

安卓应用自动原生化及混淆系统

刘惠明, 诸葛建伟

(清华大学 网络与信息安全实验室, 北京 100084)

摘 要: 提出了一种新的加固方案并实现了原型系统, 将原应用中关键代码自动转换成原生代码并编译、混淆. 由于关键代码为混淆后的原生代码, 逆向难度大大增加, 任何针对非原生代码的动态脱壳平台都无法恢复. 经过测试, 该系统兼具强度和兼容性, 对于保护应用开发者的权益, 以及数以亿计的安卓用户的安全, 具有非常重要的意义.

关键词: 安卓应用; 应用加固; 原生化; 代码混淆

中图分类号: TP311.1

文献标识码: A

文章编号: 1000-7180(2016)10-0059-04

Automatic Android Apps Translation and Obfuscation System

LIU Hui-ming, ZHUGE Jian-wei

(Network and Information Security Lab, Tsinghua University, Beijing 100084, China)

Abstract: We propose an alternative way and implement the demo system that translates the key code into native code and obfuscates it automatically. The resulting obfuscated code is immune to existing automatic reverse techniques. Our evaluation shows that this system is of both high harden strength and good compatibility, and it will be an effective protector for millions of apps' developers and users.

Key words: Android app; app harden; native code; code obfuscator

1 引言

安卓(Android)于 2007 年正式发布后, 迅速成为了世界最流行的移动平台操作系统, 安卓系统主要面向于触屏智能手机, 目前也被广泛用于平板电脑、电视及可穿戴设备. 安卓系统基于 Linux 内核开发, 由谷歌负责开发和维护, 其应用使用 Java 作为开发语言.

为了防止应用被恶意逆向, 应用加固技术应运而生. 加固技术将开发者将要发布的应用作为输入, 经过一些加密、混淆等操作, 输出加固后的应用, 一般具有一定的隐藏原应用逻辑, 防止重打包的功能. 但是如今的加固平台(腾讯、阿里、百度、梆梆加固等)通过加密原应用, 并动态解密的方法, 很容易受到动态脱壳平台的破解, 例如 DexHunter 动态脱壳系统^[1-4]. 而学术界已有的强加固方法却无法保证更好的兼容性, 比如 DI-

VILAR^[5]使用 Hook 技术修改了 Dalvik 虚拟机, 在如今的 ART 环境下, 完全无法使用.

由于现有各大公司的加固系统均无法抵抗 Dexhunter 等动态脱壳方法, 学术界的一些方法又无法保证兼容性, 对此本文提出了一种新的加固方法.

2 系统简介

我们提出的加固方法是将应用中关键函数由 DEX 字节码转换成原生代码, 使用 JNI 调用, 以消除程序逻辑的丰富语义. 虽然 DEX 字节码中有丰富语义信息, 很难进行混淆, 但是在原生代码中, 指令和数据可以混到一起, 指令也可以直接操作程序自己的任何内存空间, 甚至可以支持自修改指令, 所以程序混淆的空间非常大, 混淆之后的结果也难以逆向. 系统的核心在于自动将 Smali^[6]语言源文件

转换成 C++ 源文件,然后再使用 LLVM-Obfuscator^[7]进行混淆操作,即可以自动完成高强度的应用加固.图 1 所示为安卓应用的生成过程,图中虚线表示加固思路,即将 DEX 字节码转换成原生代码.

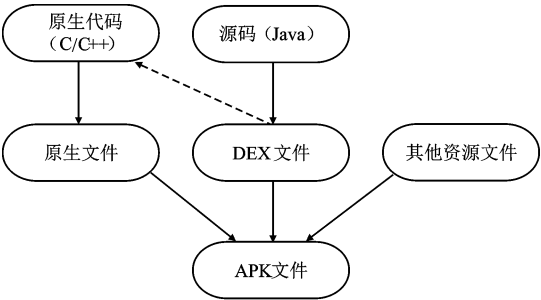


图 1 安卓应用生成过程

使用原生代码编写程序逻辑,进行混淆后通过 JNI 调用,从而保护自己的程序逻辑不被逆向的方法,已经被很多大公司的核心应用使用,比如微信和支付宝.但是由于这种方法对于应用开发者的要求较高,很难要求所有应用开发公司采取这种方法,特别是处于快速迭代开发阶段的中小型公司.而通过以上方法即能够完成自动化的应用代码原生化,对于保护市面上大部分应用的安全有很重要的意义.

自动将 Smali 语言转换成原生代码(C++)再进行混淆的思路虽然理论上能够完成绝大部分代码的本地化,但是存在以下三点问题:(1)有些函数转换成原生代码的难度比较大,特别是那些窗口操作等安卓框架中的函数;(2)大部分窗口操作、界面渲染等安卓框架代码也并不需要保密和加固,因为这些程序逻辑在使用的时候就已经被用户熟知了;(3)如果将上面提到的那些安卓框架函数强行转化成原生代码,由于多了函数调用,可能会影响性能.

正是由于以上原因,并不需要加固大部分窗口操作、界面渲染等安卓框架代码,而那些核心通信和加密代码,通常也不会包含太多安卓框架的函数调用,比较易于转换成原生代码,同时也是最需要加固的应用逻辑.所以本系统中包含了函数选择器模块,将这些不包含安卓框架调用的函数,也很可能是关键代码的程序选取出来,自动转换成原生代码,混淆之后使用 JNI 调用.而正因为选取了与安卓框架关系较小的函数进行强加固,所以也能够较好地保证本方法的兼容性,不会因为安卓的版本问题造成太大的困扰.图 2 为系统原理及详细流程.

本文的主要创新和贡献有如下几点.

(1)提出了一种兼具高强度和兼容性的安卓应

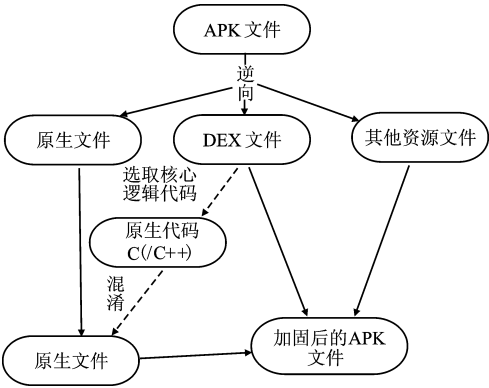


图 2 系统原理及详细流程

用加固方法.

(2)实现了 Smali 语言转换成 C++ 语言的转换器.

(3)实现了安卓应用原生化和混淆原型系统,可以完成自动化的应用加固.

3 系统设计与实现

安卓应用原生化和混淆系统的框架如图 3 所示,其中包含了 ApkInputer、ApkReverser、FuncRecognizer、FuncNativer、FuncObfuscator 和 ApkOutputer 模块.

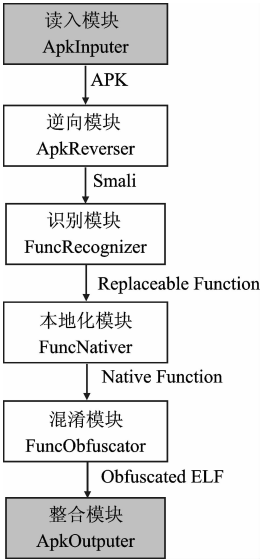


图 3 系统框架

首先,读入模块将 APK 导入到系统工作文件夹中,方便后续操作的进行.之后逆向模块开始工作,利用自行编译的 Apktool 库文件,将应用中 DEX 文件逆向为 Smali 代码,而其他原生文件(如果存在的话)和资源文件不变.

之后,函数识别模块(FuncRecognizer)开始工

作,遍历逆向得到的所有 Smali 文件,从中找出特定逻辑函数. 这些函数的特征是不包含安卓框架函数调用,仅仅是字符串或者逻辑操作,而这些函数,又恰恰很有可能是核心加密、通信逻辑函数,需要高强度混淆和加固. 模块会将这些函数保存到独立文件中,方便本地化模块的工作;删除原函数,并且将原 Smali 文件中的对于这些函数的调用修改成我们将会生成的原生函数,处理情况如图 4 所示. 该函数名一般不变,之后的模块在原生代码中生成同样的函数名.

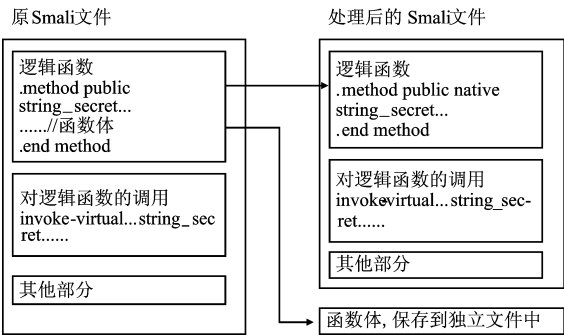


图 4 函数识别模块对 Smali 文件的处理

函数识别模块生成的函数体文件(扩展名为 sf, 即 smali function),之后被输入本地化模块,使用自行开发的 smali2cpp 库将 smali 函数转换成 C++ 函数体.

Smali2cpp 库的原理也比较直观. 因为 Dalvik 指令的个数并不多,可以直接进行字符串匹配,将指令直接替换成 C++ 语言. 对于那些 Smali 语言特有的指令,比如声明局部变量个数,通过匹配的方式,对应到 C++ 的函数变量的声明. 而 Dalvik 作为基于寄存器的指令集,而且寄存器都是 32 比特,大部分指令都可以直接替换成 C++ 命令,特别是那些进行数值或字符串运算的逻辑函数,转换的正确率非常高. 对一个测试函数的转换情况如图 5 所示.

从图 5 中可以看出,smali2cpp 模块将 Smali 函数逐行翻译为了 C++ 函数. 其中主要难点在于 Java 库函数的翻译,而这也正是选择 C++ 作为目标语言的原因. NDK 同时支持 C 和 C++,但 C++ 支持面向对象编程并且拥有很多简化操作的库文件,所以如果选择 C++ 作为转化目标语言,Java 类库的函数翻译会更加简便. 比如图 5 中的 string 操作,直接被翻译为 C++ 标准库中的 string,不仅简洁实用,也具有很好的鲁棒性.

原生化模块将逻辑函数翻译为原生代码,保存

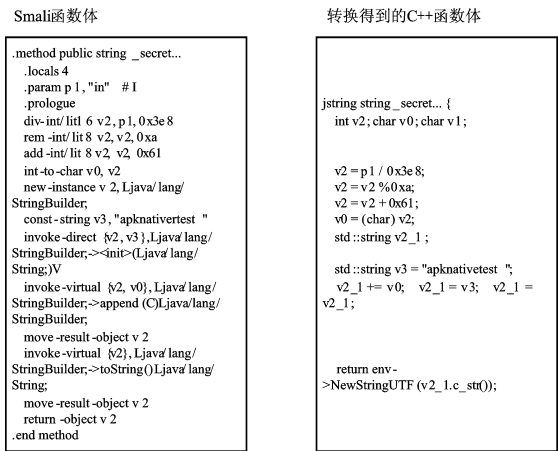


图 5 Smali2cpp 测试结果图

到文件中. 之后混淆模块会将这些函数逻辑作为输入,结合之前得到的一些函数包名等信息,整合起来作为 NDK 输入文件. 然后混淆模块利用 LLVM-Obfuscator 混淆编译器,输入混淆参数(实际配置中不仅加入了所有官方混淆参数,还加入了一 xse 字符串混淆参数,能够隐藏程序关键字字符串,增大混淆强度),之后自动构建 NDK 工程得到输出的强混淆原生库.

至此,所有需要的文件都已经生成. 最后,整合模块(ApkOutputer)将原生库、处理后的 smali 文件等替换或者添加到原应用中. 使用之前提到的 Apktool 库,将原应用打包成 APK. 此时的输出应用中的关键代码已经被本系统自动替换成了强混淆的原生库.

4 系统测试

本系统能够自动完成对测试应用的关键代码的识别、翻译和混淆,随后自动整合、打包并输出加固后的 APK. 经测试,混淆加固后的 APK 在 Android 4.2, 4.3, 4.4, 5.0, 6.0 版本上均能够成功运行,混淆强度很高.

系统关键部分为 Smali 代码翻译为 C++ 源码的 FuncNativer 模块,图 5 证明了其有效性,而该模块的运行时间和系统处理一个测试程序(1.9M)运行总时间与需要转换的 Smali 逻辑代码行数的关系如表 1 所示.

表 1 系统运行时间				
Smali 行数/行	20	100	1 000	10 000
smali2cpp 运行时间/s	0.04	0.08	0.15	0.31
系统运行总时间/s	10.65	10.72	10.83	10.98

从表 1 中可以发现,系统运行速度很快,而且随

着需要处理的 Smali 代码行数增加,时间不会迅速增加,而是非常平缓地上升. 而且 10 000 行已经能够满足绝大部分应用核心逻辑的混淆要求.

而另一项系统关键性能是混淆之后的应用与原应用的文件大小. 对于应用发布者来讲,实现同样的功能,应用占用空间越小,显然用户体验更佳. 不仅应用下载更快,对于本来就有限的手机磁盘空间的占用也更少. 下图是使用本系统混淆加固前后的测试 APK 中各文件大小及总大小.

从表 2 中可以发现,混淆之后应用的体积增加很小. 对本方法和原型系统的详细讨论将会在下一节进行.

表 2 应用大小变化情况				
测试部分	原生库	DEX	其他	总大小
原大小/kB	0.0	757.5	1 124.6	1 882.1
混淆后/kB	58.7	610.5	1 242.3	1 911.5

5 结果分析和讨论

经过测试,本系统能够有效地完成安卓应用的强混淆,且不会增加太多应用体积,完全可用于实际应用,保护应用开发者的权益.

本系统的优点有如下几点.

(1)加固强度大. 由于转成 smali 再逐行翻译为 c++,理论上比直接写成 C++更难以让逆向人员理解. 加固后的代码使用 IDA 等工具几乎不可读,其中关键字串也无法通过二进制搜索直接得到.

(2)兼容性好. 使用函数选择器,只加固那些易于加固的代码(上面提到,这些代码很可能也是关键代码). 而且加固原理核心在于 JNI,对安卓版本依赖很小.

(3)可个性化可配置加固. 现在的加固平台,使用固定模式加固. 若某一加固应用被脱壳成功,其他所有平台加固的应用都可以用同样的方法被逆向脱壳. 而本系统可以扫描所有函数,提供用户配置选项进行特定函数的特定加固、混淆. 并且由于 LLVM-Obfuscator 可配置各种不同编译选项,所以对于一个应用的逆向,可能需要多种不同的方式,极大增加了逆向难度.

(4)系统模块化设计,易于维护和扩展. 系统的每个功能都在单独的模块中,系统只定义了接口,中间的实现可任意修改.

本方法能够将应用的核心逻辑代码很好地保护起来,增加攻击者的逆向难度. 比如可将应用核心通信代码混淆加固,即可很好地隐藏真实服务器地址

和通信协议,保护应用通信安全. 如果与其他反盗版技术结合起来,更能够很好地对抗现在愈演愈烈的安卓应用重打包和盗版情况. 比如可将应用签名验证放到我们混淆加固的逻辑代码中,若发现签名与正版开发者签名不同,则关闭本程序. 而又由于这些逻辑被放在强混淆之后的原生文件中,与其他原生代码掺杂在一起,所以防护强度非常高.

但是系统也存在一些值得改进的问题. Smali 语言翻译为 C++的过程中,可能会遇到一些很难处理的语义问题. 而且现在翻译功能的支持也很有限,对于链表和哈希表等复杂数据结构,还无法正确翻译. 这些问题都将会在之后系统引入抽象语法树后逐步解决. 系统混淆后的 SO 库,也可以使用很多方法进一步加固^[8],提升混淆加固强度.

6 结束语

针对现在安卓应用逆向和盗版猖獗的情况,本文提出了一种新的反制方法,并实现了原型系统. 本系统将应用 DEX 中的核心逻辑自动翻译为 C++源码,使用 Obfuscator 进行强混淆编译为原生库后放回原应用,之后自动重打包得到混淆后的应用. 经过测试,系统运行稳定,能够有效地将应用自动混淆加固,并且输出应用体积增加很小. 本文提出的方法和系统,具有加固强度大、兼容性好、可配置和模块化的优点. 如果再与其他应用反盗版技术结合起来,将会有更好的强度和更广泛的应用前景.

参考文献:

[1] IDC. Smartphone OS Market Share [EB/OL]. [2016-01-08]. <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>.

[2] Statista. Number of apps available in leading app stores as of July 2015 [EB/OL]. [2016-01-08]. <http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>.

[3] AdMob by Google. AdMob by the nubers [EB/OL]. [2016-01-20]. <https://www.google.com/admob/>.

[4] Yueqian Zhang, Xiaopu Luo, Haoyang Yin. Dex-Hunter: Toward extracting hidden code from packed android applications[C]// Proceedings of the 2015 European Symposium on Research in Computer Security: Springer International Publishing. Vienna, Austria: IEEE,2015: 293-311.

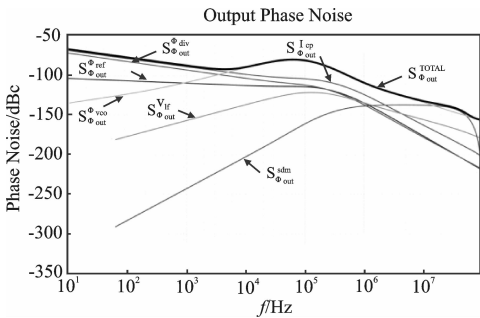


图 11 输出相位噪声

本文设计了一个 10 GHz 的扩频时钟发生器. 它基于传统的小数分频锁相环,通过 $\Delta\Sigma$ 调制器改变多模分频器的分频比来实现扩频功能. 峰值降落是一个非常重要的指标,它主要和调制波形有关. 通过选择三角波调制以及仔细优化环路带宽,得到了 29.3 dB 的峰值降落,可以有效降低 EMI. 频率偏差为下扩频 $5\,000\times10^{-6}$,调制频率为 30.525 kHz. 整个芯片的面积为 $470\,\mu\text{m}\times1\,450\,\mu\text{m}$,在 1.2 V 电源电压下总功耗为 20 mW. 在 1 MHz 频偏处,仿真得到的相位噪声为 $-110\,\text{dBc}/\text{Hz}$.

参考文献:

[1] Keith B Hardin, John T Fessler, Donald R Bush. Spread spectrum clock generation for the reduction of radiated emissions [C]// Proc IEEE Int Symp Electromagnetic Compatibility. [s. l.]: IEEE, 1994: 227-231.

[2] Hsiang-Hui Chang, I-Hui Hua, Shen-Iuan Liu. A spread-spectrum clock generator with triangular modulation [J]. IEEE J Solid-State Circuits, 2003, 38 (4): 673-676.

[3] Tsuyoshi Ebuchi, Yoshihide Komatsu, Tatsuo Okamoto, et al. A 125 - 1250 MHz process-independent adaptive bandwidth spread spectrum clock generator with digital controlled self-calibration [J]. IEEE J

Solid-State Circuits, 2009, 44(3):763-774.

[4] Jonghoon Kim, Dong Gun Kam, Pil Jung Jun, et al. Spread spectrum clock generator with delay cell array to reduce electromagnetic interference [J]. Proc IEEE Int Symp Electromagnetic Compatibility, 2005, 47 (4): 908-920.

[5] Yasushi Matsumoto, Katsumi Fujii, Akira Sugiura. An analytical method for determining the optimal modulating waveform for dithered clock generation [J]. Proc IEEE Int Symp Electromagnetic Compatibility, 2005, 47(3): 577-584.

[6] Chien-Heng Wong, Tai-Cheng Lee. A 6-GHz Self-Oscillating Spread-Spectrum Clock Generator [J]. IEEE Trans Circuits Syst I, 2013, 60(5): 1264-1273.

[7] Takashi Kawamoto, Masato Suzuki, Takayuki Noto. 1.9-ps jitter, 10.0-dBm-EMI reduction spread-spectrum clock generator with autocalibration VCO technique for serial-ATA application [J]. Symp VLSI Circuits Dig, 2014, 22(5): 1118-1126.

[8] I-Ting Lee, Shih-Han Ku, Shen-Iuan Liu. An all-digital spread-spectrum μm clock generator with self-calibrated bandwidth [J]. IEEE Trans Circuits Syst I, 2013, 60(11): 2813-2822.

[9] 王云峰,叶青,满家汉,等. 3.5GHz 锁相环的设计[J]. 微电子学与计算机,2009,26(6):43-46.

作者简介:

胡帅帅 男,(1990-),硕士研究生.研究方向为锁相环和高速串行接口电路设计. E-mail:hushuaishuai@ime.ac.cn.

周玉梅 女,(1962-),研究员,博士生导师.研究方向为高性能低功耗专用集成电路设计、高性能低功耗混合信号电路设计.

张 锋 男,(1977-),副研究员,硕士生导师.研究方向为低功耗、低抖动时钟合成电路、低功耗高速接口、存储电路设计.

(上接第 62 页)

[5] Zhou W, Wang Z, Zhou Y, et al: Divilar: Diversifying intermediate language for anti-repackaging on android platform. [C]// Proceedings of the 4th ACM conference on Data and application security and privacy. Washington DC,USA:ACM. 2014: 199-210.

[6] J. Freke, Smali, An Assembler/Disassembler for Android's dex Format [EB/OL]. [2016-01-20]. <https://github.com/JesusFreke/smali>.

[7] Obfuscator-llvm. Obfuscator[EB/OL]. [2016-01-20].

<https://github.com/obfuscator-llvm/obfuscator/tree/llvm-3.6.1>.

[8] 张译恬,王纯. 基于安卓系统 JNI 机制的 SO 库加固方案设计[J]. 电信技术, 2014,60(10): 90-93.

作者简介:

刘惠明 男,(1991-),硕士研究生.研究方向为安卓. E-mail:liuhm09@gmail.com.

诸葛建伟 男,(1980-),博士,副研究员.研究方向为系统安全.