

基于 Android 系统免 Root 主防方法的研究

刘井强, 李斌, 陈立章, 陈彬

(哈尔滨工业大学(威海)计算机科学与技术学院, 山东 威海 264209)

摘要: 为解决 Android 手机获取 Root 权限系统安全性的降低, 非 Root 授权下 Android 手机安全应用防护效果不理想的一系列问题, 研究了 ART 运行时工作原理、Java 反射机制和 AOP 原理, 设计验证了一套免 Root 主动防御机制, 避免了传统防护应用在系统内核层 Hook 系统关键函数需要 Root 授权带来的安全问题以及在应用层调用 API 效果不理想的现状, 实现了对手机应用的敏感行为拦截, 能够达到增强 Android 手机防护的效果。

关键词: Android 系统; 免 Root 主防; AOP; Hook; 行为拦截; Dalvik

中图分类号: TP309

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2016.00013

Research based on the method of Android system active defense without Root permission

LIU Jing-qiang, LI Bin, CHEN Li-zhang, CHEN Bin

(College of Computer Science and Technology, Harbin Institute of Technology, (Weihai), Weihai 264209, China)

Abstract: In order to solve the problem that the system security is reduced through getting Root permission and that the security of Android smartphone is not effective without Root permission, the working principle of ART, the Java reflection mechanism and AOP principle were studied. The new mechanism was designed and implemented to avoid the security problem caused by the traditional protection application of Hooking system function in the kernel layer and the limitations with calling API in the application layer ineffectively. The interception of sensitive behavior and the protection of smartphone were both achieved.

Key words: Android system, active defense without Root, AOP, Hook, behavior interception, Dalvik

1 引言

Android 是 Google 推出的以 Linux 内核为基础的开源移动操作系统, 采用 4 层分层架构, 从上层到底层依次为应用层、应用框架层、Android 运行时和库文件层、Linux 内核层, Android 系统层次架构如图 1 所示。

近几年智能手机的普及暴露出一系列移动平台安全问题, 尤其是 Android 平台恶意应用泄露隐私的现象更是屡见不鲜^[1], Android 平台敏感行

为检测已经成为安全研究的热点^[2]。入侵检测等被动防御技术不能在系统的脆弱性方面做出预判断, 而主动防御技术能够克服这一缺点, 通过分析敏感操作预测攻击角度, 做出主动防御策略响应^[3]。通过在 Linux 内核层对服务和进程进行注入, Android 安全应用从恶意代码定义出发, Hook 系统关键函数调用, 能够对程序恶意行为做出判断, 但前提是取得 Root 授权。鉴于开放 Root 授权后会降低手机系统安全性, 手机生产厂商反对 Root 授权。安全应用厂商以漏洞利用方式获取

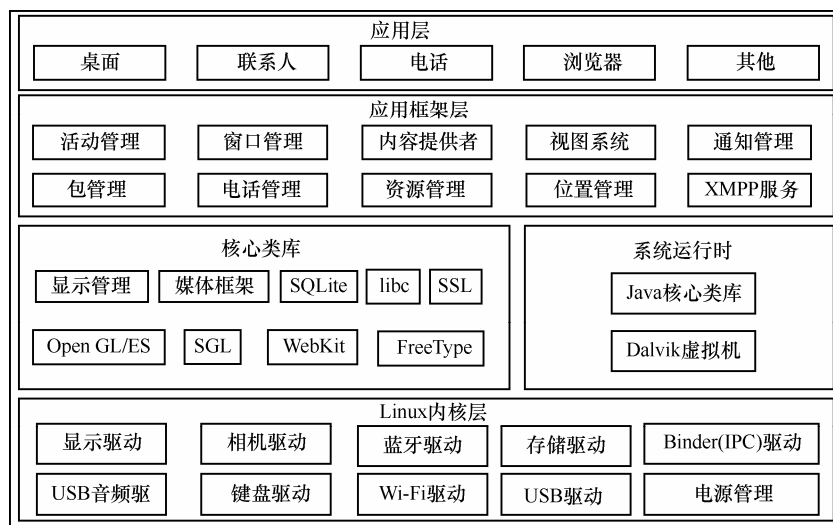


图1 Android 系统架构

Root 授权，但漏洞生命周期有限，使手机获取 Root 授权难度加大。多数 Android 恶意应用在免 Root 授权情况下仍可正常运行，如广告推广、短信推送、隐私窃取^[4]，但在相同条件下，Android 安全应用不能达到有效的手机防护效果^[5]。

一些研究者对 Android 系统的安全防护做出如下尝试。Sanz 等^[6]和 Felt 等^[7]以 Android 应用的权限声明为依据，判断软件是否存在恶意行为，这种方式仅从声明某项敏感权限的角度出发，对软件是否存在恶意行为做出判断，存在不准确性的缺点。Androguard^[8]作为流行的 Android 应用安全分析工具，对 Android 相似性应用采用归一化压缩距离（NCD, Normalized compression distance）^[9]方法，结合函数熵等方法，形式化描述函数特征，仅通过比较应用的相似性，不能从应用自身行为出发做出有效判断。语义分析和特征码分析法^[10]从被动防御角度出发，判断软件是否存在恶意行为。在代码的语义分析方面，对于 PC 端应用的语义检测，已经做了一些理论研究^[11,12]，有利于从该角度出发进行病毒变种检测分析；在特征码分析方面，移动端的防护研究在实践方面取得了一定的进展，Desnos^[13]从应用程序的敏感代码特征出发在相似性上做出比较，是一种被动防御的手段，存在软件防护滞后性的缺点。

本文利用反射机制和面向切面编程（AOP, aspect oriented programming）原理，基于安卓运行时（ART, Android runtime），将 Java 层定义的

方法修改为 Native 层定义的方法，使被修改后的方法调用由 Java 层转为 Native 层，在 Native 层改变原有程序代码的执行逻辑，可以在免 Root 授权情况下动态劫持和判断程序敏感行为。

2 普通手机安全应用隐患分析

2.1 主动防御角度的安全应用隐患分析

从主动防御角度出发，普通手机安全应用防护隐患主要存在以下两个层面。

1) 基于 Linux 内核层的安全防护。以进程注入方式 Hook 系统关键函数调用，需要 Root 授权，因此会降低系统安全性，如恶意程序在手机 Root 授权后可以静默安装其他程序^[14]。

2) 基于应用层的安全防护。通过 Android 系统顶层框架的应用程序编程接口（API, application programming interface），为开发者提供的封装类库在功能上已经被事先定义。如触发短信发送到短信接收前，该时间段内未提供可操作的 API 来检测短信的状态。

2.2 被动防御角度的安全应用隐患分析

当一种木马或病毒开始在网络中传播被发现后，第三方安全厂商（如 360、腾讯等）会对带有该木马或病毒的恶意软件进行行为分析，找出恶意软件行为特征码，形成自己的软件安全病毒库。手机安全防护应用根据病毒库进行特征码匹配，判断软件是否存在安全隐患。被动防御的缺点是只有当用户的利益受到损失后，才能发现恶

意软件的危害, 在安全应用防护方面存在滞后性, 防御效率较低^[15]。

2.3 手机生产厂商和第三方安全厂商的配合问题

随着手机隐患受到大家的广泛关注, 手机生产厂商开始陆续定制自己的安全防护功能, 该功能是一种 ROM 内置的防护措施^[16], 可以深入系统底层对手机系统进行防护, 不需要取得 Root 授权, 能够对手机的敏感行为进行拦截, 但手机生产厂商并没有第三方安全厂商在安全防护方面的技术优势。同时, 手机生产厂商没有为第三方安全厂商提供深入底层的防护接口。两方厂商配合不当, 最终导致用户的手机防护效果不能最大化。

3 基于普通手机安全应用的改进

3.1 改进原理介绍

Java VM 在虚拟栈上运行, 执行.class 文件, Dalvik VM 在寄存器上运行, 执行.class 文件经

dex 工具处理后的.dex 文件。基于 ART 运行时, 调用 libdvm.so 中的方法, 将 Dalvik VM 原有的执行流程和处理方法进行修改, 初始化原 Dalvik VM 的同时, 执行自定义方法, 将 Java 层方法修改为 Native 层方法, 为应用层调用 Native 层方法提供自定义 API 支持, Java VM、Dalvik VM 和 ART 运行时三者关系如图 2 所示。

在应用层调用 JNI 的 Native 方法, 如果该方法没有被显示声明, 程序会从 lib 库中自动加载.so 文件, 在该文件找到并注册 JNI 的 Native 方法。基于 ART 运行时, 改变原有对象方法在 Dalvik 虚拟机的定义, 并为应用层提供修改后的 API, 通过 JNI 回调应用层的统一处理方法。在应用层, 主要应用 Invoke 反射机制, 基于 AOP 原理在 API 统一处理方法中调用并重写 before、after 函数, 形成加固模块, Dalvik VM 的启动时序如图 3 所示。

解压待加固的应用程序安装包, 其主要组成如表 1 所示。

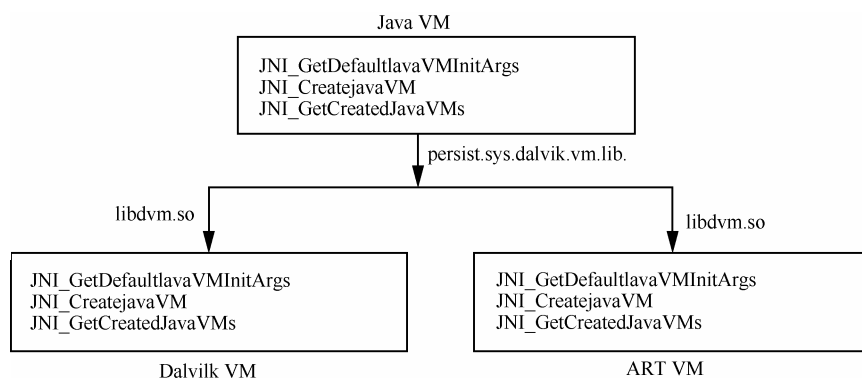


图2 Java VM、Dalvik VM 和 ART VM 的关系

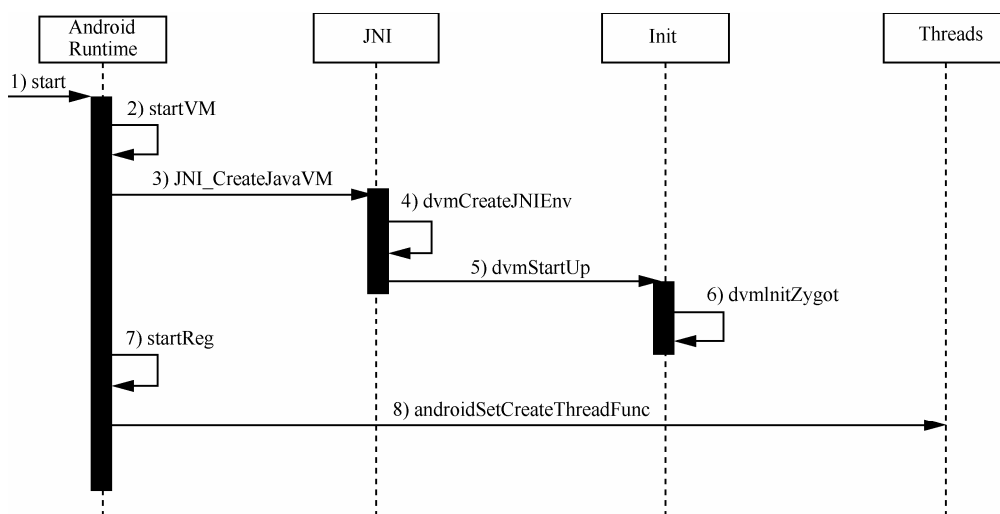


图3 Dalvik VM 启动时序

表1 APK 解压包主要文件组成

序号	文件名	说明
1	AndroidManifest.xml	清单配置文件
2	classes.dex	二进制代码文件
3	assets	附件文件
4	lib	Native 库文件
5	Res	资源文件

在清单文件中修改普通应用程序的 **Application** 入口为加固程序入口，在 **Application** 中全局初始化应用加固模型，添加所有其他类和权限声明，合并资源文件，将 **APK** 重打包并重签名，生成新的加固安装包，应用加固模型如图 4 所示。

3.2 改进模型实现

1) 加固应用执行流程

在用户启动加固应用时，**Zygote** 进程以 **Root** 身份创建并 **fork** 自身，然后加载 **Android** 运行时环境（虚拟机、类库、动态库），切换为应用相应的用户身份，加载应用包并运行应用。安装加固程序包，程序加载加固模块，预定义 **Hook** 请求，

加载并初始化系统服务，当程序向系统服务请求预定义的关键函数时，加固模块拦截该请求并进行敏感行为审核。只有审核通过后，才能继续请求调用系统服务，返回处理结果；否则，屏蔽该请求，执行加固过程如图 5 所示。加固模块执行在应用相应的用户身份下，在 **Android** 运行时层调用 **libdvm.so** 库中的方法反射得到 **Native** 层的方法，没有在内核层 **Hook** 系统的函数调用或者注入服务进程，也没有在 **Native** 层执行需要 **Root** 授权的操作，在拦截函数调用时可以免 **Root** 授权。

2) Java 层调用 Native 层方法

通过查找待 **Hook** 类的方法定义的类型名、方法名、参数信息，依次调用 **Class.forName()**、**Class.getDeclaredMethod()**、**Field.getField()**、**Constructor.getParameterTypes()** 方法，反射得到待 **Hook** 类的对象、方法、函数偏移量和待 **Hook** 方法的参数信息。**Java** 层加载 **.so** 文件，将待 **Hook** 类的对象、方法、函数偏移量和 **Hook** 方法的参数信息作为参数，传入 **Native** 层的自定义 **hookMethodNative()** 方法。**Java** 层调用 **Native** 层方法的执行流程如图 6 所示。

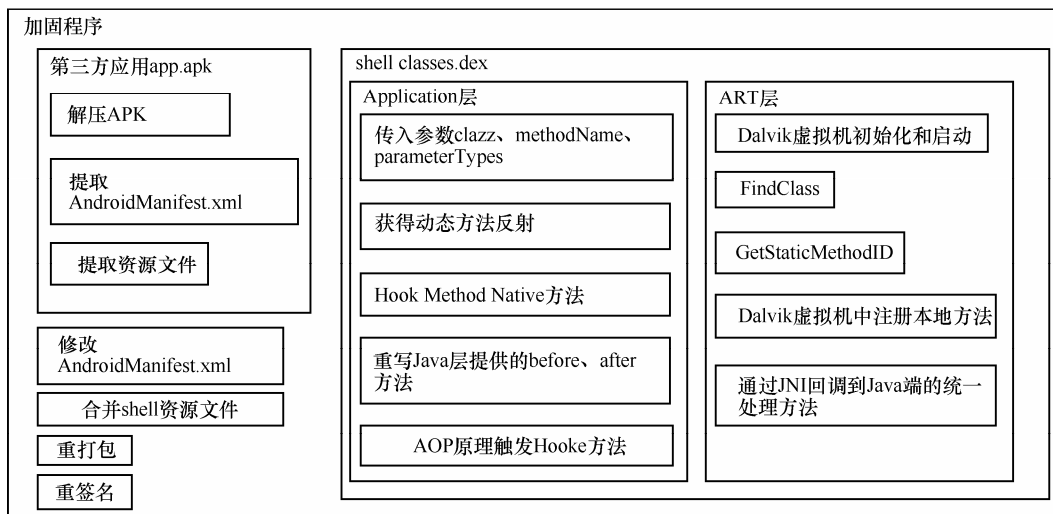


图4 应用加固模型

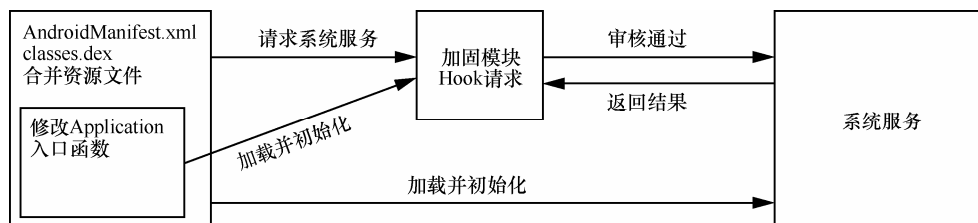


图5 加固应用执行流程

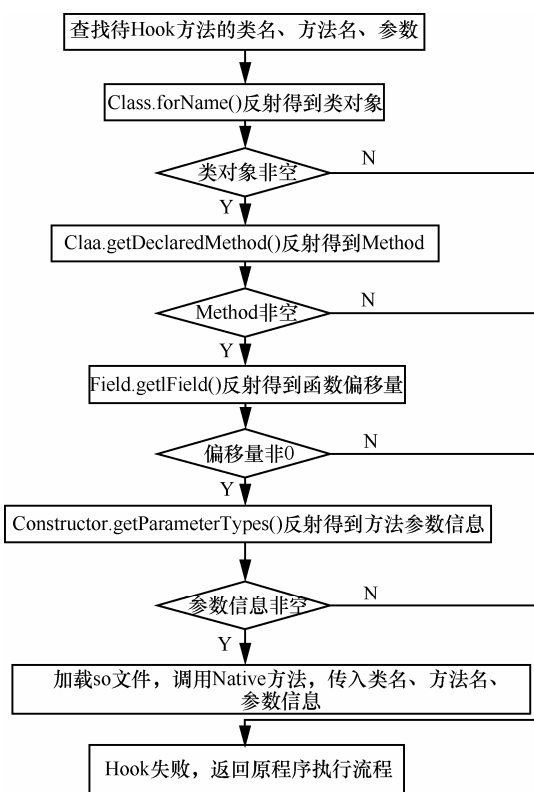


图6 Java层调用Native层方法

3) Native层Hook Java层方法

在Native层调用initNative()，初始化Native层信息，JNI注册Java中的待Hook类和方法。

```
static int register_com_protectdemo_Bridge
(JNIEnv* env) { return env->RegisterNatives (java-
Class, javaMethods, NELEM(javaMethods)); }
```

调用libdvm.so库中的dvmDecodeIndirectRef()方法来获得要注册JNI方法的类对象，调用dvmSlotToMethod()方法，根据函数的偏移量从ClassLoader中获取函数指针，获得待Hook的方法并保存被Hook的方法信息。在ART将方法标记置为ACC_NATIVE方法，让方法属性nativeFunc指向本地的methodCallbackHandler。将Java层代码的待Hook方法声明为Native方法后，执行Java层代码的方法会从Native层查找方法定义，而非在Java层执行代码，从而在Native层实现改变代码的执行逻辑。被Hook方法修改的关键代码如下。

```
SET_METHOD_FLAG(method,ACC_NATIVE);
method->nativeFunc=&methodCallbackHandler;
```

```
method->insns = (const u2*) methodInfo;
method->registersSize = method->insSize;
method->outsSize = 0;
```

此时，待Hook方法已经被成功Hook，代码的执行逻辑转为methodCallbackHandler调用本地的nativeCallbackHandler()方法，本地的nativeCallbackHandler()方法调用dvmCallMethod()方法，执行nativeCallbackHandler()方法，该方法回调Java端的handleHookedMethod()方法，其关键代码如下。

```
dvmCallMethod(self, nativeCallbackHandler,
NULL, &result, originalReflected, (int) original,
additionalInfo, thisObject, argsArray);
```

```
bridgeClass = env->FindClass("com/ protect-
Demo/android/Bridge");
```

```
nativeCallbackHandler = (Method*) env->Get-
StaticMethodID(bridgeClass,"handleHookedMethod",
"(Ljava/lang/reflect/Member;ILjava/lang/Object;Lja-
va/lang/Object;[Ljava/lang/Object;)Ljava/lang/Obje-
ct;");
```

Native层Hook Java层方法执行流程如图7所示。

4) Java层实现AOP Proxy机制

Java端的handleHookedMethod()方法通过AOP Proxy机制，实现before()和after()方法，通过在before()和after()中修改被Hook方法的执行逻辑来添加自定义的方法，实现对Hook方法的敏感行为拦截，执行流程如图8所示。

4 实验及结果分析

Android实验机未进行Root授权，分别安装主流手机安全应用软件和卸载主流手机安全应用软件，对未经允许自动发送短信程序做如下实验。

1) 将安全应用软件金山、猎豹、360、百度、腾讯推出的主流手机安全卫士安装至手机，安装用户未经允许发送短信的程序，点击短信发送按钮，没有收到安全应用软件的拦截提示，短信发送成功。如果手机安全应用开启全面防护，要求用户Root授权，加固应用执行流程界面如图9所示。

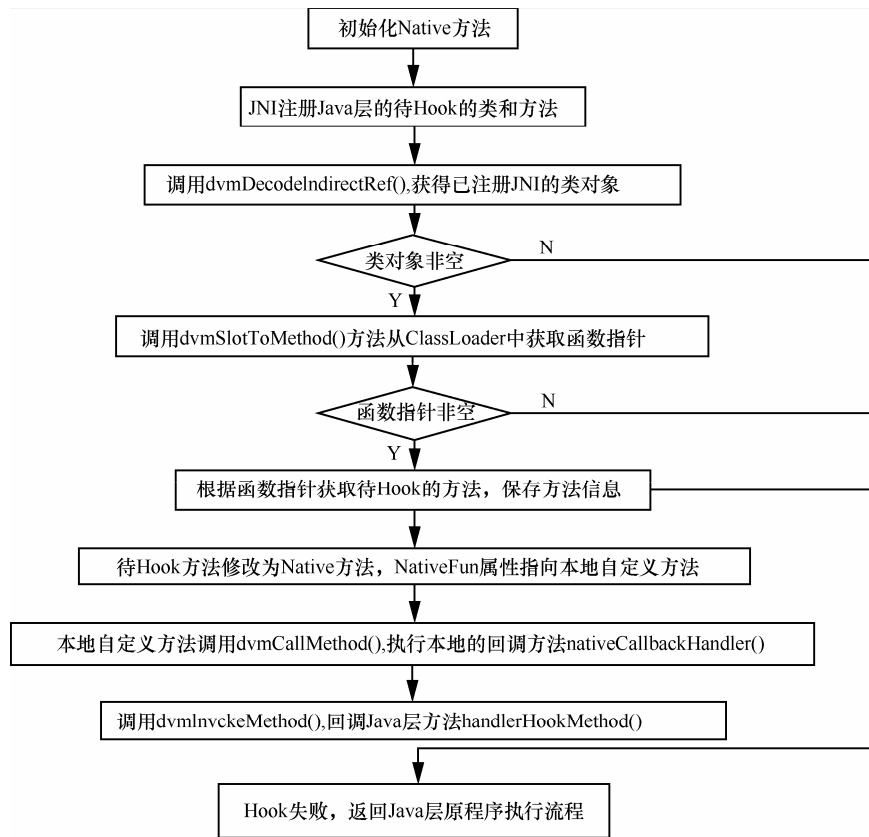


图7 Native层Hook Java层方法

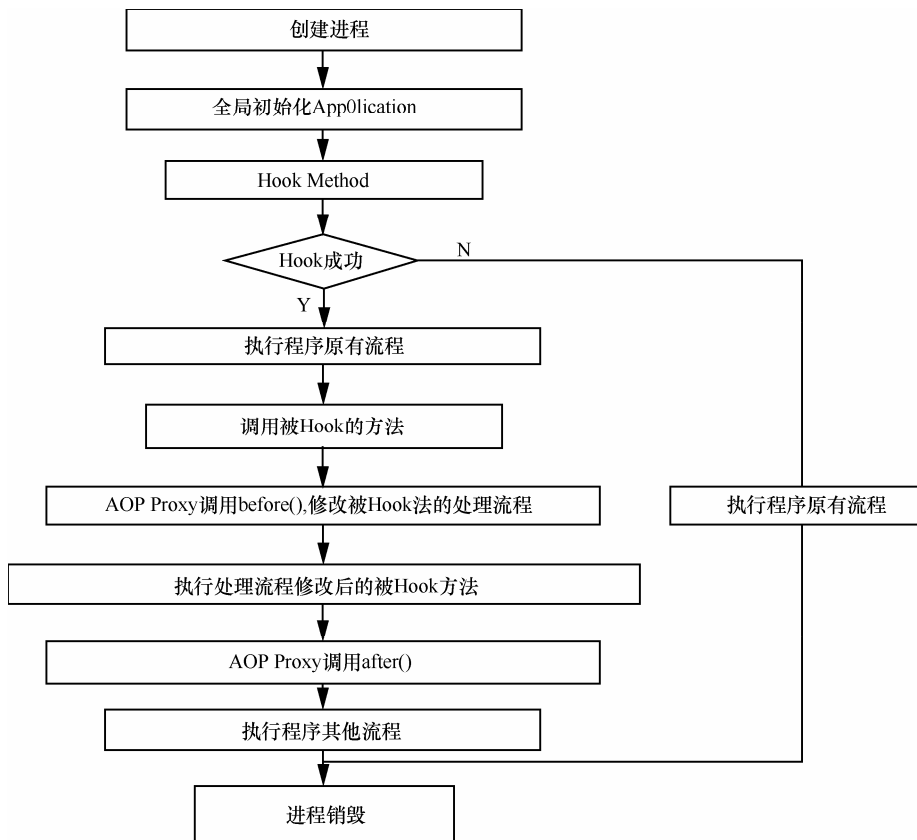


图8 Java层AOP Proxy机制



图 9 加固应用执行流程界面

2) 卸载手机安全应用软件, 安装用户未经允许发送短信的程序, 程序加固后提供关闭短信拦截和开启短信拦截功能, 短信发送行为检测程序示意如图 10 所示。



图 10 短信发送行为检测程序示意

3) 使用关闭短信拦截功能, 点击短信发送按

钮, 短信发送成功, 关闭短信发送行为检测功能如图 11 所示。

4) 卸载手机安全应用软件, 安装用户未经允许发送短信的程序, 开启短信拦截功能, 从尝试发送短信到接收方未收到短信时间段内, 短信发送过程被中断并提醒用户, 开启短信发送行为检测功能如图 12 所示。

对于上述自动发送短信程序实验可做如下分析。

1) 手机安全应用在免 Root 情况下没有检测到短信发送的敏感行为并通知用户。如果手机安装带有恶意软件的程序, 比如: 自动发送短信的恶意程序, 用户手机会在不知不觉中发送短信, 导致利益受损。另一方面, 随着第三方 Android

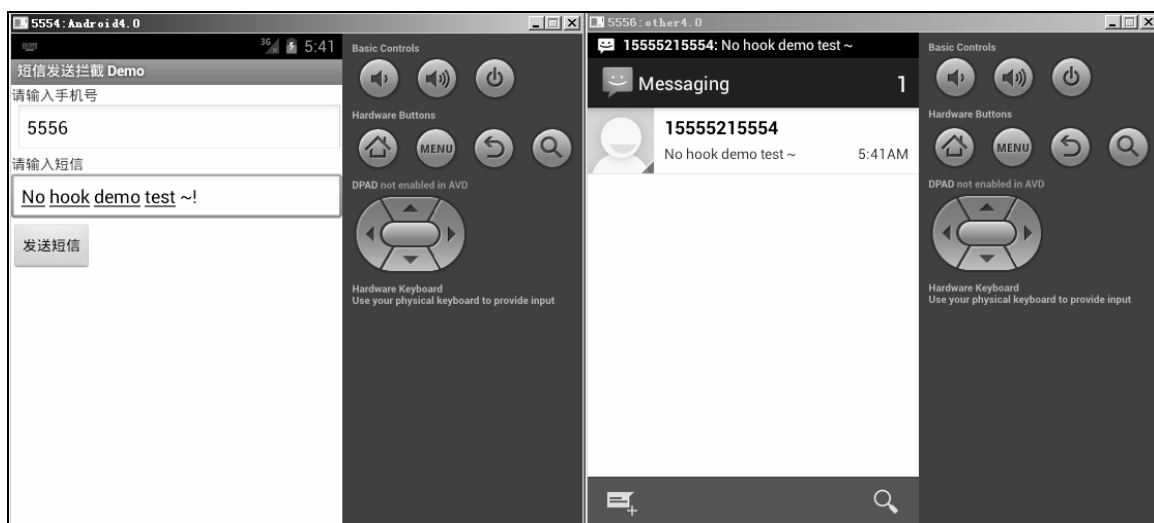


图 11 关闭短信发送行为检测功能

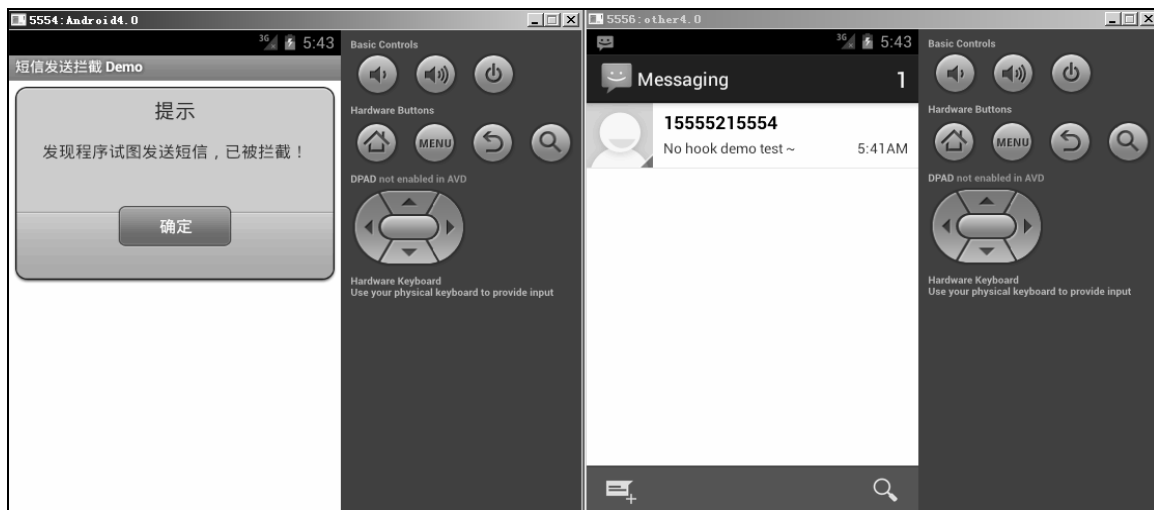


图 12 开启短信发送行为检测功能

系统的定制, 手机 Root 授权的难度增加, 如小米手机根据手机安全应提供的 Root 步骤获取权限失败。

2) 经过本文的加固方案后, 在免 Root 情况下, 可以检测到手机软件的敏感行为并通知用户, 避免了 Root 授权后手机安全风险的增加, 同时弥补了普通手机安全应用拦截敏感行为效果不理想的局限性。

5 结束语

相对于普通手机安全应用, 本文方法有以下优点:

- 1) 不需要手机 Root 授权即可运行, 进而提高手机安全性;
- 2) 主动发现并提醒手机使用者恶意应用存在的敏感行为;
- 3) 避免手机安全应用在没有 Root 授权下防护功能受限的不足。

因此, 相对于普通手机安全应用获取 Root 授权后手机安全性降低, 未取得 Root 授权时安全防护功能存在局限性的特点, 本文方法能够实现对手机应用敏感操作的行为监控, 对于 Android 手机安全应用提高免 Root 主防能力有现实意义。

参考文献:

- [1] 腾讯移动安全实验室 2015 年第三季度手机安全报告[EB/OL].

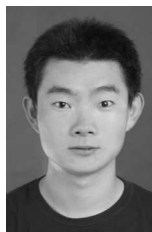
(2015-11-03). http://m.qq.com/security_lab/news_detail_335.html.

Mobile security report of Tencent mobile security laboratory in 2015, third quarter[EB/OL].(2015-11-03). http://m.qq.com/security_lab/news_detail_335.html.

- [2] 李挺, 董航, 袁春阳, 等. 基于 Dalvik 指令的 Android 恶意代码特征描述及验证[J]. 计算机研究与发展, 2014, 51(7): 1458-1466.
LI T, DONG H, YUAN C Y, et al. Description of Android malware feature based on Dalvik instructions[J]. Journal of Computer Research and Development, 2014, 51(7): 1458-1466.
- [3] 李志, 单洪, 马春来, 等. 基于攻防图的网络主动防御策略选取研究[J]. 计算机应用研究, 2015, 32(12).
LI Z, SHAN H, MA C L, et al. Network active defense strategy selection based on attack-defense graph[J]. Application Research of Computers, 2015, 32(12).
- [4] 李淑民. Android 手机隐私泄露研究[J]. 软件, 2015(2): 69-72.
LI S M. Research on privacy leakage of Android smart phones[J]. Computer Engineering & Software, 2015(2): 69-72.
- [5] 吴茂林. 微信红包折射手机安全防护软肋[J]. 通信世界, 2015(6): 17.
WU M L. We Chat ked envelope reflects a soft soft spot of mobile security protection[J]. Communications World, 2015(6): 17.
- [6] SANZ B, SANTOS I, LAORDEN C, et al. PUMA: permission usage to detect malware in Android[C]//The 5th International Conference on Computational Intelligence in Security for Information Systems (CISIS'02). Berlin: Springer, c2103: 289-298.
- [7] FELT A P, CHIN E, HANNA S, et al. Android permissions demystified[C]//The 18th ACM Conference on Computer and Communications Security, New York. c2011: 627-638.
- [8] DESNOS A. Reverse engineering, malware and goodware analysis of Android applications...and more (ninja!) [CP/OL]. (2013-03-26). <https://github.com/androguard/androguard>.

- [9] CILIBRASI R, VITANYI P M B. Clustering by compression[J]. IEEE Transactions on Information Theory, 2005, 51(4): 1523-1545.
- [10] 南秦博, 慕德俊, 侯艳艳. Android 系统恶意程序检测技术研究[J]. 现代电子技术, 2015(12): 47-50.
NAN Q B, MU D J, HOU Y Y. Research of malicious program detection technology for Android system[J]. Modern Electronics Technique, 2015(12): 47-50.
- [11] CHRISTODORESCU M, JHA S, SESHIA S A, et al. Semantics-aware malware detection[C]//The 2005 IEEE Symposium on Security and Privacy (Oakland'05). c2005: 32-46.
- [12] 王蕊, 冯登国, 杨轶, 等. 基于语义的恶意代码行为特征提取及检测方法[J]. 软件学报, 2012, 23(2): 378-393.
WANG R, FENG D G, YANG Y, et al. Semantics-based malware behavior signature extraction and detection method[J]. Journal of Software, 2012, 23(2): 378-393.
- [13] DESNOS A. Android: static analysis using similarity distance[C]//The 45th Hawaii International Conference on System Sciences(HICSS), c2012: 5394-5403.
- [14] 杨超, 刘文庆, 张伟, 等. 基于利用方式的 Android Root 漏洞分析[J]. 计算机科学, 2014, 41(S1): 343-346.
YANG C, LIU W Q, ZHANG W, et al. Utilization pattern based Android Root vulnerability analysis[J]. Computer Science, 2014, 41(S1): 343-346.
- [15] 卢文杰. 网络安全主动防御技术[J]. 网络安全技术与应用, 2015(4): 148.
LU W J. Network security active defense technology[J]. Network Security Technology & Application, 2015(4): 148.
- [16] 孙晓奇. 面向多国发布的手机软件定制系统的设计与实现[D]. 北京: 中国科学院大学, 2014.
SUN X Q. Design and implementation of software customization for global released mobile phone[D]. Beijing: University of Chinese Academy of Sciences, 2014.

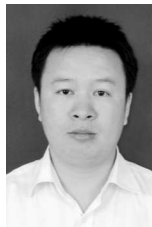
作者简介:



刘井强(1991-), 男, 山东青州人, 哈尔滨工业大学(威海)硕士生, 主要研究方向为网络安全和移动安全。



李斌(1962-), 男, 黑龙江庆安人, 哈尔滨工业大学(威海)教授, 主要研究方向为计算机网络安全。



陈立章(1980-), 男, 湖南邵东人, 哈尔滨工业大学(威海)高级工程师, 主要研究方向为网络安全与内容安全、网络管理等。



陈彬(1968-), 男, 吉林省吉林市人, 博士, 哈尔滨工业大学(威海)计算机科学与技术学院副院长、副教授, 主要研究方向为人工智能、机器学习。