекта Apktool – с распакованными ресурсами для исследования и с упакованными ресурсами для модификации

Трюки: изменение значения в методе

Даже если требуемое изменение является очень простым – может быть полезным вынести это в отдельный метод с внятным именем, так как спустя время (или если исходники патча будут потеряны) будет сложно вспоминать, почему этот параметр надо было изменить, а имя метода может напомнить о причине изменения.

Трюки: создание обёртки

- APK перед релизом часто прогоняют через ProGuard, который обфусцирует имена полей и методов, и на следующей версии APK под теми же именами могут оказаться совсем другие поля/методы
- Поэтому, полезно вместо прямого использования значения поля из своего кода создать геттер/сеттер для этого поля с внятным именем, и в своём коде использовать этот геттер/сеттер
- Аналогично и с методами – вместо прямого вызова метода по его обфусцированному имени из своего кода полезно создать метод-обёртку с внятным именем, и из своего кода вызывать уже обёртку

Трюки: методы жизненного цикла

- Иногда может быть удобным добавить свой код в жизненный цикл Activity или фрагмента вместо добавления кнопки в интерфейс приложения (так как добавить кнопку сложнее)
- Такой код будет вызываться автоматически при соответствующем событии, поэтому должен выполняться быстро

Трюки: Android Studio для патчинга

- Код Smali можно писать и руками, но есть способ получить Smali из Java автоматически при помощи пары трюков
- Jadx позволяет экспортировать результат декомпиляции в виде проекта Gradle: jadx --export-gradle --output-dir out file.apk
- Этот проект затем можно импортировать в Android Studio и писать код в нём пользуясь всеми преимуществами IDE
- Также, можно использовать Android Studio для удобного редактирования ресурсов
- Собрать APK из такого проекта скорее всего не выйдет, будет много ошибок из-за несовершенства декомпиляции

java2smali: автогенерация Smali-кода

- Это плагин к Android Studio, позволяющий компилировать код Java/Kotlin в код Smali
- При этом, требуется чтобы исходный проект собирался
- Можно создать пустой проект, в нём создать минимальную иерархию классов, полей и методов (к которой обращается новый код) как в исходном проекте (при этом классы, поля и методы должны именоваться так же как и в исходном проекте, соблюдая область видимости (private и protected/public), но не должны содержать ничего), после чего скопировать свой код из исходного проекта в новый (если всё сделано правильно – никаких ошибок в коде не будет) и уже в пустом проекте компилировать свой код в Smali при помощи java2smali

Ссылки

- https://ibotpeaches.github.io/Apktool/ (https://github.com/iBotPeaches/Apktool)
- https://nightly.link/skylot/jadx/workflows/build-artifacts/master (https://github.com/skylot/jadx)
- https://plugins.jetbrains.com/plugin/7385-java2smali (https://github.com/ollide/intellij-java2smali)
- https://developer.android.com/studio/command-line/apksigner (https://android.googlesource.com/platform/tools/apksig/)
- https://bitbucket.org/JesusFreke/smali/downloads/ (https://github.com/JesusFreke/smali)
- https://developer.android.com/studio
- http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html – полная таблица опкодов smali и их мнемоник с примерами, очень полезный документ
- https://github.com/JesusFreke/smali/wiki
- Русская шпаргалка по Smali
- https://play.google.com/store/apps/details?id=com.appsisle.developerassistant