doi 10.3969/j.issn.1671-1122.2013.07.003

基于 Android 平台的代码保护技术研究

梅瑞[1]，武学礼[2]，文伟平[1]
（1. 北京大学软件与微电子学院，北京 102600 ；
2. 中国石油集团东方地球物理勘探有限责任公司，陕西长庆 710021）

摘 要 文章针对当前基于 Android 平台的代码保护技术的滞后性，介绍了 Android 平台的基本架构，结合当前 Android 平台下基于逆向分析技术的代码篡改和注入等破坏方式，采用静态分析和动态调试机制，研究了基于 Android 平台下的代码逆向分析技术及现有代码保护技术，提出了一种结合字节码填充、标识符混淆和动态代码加载技术的高性能代码保护方案。

关键词 ：软件安全 ；代码保护 ；逆向分析 ；Android 平台
中图分类号 ：TP309 文献标识码 ：A 文章编号 ：1671-1122（2013）07-0010-06

Research on Code Protection Technology based on Android Platform

MEI Rui[1], WU Xue-li[2],WEN Wei-ping[1]
(1.*School of Software and Microelectronics,Peking University,Beijing* 102600*, China;*2.*China Petroleum Group Dongfang Geophysical Exploration Co., Ltd.,Changqing Shanxi* 710021*,China*)

Abstract: This paper target to the lag of code protection technology for Android platform. It presents the basic architecture of the Android platform, along with the damage of code tampering and injection based on the reverse analysis technology on Android platform. It use static analysis and dynamic debugging methods to study code reverse analysis and code protection technology on Android platform. A high-performance code protection scheme was presented which combined with bytecode filling, identifier confusion and dynamic code-loading technology.

Key words: software security; code protection; reverse analysis; Android platform

0 引言

近几年来，关于 Android 平台上应用程序的逆向分析技术研究已经有了很大的进展，由于 Android 应用程序主要是由 Java 语言编写而成，而且 Android 虚拟机 Dalvik 也是根据 Java 虚拟机 JVM 改进而来，于是，此前学术界对于 Java 程序的逆向分析研究和对于 Java 虚拟机 JVM 及其字节码的研究都为 Android 平台上的相应研究提供了丰富的积累和有益的借鉴。

代码保护技术可以被用来保护应用程序的知识产权不被侵犯，也可以保护应用程序的核心技术不被剽窃，还可以防止恶意代码被植入到正常程序中，今天在 Android 平台上病毒肆虐，因此基于 Android 平台的代码保护技术研究就显得更加紧迫和重要。

本文主要研究了Android 平台上的逆向分析技术，并通过这些研究讨论了如何应对这些逆向分析技术，最后提出了一种基于 Android 平台的代码保护技术方案。

1 Android 系统及应用程序架构

1.1 Android运行时的环境

Android 系统运行时，系统环境充当了非常重要的角色，该系统环境分为核心库和 Dalvik 虚拟机两部分。核心库提供了 Java 语言 API 中的大部分功能，同时也包含了Android 的一些核心 API，如 android.os、android.net、android.media 等代码包 ；Dalvik 虚拟机提供了Android 应用程序的最终运行环境，Android 应用程序最终会转换为 Dalvik 字节码在 Dalvik 虚拟机上运行（本地库

收稿日期 ：2013-05-10
项目基金 ：国家自然科学基金［61170282］
作者简介 ：梅瑞 (1984-)，男，安徽，硕士研究生，主要研究方向 ：系统与网络安全 ；武学礼 (1975-)，男，宁夏，工程师，主要研究方向 ：软件测试 ；文伟平 (1976-)，男，湖南，副教授，博士，主要研究方向 ：网络攻击与防范、恶意代码研究、信息系统逆向工程和可信计算技术等。

代码除外，本地库代码在 CPU 上直接运行）。

1.1.1 Dalvik虚拟机

Dalvik 虚拟机是由 Google 公司设计并用于 Android 平台的类 Java 虚拟机，是 Android 移动设备平台的核心组成部分之一。它可以支持已转换为 dex（即 Dalvik Executable）格式的类 Java 应用程序的运行，dex 格式是专为 Dalvik 虚拟机设计的一种压缩格式，适合内存和处理器速度有限的系统。Dalvik 是由 Dan Bornstein 编写的。Dalvik 经过优化，能够在有限的内存中同时运行多个虚拟机，并且每一个 Dalvik 应用作为一个独立的 Linux 进程执行，独立的进程可以防止在虚拟机崩溃的时候所有程序都被关闭。

1.1.2 Dalvik字节码

Android 应用程序的代码被保存在 dex 文件中，而 dex 文件中的代码格式为 Dalvik 字节码，这些字节码都是在 Dalvik 虚拟机中运行的，Dalvik 字节码和 Java 字节码是不同的。Dalvik 字节码是为了让资源有限的手持设备能够更有效率的运行而专门设计出来的。Dalvik 字节码本身并没有优化，主要是因为它是一个通用的运行在 DVM 上的字节码格式，但是一旦应用程序被安装在 Android 系统上之后，dex 字节码文件就会根据架构的类型而被优化，优化后的文件被保存为 odex 文件。这个优化的工作是由 Android 平台提供的一个工具 dexopt 来完成的。DVM 可以运行优化过的代码，也可以运行原来的代码，效果是一样的。Dalvik 字节码是按照一定的语法规则组织起来的，下面简单介绍一下常用的一些语法规则 [1-5]。

1）Dalvik 字节码中的类型

Dalvik 字节码中有两种类型，原始类型和引用类型。对象和数组是引用类型，其他的都是原始类型，如表 1 所示。

表 1 Dalvik字节码原始类型列表

DVM 字节码中的符号	安全浏览技术（SB）	备注
V	Void	只能用于返回值类型
Z	Boolean	布尔SB
B	Byte	DV SB
S	Short	短整形
C	Char	字符SB
I	Int	整形
J	Long	64 位长整形
F	Float	LMI SB
D	Double	64 位双精度浮点数

2）Dalvik 字节码中的对象

Dalvik 字节码中的对象是以 Lpackage/name/ObjectName; 的形式表示的。前面的 L 表示这是一个对象类型，package/name/ 是该对象所在的包，ObjectName 是对象的名字，“;”表示对象名称的结束。相当于 java 中的 package.name.ObjectName。例如：Ljava/lang/String 在源代码中就相当于是 java.lang.String。

3）Dalvik 字节码中的数组

Dalvik 字节码中的数组的表示形式为 [Type，其中 Type 就是一个数据类型。对于多维数组，只要增加 [的个数就行了，例如：[[I 相当于 int[][]，[[[I 相当于 int[][][]。同样的，对象数组也适用此方法，如：[Ljava/lang/String; 表示一个 String 对象数组。

4）Dalvik 字节码中的方法

DVM 中方法的表示形式为：Lpackage/name/ObjectName;->MethodName(III)Z，其中 Lpackage/name/ObjectName; 表示对象的类型，MethodName 表示方法名。III 表示参数（在此是 3 个整型参数），Z 表示返回类型（这里是布尔类型）。方法的参数是一个接着一个的，中间没有任何间隔。一个更加复杂的例子如下所示：

method(I[[IILjava/lang/String;[Ljava/lang/Object;)Ljava/lang/String; 在 java 中则为：String method(int, int[][], int, String, Object[])。

5）Dalvik 字节码中的字段

表示形式：Lpackage/name/ObjectName;->FieldName:Ljava/lang/String;

即对象名 -> 字段名 : 字段类型，其中 Lpackage/name/ObjectName; 表示对象名，FieldName 表示字段名，Ljava/lang/String; 表示字段类型，此处为 String 类型。

1.2 Android应用程序

1.2.1 Android应用程序的构建

Android 应用程序的文件格式为 apk，apk 文件是 Android 最终的运行程序，全称是 Android Package，apk 文件其实就是一个 zip 格式的文件，但后缀名被修改为了 apk，通过 UnZip 等解压软件解压后，可以看到 dex 文件。每个要安装到 Android 平台的应用程序都要被编译打包为一个单独的后缀名为 .apk 的文件，其中包含了应用程序的二进制代码、资源、配置文件等。在 Android 系统中应用程序在安装时，apk 程序会被存放在系统默认的 APP 目录中 [6]。

1.2.2 apk文件结构分析

Android 应用程序是用 Java 编程语言编写而成的，并且发布为 apk 格式的安装文件。apk 文件实际上就是一个 zip 格式的压缩文件，这个压缩文件中包含了一个 dex 文件和应用程序所使用到的各种资源文件，如图片、声音、字符串等。这个 dex 文件被命名为 classes.dex，包含了 Dalvik 字节码格式的应用程序代码。apk 文件在发布之前会被专用签名文件加密，这并不能提升其安全性，只不过是帮助识别和认证授权应用程序的开发者。一个 apk 文件的内部结构大致可以分为如下几个部分 [7, 8]：

1）META-INF目录：用于存储apk包的签名信息，保证apk包的完整性和系统的安全性；

2）res目录：存储应用程序中的各种资源文件，包括图片资源、颜色资源、菜单资源、字符串资源、风格资源等；

3）AndroidManifest.xml文件：描述了应用的名字、版本、权限、引用的库文件等信息；

4）classes.dex文件：java源码编译后生成的Java字节码文件的变换形式；

5）resources.arsc文件：编译后生成的二进制资源文件；

6）lib目录：包含使用NDK开发的本地代码文件；

其中，classes.dex文件为应用程序的执行主体，也是进行代码保护的重点。

2 Android应用程序逆向分析技术

2.1 基于静态分析的逆向分析技术

静态分析（Static Analysis）是指在不运行代码的情况下，采用词法分析、语法分析等各种技术手段对程序文件进行扫描，从而生成程序的反汇编代码，然后阅读反汇编代码来掌握程序功能的一种技术。在实际的分析过程中，完全不运行程序是不太可能的，分析人员常常需要先运行目标程序来寻找程序的突破口。静态分析强调的是静态，在整个分析的过程中，阅读反汇编代码是主要的分析工作。生成反汇编代码的工具称为反汇编工具或反编译工具，选择一个功能强大的反汇编工具不仅能获得更好的反汇编效果，而且也能为分析人员节省不少时间。

Android程序静态分析有两种方法：一种方法是阅读反汇编生成的Dalvik字节码，可以使用IDA Pro分析dex文件，或者使用文本编辑器阅读baksmali反编译生成的smali文件；另一种方法是阅读反编译生成的Java源码，可以使用dex2jar生成jar文件，然后再使用jd-gui等工具软件阅读jar文件的代码。

一个完整的Android程序反编译后的代码量可能非常庞大，要想在这浩如烟海的代码中找到程序的关键代码，主要有以下几种定位代码的方法[9]。

1）信息反馈法

所谓信息反馈法，是指先运行目标程序，然后根据程序运行时给出的反馈信息作为突破口，寻找关键代码。通常情况下，程序中用到的字符串会存储在String.xml文件或者硬编码到程序代码中，如果是前者的话，字符串在程序中会以ID的形式访问，只需在反汇编代码中搜索字符串的ID值即可找到调用代码处；如果是后者的话，在反汇编代码中直接搜索字符串即可。

2）特征函数法

这种定位代码的方法与信息反馈法类似。在信息反馈法中，无论程序给出什么样的反馈信息，终究是需要调用Android SDK中提供的相关API函数来完成的。比如弹出注册码错误的提示信息就需要调用Toast.MakeText().Show()方法，在反汇编代码中直接搜索Toast应该很快就能定位到调用代码，如果Toast在程序中有多处的话，可能需要分析人员逐个甄别。

3）顺序查看法

顺序查看法是指从软件的启动代码开始，逐行的向下分析，掌握软件的执行流程，这种分析方法在病毒分析时经常用到。

4）代码注入法

代码注入法属于动态调试方法，它的原理是手动修改apk文件的反汇编代码，加入Log输出，配合LogCat查看程序执行到特定点时的状态数据。这种方法在解密程序数据时经常使用。

2.2 基于动态调试的逆向分析技术

Android应用程序的动态调试是一种汇编级的调试，基于动态调试的逆向分析技术一般都是指基于汇编级的逆向分析，分析人员在没有应用程序源代码的情况下，动态跟踪和分析汇编代码的执行，查看寄存器和堆栈的内容，从而能够跟踪到程序的执行流程，反馈程序执行的中间结果，在基于静态分析的逆向技术难以找到突破点的时候，基于动态调试的逆向分析也是一种行之有效的方法。

动态调试中对关键信息的定位是通过运行所要分析的程序，观察程序的输出结果来判断程序的关键点，有三种方法可以较好的进行信息定位[10]：

1）日志输出法

通常，在编写应用程序的时候可以通过在代码中插入一些Log函数来输出信息，查看程序的执行情况，当然，也可以在逆向分析应用程序的时候通过在程序中插入一些Log函数来输出重要的信息，从而了解程序的关键信息。代码注入是指首先反汇编Android应用程序，然后在反汇编出来的smali文件中的特定位置处添加相应语法的Log函数的代码，最后再重新打包程序并运行，然后在DDMS的控制台查看输出的调试信息。调用Log函数的smali代码可以按照其语法形式推演出来，Log类的原型是android.util.Log，于是该类就可以表示为Landroid/util/Log;，String类可以表示为Ljava/lang/String;，Log.v()函数的两个参数都是String类，那么该函数就可以表示成Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I，于是只要将这行代码插入到需要分析的代码附

近，将参数设置成需要了解的核心信息，这样在程序运行的时候，用户就可以在 DDMS 控制台中清楚地看到所输出的调试信息。

2）栈跟踪法

栈跟踪法是一种快速定位程序关键点的方法。它主要是通过手动向反汇编出来的 smali 代码中加入栈跟踪信息输出的代码，与注入 Log 日志信息输出代码相比，栈跟踪法不需要知道精确的注入点，而只需要代码点的范围就可以了，而且栈跟踪法输出的信息要比 Log 详细得多。需要注入代码的 java 形式是 ：new Exception(“ print trace ”).printStackTrace(); 将这一句代码转化为 smali 代码格式为 ：

```
new-instance v0, Ljava/lang/Exception;
const-string v1,“ print trace ”
invoke-direct{v0,v1},Ljava/lang/Exception;-><init>(…)V
invoke-virtual{v0}, Ljava/lang/Exception;->printStackTrace()V
```

程序员只需要将这一段代码插入到需要打印信息的地方，然后重新打包签名，在程序运行过程中，就可以在 DDMS 控制台看到打印输出的栈跟踪信息了，这些信息是 warning 级别的，并且他们的 Tag 被设置为 System.error。栈跟踪信息记录了程序从启动到 printStackTrace() 被执行之间所有被调用过的方法，通过这些信息，就详细掌握了程序的执行流程。

3）剖析方法

剖析方法（Profiling Method）是在执行程序的过程中记录下每个被调用的 API 的名称，分析人员只需要查看 API 的调用序列即可知道这段代码的基本用途。Profiling Method 本身就是 Android SDK 中提供的用于调试支持的 API，它存在于 android.os.Debug 类中，提供了 startMethodProfiling() 和 StopMethodProfiling() 两个方法来开启和关闭 Profiling Method。因此也可以通过在反汇编出来的 smali 代码中插入相应的代码来进行系统的开启和关闭，例如 ：

```
android.os.Debug.startMethodProfiling(“ 123 ” );
funcA();
android.os.Debug.stopMethodProfiling();
```

上面这段代码中字符串“ 123 ” 是本次 trace 的文件名，上面的代码在执行后会在 SD 卡的根目录生成一个 123.trace 的文件，这个文件包含了 a() 方法执行过程中所有的方法调用和 CPU 占用情况等信息，可以使用 Android SDK 中提供的 traceview 工具来查看该文件，当然首先需要将该文件从虚拟机中导回到本地，然后才能够查看，查看的显示效果与在 DDMS 中直接调用的效果是一样的。另外，由于该函数需要在 SD 卡中写入文件，因此需要在 AndroidManifest.xml 文件中添加 SD 卡的写入权限。通常，可以在一个 Activity 的 OnCreate() 函数开头处加入启动代码，在 OnStop() 函数结尾处加入关闭代码，从而能够查看该 Activity 从启动到结束所有的调用流程。

3 Android 应用程序代码保护技术方案

如前文分析，针对 Android 平台应用程序的逆向分析主要依赖于如下三个方面 ：

1）由 Java 语言编写的应用程序被编译成了二进制字节码，该字节码为运行于 DVM 虚拟机上的中间代码，依托 Java 平台的反编译技术可较为容易的进行反编译，并且能够还原原始代码的逻辑结果和标识符名称等信息 ；

2）Android 平台的开源特性使得对于 Android 平台的应用程序的反汇编技术得到了较为深入的研究，通过一些成熟的反汇编算法机器工具就可以得到 smali 代码，进而进行逆向分析 ；

3）dex 文件格式早已为研究人员所熟知，因此对 dex 文件的逆向分析即可以获得 Android 应用程序原始的 Jar 文件，进而对 Java 源码文件进行逆向分析。

经过研究，文章针对以上三类逆向分析的途径，提出了如下三种代码保护机制，可以有效对抗 Android 应用程序的逆向分析。

3.1 基于标识符混淆的代码混淆技术

代码混淆是一种常用的代码保护技术，它通过选择一些语义上同等但语法上更加复杂难懂的代码形式来隐藏原有的功能代码，防止程序被逆向分析。为了达到这个保护目的，代码混淆技术就会将原有的程序代码进行同义转换，不改变程序的原有功能，但是让逆向分析更加困难。

代码混淆是一种很早以前就出现的技术，并且在一些系统平台上被广泛地研究和应用过。这种技术不仅可以被用来保护软件的知识产权，而且也被广泛用在恶意软件当中，以躲过防病毒软件的查杀，避免代码被逆向分析。

标识符包括类名、包名、方法名、字段名等，它们都是用有一定标识意义的字符串来表示的，以下 Java 代码中就包含标识符 ：

```
public class MainActivity{
    private String receive(String input){
            ……
}
private String send(String output){
            ……
}
}
```

从这段代码片段来看，即使不知道整个程序的功能，但至少可以知道这个程序的部分功能。也就是说，其中的标识符给出了程序的很多的信息。这些信息可以给逆向分析提供很多提示，从而让需要人工分析的工作量减少。如果标识符混淆使用，就可以使逆向分析出来的代码毫无价值。

混淆器 ProGuard 使用了相似的方法，它使用了最小词汇排序法来代替随机的字符串，例如：{a,b,c,…,aa,ab} 等，这种方法可以减少程序所占用的存储空间。上面那段代码被 ProGuard 进行标识符混淆之后表示如下：

```
public class a{
    private String a(String ab){
……
}
public void b(String ac){
……
}
}
```

仅从这段代码片段，分析者无法得知该代码的功能，这也就达到了让代码难以破解的目的。

3.2 字节码填充技术

字节码填充技术在 x86 平台上是一个众所周知的技术，它可以使反汇编器出现严重错误或者使其反汇编出错误的代码，从而使反汇编器失效，这是通过在字节码的特定位点插入一些“垃圾”字节来实现的。插入位点的选择需要充分考虑反汇编器所使用的反汇编算法，从而达到最优效果。目前 Android 平台上主要有两种反汇编算法，一种是线性扫描，一种是递归遍历。还有一个需要考虑的问题是“垃圾”字节在被插入之后不应该被执行，否则程序的功能就会发生改变，因此垃圾字节需要被插入在一些永远不会被执行的代码块中。

被“垃圾”字节影响到的指令的数目取决于所使用的“垃圾”字节的具体情况，如果一个“垃圾”字节是某个指令的开始，而这个指令又需要多个字节，那么这个垃圾字节就有可能影响接下来的好几条指令。例如 packed-switch-data 指令就是一个很不错的选择，因为它的长度是可变的，但是并不是每一个反汇编器都会处理这个指令，因为这是个很特殊的指令，它是一个虚设指令，不会被执行，仅仅只是存储一些其他代码将会用到的数据。

通过测试发现所有的逆向分析工具都或多或少的受到“垃圾”字节的影响，并导致部分功能或者全部功能失效，一些工具在遇到“垃圾”字节时就直接崩溃了，还有一些工具也可以在崩溃时转入异常处理。当然，逆向分析工具仍然可以通过更优化的反汇编算法来绕过“垃圾”字节码的填充。实际上，寻找更好的反汇编算法在 Dalvik 虚拟机架构中还是比较容易的，因为 Dalvik 并不支持间接跳转，如果支持间接跳转的话，就可以根据寄存器的内容来决定跳转的位置，这种情况下，想要通过静态分析来确定分支的地址就会非常困难了。

从目前来看，“垃圾”字节填充技术对于代码保护还是很有效的，分析人员必须手工检查是否存在“垃圾”字节，这是很耗时间的，因此“垃圾”字节填充目前还是一个比较有价值的代码保护技术。

3.3 动态代码加载技术

一个理想的代码保护技术应该能够将 Android 应用程序的所有元数据信息隐藏起来，或者直接就将其转换成无法解释的字节码，但是如果这样的话，Dalvik 虚拟机如何解释字节码将是个巨大的挑战。一种比较接近理想的方法是通过打包或者代码加壳来实现，这种方法在 x86 架构下已经得到了广泛的应用。

加壳软件可以将任意的二进制运行文件转换成一种中间形式，这种中间形式不能够被直接逆向分析。在程序运行时，应用程序应先释放脱壳代码或成为解密代码，将中间形式的代码解密出来，在内存中还原成原来的代码和数据然后再运行。分析人员从已加密程序中得不到任何有用的信息，必须分析解密代码后才有可能将原来的代码和数据解密出来，进而真正了解程序的功能和原始代码。

这种方法需要两个组成部分，一个是已加密过的应用程序代码包，另一个就是解密函数，而在 Android 平台上，已加密的代码包就是一个被加密的 dex 文件。

Android 平台上的代码动态加载技术可以通过两种方式来实现，一种是通过 Android 应用层的 Java 代码来实现，另外一种就是通过 JNI 接口利用 C++ 本地代码来实现。

3.3.1 通过Android应用层实现的动态代码加载

代码的动态加载可以通过 Android 的应用层用 Java 语言来实现，Android 中有两个类可以实现类的动态加载，分别是 PathClassLoader 和 DexClassLoader，这两个类都是继承自 BaseDexClassLoader 类，这两个类最终加载的都是经过 Dalvik 优化过的 dex 文件，即 odex 文件。

PathClassLoader 类只能加载已经安装到系统里面的 apk 应用程序安装包，也就是 /data/app 目录中的 apk 文件，加载到其他位置的文件在加载时会出现 ClassNotFoundExceptiong 异常。

DexClassLoader 和 PathClassLoader 的最大区别就是可以指定优化过的 dex 文件的路径，这个区别决定了 DexClassLoader 可以加载未安装的 apk 文件。因此可以使用

DexClassLoader 类来进行代码的动态加载，在运行时载入自定义类的过程如下：

1）获取待载入的包含自定义类的 dex 文件，可以是设备本地的文件，也可以是从互联网上下载的文件，还可以是在程序中以资源的形式存在的文件；

2）把获取到的 dex 文件保存到程序的内部存储空间中；

3）通过 DexClassLoader 类加载器来解析并优化前面的 dex 文件；

4）通过 DexClassLoader 的 loadClass 函数来载入类；

5）通过获得到的类的 newInstance 函数来生成一个新的对象。

通过上面的这种机制，就实现了在运行时从程序的资源文件中加载 dex 文件并将其中的类动态加载进入内存，从而使用其中的方法和数据的目的。

3.3.2 通过JNI本地代码实现动态代码加载

在 Android 应用层可以使用 DexFile 类来加载 dex 文件到当前进程中，该类的公开方法只能操作存储在文件系统中的 dex 文件，这就意味着原始 dex 文件必须要先存储在文件系统中，然后才能加载到 Dalvik 虚拟机中运行。在这种情况下，分析人员可以很容易的从存储系统中拷贝出这个原始的 dex 文件，从而突破这项技术的封锁。另一个潜在的问题是 dex 文件在运行前会被优化成 odex 文件，并存储在文件系统中。而 odex 文件被文件系统保护起来，没有特殊的权限是不能删除的，因此同样会存在信息泄露的危险。

DexFile 类提供了另外的一个方法 openDexFile(byte[])，它支持程序从任意的二进制数据流中加载一个 dex 文件，而不是从一个文件系统的文件中进行加载。但是这个方法是一个私有方法，因此并不能从 Java 代码中直接去调用它。

为了规避这个限制，可以使用 Dalvik 虚拟机提供的 JNI 调用接口，JNI 接口提供了从 Java 代码中调用 C/C++ 代码的方法。DexFile 类本身也同样使用了 JNI 来实现它自身内部的一些操作，例如上面所说的 DexFile 类中的 openDexFile(byte[]) 方法就是通过 JNI 接口调用 libdvm.so 中的同名函数来实现的。dex 文件加载器可以在本地代码中实现，这个加载器中提供这样一个方法 openDexFile（byte[]），可以导出同名的私有方法，从而突破了不能在 Dalvik 虚拟机中直接调用该私有方法的限制，如图 1 所示。

这个方法有两个优势，一个是不需要存储已解密的 dex 文件到文件系统中，可以直接通过一个字节数组来传递 dex 文件的内容。通过这个方法，dex 文件的内容只能够在内存中才能够获得，这样就避免了泄露的危险；另外一个优势就是这个方法不会产生 odex 优化文件。

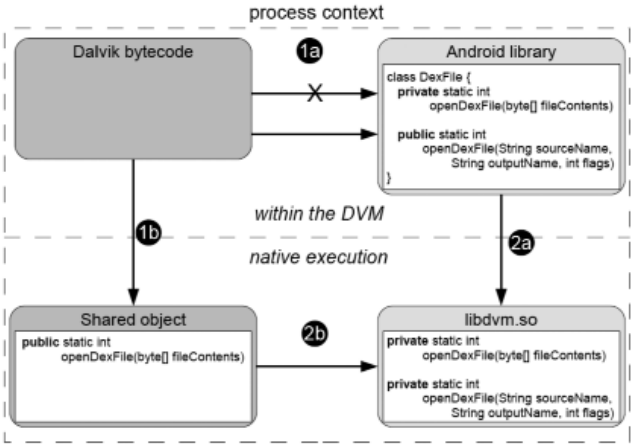

图1 通过JNI接口实现的代码动态加载示意图

4 结束语

本文首先对 Android 平台的系统架构进行了介绍，剖析了常用的 Android 应用程序的逆向分析方法，然后有针对性地提出了几种对抗逆向分析的代码保护技术，并对这些代码保护技术的原理和应用场景进行了分析。文中所研究的代码保护技术方案在实际操作过程中具有很高的可行性，可以将核心代码放入到动态类中来实现。但是，如果动态类中的代码太多，可能会影响程序的运行速度和性能，因此，下一步可以考虑将动态类加载用 JNI 层技术来实现，或者将核心代码编写成本地库由 JNI 层直接调用，当然这里都是假定通过 JNI 接口调用的本地库中的 C/C++ 代码是远远比 Java 代码要难逆向分析的。实际上，针对 C/C++ 代码已有的一些保护技术也可以充分应用到本地库代码中，从而保护本地库代码的安全。（责编 杨晨）

参考文献

[1] 张健，舒心，刘威等 .2010 年计算机病毒疫情调查报告 [J]. 信息网络安全 ,2011,(09):70-73.

[2] 刘威，刘鑫，杜振华 .2010 年我国恶意代码新特点的研究 [J]. 信息网络安全 ,2011.(09):222-225.

[3] 网秦 .2011 手机安全报告 [EB/OL]. http://www.netqin.com/upLoad/File/baogao/20120112.pdf,2012.

[4] 四川大学 .Android 编程 [EB/OL]. http://developer.android.com/guide/developing/building/index.html,2013.

[5] 徐凡 .webkit 在 win32 下的编译规则 [EB/OL]. http://www.netmite.com/android/mydroid/dalvik/docs/dalvik-bytecode.html,2013 .

[6] 王世江，盖索林 .Google Android 开发入门指南（第 2 版）[M]. 北京：人民邮电出版社 .2009.

[7] 王向辉 .Android 应用程序开发 [M]. 北京：清华大学出版社 .2010.

[8] 林城 .Go0gle Android 2.x 应用开发实战 [M]. 北京：清华大学出版社 .2011.

[9] 贾菲，刘威 . 基于 Android 平台恶意代码逆向分析技术的研究 [J]. 信息网络安全 ,2012,(04):61-84.

[10] 徐尤华，熊传玉 .Android 应用的反编译 [J]. 电脑与信息技术 ,2012,(20):50-51.