Android 加固应用脱壳技术研究

吴博，郭燕慧
（北京邮电大学计算机学院，北京 100876）

摘要：随着移动互联网的发展，移动安全加固技术逐步普及，被加固的恶意应用带来的问题日益突出，传统的应用检测在加固应用的逆向分析环节比较薄弱，难以对加固应用进行检测，这为移动互联网安全发展带来严峻的挑战。对此，本文针对具有进程状态检测和内存保护技术的加固应用，采用 Hook 技术和字节码还原技术，提出并实现了自动化脱壳系统。测试结果表明，该系统可有效的对加固应用进行脱壳。

关键词：Android 加固应用逆向分析；加固应用检测；字节码还原
中图分类号：TP309.2

Research on the Technology of shelling of Android Reinforced Application

WU Bo, GUO Yanhui
(Being University of Post and Teleommunication Information Technology Institude, Beijing 100876)

Abstract: With the development of mobile Internet, mobile security reinforcement technology has been gradually popularized, the problems caused by the Reinforced-Malware has become increasingly prominent. The reverse analysis in traditional detection of application is weak, it is difficult to detect the Reinforced-Malware, which brings severe challenges to the development of mobile Internet. In this regard, to deal with the Reinforced-application with process state detection and memory protection technology, using Hook and bytecode restore technology, this paper proposed and implemented an automatic shelling system. Test results show that the proposed system can be effectively used for the reverse analysis of the Reinforced-Application..

Key words: Reverse Analysis of Reinforce Application; Reinforced Application Detection; Bytecode Restoration

0 引言

随着移动互联网的不断发展，对移动安全技术的研究逐步成为重点，为提高移动终端应用的安全性，对移动终端平台的应用程序加固技术逐渐深入。随着加固技术的普及，恶意代码也采用加固技术对自身隐藏。

由于传统的应用检测方式，通常是对未加固应用进行动静态检测，而应用了加固技术后，对应用源码采取了加密保护，并且增加了动态保护技术，如反调试、内存保护等，很难提取静态特征和动态行为特征，使得对加固应用进行较为全面的检测变得困难。

因此，需要通过使用脱壳技术，对加固应用进行脱壳后，然后使用传统的应用检测进一步检测。但现有的脱壳技术都存在一定程度的不足，所以需要对脱壳技术进行深入研究。

1 Android 加固应用脱壳技术研究现状

在移动互联网环境错综复杂的今天，为保障移动互联网的安全，使得对加固应用的检测顺利进行，现有多种脱壳技术。现在常见的脱壳技术有：动态调试技术、基于存储状态的静态还原技术、基于进程附加的脱壳技术、基于源码编译的内存 dump 技术等[1]。下面将针对这几种脱壳技术进行简单介绍与分析，指出现脱壳技术和算法的优缺点。

作者简介：吴博(1990-)，男，硕士研究生，主要研究方向：移动安全
通信联系人：郭燕慧(1974-)，女，副教授，主要研究方向：内容安全，信息安全管理. E-mail: yhguo@bupt.edu.cn

1.1 现有的 Android 加固应用脱壳技术概述

1.1.1 动态调试技术

动态调试技术，利用 Linux 内核提供的调试接口，实现集成化调试环境，可获得被调试应用的中间状态和产生值，在此基础上对特定内存刷新函数进行监测，即可获得解密后程序字节码，即可有效的进行动态脱壳。

1.1.2 基于储存状态的静态还原技术

静态修复技术是利用进程运行时在文件存储系统中所产生的中间文件，通过对中间文件进行 HEX 解析，形成格式解释器，在此基础上对中间文件进行静态逆向过程，尽可能的对文件进行还原。

1.1.3 基于进程附加的自动脱壳技术

进程附加是指通过内核调试器与目标进程进行通信，使目标进程变得可控，在调试进程角度，目标进程的内存空间是间接可读写的，基于此在目标进程完成自解密过程后，对目标进程应用字节码读取到外存，可以有效的对加固应用进行脱壳。

1.1.4 基于 Dalvik 虚拟机的内存 dump 技术

基于 Android 源码，修改 Dalvik 虚拟机实现，利用 Dalvik 在应用程序运行时的字节码加载过程，以及运行时解密后内存刷新函数实现过程，增加字节码向外存的输出过程，可有效的获取到加壳应用解密后的字节码。

现有的脱壳技术都存在一定的不足，上述脱壳技术的优缺点归纳如表 1 所示：

表 1 常见脱壳技术优缺点对比

Tab. 1 Relative merits of the Shelling methods

脱壳技术	优点	缺点
动态调试	对目标进程的可控性高	无法自动化
基于存储状态的静态还原技术	速度快，可自动对批量应用进行	准确率低，对于复杂应用无法完整还原
基于进程附加的自动脱壳技术	可完整有效的对加固进行脱壳，且对多种加固方法具有兼容性	无法对具有反调试技术的加固应用进行脱壳
基于 Dalvik 虚拟机的内存 dump 技术	完全自动化，结果准确	源码编译复杂，对加固技术变化无法快速响应，时间代价大

1.2 加固技术实现原理概述

以 Android 应用运行原理的分层结构分析加固应用程序，其构架具体如图 1 所示。

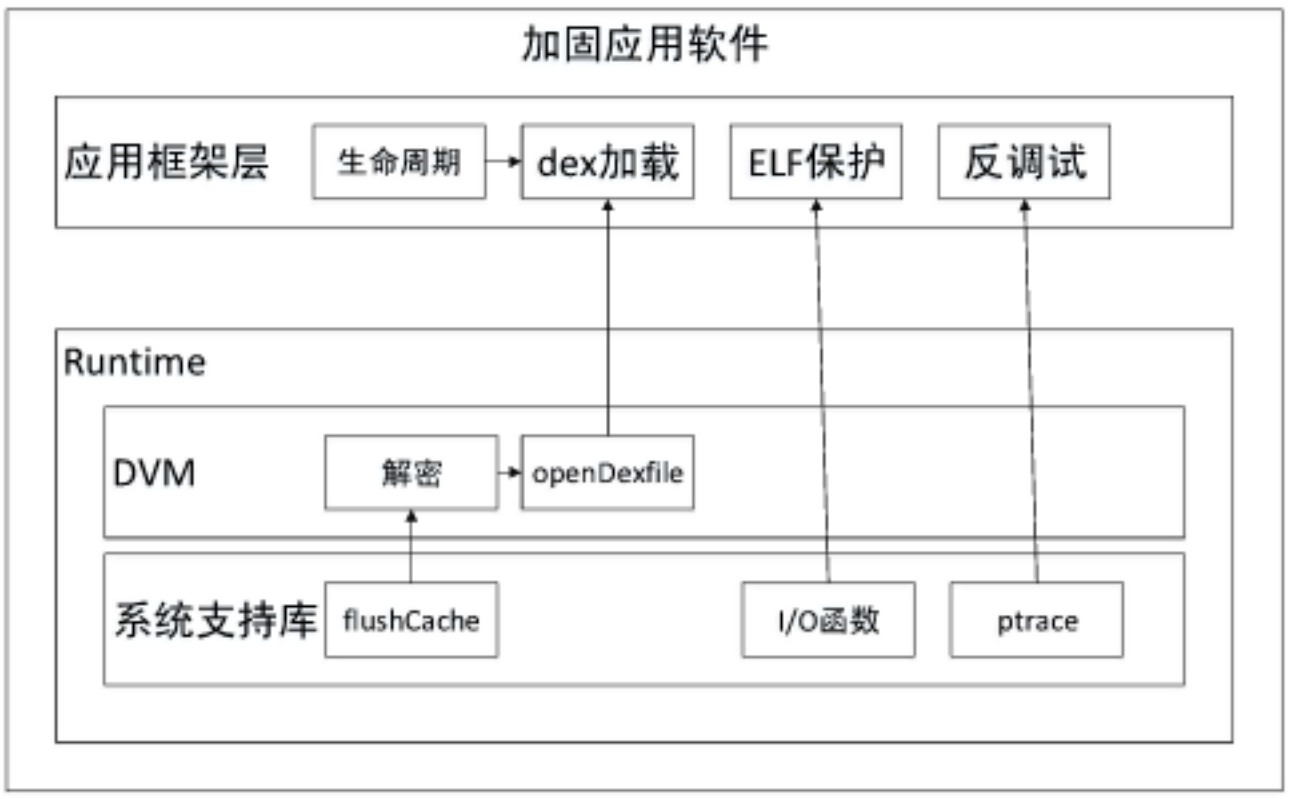

图 1　加固应用构架图

Fig. 1　The reinforced application architecture

Android 应用程序运行时在系统中以进程形式体现，进程创建完毕后从外存将 dex 文件映射到进程的内存空间中，完成一系列的初始化后，由应用框架层抽象应用的主要运行逻辑，在 DVM、支持库中对应用框架层进行具体实现[2]，从而完成程序的运行。

而加固应用利用 Android 程序运行原理，在壳进程启动后，主动解密和加载被保护程序字节码，利用应用框架层中完成进程的生命周期；在运行时层实现了具体的安全加固技术，增加自我保护技术等，以实现对应用程序的加固。

2　一种采用字节码还原技术的针对加固应用自动化脱壳系统

2.1　脱壳系统总体构架

本文所设计的基于 Hook 和字节码还原的加固应用自动化脱壳系统，采用 B/S 总体架构，功能架构总体分为四层：应用层、样本处理层、数据处理层、样本预处理层。主要功能模块包括：样本特征提取模块、数据处理模块、脱壳方案执行模块以及结果展示模块。具体架构如下图 2 所示。

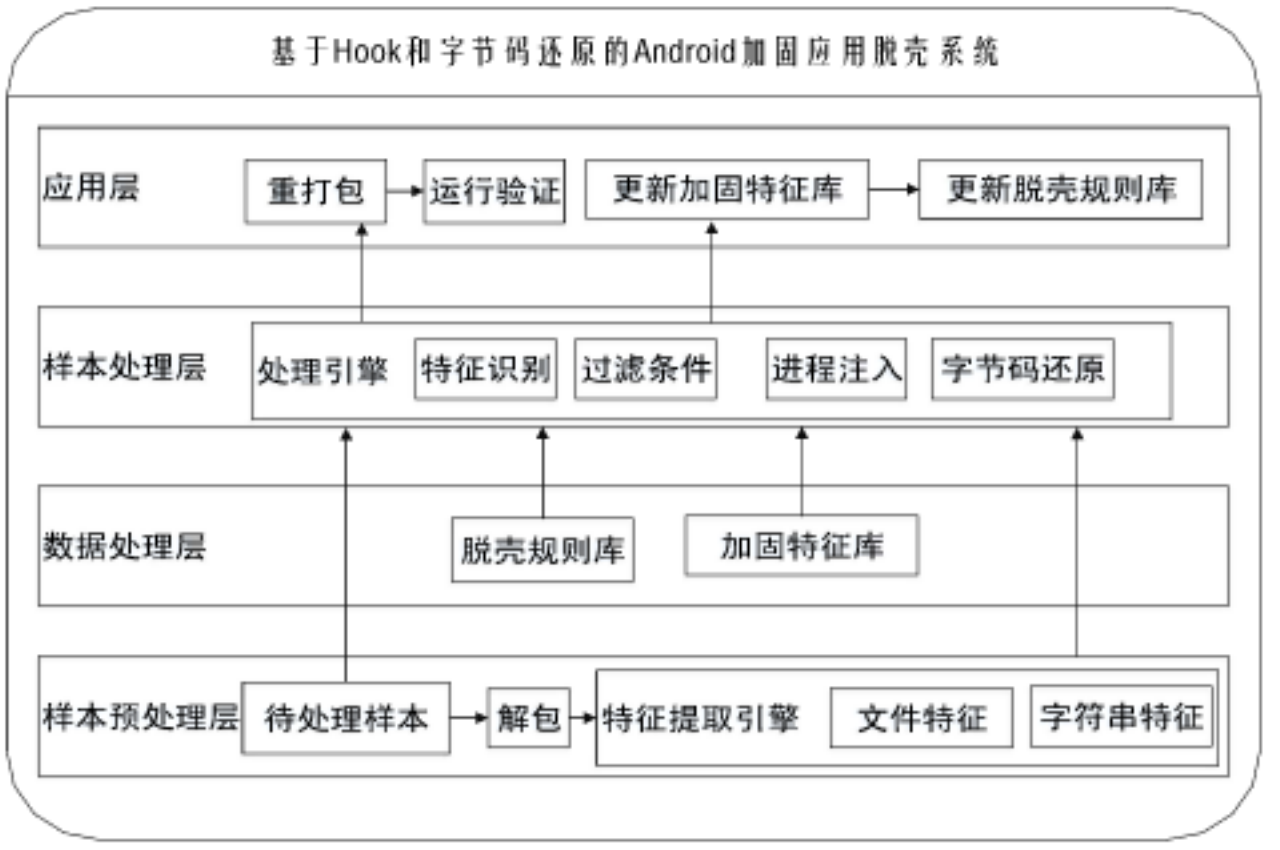

图 2　基于 Hook 和字节码还原的 Android 加固应用脱壳系统模型

Fig. 2　System Architecture Diagram

上述系统各层描述如下：

样本预处理层：根据样本解包后的样本文件，对其文件结构和文本文件中的字符串进行采集，生成加固特征，为样本处理层提供样本支持。

数据处理层：以加固方案为区别存储脱壳方案库，借助加固特征库对指定加固特征进行匹配，向样本处理层提供脱壳方案。脱壳方案库与加固特征库同步，可进行更新规则。

样本处理层：根据加固特征匹配脱壳方案库中的脱壳方案，方案涉及样本所采用的加固方案、过滤条件、Hook 位置等规则。将样本自动安装到测试设备中并启动，开始 Hook 目标进程动态字节码还原，为应用层提供结果数据。

应用层：对解密后字节码文件进行重打包、重签名后，执行验证其运行情况，并记录动态日志，最后人工判定是否需要更新脱壳方案库。

2.2 加固特征

应用程序被加固后，会在文件和实现逻辑上形成加固方案的特征，具体如下：

1．文件结构(Files Structure)，记作 RFS

对 apk 文件进行解包后，生成文件结构，其中包括 res、assets、lib、dex 文件等。其中 lib 和 assets 中或保存着特有的加固支持库和密文原字节码文件，由于加固方案不同，这些文件和目录结构会有区别。

2．应用名称(Application Name)，记作 RAN

加固方案由于实现原理所限，需要实现自定义 Application 管理应用生命周期，并在 AndroidManifest.xml 文件中配置 Application 详细名称，不同加固方案具有不同且固定的 Application 名称。

3．关键代码字符串(Key Code)，记作 RKC

加固方案中壳逻辑的实现中，会实现应用的生命周期，完成主要支持库的加载和执行，根据加固方案的不同其实现方式也有区别，但每种加固方案的实现基本固定，所以会存在特定的指令，如 Native 函数命名、生命周期函数中函数调用，将这些特定指令以字符串形式形成特征。

2.3 脱壳限定条件规则

对具有不同加固特征的加固应用执行脱壳方案时，需要根据加固特征匹配不通的脱壳规则，规则包括：

1．解密位置(Location of Decryption)，记作 ULD

应用被加固后，会对应用的字节码文件进行加密处理，最终在应用运行时会对字节码必须为解密后的明文，通过定位到解密位置获取明文数据存储位置，才可以进行进一步的字节码还原。

2．类加载器类型(Type of ClassLoader)，记作 UTC

由于 Android 系统运行原理的原因，加固应用在运行时需要通过 ClassLoader 向 Dalvik 虚拟机提供数据解析逻辑的支持，ClassLoader 存储了被执行应用的数据偏移。根据加固类型的不通所使用的 ClassLoader 也不一样，为对加固方案进行适配，以此作为判定标准之一。

3．字节码内存结构(Dex Structure in Ram)，记作 UDS

字节码被映射到进程内存空间后，根据加固方案类型的区别，会以不同的方式存储，以处理方式的区别分为三类：全密文存储、动态加解密、数据抽取动态还原。

4．增量阈值(Limit of Volumes Change)，记作ULV

ULV 指应用加固后的体积增量阈值。对应用程序加固后，应用程序的体积增量体现在加固方案所依赖的支持库和壳字节码文件上，由于在对原字节码文件加密过程中会进行压缩，会使得加固后应用体积会与原应用体积相近，通过判断体积增量阈值，可初步判定所得到的字节码文件是否正确。

3 脱壳系统模块功能及算法设计

3.1 样本特征提取模块

完成对待处理样本的加固特征提取，由于对不同加固类型的样本进行脱壳，在设定限定条件、定位解密位置时需要针对加固方案类型进行自动调整，具体提取过程如下：

首先对待处理样本进行解包，然后提取加固特征：RFS 、RAN 、RKC 。生成文件结构树 $RFS=\{p, f\}$， f 为树 RFS 中存在的结点文件， p 为 f 在 RFS 中的位置；记录 Application 名称为 RAN ；提取应用生命周期函数 attachBaseContext 中实现的内容，记行代码数为 n ，代码字符串 s ，对应代码位置 l ，生成关键代码字符串特征表 $RKC=\{n,(l,s)\}$ 。首先定义了 R_{map} 特征表、特征校验和 R_c 以及加固特征序列 R ，然后生成特征表 $R_{map}=\{RAN,RFS,RKC\}$ ，并且计算样本特征表的校验和得到 $R_c=checksum(R_{map})$ ，最后定义样本加固特征序列 $R=md5(R_{map},R_c)$ 。

3.2 数据处理模块

维护了两个可更新数据库：加固特征库和脱壳方案库，加固特征库记录着已知的加固方案特征，包括 RFS 、RAN 、RKC ，脱壳方案库记录着脱壳方案的限定条件规则，包括 ULD 、UTC 、UDS 、ULV 。为处理引擎提供方案支持之外还提供了特征匹配，可以加快匹配脱壳方案过程。出于从系统拓展和升级考虑，采用对数据库分别管理的模式。

3.3 脱壳方案执行模块

脱壳方案执行模块主要分为五个部分：匹配加固特征、设定 Hook 位置、解析字节码偏移、还原字节码、判定体积增量。其流程如图 5 所示。

1．匹配加固特征

在加固特征库中查询加固特征序列 R ，匹配成功后，通过特征库与脱壳规则库的关联关系，获取对应的脱壳规则 $U=(ULD,UTC,UDS,ULV)$ 。若未匹配成功，则返回 $U_{default}=0$ ，后续部分将按照预设 Hook 位置、还原方式进行脱壳。

2．设定 **Hook** 位置

加固方案中应用了多种动态安全技术，如反调试、防止 dump、动态解密等，在向进程附加时会被加固进程检测并拦截，所以根据 ULD 和 UDS 值，使用 Hook 对进程空间中 libdvm、libc 以及应用框架层的 handleBindApplication 进行选择性 Hook。

进行多点 Hook 从两方面考虑。一方面，通过使用 Hook 可以有效的绕过反调试技术，注入到目标进程，另一方面，目标进程的 libdvm、libc 等库以及应用框架层中相关函数涉及到应用程序的运行和加固方案的执行。

libdvm 中 dvmDexFileOpenPartial、openDexFileNative 完成对字节码文件的加载[3]，将字

节码文件映射到进程内存空间后对其进行 opt 优化；应用运行过程中 ClassLoader 调用 defineClassNative 函数，完成类的初始化和解析工作，过程中会将字节码文件数据进行解析，加固进程在这时需要对数据进行解密和偏移修复；在字节码文件被解析时系统通过调用 dexGetClassDef 获得 ClassDef 结构，而 ClassDef 定义了一个 Java 类的所有数据的内存偏移，其数据体现在字节码文件中。

由于 Android 内核原理所限，在修改内存空间数据后，必须调用 libc 中 fulshCache 函数，对内存数据进行刷新，所以在加固进程对内存数据进行加解密操作的最后，一定会刷新缓存完成数据刷新。

3. 解析字节码偏移

在 Android 平台中，对一个应用程序类的加载(dex 文件加载)，默认使用 PathClassLoader，完成加载和解析。通过分析可知，其父类为 BaseClassLoader，在其实现过程中，涉及到 DexPathList 类，这个类是调用 dexfile 的 native 实现完成对 dex 文件的加载和 findClass，并且构造 Element 结构数组，将进程运行时所需要的 lib、res 以及 dexfile 信息进行记录。BaseClassLoader 继承自 java 类 ClassLoader，从而继承 ClassLoader 的回调逻辑，实现了对类的查找、解析等逻辑。

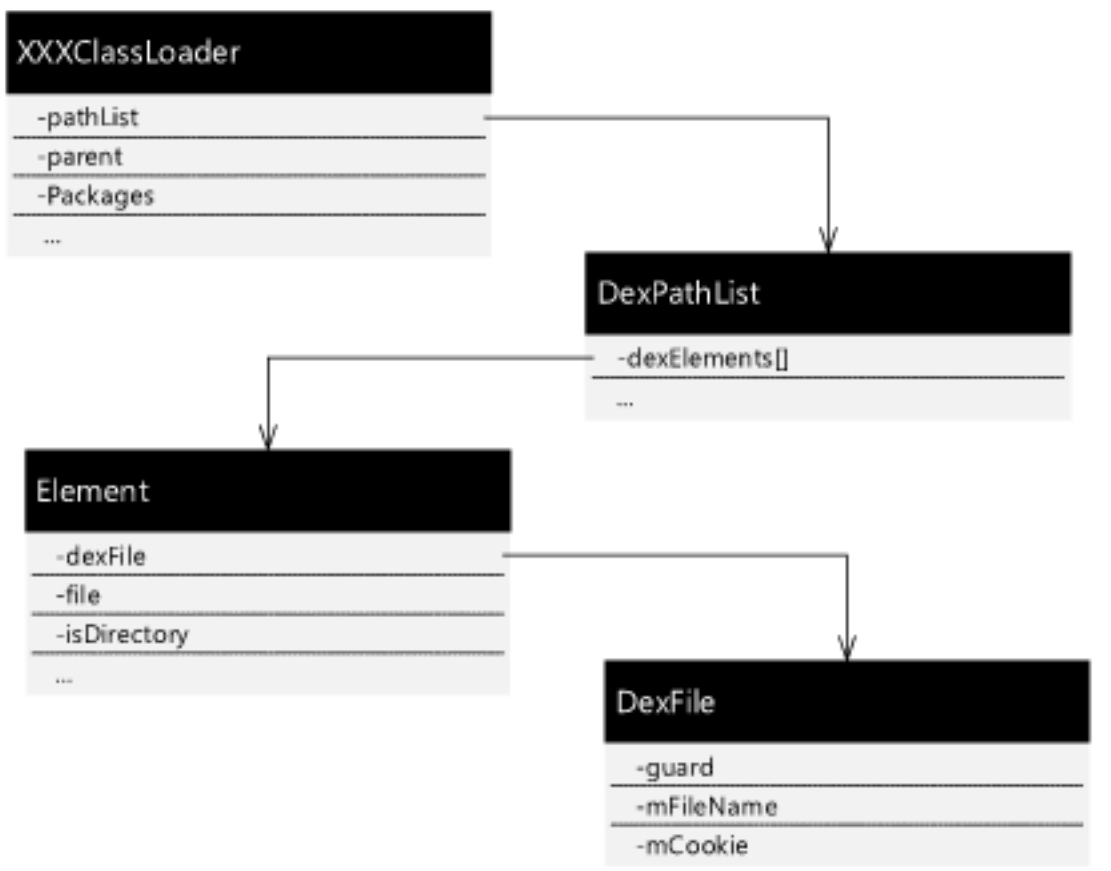

图 3　DexFile 关系图

Fig. 3　The relation diagram of DexFile

在 ClassLoader 中保存了 dex 文件在内存中的偏移，即图 3 所示的 mCookie。根据UTC，判断加固方案是否使用了自定义 ClassLoader，结合 Hook 追踪程序执行获取到目标 ClassLoader，然后通过上述结构，解析到 mCookie，就获得了字节码在内存中的真正位置。

4. 还原字节码

dex 以数据连续的形式被加载到内存的过程中，由 dvm 对其进行 opt 优化，被优化处理后的 dex 文件与 dvm 中定义的多个文件结构相关联，从而形成 dex 文件在内存中的结构，包括 DexOrJar、RawDexFile、DvmDex 以及 DexFile，其继承关系如图 4 所示：

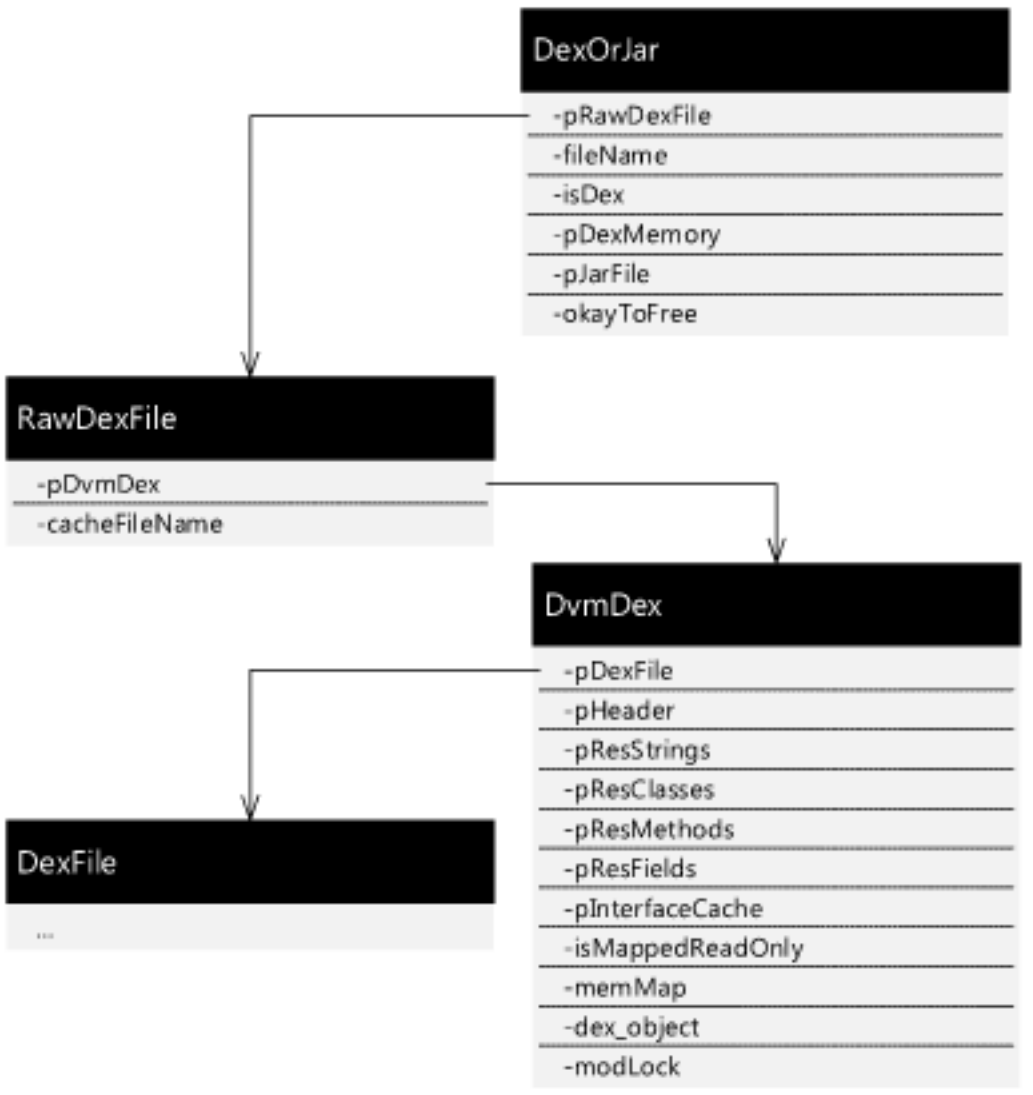

图 4　dex 内存结构继承关系图

Fig. 4　Inheritance diagram of the dex in RAM

其中 DexFile 就是 dex 文件的格式定义，从严格意义上讲，DexFile 是 dex 经过 opt 优化后的 odex 文件格式定义[4]。在 dvm 解析 dex 时，通过 DvmDex 生成 ClassObject，使得程序运行，ClassObject 是 Java 类在虚拟机中的实体，也就是源码中所定义的类的表现，也是 Android 应用程序的基本单元，其包涵了某一类的方法表、成员表、代码信息等。换而言之，ClassObject 是类在内存中的真实形态。DvmDex 作为中间载体将，ClassObject 与 DexFile 连接起来。应用运行中 dvm 解析 ClassObject，读取 DexFile 中的数据为上层做支持。

在一些加固方案中会使内存保护技术，改变 dex 在映射到内存后的存在形态，抽离其中的部分结构，打碎原本在内存中连续的数据，使 dex 文件在内存中不处于连续的形态[5]。这里通过UDS来确定其在内存中的形态并进行解析，解析步骤分为如下：

步骤一：解析 DexFile 头；

步骤二：定义 DexFile 结构体，对头进行保存；

步骤三：遍历 Header 中结构体偏移，对解析的目标数据进行保存，若遍历到的数据不存在，采取向 ClassLoader 动态 defineClass 的方式，使加固进程对目标数据进行解密；

步骤四：将新定义的 DexFile 结构体写到文件保存成字节码文件。

判定体积增量

对取得的字节码文件，与解包过程中生成的资源文件、AndroidManifest.xml、assets 等文件进行重打包，计算重打包后文件大小，并与待脱壳样本大小进行差值取绝对值，最后将所得差值与ULV进行比较，若差值不大于ULV则初步判定所得到的字节码文件属于原程序字节码，脱壳成功。否则，所得到的字节码可能为壳应用的字节码文件。

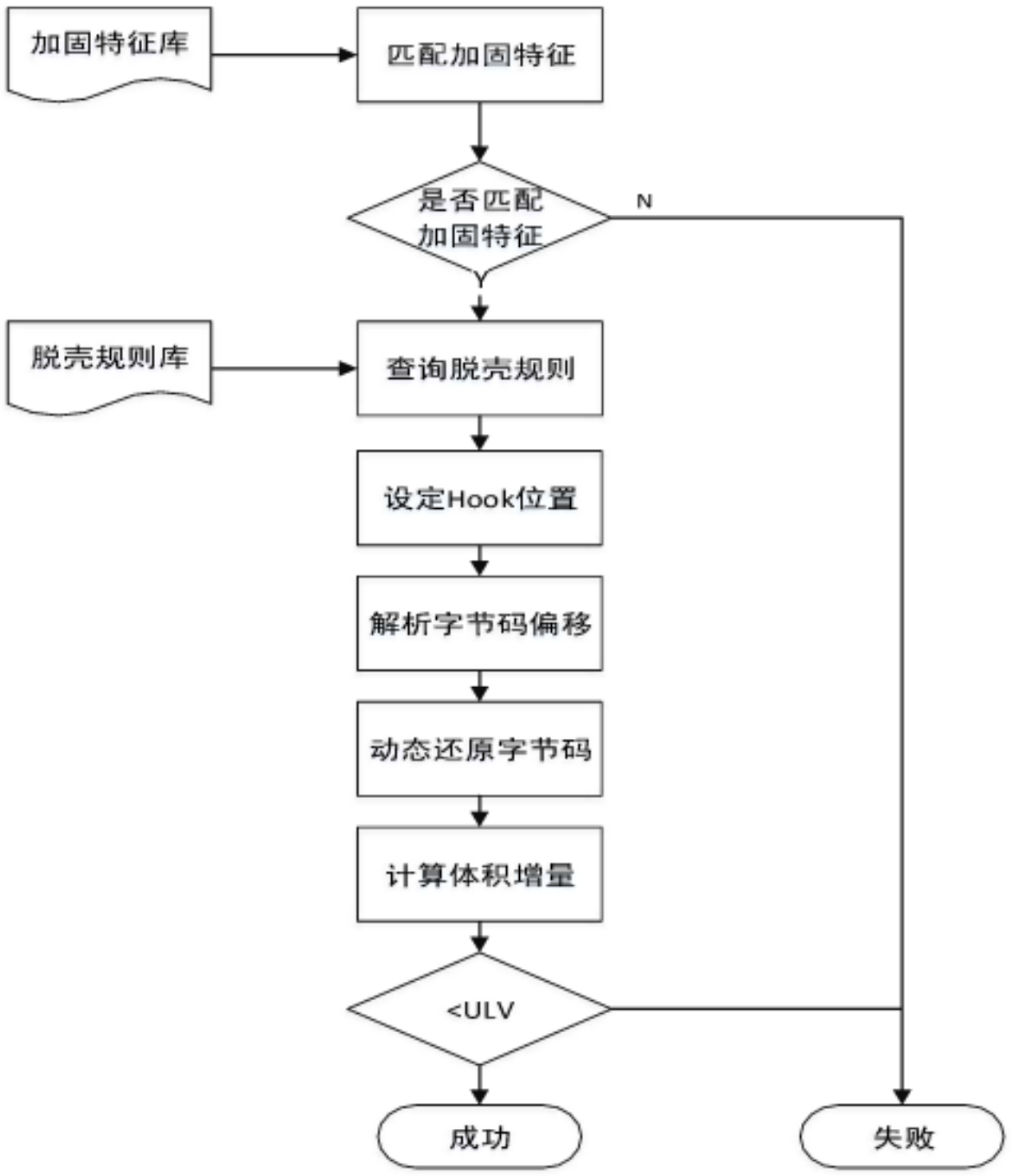

图 5 脱壳方案执行流程图

Fig.5 Shelling module process diagram

3.4 结果展示模块

若脱壳成功，则提供已脱壳应用，并将重打包应用执行日志以详情的方式展示给用户；

若脱壳失败，需要人为分析加固应用的特征，并形成脱壳方案规则，在规则库和特征库中建立相应条目，进行库的更新。

4 脱壳系统实验及评测

4.1 实验环境及测试样本

1) 实验环境

Android 4.4.2

Sumsung n900

ArmV7 指令集

2) 测试样本

由于本方案是以 Android 平台加固应用作为基本测试单位，采用 2015 年 6 月下旬主流加固厂商的加固方案所加固的 Android 应用。

3) 对比测试

加固应用分析解决方案工具 Zjdroid[6]。

4.2 总体测试报告

1、 实验测试一

本节将对实现的基于 Hook 和字节码还原的加固应用自动化脱壳系统进行系统验证。首先，构建加固特征库、脱壳规则库；在实验室现有工具的基础上，获取了 100 个采用不同种类的加固方案加固的应用样本，其中 2 个方案 A 加固样本存在程序运行错误，应用在执行过程中会退出；部署 4 台智能终端作为测试机。通过不断的调整系统限定条件，使得成功比例达到最高。最终的实验数据如下表 2 所示。

表 2 测试结果

Tab. 2 Test results

加固种类	加固应用数量/个	成功脱壳数量/个	攻击成功比例/%
A	24	24	100%
B	15	10	67%
C	30	30	100%
D	21	21	100%
E	10	10	100%

注：为防止侵权或引起其他纠纷，这里不明确指出加固方案名称

本实验表明，本系统有效的对大部分加固方案进行脱壳，在针对加固 B 方案上出现的脱壳失败情况，经分析可能有如下两方面原因：

- 样本应用加固方案进行了部分技术更新，而在系统特征库和规则库中没有匹配；
- 系统在设定限定条件、Hook 位置，被加固方案中安全技术检测到。

2、 实验测试二

以脱壳工具 Zjdroid 进行对比测试，结果表明 Zjdroid 对加固方案 B 和 C 无法有效的进行脱壳，其最终实验数据如表 3 所示。

表 3 测试结果

Tab. 3 Test results

加固种类	加固应用数量/个	成功脱壳数量/个	攻击成功比例/%
A	24	22	92%
B	15	0	0%
C	30	0	0%
D	21	21	100%
E	10	10	100%

注：为防止侵权或引起其他纠纷，这里不明确指出加固方案名称

本实验表明，在针对加固 B 方案上出现的脱壳失败情况，经分析可能有如下两方面原

因：

- 样本应用加固方案进行了部分技术更新，而在系统特征库和规则库中没有匹配；
- 系统在设定限定条件、Hook 位置，被加固方案中安全技术检测到。

5 结束语

本文就目前移动安全所面临的加固应用分析问题，提出基于 hook 的逆向方案解决方案，设计并实现了逆向方案原型，旨在有效的分析加固应用，起到了很好的效果。但方案基于 Android4.4.2 版本设计，对部分加固应用存在适配性问题，在接下来的研究当中，考虑增强现存方案的适配性，以期达到更好的检测效果。

[参考文献] (References)

[1] P. Schulz. Code Protection in Android[OL]. [2012]. http://net.cs.uni-bonn.de/fileadmin/user upload/plohmann/2012-Schulz-Code Protection in Android.pdf

[2] Protsenko M, Kreuter S, Muller T. Dynamic Self-Protection and Tamperproofing for Android Apps Using Native Code[J]. 10th International Conference, 2015, 24(17): 129-138.

[3] Junying Gan, Hui Xiao. The Design and Research of Reversing System Based on Android[J]. Second International Conference, 2012, 8(10): 692-695.

[4] 杨勇义. 基于 Android 平台的软件保护技术研究[D]. 北京：北京邮电大学，2012.

[5] 张志远，万月亮，翁越龙，糜波. Android 应用逆向分析方法研究[J]. 信息网络安全，2013，0(6) ：65-68.

[6] JackJia. Android app dynamic reverse tool based on Xposed framework[OL]. [2014-6-25]. https://github.com/halfkiss/ZjDroid