心於至善

SOUTHEAST UNIVERSITY

基于智能手机惯性传感器的用户隐私安全研究

宋睿

东南大学

学校代码: 10286
分 类 号: TP393
密 级: 公开
U D C: 004.9
学 号: 170815

東南大學

SOUTHEAST UNIVERSITY

硕士学位论文

基于智能手机惯性传感器的用户隐私安全研究

研究生姓名: 宋睿

导 师 姓 名: 宋宇波

申请学位类别 工学硕士　　学位授予单位 东 南 大 学

一级学科名称 网络空间安全　　论文答辩日期 2019 年 6 月 3 日

二级学科名称　　学位授予日期 2019 年 6 月 3 日

答辩委员会主席　　评 阅 人

2019 年 6 月 3 日

学校代码: 10286
分 类 号: TP393
密 级: 公开
U D C: 004.9
学 号: 170815

东南大學

SOUTHEAST UNIVERSITY

硕士学位论文

基于智能手机惯性传感器的用户隐私安全研究

研究生姓名: 宋睿

导 师 姓 名: 宋宇波

申请学位类别 工学硕士　　学位授予单位 东 南 大 学

一级学科名称 网络空间安全　　论文答辩日期 2019 年 6 月 3 日

二级学科名称 _____　　学位授予日期 2019 年 6 月 3 日

答辩委员会主席 _____　　评 阅 人 _____

_____

2019 年 6 月 3 日

東南大學

硕士学位论文

基于智能手机惯性传感器的用户隐私安全研究

专 业 名 称: 网络空间安全

研究生姓名: 宋 睿

导 师 姓 名: 宋 宇 波

RESEARCH ON USER PRIVACY SECURITY BASED ON SMARTPHONE INERTIAL SENSORS

A Thesis submitted to

Southeast University

For the Academic Degree of Master of Engineering

BY

SONG Rui

Supervised by:

A. Prof. SONG Yu-bo

School of Cybersecurity

Southeast University

2019/6/3

东南大学学位论文独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得东南大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

研究生签名：_____ 日 期：_____

东南大学学位论文使用授权声明

东南大学、中国科学技术信息研究所、国家图书馆有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括刊登）论文的全部或部分内容。论文的公布（包括刊登）授权东南大学研究生院办理。

研究生签名：_____ 导师签名：_____ 日 期：_____

摘　要

Android 提供了一个具有权限控制的安全系统，但是有许多漏洞具有过多的权限和大量与权限相关的 API。为了解决这些漏洞，已经对可能危及用户隐私风险的 API 进行了权限控制研究。然而，不可能向不安全的应用程序添加新的安全功能，并且存在在应用程序的进程中发生开销的缺点，因为要求用户实时许可并且减少用户的便利性。在本文中，我们提出了一个 AppWrapper 工具包。该工具包可以使用 appwrapping 技术向用户/管理员的不安全应用程序的所需位置（活动中的方法级别）添加安全功能。而且，使用动态策略管理，可以轻松应用安全策略，而无需再次添加安全功能。此外，通过提供考虑用户便利性的实时应用程序日志功能，可以根据不安全应用程序的进度流程确认需要安全功能的位置，并通过设置创建策略文件政策。除了内置安全性和 Android 应用程序的应用程序外，商业应用程序的实验已经显示出 100％的成功率。平均而言，通过建议的框架添加安全功能花了 1.86 秒，文件大小增加了大约 2.11％，表明随着最小文件大小的增加可以在短时间内添加安全功能。

关键词： Appwrapping，动态策略，移动安全

Abstract

Android provides a security system with permission control, but there are a number of vulnerabilities that have excessive permission rights and a large number of per- permission related APIs. To address these vulnerabilities, permission control studies have been conducted on APIs that are at risk of compromising user privacy. However, it is impossible to add a new security function to an insecure application, and there is a disadvantage that an overhead occurs in the progress of the app because the user is required to permit permission in real time and the users'convenience is decreased. In this paper, we propose an AppWrapper toolkit. The toolkit can add security functions to the user/administrator's desired locations (method level in activities) of an insecure app using the appwrapping technique. And, using dynamic policy management, it is easy to apply secure policies without adding security functions again. In addition, by providing a real-time app log function that considers the convenience of users, it is possible to confirm the location where the security function is required according to the progress flow of the insecure app, and to create a policy file by setting the policy. Experiments on commercial apps have shown 100% success rate, except for apps with built-in security and Android apps. On the average, it took 1.86 seconds to add the security function through the proposed framework, and the file size increased by about 2.11%, indicating that the security function can be added in a short time with the increase of the minimum file size.

Keywords: App Wrapping, Dynamic Policy, Mobile Security

目录

插图目录

表格目录

算法目录

第一章　绪论

1.1　研究背景

Android 提供了具有权限控制的安全系统。但是，应用程序开发人员可以在应用程序（a.k.a 应用程序）中创建漏洞，因为除了应用程序所需的权限信息之外，它还设置了太多权限。为了解决这个问题，已经对字节码级别的权限控制和方法级安全功能进行了研究[1][2][3][4]。由于权限控制方法控制与权限相关的所有 API，因此即使在不需要控制的部分中也生成 API 控制，从而导致开销。此外，由于它控制与危险 API 相关的权限，因此存在执行权限控制的缺点，而不管应用程序的操作流程如何。此外，由于基于 API 的控制，无法添加和控制应用程序的方法级安全功能。此外，当应用程序正在使用时，会显示权限控制窗口，这会降低用户的便利性。

在字节码级别添加方法级安全性的先前方法[4] 是使用 appwrapping 在字节码级别添加适当的安全功能，其中不安全应用程序需要安全功能。此方法在通过反编译缺少安全功能的应用程序（不安全应用程序）获得的字节码（smali 文件）中需要安全性的地方重新打包（重新编译和签名）适当的安全功能。但是，根据静态策略，只在必要时添加单个安全功能。此外，每次更改安全功能时，都应添加并应用新的安全功能。因此，每当添加了安全功能的应用程序的安全策略发生更改时，必须添加安全功能并重新打包，从而使策略管理变得困难且用户友好性降低。

1.2　国内外研究现状

Android 提供依赖权限控制的安全系统; 但是，它需要的权限多于应用程序所需的最低权限，并且很难将每个权限的 API 数量微调到平均七个 API[5]。为了解决这些漏洞，权限控制研究为用户提供了一种控制与可利用或具有隐私问题的 API 相关的权限的方法。

Zhang 等提出了一种静态分析安全漏洞应用程序的方法[2]。该方法简要描述如下：提前检查与权限相关的 API 可能泄漏敏感信息的位置，并将权限控制代码添加到这些位置。权限控制允许用户通过显示警报窗口来选择是允许还是拒绝权限。该方法需要静态分析要执行的 app 的初步任务，并且存在仅可以控制与许可相关的 API 的限制。此外，通过强制最终用户确定是否允许该许可来丢失 UI 的便利性。

Backes 等为应用程序添加了监控代码以进行权限控制[1]。该代码监视整个应用程序中发生的与权限相关的 API 调用，并将其与预配置的策略文件进行比较。当调用与策略文件中声明的权限相关的 API 时，它会引发警报窗口。警报窗口使用户可以选择显示多少信息。例如，在位置信息的情况下，可以选择是否提供近似位置信息或准确位置信息。然而，这种方法的

缺点是必须实时操作监视服务以进行许可控制。此外，如在先前的研究中，可以仅控制与许可相关的 API，并且要求用户决定是否允许减少用户便利性的许可。

Neisse 等人提出了一种基于预先配置的权限控制策略来控制权限的方法[3]。当 API 调用具有潜在的隐私威胁时，该策略将设置要提供的信息级别。设置策略是权限控制库始终处于活动状态，始终监视 API 调用，并在存在相应调用时调整根据策略提供的信息级别。它为九个项目提供用户界面，包括结构，操作，威胁规则和策略设置角色。但是，无论应用程序的流程如何，都可以为隐私威胁 API 设置策略。根据应用程序的进度，可能无法将安全功能添加到需要安全性的位置。

还有一些研究在字节码级别的方法单元上需要安全性的位置添加安全性功能，而不是权限控制。Lee 等人提出了一种解决方案，使用 appwrapping 技术在方法上添加必要的安全功能，该技术在没有应用程序的 Android 源代码的情况下在字节码级别添加安全功能[4]。这包括将所需的安全功能提取到小字代码中，这些代码是字节码级别，并添加安全功能以解决所需方法位置缺乏安全性的问题。但是，它基于静态策略运行，每次更改策略时，都会重复安全功能提取，添加和应用程序; 此外，在更改策略时需要重新打包。另外，缺点在于必须事先知道 app 的进度以便适当地应用策略。

1.3 论文研究内容

在本文中，我们提出了一个动态的基于策略的自动 AppWrapper 工具包，它可以在需要字节码级别安全性的位置为不安全的应用程序添加安全功能，并管理安全功能，而无需通过动态策略管理进行重新打包。建议的 AppWrapper 工具包的目的如下:

1. 在方法级别添加安全功能：在不安全应用程序的所有活动类的方法单元中添加安全功能执行代码。可执行代码使用 Java 反射技术调用安全性函数。添加的安全功能可执行代码根据动态策略执行。

2. 动态策略管理：在方法单元中添加安全功能执行代码后，通过检查设置策略的策略文件和执行安全功能执行代码的位置来确定安全功能的执行（方法单元）活动类）。该策略具有安全功能 API 以及将在其中执行安全功能的应用程序的位置（类和方法名称）以及要执行的安全功能。只需更改策略，即可在不重新打包应用程序的情况下应用更改的策略。

3. 实时应用程序操作流程检查和策略设置：通过在方法级别添加实时应用程序日志功能，用户/管理员可以在运行应用程序时实时检查日志并了解应用程序的进度流程（类和方法名称）。在运行应用程序时，用户必须同时检查应用程序的当前类和方法名称，并将安全功能添加到需要安全性的方法的位置。

1.4　论文组织结构

本文的其余部分如下。

在第 2 节中，总结了相关的研究和研究。

在第 3 节中，描述了提议的框架。

在第 4 节中，给出了提出的框架的性能分析的实验结果。

第 5 节讨论了几个问题。

最后，我们在第 6 节中提出我们的结论。

第二章　相关技术研究

本章介绍现代手机操作系统的权限控制，具体包括基本的安全机制、Android 系统和 iOS 系统的权限管理以及它们对内置传感器的权限控制策略。

2.1　智能终端权限控制

Android 和 iOS 分别是基于 Linux 内核和 Unix 内核进行设计的现代操作系统，它们继承了类 Unix 操作系统的基础安全框架，又在此基础上各自发展出了一些新的安全措施和隐私保护机制。现代手机操作系统安全机制的一个重要功能就是权限管理。通过设定访问设备中各种软硬件资源的权限需求、权限组和权限等级，操作系统就能够将软硬件资源的访问牢牢地控制住。在多年的攻防实践中，系统安全工程师们不断地修补漏洞，构筑更完善的安全体系。在经过长足的发展后，现在的手机操作系统安全机制已经基本成熟，能够预防一定强度的网络攻击，保护用户的隐私信息。

因此，想要实现对用户输入内容的推测或窃取，就必须对手机操作系统的安全机制和防护体系有具体的认识。只有通过分析和理解手机操作系统的安全机制和权限控制原理，才能找到合适的信息源辅助数据窃取和分析工作。本节首先简要介绍手机操作系统的安全机制，然后分别对 Android 和 iOS 两个系统的权限控制机制进行详细介绍，最后说明这两个操作系统对内置动作传感器的权限控制。

2.1.1　操作系统安全机制

Android 操作系统采用一种名为 SandBox（沙盒）的机制来进行进程隔离和应用隔离。一般情况下，各个运行中的应用程序都以独立进程运行在各自的虚拟机中。每个应用程序在首次安装时都会被分配一个 UserID，这个 ID 在全局中是唯一固定且保持不变的。并且，该 UserID 与 Linux 内核中的进程 UID 用户名一一对应。而手机中运行的不同进程无法访问其他进程的软硬件资源，也无法和其他程序共享数据。因此，除了相关进程的程序外，其他应用程序是不能直接获得用户在屏幕软键盘上的输入的，这就有效地防止了恶意程序对用户隐私的窥探。

iOS 系统的隔离机制则更为激进，它采用了被称为“岛式存储”的存储结构。Android 系统虽然运行中的程序无法共享资源，但还存在统一的资源管理器，由系统维护着基本的数据目录。而 iOS 系统中，各个应用程序独自维护自己的数据资源和储存文件，在不经用户许可的情况下无法和其他程序交换文件。在 iOS 中，SandBox 的本质是一个文件夹，其路径是固定的，文件夹名字使用 UUID（全球唯一标识符）随机分配。如果某个应用程序要求其他应用程序目录下的文件资源时，需要严格的权限检查和用户提醒，合格后才能将该文件以文件副本的方式拷贝到自己的目录下。

2.1.2 Android 权限控制

Android 是一个权限分隔的操作系统，其中每个应用都有其独特的系统标识（Linux 用户 ID 和组 ID）。系统各部分也分隔为不同的标识。Android 据此将不同的应用以及应用与系统分隔开来。在默认情况下任何应用都没有权限执行对其他应用、操作系统或用户有不利影响的任何操作。这包括读取或写入用户的私有数据（例如联系人或电子邮件）、读取或写入其他应用程序的文件、执行网络访问、使设备保持唤醒状态等。由于每个 Android 应用都是在进程沙盒中运行，因此应用必须显式共享资源和数据。它们的方法是声明需要哪些权限来获取基本 SandBox 未提供的额外功能。应用以静态方式声明它们需要的权限，然后 Android 系统提示用户同意。

基本 Android 应用默认情况下不会关联权限，这意味着它无法执行对用户体验或设备上任何数据产生不利影响的任何操作。要利用受保护的设备功能，必须在应用清单中包含一个或多个 <uses-permission> 标记。在 Android 操作系统中，系统权限通常以其涉及到的数据性质或使用到的硬件的特性而分为安全或危险的。Android 操作系统会判定权限是否涉及到使用者的个人私密信息，或是否存在破坏设备的软硬件环境的可能性。对于操作系统认为安全的权限，一般会直接赋予应用程序而不会触发警告消息；而对于系统认为危险的权限，系统将触发警告或提示信息，明确告知风险和责任，用户同意后方可赋予该应用。

2.1.3 iOS 权限控制

iOS 的权限管理与 Android 大致类似。主要的不同点在于，iOS 系统中所有软件需要读取的数据都会在读取的瞬间触发系统的强制提示，这个时候用户会看到一个系统提示框询问是否授予软件此项权限。如果用户选否，那么软件将无法获取任何对应的内容。相关授权在进行过第一次询问之后，用户可以在隐私设置中调整软件的对应授权。值得一提的是，越狱之后由于软件可以获取整个机器内置储存的访问权限，从某种程度上来说，软件可以直接读取对应的数据库内容而无需通过 API 进行访问，这个时候的隐私选项和权限控制也就形同虚设了。与 Android 系统相比还有些不同在于，iOS 系统并不提供对于短信、通话记录等 Android 系统提供读取的接口，所以有些东西，除非用户越狱，是一定无法被应用读取的。

2.2 本章小结

本章介绍了现代手机操作系统的权限控制，具体包括基本的安全机制、Android 系统和 iOS 系统的权限管理以及它们对内置传感器的权限控制策略。

第三章　系统框架设计

本节介绍 AppWrapper 工具包。与传统的 appwrapping 技术不同，此工具包可以在字节码级别添加安全功能，以使方法单元上的应用程序不安全，并控制是否通过动态策略管理启用安全功能。如图3.1所示，AppWrapper 工具包由三部分组成：自动 app 包装，实时 app 流确认和策略设置，以及动态策略管理。

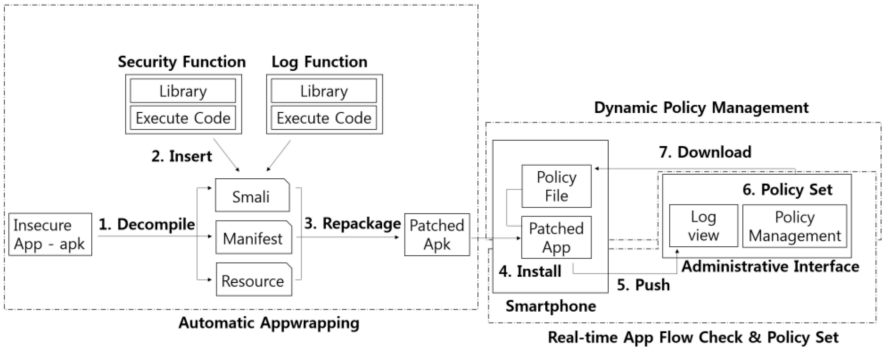


图 3.1 AppWrapper 概述

1. 反编译不安全应用程序的.apk 以获取 smali 文件作为字节码级别，AndroidManifest.xml 文件和资源文件。

2. 在步骤 1 中添加在 smali code 和 smali 文件中提取的安全性功能和日志应用程序功能。

3. 使用安全功能和日志功能，现有 AndroidManifest.xml 文件和资源文件重新打包 smali 文件，以创建修补的.apk 文件。

4. 在智能手机上安装生成的.apk 文件。

5. 运行已修补的应用程序时，驱动信息（活动类名称，方法名称）将根据应用程序进度的流程发送到 AppWrapper 工具箱的 UI 的日志视图。

6. 检查日志视图并根据应用程序的进度流程将策略设置为在需要安全功能的位置运行安全功能。

7. 一旦设置了所有必要的安全策略，请将策略文件下载到用户的电话并运行修补的应用程序。修补应用程序的安全功能根据应用程序的进度与下载的策略文件一起执行。

该流程在以下小节中详细描述。

3.1 应用程序自动包装

安全和日志记录功能是在字节码级别添加的，没有 Android 的原始源代码。有必要对不安全应用程序的.apk 文件进行反编译和重新打包（重新编译和签名）。使用 apktool.jar 文件执行反编译和重新编译过程。通过重新编译获得的.apk 文件使用 signapk.jar 文件进行签名，并转换为可以安装在用户手机中的.apk 文件。签名时需要签名密钥。

通过反编译.apk 获得的 smali 文件是在字节码级别用 smali 编写的代码集合，其中添加了安全功能和日志功能。添加的位置是活动中的所有方法单位。它被添加到 AndroidManifest.xml 文件中声明的所有活动类中的所有方法中。安全和日志函数的 smali 代码被添加到每个方法的开头，以最小化与现有代码的冲突。在 smali 中，.method 和 end 方法语法分别表示方法的开始和结束。

在 smali 中，寄存器区域根据每个方法中声明的局部变量和输入参数的数量进行分配。在要分配的寄存器区域中有 4 位，8 位和 16 位，并且对于这些分配中的每一个，指令是不同的。安全性和日志功能代码使用三个局部变量。对于 4 位寄存器，可以使用从 0 到 15 的 16 个地址编号.Ifa16th addressisusedina4bit 寄存器分配方法，发生寄存器错误，重新编译过程失败。为了避免这个问题，通过计算方法中局部变量和参数的数量，将安全性和日志函数添加到仅 13 个或更少的局部变量分配方法中。这允许添加安全性和日志功能，然后重新打包而不会出现错误。

3.2 动态策略执行

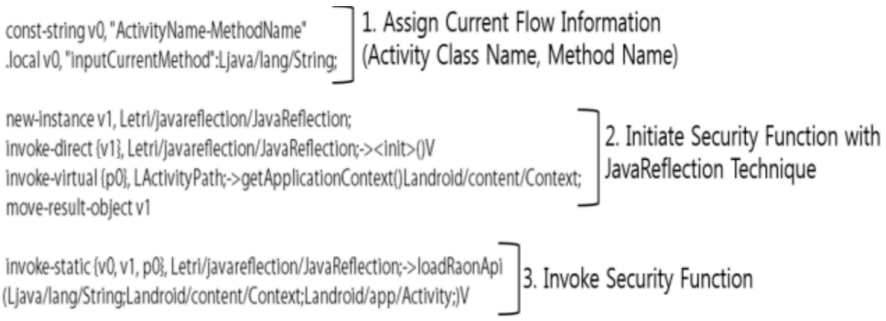


图 3.2 使用 java 反射的 Smali 安全功能代码

通过 appwrapping 添加的安全功能由策略决定。现有的应用程序技术在动态策略管理中存在困难，因为所有与安全相关的代码都包含在基于静态策略添加的安全功能中。但是，建议的 AppWrapper 可以使用 Java 反射技术基于动态策略调用和管理各种安全功能。Java 反射技术是一种用于动态加载和使用类的流行技术。通过应用这种技术调用安全函数的 smali 代码如图3.2所示。当调用安全函数时，它被指定为变量，以便它可以作为参数传输到哪个位置（类名和方法名）叫做。然后，使用 Java 反射技术初始化安全性函数库，并且与先前分配的变量（类名和方法名）一起调用安全性函数。通过该流程，可以根据 app 的进度在方法单元中

表 3.1 策略文件样本

#	活动类	方法名	安全库	安全函数
1	MainActivity	onResume	SecurityApi, SecurityFuntion	FIDO Authentication
2	LoginActivity	onCreate	SecurityApi, SecurityFuntion	Office Model Login
3	LoginActivity	onResume	SecurityApi, SecurityFuntion	Office Model Logout

执行安全功能，并且可以灵活地应用各种安全功能。另外，在添加第一个安全功能之后，不需要再次添加安全功能和重新打包过程。由于已将所有安全功能添加到所有方法单元，因此当将新创建的具有已更改策略的策略文件下载到用户的电话时，将自动应用新策略。

可以通过管理界面执行策略设置和管理，如图3.3所示。管理界面由三个单独的屏幕组成：活动类列表和所选活动类和安全功能列表的方法列表。要设置策略，请选择要添加安全功能的活动类，然后选择在活动中执行安全功能的方法位置。然后选择要执行的安全功能。策略设置完成后，将创建策略文件。策略文件包括要执行的安全功能库信息和安全功能信息，以及执行安全功能的位置信息（活动类，方法名称）。表3.1显示了一个示例策略文件。安全功能库信息用于使用上述 Java 反射技术动态调用安全功能。

3.3　检查实时应用程序行为并设置策略

将安全功能添加到不安全的应用程序时，将检查应用程序的进度。为此，添加了一个日志功能以查看实时应用程序行为。日志功能使用 SSL 连接到 AppWrapper 工具包，并根据应用程序进度流传输日志信息。通过 appwrapping 将日志功能添加到修补的应用程序中。启动修补应用程序后，您可以根据应用程序的流程检查日志信息（类名，方法名称），如图3.4所示。如果单击要设置策略的位置的日志，您将切换到图3.5的策略设置用户界面屏幕。如果设置策略，则根据应用程序进度流程将安全功能添加到缺乏安全性的位置。

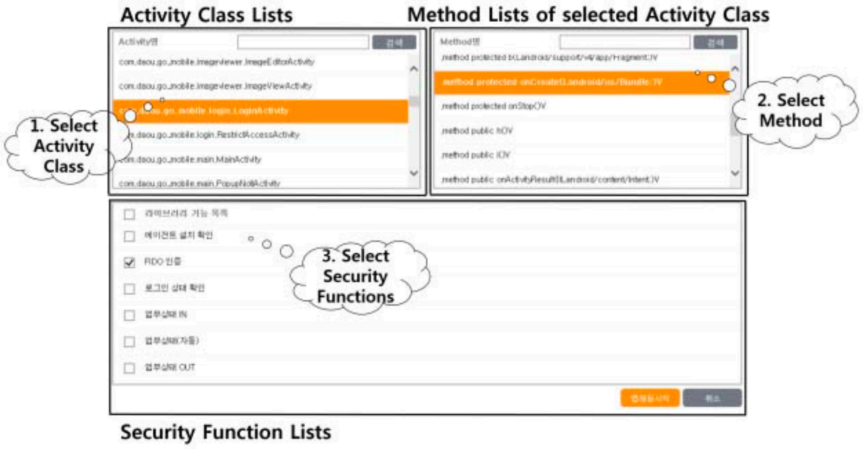


图 3.3 动态策略管理界面

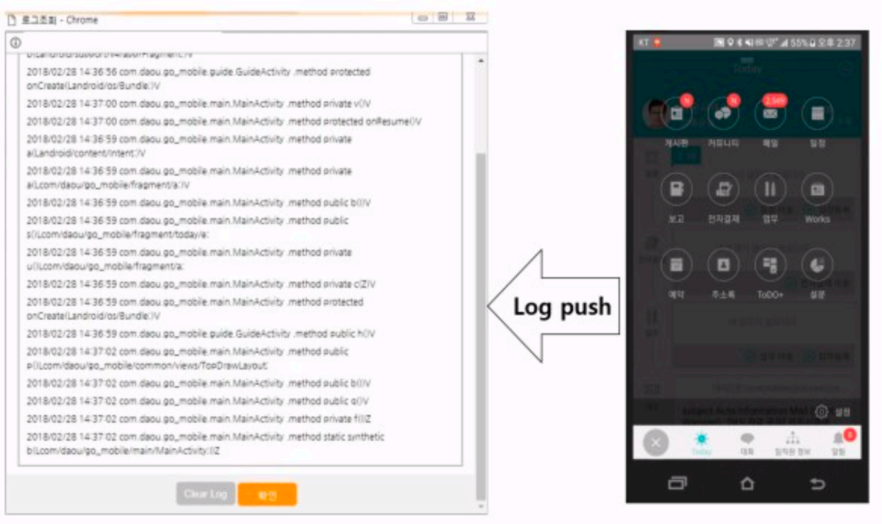


图 3.4 实时日志视图管理界面

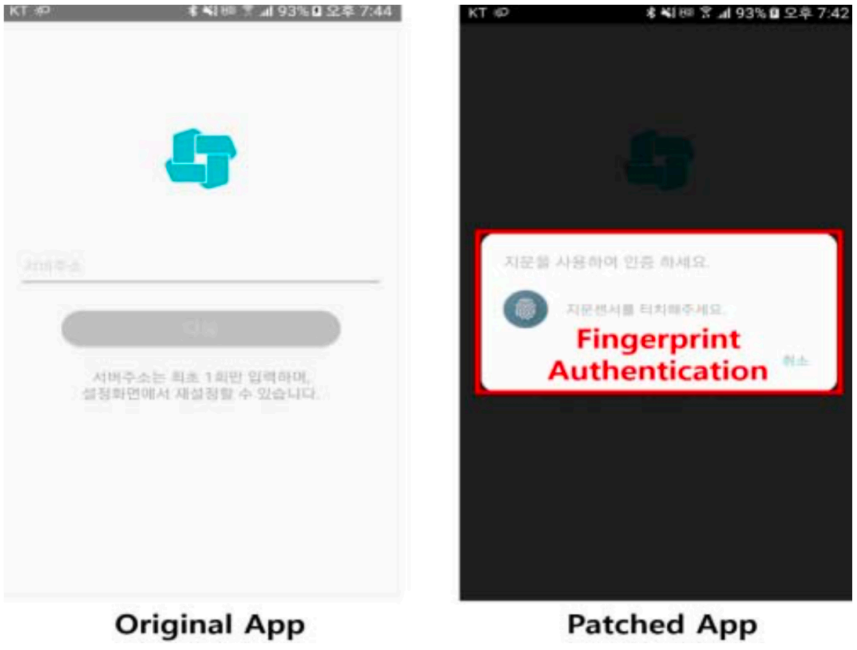


图 3.5 安全功能补丁前/后的应用启动屏幕

第四章　实验与系统评估

为了评估 AppWrapper 工具包的性能，我们尝试了 Android 商业应用。实验中使用的应用程序是 Google Play 韩国市场中 33 个类别的前三个应用程序，并对表格4.1中列出的 99 个应用程序进行了实验。在实验中，我们使用了三星 Galaxy S6 和 Android 7.0 版。计算机系统使用 Windows 7 64 位，CPU 为 Intel i7-4770 3.4 GHz，内存为 8 GB。实验证实，安装并正常执行 99 .apk 文件的反编译，安全修补和重新打包后创建的.apk 文件。图3.5显示了添加安全功能之前和之后的应用程序启动屏幕。

4.1　实验结果

通过使用 AppWrapper 工具包为 99 个应用程序添加安全功能，除了 20 个安全应用程序和一个 Android 内置应用程序外，成功率为 100 %。对于具有安全功能的应用程序，包含.apk 文件以防止修改。此功能可防止在执行应用程序后在字节码级别进行修改并验证签名密钥。在前一种情况下，smali 文件不会被反编译。后者在运行应用程序后，应用程序服务器检查签名密钥，生成异常应用程序警告，并关闭应用程序。Android 内置应用程序默认安装在 Android 上，该应用程序只能安装在 Google Play Market 上。

4.2　处理时间

通过将安全功能添加到每个步骤（反编译，安全补丁，重新打包（重新编译，签名））来测量安全功能添加的处理时间。计算与应用程序大小部分对应的每个应用程序的平均处理时间，以确定每步的处理时间。如表4.3所示，添加安全可执行代码花费的时间最少。此过程从整个阶段花费的总时间的最大 3.49 % 至少花费 2.02 %，并且在最少 1 秒内最多 2.6 秒内，安全性被添加到 AndroidManifest 中声明的所有活动方法中。该应用程序的 xml 文件。

表 4.1 99 个样本的分布

数量	样本大小	比例%
36	< 10 MB	37
19	10 –20 MB	19
16	20 –30 MB	16
11	30 –40 MB	11
8	40 –50 MB	8
9	> 50 MB	9

表 4.2 处理时间

App	反编译	安全补丁	重编译	签名	总时间
< 10 MB	7.340	1.010	18.346	2.254	28.921
10 −20 MB	11.994	1.516	31.351	3.951	48.946
20 −30 MB	13.648	1.841	43.189	7.590	66.268
30 −40 MB	17.479	2.383	55.595	10.480	85.937
40 −50 MB	18.068	2.662	61.248	13.886	95.864
> 50 MB	16.354	1.752	52.538	15.690	86.334

表 4.3 文件大小

App	APK 大小	安全补丁后大小	主 Class 文件大小	安全补丁后大小
< 10 MB	6,345KB	6,538KB	34KB	41KB
10 −20 MB	14,696KB	14,942KB	60KB	70KB
20 −30 MB	25,737KB	27,155KB	25KB	31KB
30 −40 MB	34,974KB	35,553KB	17KB	22KB
40 −50 MB	47,285KB	47,620KB	27KB	35KB
> 50 MB	75,538KB	73,984KB	147KB	165KB

随着应用程序的大小增加，每个步骤的处理时间通常会增加，并且安全补丁处理时间也会增加。这是因为随着应用程序大小的增加，活动类数量会增加，并且随着更多安全功能的添加，安全补丁时间似乎也会增加。

在从反编译到重新打包的时间内，应用程序大小为 10 MB 或更小时所需的最小时间跨度平均为 28 秒。相反，对于 40-50 MB 的应用程序大小，最耗时的部分花了大约 95 秒。这是因为反编译和重新打包所需的时间增加了。安全补丁时间约占总时间的 2-3 %, 并且所提出的框架的安全补丁处理时间并不重要。此外，如果要在以后更改策略，则不需要重新修补安全补丁; 因此，重新包装不需要时间。

4.3 文件大小

将安全功能添加到 AndroidManifest.xml 中声明的所有活动中的所有方法后重新打包的.apk 文件和主类文件的大小变化并不重要。表 4 显示了根据.apk 文件大小分类的.apk 文件大小和主类文件大小的变化。

首先，小于 10 MB 的.apk 文件大小显示平均文件大小增加 3 %; 但是，主类文件大小增加了 15.7 %。尽管每个大小的文件大小和主类大小增加率不同，但.apk 文件大小增加了 0.6 % 到 5.2 %, 文件的总体平均值显示文件大小增加了 2.54 %。主类文件的大小增加了 10.8 % 到 22.8 %。

尽管应用安全补丁以使各种安全功能能够在活动的所有方法中执行，但与现有的 Lee 研

究相比，文件大小仅增加了 2.05％（平均五个办公应用程序，增加 0.49％）。很难直接与现有研究的五组进行比较; 但是，建议的框架为 78 个应用程序的所有方法单元增加了安全性，并且文件大小增加最小。这是因为它使用 Java 反射技术调用安全函数库。通过策略文件控制安全功能也很重要，这样只有最不安全的代码才会添加到不安全应用程序活动中的所有方法中。

第五章　总结与展望

5.1　问题讨论

5.1.1　性能问题

该研究增加了基于静态策略的安全功能，并在添加安全功能处理时间和安全功能后比较文件大小。基于权限的研究不提供方法级安全功能，因此很难直接比较性能。

基于静态策略的安全功能研究并未在整个方法单元中添加安全功能，而是仅根据预设策略将安全功能添加到需要安全性的方法位置。相反，建议的框架为 AndroidManifest.xml 中声明的所有活动类的方法单元增加了安全性。然而，所提出的框架的安全补丁的处理时间比先前的研究（将安全功能添加到平均 0.15 秒的两种方法）从最小 1 秒到最大 2.6 秒更长。但是，考虑到方法单元的所有添加，开销很小。

在文件大小的情况下，原始应用程序中修补应用程序的大小更改比基于静态策略的研究高约 1.5%。但是，考虑到所有方法都添加了所有安全方法，应用程序的最小大小已经增加，并且由于使用 Java 反射技术添加了简化的安全功能，仅增加了应用程序大小的 2.05‰。此外，提供管理界面以检查策略管理和应用程序进度流，从而提高用户便利性。

5.1.2　实验限制

在实验应用程序中，排除了具有自己的安全功能（修改预防）的 20 个应用程序。由于冗余，未考虑向安全内置应用程序添加安全功能。修改防止功能是不正确地反编译 app 的功能和在执行 app 之后检查签名密钥值的功能。在后一种情况下，可以在通过绕过签名密钥值验证逻辑禁用伪造防止功能之后添加安全功能。但是，没有执行绕过这些应用程序的防伪功能，因为它可能看起来是恶意的。

5.1.3　易受攻击的代码补丁

建议的框架显示了如何为不安全的应用程序添加安全功能。在以前的研究中，已经有研究在字节码级别找到并修补误用的加密 API[6]。虽然本文未涉及，但也可以通过检测在检查应用程序的进度流程或者代码中需要安全更新的过程中需要安全更新的情况来更新代码。例如，它是用 SSL（安全通信）代码更新在一般通信中实现的代码的功能。

5.1.4 法律问题

确定该应用是否违反了 Google Play 的政策，禁止在没有 Android 源代码的情况下进行修改。我们的 App Wrapper 工具包用于向商业不安全的应用程序添加安全功能。Google Play 中没有规定在没有 Android 源代码的情况下修改应用。此外，根据 Google Android 部署政策 [10]，在分发应用时只需注意几个方面，并且对应用修订和再分发限制没有任何限制。

5.2 总结

Android 提供了基于权限的安全系统，但存在多个漏洞。为了解决这些漏洞，已经开展了通过静态策略进行权限控制和修补安全功能的研究。在权限控制的情况下，不可能为每个方法添加安全功能，并且每当发生权限控制时就生成许可警告窗口，从而导致应用的处理流程的开销和用户便利性的降低。在通过静态策略添加安全功能的情况下，每次更改安全策略时都应该重新打包应用程序。

在本文中，我们提出了 AppWrapper 工具包来解决现有研究的局限性。建议的框架可以在需要字节码级别安全性的位置为不安全的应用程序添加安全功能，并管理安全功能，而无需通过动态策略管理进行应用程序重新打包。有一个优点是重新包装的阶段不是必需的。它还提供了一个日志视图管理界面，可根据应用程序进度流添加安全功能。在界面中，只需选择添加安全功能的位置，即可切换到策略管理界面并设置策略，从而提高用户便利性。

Google Play Market 的商业应用上的实验最大限度地减少了处理时间和文件大小的开销。尽管安全功能已添加到 AndroidManifest.xml 文件中所有活动类的方法单元中，但安全修补程序至少需要 1 到 2.6 秒，而.apk 文件大小仅增加 2.05‰。未来的工作将包括修补易受攻击代码的能力。

致谢

感谢每一个给予帮助的人。

参考文献

[1] Backes, Michael, Gerling, Sebastian, Hammer, Christian, et al. AppGuard – Enforcing User Requirements on Android Apps[C]. In: Piterman, Nir and Smolka, Scott A., eds., Tools and Algorithms for the Construction and Analysis of Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. 543–548.

[2] Zhang, Mu and Yin, Heng. Efficient, context-aware privacy leakage confinement for android applications without firmware modding[J]. 2014.

[3] Neisse, Ricardo, Steri, Gary, Geneiatakis, Dimitris, et al. A privacy enforcing framework for android applications[J]. Computers and Security, 2016, 62:257–277.

[4] Lee, Sung-Hoon, Kim, Seung-Hyun, Kim, SooHyung, et al. Appwrapping Providing Fine-Grained Security Policy Enforcement Per Method Unit in Android[C]. 2017. 36–39.

[5] Felt, Adrienne, Chin, Erika, Hanna, Steve, et al. Android Permissions Demystified[C]. 2011. 627–638.

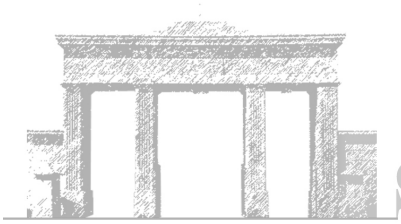
[6] Ma, Siqi, Lo, David, Li, Teng, et al. CDRep: Automatic Repair of Cryptographic Misuses in Android Applications[C]. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. New York, NY, USA: ACM, 2016. 711–722.

作者攻读硕士学位期间的研究成果

发表的论文

[1] 第一作者, “灵犀一指：理论与应用”, 武侠学报，2015 年 5 月。

心求至善



SOUTHEAST UNIVERSITY