

# 目录

前言	1.1
概述	1.2
安卓背景知识	1.3
安卓基本框架	1.3.1
apk编译打包流程	1.3.2
相关知识	1.3.3
apk文件	1.3.3.1
dex文件	1.3.3.2
安卓虚拟机	1.3.3.3
Dalvik	1.3.3.3.1
ART	1.3.3.3.2
smali	1.3.3.4
安卓正向安全	1.4
加固手段发展历史	1.4.1
VMP	1.4.1.1
防静态分析	1.4.2
加壳	1.4.2.1
为何要加壳	1.4.2.1.1
常见加固服务提供商	1.4.2.1.2
代码混淆	1.4.2.2
ProGuard	1.4.2.2.1
Obfuscator-LLVM	1.4.2.2.2
防动态调试	1.4.3
安卓逆向破解	1.5
系列子教程	1.6
附录	1.7
参考资料	1.7.1

# 安卓应用的安全和破解

- 最新版本: v4.0.1
- 更新时间: 20230914

## 简介

总结安卓应用的正向安装和逆向破解；先是安卓正向和逆向的概述；然后是安卓背景知识介绍，包括安卓基本框架、apk编译打包流程、以及相关知识，包括apk文件、dex文件、安卓虚拟机的Dalvik和ART，以及Smali。接着介绍安装正向的加固手段发展历史，包括VMP；以及防静态分析，包括加壳加固，为何要加壳，以及常见的加固服务提供商，和代码混淆的ProGuard和Obfuscator-LLVM，以及防动态调试。然后是安卓逆向破解，以及系列子教程。

## 源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

### HonKit源码

- [crifan/android\\_app\\_security\\_crack: 安卓应用的安全和破解](#)

### 如何使用此HonKit源码去生成发布为电子书

详见：[crifan/honkit\\_template: demo how to use crifan honkit template and demo](#)

### 在线浏览

- [安卓应用的安全和破解 book.crifan.org](#)
- [安卓应用的安全和破解 crifan.github.io](#)

### 离线下载阅读

- [安卓应用的安全和破解 PDF](#)
- [安卓应用的安全和破解 ePUB](#)
- [安卓应用的安全和破解 Mobi](#)

## 版权和用途说明

此电子书教程的全部内容，如无特别说明，均为本人原创。其中部分内容参考自网络，均已备注了出处。如发现有侵权，请通过邮箱联系我 admin 艾特 crifan.com，我会尽快删除。谢谢合作。

各种技术类教程，仅作为学习和研究使用。请勿用于任何非法用途。如有非法用途，均与本人无关。

## 鸣谢

感谢我的老婆陈雪的包容理解和悉心照料，才使得我 crifan 有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

## 其他

## 作者的其他电子书

本人 crifan 还写了其他 150+ 本电子书教程，感兴趣可移步至：

[crifan/crifan\\_ebook\\_readme: Crifan的电子书的使用说明](#)

## 关于作者

关于作者更多介绍，详见：

[关于CrifanLi李茂 – 在路上](#)

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新： 2023-09-14 21:02:01

# 概述

## 不准把技术用于非法用途

警告△：相关的安卓逆向破解技术仅限于技术研究使用，不准用于非法目的，否则后果自负。

先后接触过一些安卓app的逆向破解，现去整理出相关安卓app的安全和逆向相关知识。

关于安卓，大体可以分为正向的安全和逆向的破解这两大分支：

- 安卓
  - 正向=安全
    - 防静态分析
      - 防砸壳：加壳=加固
        - 加固手段的发展历史
      - 防反编译
        - 代码混淆
          - Proguard
          - Ollvm
    - 防动态调试
      - 防调试和运行
        - 反调试
        - Root检测
          - 检测 Xposed 、 Frida 等
      - 防抓包=反抓包
        - ssl pinning = 证书绑定
  - 逆向=破解
    - 静态分析
      - 按文件格式类型分
        - 针对 apk
          - 查看apk信息
            - aapt
          - 解包工具：输出 dex 、 so 、 smali
            - apktool
        - 反编译工具
          - 直接apk转java
            - jadx
            - JEB
            - GDA
      - 针对 dex
        - 各种导出dex的工具=砸壳工具=脱壳工具
          - FDex2
          - DumpDex
          - DexExtractor
        - 各种反编译dex的工具
          - dex转jar
            - dex2jar
          - dex转smali
            - baksmali
          - dex直接转java
            - jadx
            - GDA
      - 针对 jar

- jar转java=各种反编译工具
  - Procyon
  - CFR
  - JD-GUI
- 针对 so 库文件=（往往是ARM64架构）的ELF文件
  - 导出静态资源
    - readelf
    - objdump
    - razbin2
  - 反编译代码逻辑
    - IDA
    - Hopper
    - Radare2
    - Ghidra
- 涉及子领域
  - 反代码混淆
- 动态调试
  - （绕过限制去）抓包
    - 绕过证书绑定的Exposed插件或Frida的脚本
  - 反root检测
  - 反反调试
  - 各种动态调试手段
    - 调试smali: AS+smalidea插件
      - app进程可调试
      - Magisk插件: MagiskHide Props Config
    - 调试Frida: Frida
      - Frida环境搭建
      - Magisk插件: MagiskFrida
    - 调试lldb: LLDB
    - 调试Xposed插件
      - Xposed
      - EdXposed
      - LSPosed
    - 模拟代码执行
      - Unidbg
      - Unicorn
    - 辅助调试工具
      - adb
      - DDMS
- 输出
  - 重新打包apk
    - apktool
    - keytool
    - jarsigner 或 apksigner
    - zipalign
  - 用代码重写逻辑
    - Python代码
    - C/C++代码
  - 改机定制ROM
    - AOSP源码 编译
    - JNI 开发
    - NDK 开发



# 安卓背景知识

在介绍安卓的安全技术和破解技术之前，需要先去了解相关的背景知识。

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2019-05-02 14:54:26

# 安卓基本框架

为了更好的理解，安卓破解相关知识和工具，先要了解安卓的基本框架，以及每一层中都是什么东西：

## 安卓的基本架构

- 内核层：
  - 支持多进程和多线程的Linux内核
  - 每个应用程序都有自己的Linux ID，并在单独的进程中运行
  - 具有相同ID的两个应用程序可以彼此交换数据。
- 系统运行层
  - 主要包括一些开源类库以及Android运行时环境
    - 其中虚拟机（Dalvik、ART）中运行的应用程序格式为dex的二进制文件
- 应用框架层
  - 具有Java接口的应用程序框架
    - 主要组成
      - Java层的Android SDK
      - Native层的Android NDK
- 应用层
  - 预安装一些核心应用程序

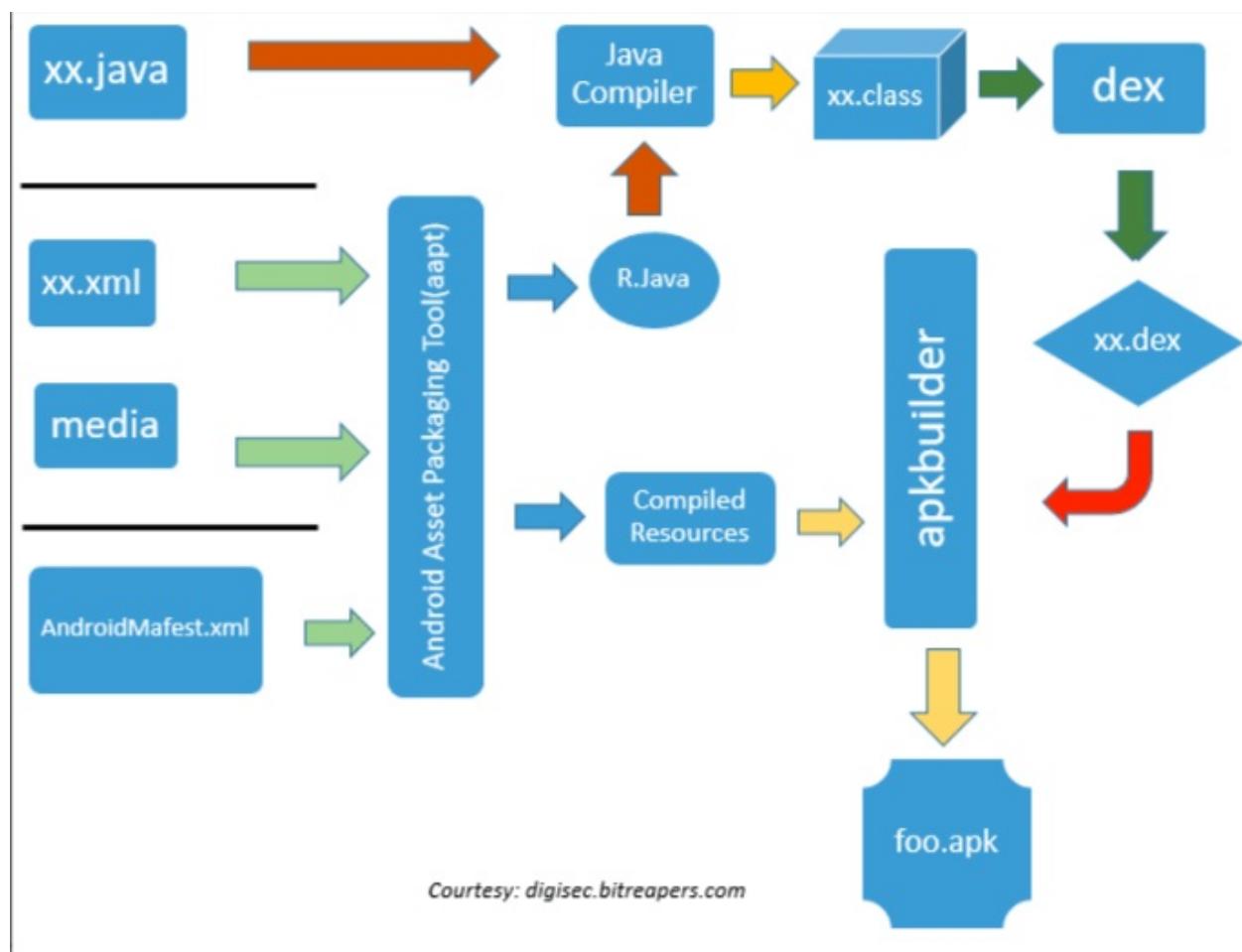
后续的很多破解工具，则是针对Dalvik虚拟机和dex文件去破解的。

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2023-08-24 22:24:22

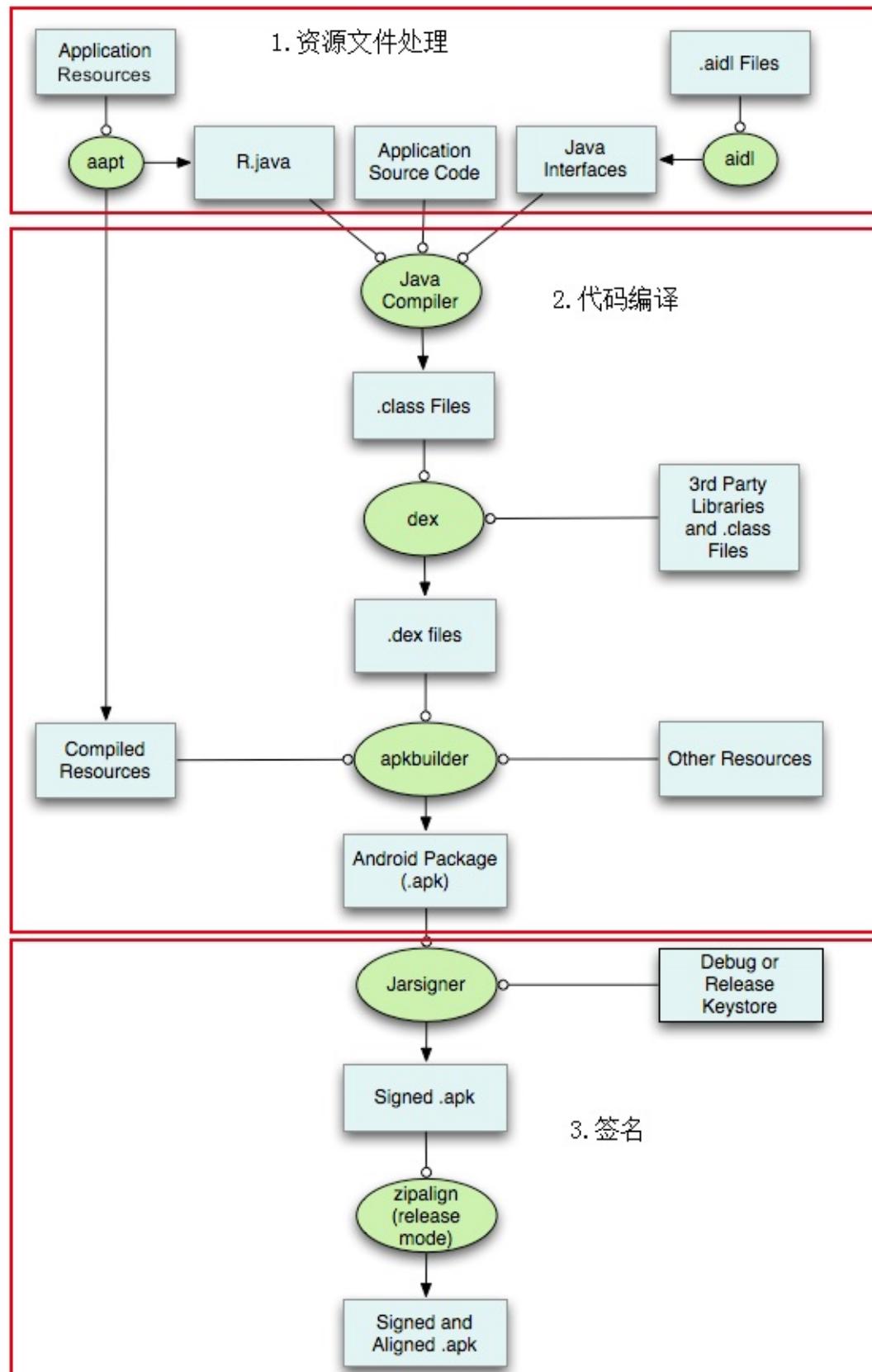
## apk编译打包流程

### APK打包流程

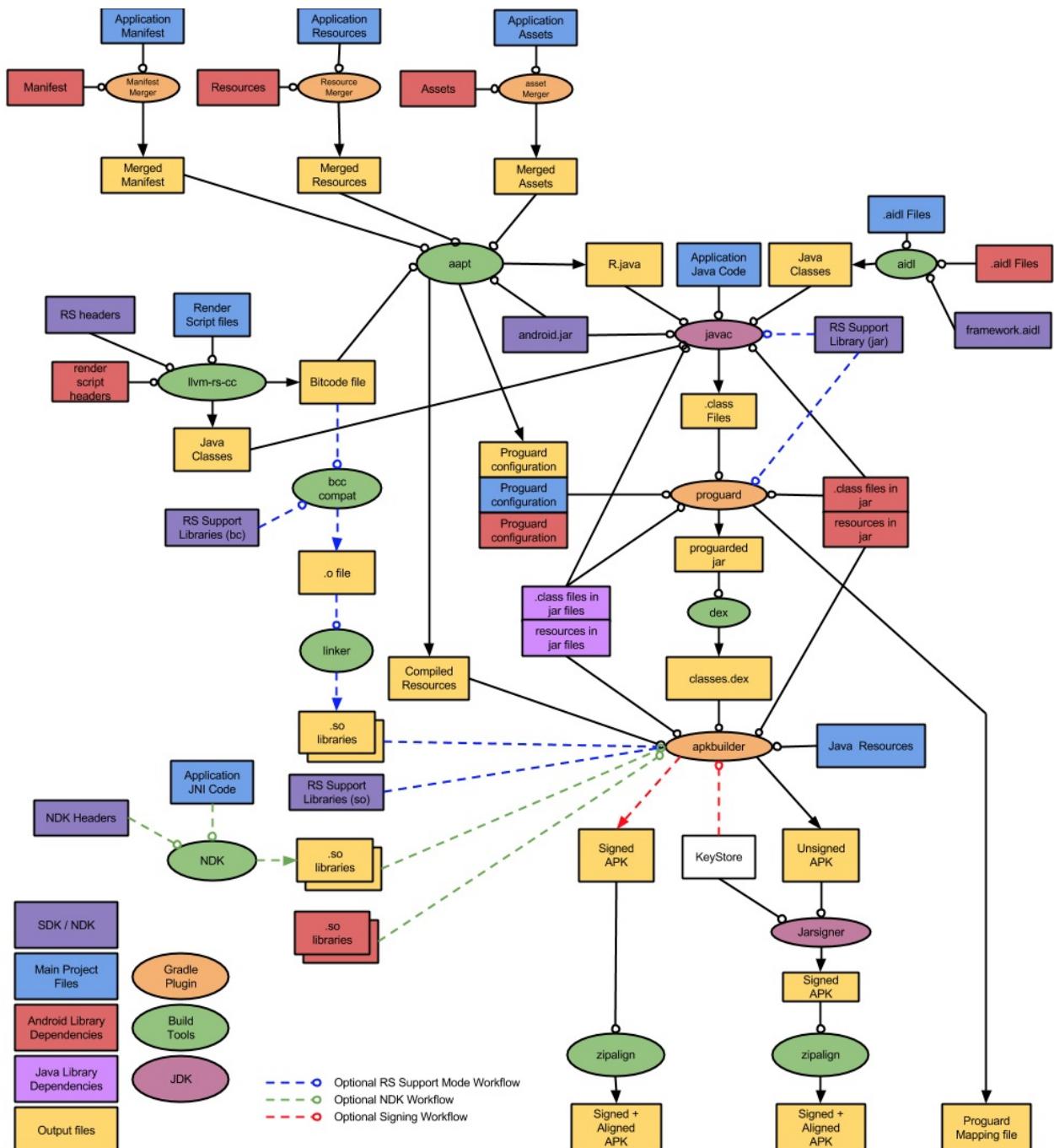
apk 打包流程图= apk 产生过程:



和



和安卓编译流程：



具体的解释是：

- 资源处理
  - 这一过程中主要
    - 使用appt工具进行资源文件的处理
      - 分析AndroidManifest.xml中的资源文件
      - 生成R.java和resources.arsc文件
    - aidl工具负责处理aidl文件
      - 生成对应的java接口文件
- 代码编译
  - 将上一过程中产生的R.java、java接口文件以及工程源代码一起通过Java Compiler编译成.class文件，打成Jar包
    - 这部分可以加入代码混淆）
      - 比如用 ProGuard
  - 然后与第三方库的Jar包一起通过dx工具转换成.dex文件
    - 注：如果apk的方法数超过了65535，会生成多个dex文件

- 反编译的话需要对这多个dex文件均进行转换Jar包处理
- 通过apkbuilder工具将aapt生成的resources.arsc、classes.dex（可能多个）、其他的资源一块打包生成未经签名的apk文件。
- 添加签名
  - 通过Jarsigner对生成的未签名的apk进行签名。
  - 再通过zipalign对签名后的apk进行对其处理，使apk中所有资源文件距离文件起始偏移为4字节的整数倍，从而在通过内存映射访问apk文件时会更快。

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2023-08-24 22:44:04

## 相关知识

crifan.org, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新: 2019-05-02 11:00:12

# apk文件

- apk = android Application Package = APK
  - apk文件是什么：是安卓app的安装文件
  - 本质：(apk文件其实就是一个) zip压缩包
    - 意味着
      - 可以用解压缩工具把apk当做zip文件一样去解压
        - 解压后，得到一堆安卓相关文件
      - 可以在 apktool 等工具破解和修改了安卓文件后，再重新用压缩文件工具或 apktool 等工具，重新打包为 apk 文件

## apk内容结构

内容入口	含义解释
AndroidManifest.xml	二进制xml文件，提供设备运行应用程序所需的各种信息
classes.dex	以dex格式编译的应用程序代码
resources.arsc	包含预编译应用程序资源的二进制XML文件
res/	此文件夹中包含未编译到resources.arsc文件中的资源
assets/	此文件夹包含应用程序的原始资产，由AssetManager提供对这些资产文件的访问
META-INF/	它包含MANIFEST.MF文件，该文件存储有关JAR内容的元数据。APK的签名也存储在此文件夹中
lib/	该文件夹包含已编译的代码，例如本地代码库

## apk产生的大概过程

简述安卓apk的产生过程：

- 概述：.java -> .class -> .dex -> .apk
- 详解
  - java源代码
    - java编译器 编译
  - class文件
    - dx 工具转换和打包压缩
      - 加上 第三方的，其他的库文件
  - dex文件
    - apkbuilder打包
      - 加上 其他资源文件resources.arsc，其他库等
  - (未签名的) apk文件
    - jarsigner去签名 + zipalign去处理
  - (已签名的) apk文件
    - 可以用于发布和上架各种安卓应用市场
      - 供普通用户下载安装试用



# dex文件

## 什么是dex文件

简答：

- dex = Dalvik Executable format = dex文件 = dex格式
  - dex 之于 Android，类似于 class 之于 Java
    - 注：java的class文件内部是Java的字节码(Java bytecode)
    - dex = Dalvik Executable
      - 相关：dex文件 = dex字节码
      - dex 反汇编后是：smali代码
        - 即：Android（虚拟机中的dex文件）反汇编（后的）代码：smali
  - 文档
    - dex格式
      - Dalvik 可执行文件格式 | Android 开源项目 | Android Open Source Project
        - <https://source.android.com/devices/tech/dalvik/dex-format>
      - 字节码
        - Dalvik 字节码 | Android 开源项目 | Android Open Source Project
          - <https://source.android.com/devices/tech/dalvik/dalvik-bytecode>

详解：

安卓系统中，用 Dalvik虚拟机 ( DVM = Dalvik Virtual Machine )去把 java 源码编译为 dex 可执行文件(Dalvik Executable)。

而dex文件中保存的就是：编译后了的安卓程序代码文件

## Dex文件内部格式

1. File Header
2. String Table
3. Class List
4. Field Table
5. Method Table
6. Class Definition Table
7. Field List
8. Method List
9. Code Header
10. Local Variable List

## 相关工具

Android自带 dexdump：用来反编译 dex 文件

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook 最后更新：2021-07-18 09:55:45

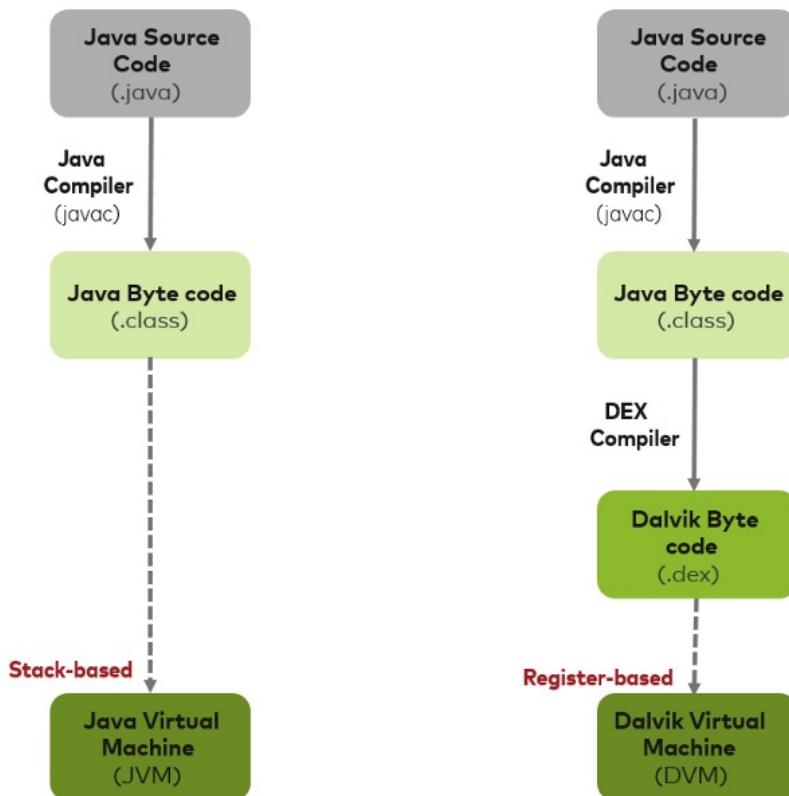
# 安卓虚拟机

- 历史背景
  - Android
    - 代码语言: Java
      - Java 的虚拟机是: JVM
    - Android: 出于性能考虑, 没用 JVM, 用了自己的虚拟机 VM
  - 安卓虚拟机 = Android虚拟机 = Android VM
    - 旧: Android < 5.0 : Dalvik
    - 新: Android >= 5.0 : ART
  - 资料
    - 官网
      - Android Runtime (ART) 和 Dalvik | Android 开源项目
      - <https://source.android.com/devices/tech/dalvik>

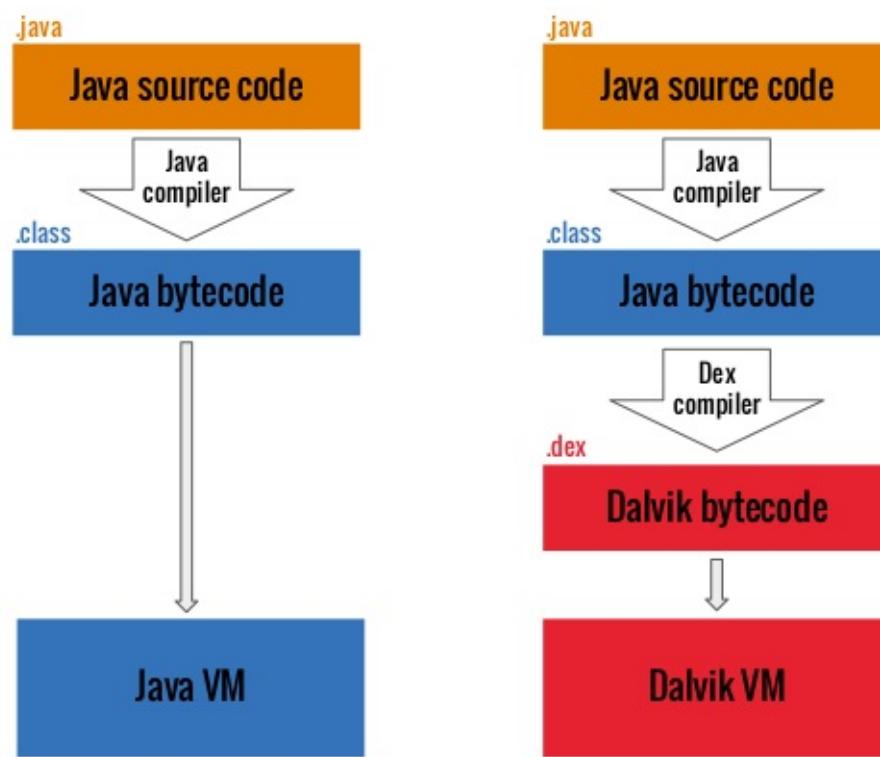
crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2023-08-24 23:10:57

# Dalvik

- Dalvik = Dalvik VM = DVM
  - 概述
    - Dalvik 是 google 专门为 Android 操作系统设计的一个虚拟机，经过深度的优化。虽然Android上的程序是使用java来开发的，但是Dalvik和标准的java虚拟机JVM还是两回事
  - JVM VS DVM
    - 对比： JVM vs DVM
      - 执行的字节码文件不一样
        - JVM 执行的是 .class 文件= Java Bytecode
        - DVM 执行的是 .dex 文件= Dalvik Bytecode
      - 文件类型变化
        - JVM : .java → .class → .jar
        - DVM : .java → .class → .dex → .apk
      - 运行环境不同
        - DVM : 允许运行多个虚拟机实例
          - 每一个应用启动都运行一个单独的虚拟机，并且运行在一个独立的进程中
        - JVM : 只能运行一个实例
          - 也就是所有应用都运行在同一个 JVM 中
      - 编译流程对比



## JVM vs DVM



- JVM:
  - 基础：基于栈帧 Stack-based
  - 文件格式：java字节码 = java bytecode
  - 效率：相对低
- DVM:
  - 基础：基于寄存器 Register-based
  - 文件格式：dex
  - 效率：DVM 效率比 JVM 高
    - 速度更快，占用空间更少

# ART

- ART = Android RunTime
  - 是什么: Android的新一代的VM虚拟机
    - 在 Dalvik 的基础上做了一些优化, 提高了运行效率
    - 用于替代旧的: Dalvik
  - 特点
    - 预先编译 AOT
    - 垃圾回收方面的优化
    - 开发和调试方面的优化
      - 支持采样分析器
      - 支持更多调试功能
      - 优化了异常和崩溃报告中的诊断详细信息
  - 对比: Dalvik VS ART
    - 效率对比
      - Dalvik : 应用每次运行的时候, 字节码 都需要通过 即时编译器 = JIT = Just In Time 转换为 机器码
        - 这会拖慢应用的运行效率
      - ART : 应用在第一次安装的时候, 字节码 就会预先编译成 机器码
        - 使其成为真正的本地应用
        - 这个过程叫做 预编译 = AOT = Ahead-Of-Time
          - 所用工具是: dex2oat
          - 这样的话, 应用的启动(首次)和执行都会变得更加快速
        - 文件类型变化: .java → .class → .dex → .oat
          - .oat : optimized android runtime machine code
    - 效果对比
      - Dalvik是运行时解释dex文件
        - 安装比较快
        - 开启应用比较慢
        - 应用占用空间小
      - ART是安装时字节码预编译成机器码存储在本地, 执行的时候直接就可以运行的
        - 安装慢
        - 开启应用快
        - 占用空间大
    - 类比
      - 就像骑自行车
        - Dalvik ~= 折叠自行车: 每次骑之前, 都要先组装/展开才能骑
          - 空间占用小, 但启动慢
        - ART ~= 组装好的自行车: 每次直接骑着就走了
          - 空间占用大, 但启动快

## smali

- Smali
  - 是什么：一种 汇编语法 / 汇编文件
    - 一种语法： Smali语法
      - 来源：是 Dalvik 的 VM 的字节码，即 dex 文件中的 bytecode = 二进制数据，反汇编后得到的：Smali代码
      - 语法：一种宽松式的Jasmin/dedexer语法
        - 它实现了 .dex 格式所有功能（注解，调试信息，线路信息等）
    - 对应文件叫： Smali文件
  - 举例
    - java源码： int x = 42
    - Dalvik编译后的，dex中 二进制数据 = bytecode = 字节码： 13 00 2A 00
      - 二进制，人类很难读懂
    - 用 baksmlai 反汇编后的，smali代码： const/16 v0, 42
      - smali代码，人类基本可读
  - 学习Smali的用途
    - 分析Apk：静态分析，不够
      - 需要动态分析，涉及Smali
    - 修改Apk逻辑：修改Smali代码，重新编译打包Apk
      - Android逆向基础：掌握Smali
        - 能阅读 smali 代码对进行 android 逆向十分重要
  - 官网
    - JesusFreke/smali: smali/baksmali
      - <https://github.com/JesusFreke/smali>
- smali / baksmali
  - GitHub
    - [JesusFreke/smali: smali/baksmali](#)
      - smali/baksmali is an assembler/disassembler for the dex format used by dalvik, Android's Java VM implementation
  - 针对dex
    - smali : assembler = 汇编器
      - smali语言 = 汇编语言
    - baksmali : disassembler = 反汇编器
      - 安卓系统里的Java虚拟机（Dalvik）所使用的一种 .dex 格式文件的反汇编器

## Smali基本语法

- 官网文档
  - TypesMethodsAndFields · JesusFreke/smali Wiki
    - <https://github.com/JesusFreke/smali/wiki/TypesMethodsAndFields>

## 数据类型 Types

Smali	Java	备注
v	void	只能用于返回值类型
Z	boolean	
B	byte	
S	short	

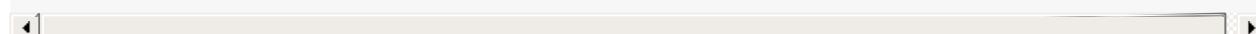
C	char	
I	int	
J	long	
F	float	
D	double	
Lpackage/name;	对象类型	L 表示这是一个对象类型， package/name 表示该对象所在的包， ; 表示对象名称的结束 Lpackage/name/ObjectName; 相当于 java 中的 package.name.ObjectName;
[类型	数组	[I 表示一个 int 型 数组， [Ljava/lang/String 表示一个 String 的对象 数组`

## 寄存器

- 官网文档
  - Registers · JesusFreke/smali Wiki
    - <https://github.com/JesusFreke/smali/wiki/Registers>
- Java中变量都是存放在内存中的
  - Android为了提高性能，变量都是存放在寄存器中的
    - 寄存器为32位，可以支持任何类型
- 寄存器
  - 类型
    - 本地寄存器
      - 用v开头数字结尾的符号来表示
      - 举例
        - v0, v1, v2
    - 参数寄存器
      - 用p开头数字结尾的符号来表示
      - 举例
        - p0,p1,p2
  - 注意
    - 在非static 方法中， p0代指this， p1为方法的第一个参数
    - 在 static 方法中， p0为方法的第一个参数
  - 说明
    - 指定有多少寄存器是可用
      - .registers : 指定了方法中寄存器的总数
      - .locals : 表明了方法中非参寄存器的总数，出现在方法中的第一行

## Smali代码示例

```
const/4 v0, 0x1 //把值0x1存到v0本地寄存器
input boolean v0, p0, Lcom/aaa/>IsRegistered:Z //把v0中的值赋给com.aaa.IsRegistered, p0代表this, 相当于this.Isregistered=true
```



## 成员变量? Fields

- 格式
 

```
.field public/private [static][final] varName: 类型
```
- 指令

- 获取指令
  - ige, sge, ige-boolean, sge-boolean, ige-object, sge-object
- 操作指令
  - ige, sge, ige-boolean, sge-boolean, ige-object, sge-object
  - array的操作是aget和aput

## Smali代码示例

```
sget object v0, Lcom/aaa; > ID Ljava/lang/String;
```

- 获取ID这个String类型的成员变量并放到v0这个寄存器中

```
iget object v0, Lcom/aaa; > view Lcom/aaa/view;
```

- iget-object比sget-object多一个参数p0, 这个参数代表变量所在类的实例。这里p0就是this

```
const/4 v3, 0x0
sput object v3, Lcom/aaa; > timer Lcom/aaa/timer;
```

- 相当于java代码

```
this.timer = null;
```

```
.local v0, args Landroid/os/Message;
const/4 v1, 0x12
iget v1, v0, Landroid/os/Message; > what:I
```

- 相当于java代码

```
args.what = 18;
```

- 其中args为Message的实例

## 方法/函数 Methods

- 函数定义格式

```
method public/private [static][final] methodName() 美型
    end method
```

- 函数类型
  - direct method = private方法
  - virtual method = 其余的方法
- 函数调用
  - 格式

```
invoke 指令类型 {参数1, 参数2, ...}, 类名; 方法名
```

- 包含
  - invoke-direct
  - invoke-virtual
  - invoke-static
  - invoke-super
  - invoke-interface
- 函数返回结果

- 要用指令move-result或move-result-object来保存函数返回的结果

Smali代码示例：

```
.method private ifRegistered()
    .locals 2           // 本地寄存器的个数
    .prologue
    const/4 v0, 0x1    // v0赋值为1
    if eqz v0, cond_0  // 判断v0是否等于0，等于0则跳到cond_0执行
    const/4 v1, 0x1    // 符合条件分支
    goto_0              // 标签
    return v1           // 返回v1的值
    cond_0              // 标签
    const/4 v1, 0x0    // cond_0分支
    goto_0              // 跳到goto_0执行
.end method
```

```
const string v0, "NDKLIB"
invoke static {v0}, Ljava/lang/System;::loadLibrary(Ljava/lang/String;)V
```

- 相当于java代码

```
System.loadLibrary("NDKLIB")
```

```
const string v0, "Eric"
invoke static {v0}, Lcom/pbi;::t(Ljava/lang/String;)Ljava/lang/String;
move result object v2
```

- 表示将方法 t 返回的 String 对象 保存到 v2 中

# 安卓正向安全

此处主要介绍：

- 安卓的 正向=安全
  - 防静态分析
    - 防砸壳：加固
      - 加固手段的发展历史
      - 加壳
    - 防反编译
      - 代码混淆
        - Proguard
        - Ollvm
  - 防动态调试
    - 防调试和运行
      - 反调试
      - Root检测
        - 检测 Xposed 、 Frida 等
    - 防抓包=反抓包
      - ssl pinning = 证书绑定

## 如何防护安卓的安全

- 首要的：加强业务逻辑
  - api接口通讯
    - 全部接口都实现https
      - 且做 证书绑定 = ssl pinning
- 其次：加强安全防破解技术
  - 代码
    - 代码混淆
      - 首选： Obfuscator-LLVM = ollvm
      - 其次： ProGuard
  - 其他防护
    - VMP
      - 给 dex (中的核心逻辑) 做VMP
      - 给 so 库 (中的核心逻辑) 做VMP
    - 加壳
      - 用第三方加壳服务或自己实现
        - 第三方加壳服务商
          - 腾讯乐固legu
          - 360加固保
          - 网易易盾
          - 等

# 加固手段发展历史

- 加固 = 应用加固
  - 含义：给原有的安卓应用，增加了保护手段，增强安全性，防止被轻易破解
  - 目的：
    - 比如
      - 使得别人即使反编译安卓应用得到了的jar包，也看不到原始的项目的源码
- 加固 的英文说法
  - 自己暂时用： `harden`
  - 被加固了的（apk）就叫： `hardened`
- 主要包括
  - 代码混淆
    - `java` 层
    - `so` 层= `native` 层
  - 加壳
    - 注：壳是一段保护软件不被非法修改或反编译的程序

下面整理安卓的加固技术的历史发展。

## 概述

常见Android的 APK 的五代加固技术比较：

**市面上最常見的 Android (APK) 五代加固技术比較**

阶段	名称	开发过程	核心逻辑 (执行过程)	不足	优势	破解状况	实例
第一代 动态 加载	程序切分成加载 (Loader) 与关键逻辑 (Payload) 两部分，并分别打包。	运行时加载部分 (Loader) 会先运行，释放出关键逻辑 (Payload)，然后java的动态加载技术进行加载，并转交控制权。	1. Payload部分必须解压及释放在文件系统 (直接载入)。 2. 通过自定义虚拟机，截获关键函数，在加载Payload后把解密后的内容复制。	加密Payload保存	目前基本上已经被破解，部分反编译工具已经集成修复功能。	早期版本的爱加密	
第二代 内存 不落 地加 载	加载Loader，初始化StubApplication，解密和加载Application。初始化后Application用原始Application替代StubApplication，最后正常加载其他组件。	1. 栈拦截IO相关的函数 (如read, write)。在这些函数中提供透明加解密； 2. 直接调用虚拟机提供的函数进行不落地的加载。	1. 在启动过程中需要处理大量解密操作，容易造成黑屏或死机。 2. Payload加载后，在内存中是连接，内存dump即可看到解密。	当前市面上最为常见，通常作为一项基础性的免费服务向用户提供。	已经出现专业人士自行研究的手工脱壳方法，但尚未出现自动脱壳工具，破解难度仍然比较大。	市面上流行的大多数在线加固服务，如腾讯乐固，360加固，百度加固，阿里聚安全，网易易盾等。	
第三代 指令 抽取	首先，将保护级别降到了函数级别。然后将原始DEX内的函数 (Code item) 清除、单独移除到一个文件中。运行阶段将函数内容重新恢复到对应的函数体。	1. 加载之后恢复函数内容到DEX壳所在的内存区域。 2. 加载之后将函数内容恢复到虚拟机内部的结构体上。这个结构体上有一个指针指向函数内容 (Codeitem)。可以通过修改这个指针修改对应的函数内容。 3. 读取虚拟机内与查找执行代码相关的函数，返回函数内存。	1. 指令分离方案需要虚拟机的JIT性能优化冲突，达不到性能最佳的性能。 2. 依然使用Java虚拟机进行函数内容执行，无法对抽象类虚拟机。 3. 指令抽离技术使用了大量的虚拟机内部结构与未被文档的特性。再加上Android复杂的厂商定制，带来大量的兼容性问题。	在对自定义虚拟机记录函数执行时函数的内容 (Codeitem)。遍历所有所有的函数，从而获得全部抽离的内容，最终组装成完整的DEX文件。可通过自动化完成整个过程。	部分被破解，已经出现专业人士自行研究的手工破解方法，但是迄今为止尚未出现自动脱壳工具。	1. 现在的免费版“爱加密”。 2. 柳柳安全免费版。	
第四代 指令 转换	1. DEX文件内的函数被标记为native，内容被抽离并转换成一个符合JNI要求的动态库。动态库内通过JNI和Android系统进行交互。 2. DEX文件内的函数被标记为native，内容被抽离并转换成自定义的指令格式。该格式使用自定义接收器执行，和A一样需要使用JNI和Android系统进行调用。		1. 不论使用指令转换/ARM加固的A方案或者B方案，其必须通过虚拟机提供的JNI接口与虚拟机进行交互。攻击者可以直接将指令转换/ARM加固的方案当作黑盒，通过自定义的JNICALL对象，对函数内部进行探测、记录和分析，进而得到完整DEX程序。 2. 第四代APM加固技术一般配合第三代加固技术使用，所以第三代的所有兼容性问题，指令转换/ARM加固也存在。	部分被破解，已经出现专业人士自行研究的手工破解方法，但是迄今为止尚未出现自动脱壳工具。	大部分需要定制收费的加密服务（如爱加密，柳柳安全，中国移动加固，以及手机银行自助加固等）		
第五代 虚拟 机器 码保 护	基于第四代方案的A方式 (Java或Kotlin -> C/C++)，基于LLVM编译工具链（同时支持C/C++、Swift、Objective-C）；通过对IR进行指令转换、生成或自定义指令集 (IR->VM)。App内部隔离出独立的执行环境。该核心代码的运行程序在此独立的执行环境里运行。		1. 无法摆脱对JNI的依赖。因此依然存在第四代加固技术的缺陷，存在被记录修复的可能性。 2. 由Java转换成等价的C/C++，会导致体积呈线性增大、性能有所下降。	1. 由Java转C/C++后的代码，由于虚拟机的保护，逆向难度会上升一个数量级。 2. 对于C/C++部分逻辑，只能大大投入时间去破译虚拟机的指令含义。	大多数未被破解	极为少数，需要特殊定制的加固服务，通常用于银行金融机构等关乎国家安全的重点领域。	

- 加固的发展历史
  - 目前加固技术基本都发展到第三代
    - 前2代的加固技术破解难度不大，基本被淘汰
  - 第三代加固技术，由于各加固服务商加固原理大致相同
- 第三代加固技术主要有2种方式：
  - 对源apk整体做一个加固，放到指定位置，运行的时候再解密动态加载
    - 对apk加固的破解，叫做：脱壳=去壳
      - == Dex Method代码动态解密
  - 对so进行加固，在so加载内存的时候进行解密释放
    - 对so的加固的破解，叫做：so库反编译
      - == So代码膨胀混淆

## 不同历史阶段

用人类历史发展的阶段去类比解释如下：

## 原始社会时期

主要方式：代码混淆

## 奴隶社会时期

主要方式：自我校验

## 封建社会时期

主要方式：dex文件变形

## 资本主义社会时期

### 1. Dex保护

#### i. 隐藏dex文件

- 既然dex文件中包含了核心逻辑，那么把dex隐藏，再通过另外的方式加载起来，是不是就能达到保护dex的目的了呢？于是这成为一些第三方加固产品保护应用的方式。
- 他们通过加密甚至压缩（早期是不存在压缩的，只是单纯的加密）方式把dex转换为另外一个文件。而被加固后的apk里面的dex则是那些第三方加固产品用来启动和加载隐藏dex的入口，也就是壳。
- 感觉小花生v3.6.9 和 康美通 v.4.4.0就是这类？
- 总之是看不到dex文件

#### ii. 对dex文件进行变形

- 这里所说的变形，不同于封建社会时期提到的变形。这种办法不隐藏dex，而是让dex保留在外面，但是当破解者去分析这个dex的时候，会发现dex里面的内容是不完整的。

#### iii. 对dex结构进行变形

- 此类方法是比较复杂的，了解dex结构的人应该很清楚，dex结构中包含DexClassDef、ClassDataItem、DexCode，这些是dalvik虚拟机运行一个dex必不可少的部分，特别是DexCode，DexCode包含了虚拟机运行的字节码指令。
- 部分第三方加固产品开始尝试这种方式，他们的保护方案中可能抽取了DexCode中的部分，然后对字节码指令添加nop，或者连ClassDataItem和DexCode一同抽取，或者对上面提到的三个部分都做处理。抽取完之后，还要做修正、修复等工作，总之很烦锁。因为dex运行时有很多关于dex的校验，即使校验通过还有一些偏移问题。
- Dex都被抽取修改后为什么还能运行呢？那是因为在运行之前或者运行之中对这个内存中的dex做修正。修正工作也很复杂，一般选择在运行之前做修正，这样可以减少很大的工作量，甚至可能还需要借助hook来帮忙。

### 2. So保护

#### i. 修改Elf头、节表

- 相关工具：
  - O10 Editor
  - IDA

#### ii. 选择开源加壳工具

- 最常用的：
  - UPX壳
  - 支持arm架构的ELF加固

#### iii. 进程防调试、或增加调试难度

- 调试一个进程首先要ptrace这个进程
- 防止进程被ptrace

## 社会主义时期

### • 之前遗留问题

#### 1. 隐藏dex遗留的问题

- 破解办法：
  - 实现自定义rom
  - 利用Inject原理将目标进程注入,代码进行hook系统函数来达到脱壳的目的
    - FDex2, DumpDex感觉就是用的这个机制?
- 2. Dex结构变形带来的弊端
  - 安卓5.0新增了ART
    - ART可以直接将dex编译为本地指令运行
  - Dex结构变形遗留的问题很明显
    - 兼容性
    - ART模式下的编译问题
- 3. ELF简单修改遗留问题
- 4. UPX方面的劣势
  - 虽然upx是最为so加壳的首选,但是upx代码逻辑复杂,很难达到定制,特别是让它同时支持多种架构
    - -》基于上述原因一些第三方加固产品只是简单的利用upx加壳,并修改一些数据。不过很容易被有upx经验的人识破并脱壳
- 新防护技术
  - llvm混淆
    - 效果非常好,可以实现,即使被反编译后,也很难看懂代码逻辑
  - VMP

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2023-09-03 17:55:53

# VMP

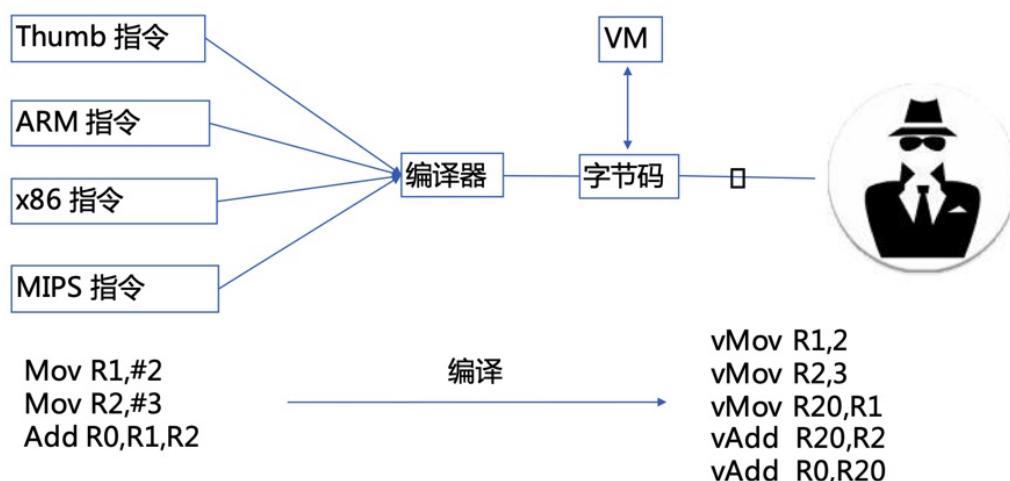
新出的安卓加密技术，叫做： VMP

- VMP

- 名称： VMP = Virtual Machine Protection = 虚拟机保护 = 虚拟软件保护技术 = 代码虚拟化
- 是什么： (安卓) 代码加固领域的技术
- 背景和起源
  - 俄罗斯的著名软件保护软件 VmProtect=虚拟机保护
    - 主页：[VMProtect Software Protection](#)
    - 以此为开端引起了软件保护壳领域的革命，各大软件保护壳都将虚拟机保护这一新颖的技术加入到自己的产品中
- 为什么（要指令虚拟化）？
  - 软件保护壳的发展阶段
    - 第一阶段
      - 当壳完成解密目标代码时，它将不会再次控制程序，被保护程序的明文将在内存中展开。在此之前，壳可以调用一切系统手段来防治黑客的调试与逆向
    - 第二阶段
      - 可以实现分段式的加解密，壳运行完毕后，并不会消失而仍然会在程序运行到某个点时再次启动
    - 第三阶段
      - 其实最简单的解释是，将被保护的指令使用一套自定义的字节码(逻辑上等价)来替换掉程序中原有的指令，而字节码在执行的时候又由程序中的解释器来解释执行，自定义的字节码只有自己的解释器才能识别，也是因为这一点，基于虚拟机的保护相对其他保护而言要更加难分析

- 核心原理

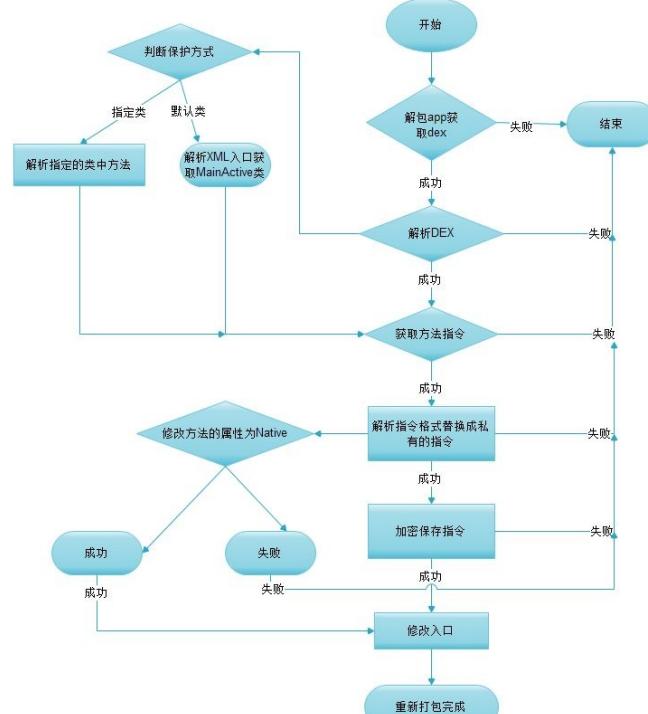
- 代码虚拟化 = 基于 Dalvik 的解释器实现自己定义的指令
  - 说明
    - 将程序代码编译为虚拟机指令即虚拟代码(自己定义的代码集)，通过虚拟CPU解释并执行的一种方式
    - 自定义一套虚拟机指令和对应的解释器，并将标准的指令转换成自己的指令，然后由解释器将自己的指令给对应的解释器
  - 举例
    - x86或arm体系架构的标准汇编指令 (mov、add、pop等)，已变成了自定义加密汇编指令 (xchg、db、dq等)
  - 图解



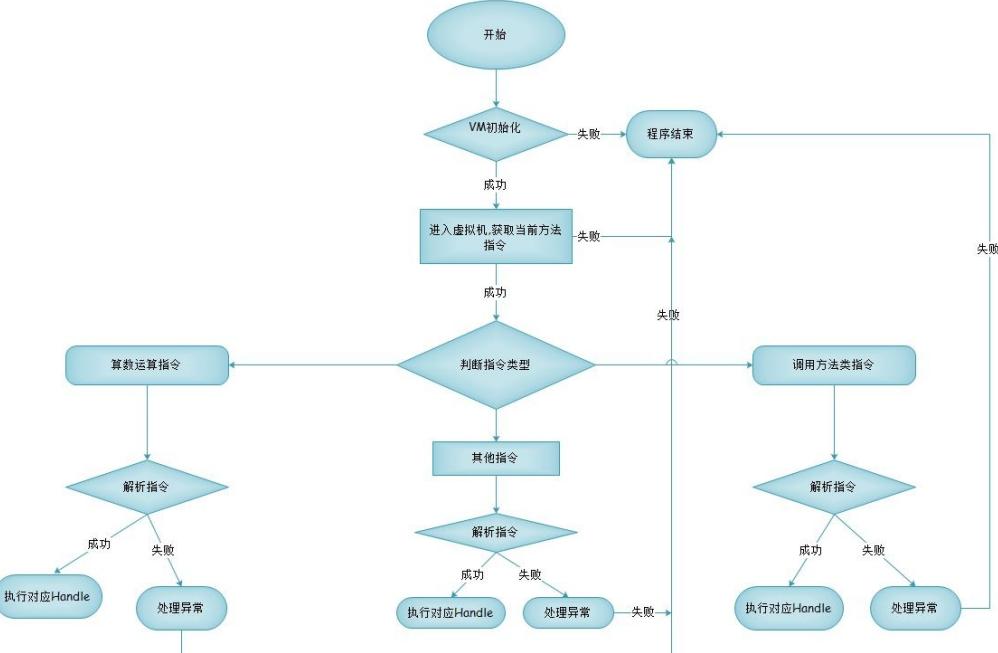
- 运行机制



- 运行流程
  - 加固端



- 解释器



- 缺点
  - 存在一定兼容性问题
  - 会降低代码执行效率

- 应用
  - 说明
    - 由于兼容性和效率等问题，所以VMP一般只用于关键函数
      - 根据保护的内容，可以分
        - DEX 的 VMP
        - SO 的 VMP
    - 多数VMP的实现都是：直接把 `smali` 翻译成 `c` 实现
  - 举例
    - 爱加密
      - 所说的第四代vmp是先提取dex中的虚拟指令集，将dex中提取指令的方法清空，并将方法修改为 native方法；然后通过爱加密自定义指令替换规则，替换提取的指令并保存到其他文件中
    - 通付盾
      - 实现了自定义指令集和自定义虚拟机运行环境的动态代码保护方案
      - 产品
        - 通付盾安全虚拟机 PayegisVM 3.0

- 虚拟机

- 背景：VMP的核心要点是，设计一个虚拟机，实现自定义指令的功能
- 包含几大模块
  - VM 虚拟机核心
  - VM 编译器
    - 如何设计一个编译器？



- 编译器工作流程
  - 1. 反汇编ARM
  - 2. 生成中间代码
  - 3. 处理定位
  - 4. 生成opcode
- VM 链接器
- VM 各种stub
- 想实现一个基于虚拟机的保护壳，涉及内容
  - 随机VCode与Handle的关系映射
  - Handle混淆与乱序
  - 代码变形
  - 重定位

- 资料

- 网上某个开源实现
  - GitHub主页
    - eaglx/VMPROTECT: Obfuscation method using virtual machine.
    - <https://github.com/eaglx/VMPROTECT>

# 防静态分析

此处去介绍安卓的正向安全中的：

- 防静态分析
  - 防砸壳
    - 加壳
  - 防反编译
    - 代码混淆
      - Proguard
      - Ollvm

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2023-08-25 22:33:28

# 加壳

## 加壳 的英文说法

- pack = 加壳
  - 加壳过程叫做: pack = packing
  - 加壳的动作叫做: packer
- 其他叫法: shelling
  - 壳 = shell

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2023-09-03 17:52:45

# 为什么要加壳/加固

- 安卓应用主要基于Java开发
  - 极易被破解
    - 造成影响
      - 代码或关键接口暴露
      - 甚至被别人加入广告，病毒等二次打包发布
    - 给公司和用户均带来巨大的风险
  - 应对破解的最便捷有效的方式
    - 加固=加壳
      - 通过加固可以在一定程度上达到反编译和防止被二次打包的效果
- 其他一些原因
  - 处于学习目的，想要了解、分析、学习某个安卓app的内部设计和代码逻辑
    - 所以需要反编译和破解
    - 所以防止别人破解要加密和加固

但是加固也有些缺点：

- 加固后对应用的影响
  - 体积
    - 变大（一些）
  - 启动速度
    - 变慢（一些）
      - 效率（略）降低
  - 兼容性
    - 部分方案加固后，会导致无法正常某些平台的正常运行
  - 使用成本
    - 有些加固方案需要收费
  - 影响部分应用市场的上架
    - 有部分的市场会拒绝加壳后的应用上架

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2023-09-03 17:53:04

## 常见加固服务提供商

市面上有很多家公司、厂家提供了免费或收费的加固方案：

## 各家加固方案的总结和对比

此处先列出：

### 总体效果对比

- 体积（体积小的为优）： 360 > 腾讯 > 爱加密 > 阿里 > 楠楠
- 兼容性： 阿里 > 腾讯 > 360 = 楠楠 > 爱加密
- 启动速度（时间短为优）： 阿里 > 爱加密 > 360 = 楠楠 > 腾讯
- 漏洞： 腾讯 > 爱加密 > 360 > 楠楠 > 阿里

### 实现原理对比

引用别人总结的（截至2015年）各家加固方案的技术原理，供参考：

- **360**：基本上是把原始的 `dex` 加密存在了一个 `so` 中，加载之前解密
- **阿里**：把一些 `class_data_item` 和 `code_item` 拆出去了，打开 `dex` 时会修复之间的关系。同时一些 `annotation_off` 是无效的来防止静态解析
- **百度**：把一些 `class_data_item` 拆走了，与阿里很像，同时它还会抹去 `dex` 文件的头部；它也会选择个别方法重新包装，达到调用前还原，调用后抹去的效果。我们可以通过对 `DoInvoke` (`ART`) 和 `dvmMterp_invokeMethod` (`DVM`) 监控来获取到相关代码
- **楠楠和爱加密**：与360的做法很像，楠楠把一堆 `read`、`write`、`mmap` 等 `libc` 函数 hook 了，防止读取相关 `dex` 的区域，爱加密的字符串会变，但是只是文件名变目录不变
- **腾讯**：针对于被保护的类或方法造了一个假的 `class_data_item`，不包含被保护的内容。真正的 `class_data_item` 会在运行的时候释放并连接上去，但是 `code_item` 却始终存在于 `dex` 文件里，它用无效数据填充 `annotation_off` 和 `debug_info_off` 来实现干扰反编译

## 各家加固方案详解

### 360加固宝

- 主页：<http://jiagu.360.cn/>
  - 还有其他产品：
    - 针对手游的：
      - 360手游保
      - <http://shouyouobao.360.cn>
    - 针对网页的：
      - H5加固

### 腾讯乐固legu

- 主页：<http://legu.qcloud.com/>
  - 应用安全 = Mobile Security = MS
    - 功能：为用户提供移动应用 (APP) 全生命周期的一站式安全解决方案
    - 涵盖服务
      - 应用加固
      - 安全测评

- 兼容性测试
- 盗版监控
- 崩溃监测
- 安全组件
- 特点
  - 坚固
    - 应用加固在不改 Android 应用源代码的情况下，将针对应用各种安全缺陷的加固保护技术集成到应用 APK，从而提升应用的整体安全水平，力保应用不被盗版侵权
  - 稳定
    - 应用安全提供的安全能力可在复杂环境下稳定运行，兼容性高、崩溃率低；不仅支持 arm、aarch64、x86、x64，还支持 android 2.0 到 android N 等几乎全系统全机型
- 机制和原理分析：
  - 乐固做了一些反调试的东西，很多情况并不是反调试越厉害加固就越好
  - 乐固仍然是常规规的函数调用和返回方式，流程清晰很多
- [腾讯云 移动应用安全 购买指南 产品文档](#)
  - 加壳 保护 功能
    - 反编译保护
      - DEX 反编译保护
        - 壳加密算法保护
        - AndroidManifest.xml 防篡改
        - DEX 文件整体加固保护
        - DEX 虚拟化加固(VMP)
      - SO 反编译保护
        - SO 库加壳保护
        - SO 库内存动态清除
        - SO 库与应用绑定保护
        - 高级 SO 混淆保护
        - SO 库字符串加密
      - 防篡改保护
        - APK 防篡改保护
          - APK 防二次打包保护
          - APK 签名文件校验保护
        - 源代码防篡改保护
          - DEX 文件防篡改
          - SO 库防篡改
        - 资源防篡改保护
          - assets 资源防篡改
          - res 资源防篡改
          - raw 资源防篡改
          - 配置文件防篡改
      - 防调试保护
        - 防调试保护
          - 防模拟器保护
          - 加固壳防动态调试
          - 防线程动态调试保护
          - 防进程动态调试保护
          - 防 JDWP 调试
          - 防注入保护
          - 防内存 dump 保护
          - 防内存数据读取
          - 防内存数据修改
      - 数据与资源保护
        - 资源防窃取保护

- assets 资源防窃取
- res 资源防窃取
- raw 资源防窃取
- SSL 证书防窃取
- 本地数据保护
  - 本地 databases 目录数据库文件加密
  - 防日志泄漏
  - 应用防截屏/录屏

## 网易易盾

- 网易易盾
  - 主页: [Android应用加固APK加固防篡改APP加固网易易盾](#)
  - 产品介绍
    - 防逆向
      - 多重指令转换VMP虚拟机保护技术, 对关键代码、核心逻辑进行加密保护, 避免通过IDA, Readelf等逆向工具分析获取源码
    - 防篡改
      - 对APP应用每个文件分配唯一识别指纹, 替换任何一个文件会导致无法运行, 防止广告病毒植入、二次打包、功能屏蔽等恶意破解
    - 防调试
      - 多重加密技术防止代码注入, 防止JAVA层/C层动态调试, 可有效抵挡动态调试、内存DUMP、代码注入、HOOK等恶意攻击
    - 数据保护
      - 提供安全键盘、通讯协议加密、数据存储加密、异常进程动态跟踪等功能技术, 在各个环节有效阻止数据被捕获、劫持和篡改
  - 功能介绍
    - DEX安全保护
      - VMP虚拟机保护
      - Java2C保护
      - DEX函数抽取加密
    - SO库加密保护
      - SO代码高级加密
      - SO函数动态加密
      - 防HOOK攻击
      - 防脱壳
    - 资源文件加密
      - assets资源文件加密
      - H5文件加密
      - XML配置文件保护
    - 防调试
      - 防动态调试
      - 防内存DUMP
      - 防动态注入
    - 数据保护
      - 日志防泄露
      - 防截屏保护
      - 数据文件加密
  - 应用场景
    - 应用程序被破解
    - 核心代码被窃取
    - 恶意代码注入
    - 核心数据泄露

- 安全检测未合规
- 核心优势
  - 高安全性
    - 加固强度高，有效对抗多种反编译逆向工具，防止APP被破解剽窃
  - 高兼容性
    - 支持arm、x86及64位多种CPU架构，完美支持Android4.0到最新系统
  - 高稳定性
    - 积累丰富的网易内部APP服务经验，加固后性能几乎无影响
  - 极速便捷
    - 提供工具和命令行操作，编译、加壳一体化快速完成
  - 灵活定制
    - 提供多种加固项和定制加固服务，自由选取灵活定制，满足不同行业需求
  - 国际认证
    - 拥有ISO27001、CSA-STAR国际权威标准认证，安全合规双重保障

## 爱加密

- <http://ijiami.cn>
- 移动应用安全加固
  - 安卓：<http://ijiami.cn/android>
    - 核心技术
      - 防逆向
        - 通过DEX加花和加壳、SO文件高级混淆和加壳等技术对DEX和SO文件进行保护，防止被IDA等逆向工具分析
      - 防调试
        - 多重加密技术防止代码注入，防JAVA层/C层动态调试、防代码注入和防HOOK攻击
      - 页面数据防护
        - 应用防劫持、应用防截屏、虚拟键盘SDK产品和技术，防界面劫持插件对组件进行全方位监听
      - 防篡改
        - 在加固时提取APP内各文件的文件特征值，当文件运行时，系统解密加密文件提取特征值进行文件校验
      - 数据防泄漏
        - 使用多种加密算法，包括国际通用算法及自主研发的加密算法等，保护本地数据
      - 传输数据防护
        - 在客户端和服务器分别嵌入数据加密SDK，保证通道中传输的数据为高强度加密后的数据
    - 主要功能
      - 概述
        - 提升APP安全性
          - 源代码保护、SO库保护、DEX文件保护、数据加密保护
        - 确保APP业务安全
          - 防盗版保护、防篡改、页面防劫持技术、防截屏技术、环境清场、短信防劫持
        - 保障APP数据安全
          - 密钥白盒技术、APP通讯链路加密、数据本地加密技术、安全键盘
        - 确保APP的整体优化
          - APP包体大小不超过原包“±5%”、全面的兼容性测试、全面的性能测试
      - 功能点
        - 防逆向
          - DEX整体加密保护
          - DEX代码分离保护
          - DEX混合加密保护
          - DEX VMP保护
          - 双重VMP保护
          - Java2CPP

- SO加壳
- SO Linker
- SO防调用
- SO VMP
- 防篡改
  - DEX文件防篡改
  - SO库文件防篡改
  - H5文件防篡改
  - 资源文件防篡改
  - 资源文件加密
  - 签名保护
- 防调试
  - 防动态调试
  - 防内存代码注入
  - 防模拟器
  - 防加速器
- 数据防泄漏
  - 防内存数据读取
  - 防内存数据修改
  - 防日志泄漏
  - 本地sharepreferences数据加密
  - 本地SQLite数据加密
- 页面数据防护
  - 防劫持
  - 防截屏
  - 安全键盘SDK
- 传输数据防护
  - 通信协议加密SDK
  - 密钥白盒
- 产品优势
  - 最新第六代高级双重VMP加密技术
  - 6种加密方式满足不同用户、行业的使用需求
  - 加密后包增量大小不超过原包“±5%”
  - 兼容性高达99%，实现ART全面兼容
  - 交付方式灵活，支持本地部署或者云部署，云部署支持私有云和公有云
  - 通过密钥白盒技术实现最强的加密强度
  - 获得上千家知名行业客户认可的移动安全技术方案
- iOS: <http://ijiami.cn/iosProtect>
- SDK: <http://ijiami.cn/sdkProtection>
- SO库: <http://ijiami.cn/soProtect>
  - SO文件加壳保护
    - 对SO文件进行加壳保护，加壳后使用ida工具无法看出SO库文件的导入导出函数以及定位源码，有效防止黑客反编译，解包后看到真正源码
  - SO文件混淆保护
    - 爱加密基于移动安全领域的先进的技术和经验，针对黑客在分析阶段的攻击手段和行为进行分析，利用SO混淆编译器，可以有效的增加黑客信息搜集的难度和复杂度，防止应用被破解，降低APP安全风险

## 梆梆安全

- 主页
  - 梆梆安全 - 移动安全领导品牌，保护智能生活，共建智慧城市！
- 产品
  - 泰固

- 移动应用安全加固
  - 针对目前移动应用普遍存在的破解、篡改、盗版、钓鱼欺诈、内存调试、数据窃取等各类安全风险，梆梆安全为开发者提供全面的移动应用加固加密技术和攻击防范服务
  - 核心加固技术
    - 防逆向（Anti-RE）：抽取classes.dex中的所有代码，剥离敏感函数功能，混淆关键逻辑代码，整体文件深度加密加壳，防止通过apktool, dex2jar, JEB等静态工具来查看应用的Java层代码，防止通过IDA, readelf等工具对so里面的逻辑进行分析，保护native代码。
    - 防篡改（Anti-tamper）：每个代码文件、资源文件、配置文件都会分配唯一识别指纹，替换任何一个文件，将会导致应用无法运行，存档替换、病毒广告植入、内购破解、功能屏蔽等恶意行为将无法实施。
    - 防调试（Anti-debug）：多重加密技术防止代码注入，彻底屏蔽游戏外挂、应用辅助工具，避免钓鱼攻击、交易劫持、数据修改等调试行为。
    - 防窃取（Storage Encryption）：支持存储数据加密，提供输入键盘保护、通讯协议加密、内存数据变换、异常进程动态跟踪等安防技术，有效防止针对应用动、静态数据的捕获、劫持和篡改。
  - 加固服务策略
    - 提供灵活多样的App加固方案
      - 防逆向保护
        - DEX文件加壳保护
        - DEX函数抽取加密
        - HTML开发框架保护
        - SO文件加壳保护
        - SO代码压缩及加密
        - SO库设备绑定
        - 源代码深度混淆
      - 防篡改保护
        - 开发者签名校验
        - 代码、资源文件、配置文件完整性校验
        - 数据透明加密及设备绑定
        - 配置文件、数据库文件加密
        - 视频、音频、图像等文件加密
      - 防调试保护
        - 防止进程/线程附加
        - 防止进程注入
        - 防HOOK攻击
        - 防内存Dump
        - App完整性保护
        - SO数据动态清除
      - 防窃取保护
        - 本地数据文件加密
        - 键盘数据加密
        - 通讯协议加密
        - 密钥白盒加密
    - 移动应用源代码加固
      - 未经混淆的源代码受到攻击后，易暴露程序中关键算法、核心业务逻辑、数据结构和模块的控制流布局等敏感内容

## 顶象安全

- 主页
  - [顶象技术 - 金融业务安全的实践者，专注智能风控与全链路反欺诈](#)
- 功能
  - [Android加固保护 - 顶象技术](#)
    - 一套纵深防御体系

- 分别从防逆向、防调试和防篡改等几个维度提供安全保护
- 同时针对每个维度提供又进行了不同层次的划分，加固策可依据实际场景进行动态调配，安全和性能达到完美平衡。
- 对APK提供DEX保护、SO保护、数据加密、资源防篡改、运行时保护等多角度全方位的保护
- 核心功能
  - Dex加壳保护
    - 对dex文件整体进行加密隐藏，并进行反编译保护处理
  - Dex VMP保护
    - 将函数方法指令翻译为自定义指令集并加密存储，在运行过程中交由顶象dex虚拟机解释执行
  - SO加壳保护
    - 加密隐藏SO文件中的导入导出符号表以及常量字符串，同时对SO进行反编译保护处理
  - SO VMP保护
    - 对二进制函数指令翻译为自定义指令集并加密存储，在运行过程中交由顶象dex虚拟机解释执行
- 特点与优势
  - 平台兼容性高
    - 支持ARM, ARM64, x86, x86\_64, mips等多种cpu框架
  - 语言支持丰富
    - 支持包括Java, Kotlin, C/C++, Objective-C, Swift等在内的多种源码类型或混合型项目
  - 操作流程便捷
    - 把编译好的App上传并选择好加固方案即可完成整个加固流程，操作简单易懂，非技术人员也能轻松掌握
  - 体积增量小
    - 整体加固增量不会超过100kb，一般情况下是80kb
- 应对风险
  - 程序逻辑被破解
  - 核心代码被窃取
  - API接口暴露
  - 恶意代码注入

## 几维安全

- 主页
  - [几维安全 - 让万物互联更安全](#)
- 功能
  - APP应用加固
    - 重点
      - 高效、专业、兼容好
      - 5分钟极速加密，轻松集成DEX加密、反调试、防盗版等多重安全防护
    - 产品简介
      - 移动应用安全加固是一项面向互联网企业和个人开发者的在线加密服务，现支持安卓应用加密，用户只需提供APK包即可快速集成防静态工具分析、Dex文件保护、So文件加壳、内存保护、反调试、防二次打包等多项安全功能。支持对金融、手游、电商、社交等多个行业的应用做加固保护，避免核心代码被反编译，请求协议被伪造，APK包被植入恶意代码等诸多安全问题。
    - 功能特点
      - Dex文件保护
        - 对DEX文件进行加密保护，防止被Dex2Jar等工具逆向破解
      - Dex-Java2C
        - 将Java代码翻译为C代码，并实施Native层的代码混淆保护
      - SO文件加壳
        - 对SO文件进行整体加壳保护，防止IDA Pro等工具逆向分析
      - 防二次打包
        - 集成正版签名校验功能，运行时动态校验，防止被植入恶意代码
      - 内存加密
        - 防止内存数据被篡改或Dump，比如Dump解密后的Java代码

- 反调试
  - 拒绝调试器对当前应用的附加操作，防止程序被恶意调试分析
- 自身虚拟化保护
  - 专业版采用代码虚拟化技术对自身代码进行保护，防止逆向分析
- 兼容性良好
  - 测试覆盖200+机型，兼容性达到99%，支持ART模式
- 应对风险
  - Dex文件反编译
  - So文件反编译
  - 核心技术窃取
  - 通信模块破解
  - API接口暴露
  - 密钥窃取
  - 注入恶意代码
  - 伪造盗版应用
- SO源码混淆保护
  - 重点
    - 基于NDK项目源码，通过安装NDK插件即可集成代码混淆、轻量虚拟化、字符串加密等多项高强度的安全保护
  - 产品简介
    - SO库源代码保护是一款离线的安全编译器工具，主要用于保护Android NDK项目中的核心代码，避免因逆向工程或破解，造成核心技术被泄漏、代码执行流程被分析等安全问题。该安全编译器和普通编译器相似，基于项目源代码可将C、C++代码编译成二进制代码，不同之处在于，安全编译器在编译的时，能够对代码进行混淆、轻量虚拟化、字符串加密等安全保护。从而避免攻击者通过IDA Pro等逆向工具反编译二进制代码，分析业务代码执行流程，进一步篡改或窃取核心技术。

## 阿里聚安全

- <http://jaq.alibaba.com/>
  - 20180801已下线
- 原理分析：
  - Ali 利用 SP 来储存返回地址和参数所以整个流程很混乱，基本看上去就是在各种 JUMP

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2023-08-25 22:55:15

# 代码混淆

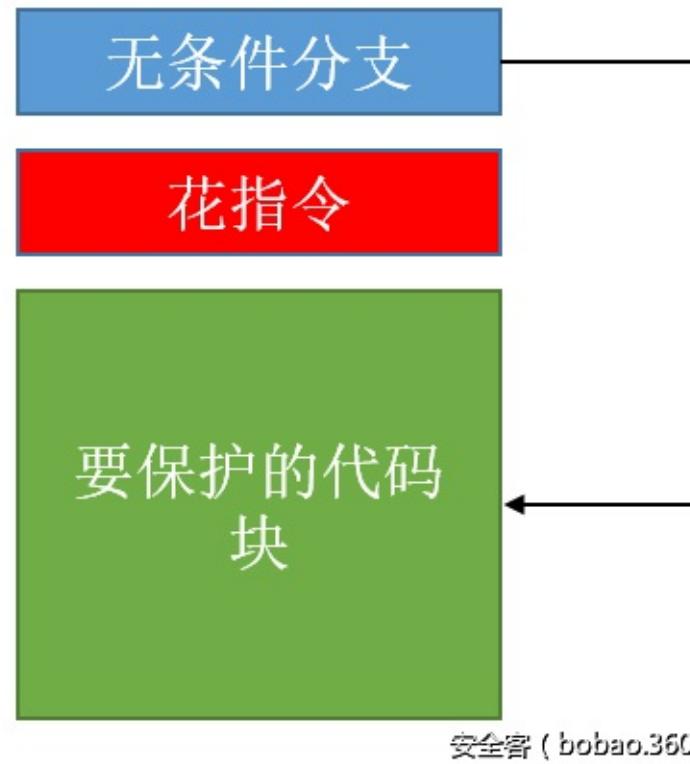
- 混淆 = 代码混淆 = 花指令
  - 含义：把原先的代码通过变量替换（成a,b,c等）方式，使得代码不可读，很难读
  - 目的：增加破解人员读懂原先代码逻辑的难度

下面详细介绍安卓代码混淆的技术方案：

- ProGuard
- Obfuscator-LLVM

## 花指令

- 加花 = 花指令
  - 名称：
    - 花指令
    - 又称：
      - 垃圾指令
      - 指令 其实就是 字节
      - = Junk Bytes = JunkBytes
      - 垃圾代码 = Junk Code = JunkCode
    - 把 花指令 加到代码中的动作： 加花
  - 起源： 花指令 这个词来源于汇编语言
  - 含义：在真实代码中插入一些（垃圾的、无用的）代码 / 指令 / 字节，但又不会改变程序的原始逻辑，确保原有程序的正确执行
  - 目的：反汇编工具在反汇编时会出错，导致反汇编工具失效，提高破解难度
    - 隐藏掉不想被逆向工程的代码块(或其它功能)的一种方法，使得程序无法很容易地反编译，即使被反编译后，也难以理解程序内容，达到混淆视听的效果，增加破解和逆向的难度
  - 主要思想
    - 当花指令跟正常指令的开始几个字节被反汇编工具识别成一条指令的时候，才可以使得反汇编工具报错
    - 插入的花指令都是一些随机的但是不完整的指令
  - 特点
    - 花指令必须要满足两个条件
      - 在程序运行时，花指令是位于一个永远也不会被执行的路径中
      - 这些花指令也是合法指令的一部分，只不过它们是不完整指令而已
  - 实现思路
    - 在每个要保护的代码块之前插入无条件分支语句和花指令



安全客 ( bobao.360.cn )

- 实际案例

- Dalvik Bytecode Obfuscation on Android中，插入fill-array-data-payload花指令，导致反编译工具失效

```

private String exec(String paramString)
{
    throw new RuntimeException("Generated by Dex2jar, and Some Exception Caught :java.lang.NullPointerException\n\tat
com.googlecode.dex2jar.ir.ts.ExceptionHandlerCorrectTransformer.transform(ExceptionHandlerCorrectTransformer.java:66)\n\tat
com.googlecode.dex2jar.v3.V3MethodAdapter.visitMethodEnd(V3MethodAdapter.java:214)\n\tat
com.googlecode.dex2jar.v3.V3ClassAdapter$2.visitMethodEnd(V3ClassAdapter.java:261)\n\tat
com.googlecode.dex2jar.reader.DexFileReader.visitMethod(DexFileReader.java:702)\n\tat
com.googlecode.dex2jar.reader.DexFileReader.acceptMethod(DexFileReader.java:446)\n\tat
com.googlecode.dex2jar.reader.DexFileReader.accept(DexFileReader.java:333)\n\tat
com.googlecode.dex2jar.v3.Dex2jar.doTranslate(Dex2jar.java:82)\n\tat com.googlecode.dex2jar.v3.Dex2jar.to(Dex2jar.java:219)\n\tat
com.googlecode.dex2jar.v3.Dex2jar.to(Dex2jar.java:210)\n\tat
com.googlecode.dex2jar.tools.Dex2jarCmd.doCommandLine(Dex2jarCmd.java:108)\n\tat
com.googlecode.dex2jar.tools.BaseCmd.doMain(BaseCmd.java:118)\n\tat
com.googlecode.dex2jar.tools.Dex2jarCmd.main(Dex2jarCmd.java:34)\n";
}
  
```

安全客 ( bobao.360.cn )

- 现存服务提供商

- 举例

- 网易易盾的：安卓dex加花保护

- 相关背景

- 反汇编工具常用算法

- 线性扫描算法

- 逻辑：依次按顺序逐个地将每一条指令反汇编成汇编指令

- 结果：容易把花指令错误识别，导致反汇编出错

- 举例

- 指令

1 0003bc: 1250	0000: const/4 v0, #int 5 // #5
2 0003be: 2900 0400	0001: goto/16 0005 // +0004
3 0003c2: 0001	0003: <Junkbytes>
4 0003c4: 0000	0004: <Junkbytes>
5 0003c6: d800 000	0005: add-int/lit8 v0, v0, #int 1 // #01
6 0003ca: 0f00	0007: return v0

安全客 ( bobao.360.cn )

- 反汇编后出错

```

1 0003bc: 1250          |0000: const/4 v0, #int 5 // #5
2 0003be: 2900 0400      |0001: goto/16 0005 // +0004
3 0003c2: 0001 0000 d800 0001 |0003: packed-switch-data (4 units)
4 0003ca: 0f00          |0007: return v0
                                         安全客 (bobao.360.cn)

```

#### ■ 递归扫描算法

- 逻辑：按顺序逐个反汇编指令
  - 如果某个地方出现了分支，就会把这个分支地址记录下来，然后对这些反汇编过程中碰到的分支进行反汇编
  - 结果：反汇编能力更强
- 总结
  - 常用Android逆向工具中的反汇编算法

线性扫描算法	递归扫描算法
DexDump	baksmali
jeb	Androguard
ded	IDA Pro
	Radar2 安全客 (bobao.360.cn)

# ProGuard

## ProGuard的作用

- 压缩=Shrinking
  - 移除未被使用的类、属性、方法等，并且会在优化动作执行之后再次执行（因为优化后可能会再次暴露一些未被使用的类和成员）
- 优化=Optimization
  - 优化字节码，并删除未使用的结构
- 混淆=Obfuscation
  - 将类名、属性名、方法名混淆为难以读懂的字母，比如a,b,c等，增大反编译难度

## ProGuard的输出文件说明

- dump.txt：说明 APK 中所有类文件的内部结构
- mapping.txt：提供原始与混淆过的类、方法和字段名称之间的转换和对应关系
- seeds.txt：列出未进行混淆的类和成员
- usage.txt：列出从 APK 移除的代码

## 注意事项

- 有些库，混淆后，导致代码不可用
  - 所以有些好的库，专门支持了ProGuard
    - 举例
      - okhttp
      - GitHub
      - <https://github.com/square/okhttp>
        - If you are using R8 or ProGuard add the options from okhttp3.pro
        - R8 proguard - OkHttp
  - 有些库不够好，需要自己额外处理
    - 举例
      - PermissionGen
      - <https://github.com/lovedise/PermissionGen>
        - 某人：打包混淆以后就废了，需要自己加配置，避免部分模块被混淆，才勉强可用

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2021-06-27 16:44:06

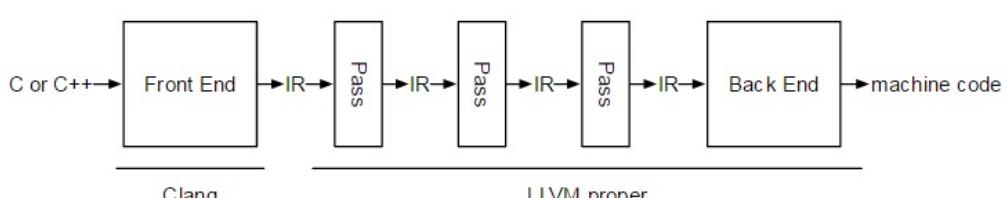
## Obfuscator-LLVM

- Obfuscator-LLVM = ollvm
  - 功能特性
    - 指令替换
      - 参数: -mllvm -sub
      - 文档: [Instructions Substitution](#)
    - Bogus控制流
      - 参数: -mllvm -bcf
      - 文档: [Bogus Control Flow](#)
    - 控制流扁平化 = 控制流平坦化
      - 参数: -mllvm -fca
      - 文档: [Control Flow Flattening](#)
    - 函数注解
      - 文档: [Functions annotations](#)
  - 应用
    - 市场上一些加固厂商(比如360加固宝、梆梆加固)会使用改进的Obfuscator-LLVM对它们so文件中的一些关键函数采用Obfuscator-LLVM混淆，增加逆向的难度
    - 简单一点的是，用Obfuscator-LLVM混淆native代码，膨胀so并插入花指令
  - 文档
    - Github
      - obfuscator-llvm/obfuscator
        - <https://github.com/obfuscator-llvm/obfuscator>
      - Home · obfuscator-llvm/obfuscator Wiki
        - <https://github.com/obfuscator-llvm/obfuscator/wiki>
    - 最新版
      - obfuscator-llvm/obfuscator at llvm-4.0
        - <https://github.com/obfuscator-llvm/obfuscator/tree/llvm-4.0>

## 相关: LLVM

- LLVM = Low Level Virtual Machine
  - 概述: a open source toolkit for the construction of highly optimized compilers, optimizers, and runtime environments
  - 其下很多子项目

- LLVM Core
  - LLVM总体架构



- Clang
- LLDB
- libc++ 和 libc++ ABI
- compiler-rt
- MLIR
- OpenMP
- polly

- [libclc](#)
- [klee](#)
- [LLD](#)
- 官网
  - <http://www.llvm.org/>

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2023-09-04 09:14:32

# 防动态调试

此处主要介绍安卓的正向安全中的：

- 防动态调试
  - 防调试和运行
    - 反调试
    - Root检测
      - 检测 Exposed 、 Frida 等
  - 防抓包=反抓包
    - ssl pinning = 证书绑定

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2023-08-24 23:16:28

# 安卓逆向破解

此处主要介绍安卓的逆向破解方面的内容：

## 概述

- 安卓的逆向破解
  - 静态分析
    - 按文件格式类型分
      - 针对 apk
        - 查看apk信息
          - aapt
        - 解包工具：输出 dex、so、smali
          - apktool
        - 反编译工具
          - 直接apk转java
            - jadx
            - JEB
            - GDA
        - 针对 dex
          - 各种导出dex的工具=砸壳工具=脱壳工具
            - FDex2
            - DumpDex
            - DexExtractor
          - 各种反编译dex的工具
            - dex转jar
              - dex2jar
            - dex转smali
              - baksmali
            - dex直接转java
              - jadx
              - GDA
          - 针对 jar
            - jar转java=各种反编译工具
              - Procyon
              - CFR
              - JD-GUI
          - 针对 so 库文件=（往往是ARM64架构）的ELF文件
            - 导出静态资源
              - readelf
              - objdump
              - razbin2
            - 反编译代码逻辑
              - IDA
              - Hopper
              - Radare2
              - Ghidra
          - 涉及子领域
            - 反代码混淆
      - 动态调试

- (绕过限制去) 抓包
  - 绕过证书绑定的Exposed插件或Frida的脚本
- 反root检测
- 反反调试
- 各种动态调试手段
  - 调试smali: AS+smalidea插件
    - app进程可调试
      - Magisk插件: MagiskHide Props Config
  - 调试Frida: Frida
    - Frida环境搭建
      - Magisk插件: MagiskFrida
  - 调试lldb: LLDB
  - 调试Xposed插件
    - Xposed
    - EdXposed
    - LSPosed
  - 模拟代码执行
    - Unidbg
      - Unicorn
  - 辅助调试工具
    - adb
    - DDMS
- 输出
  - 重新打包apk
    - apktool
    - keytool
    - jarsigner 或 apksigner
    - zipalign
  - 用代码重写逻辑
    - Python代码
    - C/C++代码
  - 改机定制ROM
    - AOSP源码 编译
    - JNI 开发
    - NDK 开发

另外：

- 砸壳思路=脱壳方案
  - 目前多数都是基于 Hook 框架，去从安卓app中导出dex文件
    - 典型的hook框架
      - Xposed : 从根上hook了Android Java虚拟机
      - Cydia : 支持 JNI 和 java 层的hook功能
    - 再去从dex中转换出java代码

## 详解

具体内容，详见独立子教程：

[Android逆向开发](#)



# 系列子教程

此处列出已完成的Android安全和逆向相关系列的 子教程：

- 子教程
  - 安卓
    - 正向
      - [Android开发总结](#)
    - 逆向
      - [Android逆向开发](#)
      - [Android逆向：开启root](#)
      - [Android逆向：静态分析](#)
      - [安卓反编译利器：jad](#)
      - [Android逆向：动态调试](#)
      - [Android逆向：重新打包apk](#)
    - 相关
      - [安卓模拟器](#)
      - [好用的安卓模拟器：夜神Nox](#)
    - 框架
      - [安卓逆向调试：Xposed框架](#)
  - 通用
    - 编程语言
      - C
        - [C语言开发心得](#)
      - ARM
        - [最流行汇编语言：ARM](#)
    - 抓包工具
      - [app抓包利器：Charles](#)
    - 静态分析 (+动态调试)
      - [逆向利器：IDA](#)
    - 动态调试
      - [主流调试器：LLDB](#)
      - [逆向调试利器：Frida](#)
      - [CPU模拟利器：Unicorn](#)

crifan.org, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2023-09-14 20:34:11

## 附录

下面列出相关参考资料。

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2019-05-02 15:25:25

# 参考资料

- 【整理】安卓安全技术：VMP android
- 【已解决】mac中安装最新版本的安卓反编译工具：Apktool
- 【已解决】用Python代码实现少儿趣配音的请求参数sign的计算逻辑
- 【整理】安卓安全 Smali
- 【整理】安卓安全和破解相关：加花 花指令
- 【整理】安卓 防护 加壳 服务商 总结
- 
- 【已解决】mac中试用FDex2去hook导出安卓app的dex等文件
- 【已解决】从不同版本的小花生apk中反编译出包含业务逻辑代码的dex和jar包源码
- 【已解决】小花生安卓app的v3.4.8版破解后找到源码中是否包含J字段的加密逻辑
- 【已解决】尝试破解小花生app安卓apk希望看到api返回的json中的J的解密算法得到明文
- 【已解决】如何反混淆即还原反编译后混淆的安卓代码
- 【记录】从反编译安卓apk得到的java源码代码中尝试找返回json中J加密的逻辑和线索
- 【已解决】小花生app中api请求返回json的C, J, M, ST的含义和如何破解解密
- 【记录】爬取小花生app中自主阅读馆和亲子阅读馆中的有音频的绘本数据
- 【未解决】dex2jar反编译dex后jar文件包含java代码：throw new VerifyError bad dex opcode
- 【已解决】python实现java的MessageGZIP.uncompressToString即gzip的解码
- 【已解决】Python中实现java中的Base64.decode解码加密字符串
- 【已解决】从反编译小花生apk得到的包含业务逻辑代码中找到J字段解码的逻辑并用Python实现
- 【无需解决】安卓apk反编译出的smali反向破解出java原始代码
- 【已解决】安卓app如何脱壳如何破解加固
- 【已解决】找小花生app的旧版本apk并尝试能否安装使用
- 【部分解决】尝试破解安卓apk康美通去得到java源码
- 【整理】JVM参数-Xms和-Xmx参数的含义
- 【未解决】用ART, oat, dex2oat相关机制去破解新一代360、腾讯等安卓apk的加固
- 【整理】把jar包转换为java源代码的java反编译器的整理和对比
- 【已解决】用基于Procyon的Luyten反编译安卓jar包得到java源码
- 【未解决】小花生中如何得到getToken的计算逻辑以便得到正确的md5值可以正常请求接口
- 【已解决】mac版JD-GUI查看并导出jar包的java源代码
- 【已解决】用java反编译器CFR从jar包导出java源代码
- 【已解决】用Procyon命令行去从jar包导出java源代码
- 【记录】从安卓的apk中解压出各种项目文件
- 【已解决】用WrBug的DumpDex从app中hook导出dex文件
- 【已解决】mac中用dex2jar反编译dex文件导出jar包文件
- 【已解决】夜神安卓模拟器中导出文件到mac电脑
- 【已解决】Nox夜神安卓模拟器中/mnt/shared对应Mac的共享目录在哪里
- 【基本解决】尝试破解安卓apk马蜂窝去得到java源码
- 【已解决】搞懂安卓app混淆和加固常见做法和相关逻辑
- 
- 破解某小说App（一） - 挖金
- 最新乐加固脱壳详细教程（有图有真相） - Android安全 - 逆向未来技术社区 - Powered by Discuz!
- Android逆向助手反编译APK - 长江某菜鸟的博客 - CSDN博客
- 加固保-移动应用安全资讯
- android - Is there a way to get the source code from an APK file? - Stack Overflow
- spriteviki/Dex2oatHunter: Automatic Unpacking Tool for Android Dex Files
- Dex : Java Data Visualization
- What are .dex files in Android? - Stack Overflow
- Dalvik 可执行文件格式 | Android Open Source Project
- ART 和 Dalvik | Android Open Source Project

- [压缩代码和资源 | Android Developers](#)
- [Android应用加固产品使用对比 - 『移动安全区』 - 吾爱破解 - LCG - LSG |安卓破解|病毒分析|破解软件|www.52pojie.cn](#)
- [S3cuRiTy-Er1C/JebScripts: Jeb public scripts](#)
- [flankerhqd/jebPlugins: Various Jeb plugins, including obfuscation restore](#)
- [enovella/jebscripts: A set of JEB Python/Java scripts for reverse engineering Android obfuscated code](#)
- [CalebFenton/simplify: Generic Android Deobfuscator](#)
- [Android 反混淆神器JEB2的使用简介 - 飞少的博客 | Jack's Blog](#)
- [【技术分享】Android程序反混淆利器——Simplify工具 - 安全客, 安全资讯平台](#)
- [\[原创\]JEB2反混淆神器 - 『Android安全』-看雪安全论坛](#)
- [Android | 使用Java Deobfuscator对JEB Decompiler反混淆\\_ - AppScan.IO | Janus移动安全中心](#)
- [Deobfuscating Android Triada malware – JEB Decompiler in Action](#)
- [ANDROID 逆向实例（八） – 乐固加固脱壳（2017.01） ~ and-rev](#)
- [乐固加固脱壳实战 - faTe's Home](#)
- [Android APK脱壳--腾讯乐固、360加固一键脱壳 - 知乎](#)
- [Android APK脱壳--腾讯乐固、360加固一键脱壳 | 辉天神龙](#)
- [安卓逆向调试：XPosed框架](#)
- [好用的安卓模拟器：夜神Nox](#)
- [Android逆向入门流程 - 简书](#)
- [swdunlop/AndBug: Android Debugging Library](#)
- [Android安全专项-AndBug动态调试工具 - doctorq - CSDN博客](#)
- [androguard/androguard: Reverse engineering, Malware and goodware analysis of Android applications](#)
- [Welcome to Androguard's documentation! — Androguard 3.4.0 documentation](#)
- [Android逆向之旅—Native层的Hook神器Cydia Substrate使用详解 | 尼古拉斯.赵四](#)
- [AndroidSecNotes/Android Hook 框架（Cydia篇）.md at master · JnuSimba/AndroidSecNotes](#)
- [看雪 - 安卓黑科技之HOOK详解](#)
- [Android逆向工程工具Dare的使用方法（Mac OS X中） - qysh123的专栏 - CSDN博客](#)
- [Dare Homepage](#)
- [Dare downloads](#)
- [Google反编译新工具——Enjarify - liuweiballack的专栏 - CSDN博客](#)
- [android:tools:enjarify WooYun WiKi](#)
- [Android 逆向工具 dex2jar, enjarify 和 AXMLPrinter2 // Neurohazard](#)
- [Dedexer user's manual](#)
- [dedexer - Browse Files at SourceForge.net](#)
- [Dedexer:Dex文件反编译工具介绍 - amos\\_tl - ITeye博客](#)
- [2015移动安全挑战赛（阿里&看雪主办）全程回顾（3） - Group of Software Security In Progress](#)
- [关于Indroid的编译 · Issue #1 · romangol/InDroid](#)
- [iSECPartners/Introspy-Android: Security profiling for blackbox Android](#)
- [Introspy-Android](#)
- [目前最全面的Android安全工具清单 - IT经理网](#)
- [翻译Android 应用逆向工具总结-『外文翻译』 -看雪安全论坛](#)
- [Infografía sobre el funcionamiento interno del robot de Android | SmallVille](#)
- [intellij-community/plugins/java-decompiler/engine at master · JetBrains/intellij-community](#)
- [fesh0r/fernflower: Unofficial mirror of FernFlower Java decompiler \(All pulls should be submitted upstream\)](#)
- [《Smali Viewer 用户指南》 | AVL Team](#)
- [AndroidDevTools - Android开发工具 Android SDK下载 Android Studio下载 Gradle下载 SDK Tools下载](#)
- [SmaliViewer.Zip](#)
- [Application Hardening - Mobile App Hardening | Promon](#)
- [Cydia Substrate使用手册 - 简书](#)
- [看雪安全论坛 18年专注——顶级软件逆向论坛](#)
- [一张图看懂Android编译流程 - 简书](#)
- [原创 如何使用Xposed+JustTrustMe来突破SSL Pinning-『WEB安全』 -看雪安全论坛](#)
- [当你写爬虫抓不到APP请求包的时候该怎么办？【中级篇】 - 知乎](#)
- [tls - What is certificate pinning? - Information Security Stack Exchange](#)

- [permissionen权限管理混淆处理\\_dobiman的博客-CSDN博客](#)
- [第三方免费加固横向对比 – Android – 掘金](#)
- [Android DEX-VMP 虚拟保护技术 | GeneBlue's Blog](#)
- [Android最新的VMP加固技术一般是怎么实现的? - 知乎](#)
- [Android Vmp加固实现流程图\\_zhangmiaoping23的专栏-CSDN博客\\_android vmp](#)
- [\[原创\]某Android DEX vmp加固逆向分析-Android安全-看雪论坛-安全社区|安全招聘|bbs.pediy.com](#)
- [Study of android malicious in dynamic unpacking](#)
- [Android SO Virtualization Protection | KIWISEC](#)
- [eaglx/VMPROTECT: Obfuscation method using virtual machine.](#)
- [Home · obfuscator-llvm/obfuscator Wiki](#)
- [Obfuscator-llvm源码分析 - 知乎](#)
- [\[原创\]jollvm的混淆反混淆和定制修改-Android安全-看雪论坛-安全社区|安全招聘|bbs.pediy.com](#)
- [技术前沿|虚拟机保护技术（VMP）的实践与体会-支付产业网](#)
- [【更新】讨论android加固内存dump的技术及vmp壳的防护强度 - 看雪安全论坛](#)
- [\[原创\]dex vmp虚拟化-『Android安全』-看雪安全论坛](#)
- [第五代加固技术 ARM代码虚拟化保护技术](#)
- [DexHunter的原理分析和使用说明（一）\\_Fly20141201. 的专栏-CSDN博客\\_dexhunter](#)
- [\[原创\]Android dex文件通用自动脱壳器-Android安全-看雪论坛-安全社区|安全招聘|bbs.pediy.com](#)
- [从Android运行时出发，打造我们的脱壳神器-Harries Blog™](#)
- [\[原创\]FART：ART环境下基于主动调用的自动化脱壳方案-Android安全-看雪论坛-安全社区|安全招聘|bbs.pediy.com](#)
- [\[原创\]记录一次非常简单的so层小逆向，适合小白入门-『Android安全』-看雪安全论坛](#)
- [Smali 语法解析——Hello World - 掘金](#)
- [JesusFreke/smali: smali/baksmali](#)
- [Smali语法介绍\\_Java\\_singwhatiwanna-CSDN博客](#)
- [Android逆向基础：Smali语法 - 简书](#)
- [dalvik - What is Smali Code Android - Stack Overflow](#)
- [Smali: Assembler for Android's VM | Medium](#)
- [\[Android\]\[Security\] Android 逆向之 smali | wOw的博客 \(wossoneri.github.io\)](#)
- [Android Runtime \(ART\) 和 Dalvik | Android 开源项目](#)
- [Dalvik 可执行文件格式 | Android 开源项目 | Android Open Source Project](#)
- [通付盾: 产品体系\\_应用加固 \(Andorid/iOS/H5/SDK\)](#)
- [移动应用安全加固移动应用APP加固应用安全安全市场华为云市场-华为云](#)
- [爱加密：深入上海互联网移动应用分析，协助构建移动安全互联-爱加密移动应用安全保护平台|app防反编译|app加壳|app防破解](#)
- [安卓dex加固保护\\_网易易盾](#)
- [android dex加固保护\\_网易易盾](#)
- [Android 代码混淆并加花 - 第四维空间testing - 51Testing软件测试网 51Testing软件测试网-软件测试人的精神家园](#)
- [android打包上架之预防反编译（花指令）\\_移动开发\\_qinzhuoheng的专栏-CSDN博客](#)
- [【技术分享】Android代码混淆技术总结（一） - 安全客，安全资讯平台](#)
- [花指令 - CTF Wiki](#)
- [纯手工混淆C/C++代码（上） - 知乎](#)
- [Unicorn实战（一）：去掉libcms.so的花指令 – Leadroyal's website](#)
- [【技术分享】Android SO 高阶黑盒利用 - 安全客，安全资讯平台](#)
- [FDex2 core code MainHook - Programmer Sought](#)
- [Shelling Android APK - Le Tencent solid, a reinforcement key 360 husking\(Others-Community\) \(titanwolf.org\)](#)
- [关于Dalvik、ART、DEX、ODEX、JIT、AOT、OAT\\_玛斯特·布兰迪的博客-CSDN博客](#)
- [Android .dex、.odex、Dalvik、ART、AOT、OATAndroid dex odex art龙腾腾的博客-CSDN博客](#)
- [安卓黑科技之HOOK详解](#)
- [...](#)

