申请上海交通大学硕士学位论文

Android 软件安全分析和系统安全增强研究

学　　校： 上海交通大学

院　　系： 电子信息与电气工程学院

班　　级： B0903391

学　　号： 1030993008

硕 士 生： 罗宇皓

专　　业： 计算机系统结构

导　　师： 谷大武（教授）

上海交通大学电子信息与电气工程学院

2013 年 3 月

A Dissertation Submitted to Shanghai Jiao Tong University for the Degree of Master

Securiy Analysis and Enhancement for Android System and Applications

Author: Luo Yuhao

Specialty: Computer Science & Engineering

Advisor: Prof. Gu Dawu

School of Electronics and Electric Engineering

Shanghai Jiao Tong University

Shanghai, P.R.China

March 1, 2013

附件四

上海交通大学
学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：2014 年 1 月 9 日

附件五

上海交通大学

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

保密□，在___年解密后适用本授权书。

不保密☑。

（请在以上方框内打“√”）

学位论文作者签名：罗宇航　　指导教师签名：

日期：2014年 2月19日　　日期：2014年 2月19日

上海交通大学 硕士 学位论文答辩决议书

姓 名	罗宇皓	学 号	1090339008	所在学科	计算机系统结构
指导教师	谷大武	答辩日期	2014-1-9	答辩地点	电信群楼3-414
论文题目	Android软件安全分析和系统安全增强研究				

投票表决结果：5/5/5 （同意票数/实到委员数/应到委员数） 答辩结论：☑通过 ☐未通过

评语和决议：

Android 是目前世界上使用最广泛的移动智能操作系统，Android 平台上的安全问题是目前软件安全领域具有挑战性的研究热点之一。该文针对 Android 软件安全分析和安全增强两个方面进行研究，提出了一套有效的软件恶意行为分析方法和安全增强方案，具有很好的实际应用价值。因此，罗宇皓同学的硕士学位论文选题具有一定的理论意义和实用参考价值。

该论文首先提出了一套 Android 平台恶意软件行为的系统化安全分析方法，并给出了一个有代表性的恶意软件分析实例；其次，该论文以保护隐私数据为核心思想，设计了一套 Android 系统安全增强方案。

针对 Android 平台恶意软件行为的安全分析方法这一研究，该论文提取了恶意软件行为的一般特征进行分析，并且通过给出一个恶意软件的实例，验证了该方法的可靠性和高效。该方法包括鉴别、代码反保护、代码分析、行为提取等步骤，通过该方法有效的去除了恶意软件实例的代码混淆保护，提取出了恶意软件的全部工作流程、通讯协议，以及加密算法和密钥，还原出了恶意软件通过加密通讯泄露用户隐私数据的全过程原始数据。

针对 Android 系统安全增强这一研究，该论文在总结当前安全问题和相关研究的基础上，提出了一套以数据为中心的安全增强方案，该方案的设计主要包括可信内核，可信数据库，以及应用程序管理三个模块。该方案在提升数据安全性的同时，也保证了系统的运行效率和易用性。

论文工作量饱满，研究结果表明：罗宇皓同学在 Android 软件安全分析和系统安全增强方向做了大量研究，已经掌握了本领域扎实的基本理论和系统的专门知识，具有从事科研工作的能力。论文答辩中思路清楚，回答问题正确。经答辩委员会无记名投票表决，一致同意通过该同学的硕士学位论文答辩，建议授予其工学硕士学位。

年 月 日

答辩委员会成员	职务	姓名	职称	单位	签名
	主席	刘胜利	教授	上海交通大学	刘胜利
	委员	李小勇	副教授	上海交通大学	李小勇
	委员	张爱新	副研究员	上海交通大学	张爱新
	委员	顾海华	高级工程师	上海交通大学	顾海华
	委员	谷大武	教授	上海交通大学	谷大武
	秘书	刘军荣	助理研究员	上海交通大学	刘军荣

Android 软件安全分析和系统安全增强研究

摘 要

Android 是 Google 针对移动互联网推出的一套操作系统平台。近年来，随着移动智能技术的高速发展， Android 已经成为目前世界上使用最广泛的移动智能操作系统。在高速发展的同时，Android 平台上的安全问题也逐渐的凸显，虽然 Android 已有的安全机制包含了权限控制等安全措施，但仍然存在暴露用户隐私的问题，且其安全策略不利于普通用户控制和配置。

本文对 Android 平台上安全问题进行了深入研究，特别针对 Android 软件安全分析和安全加固两个方面进行分析。首先提出了一套系统化分析 Android 平台软件恶意行为的安全分析方法，该方法提取恶意行为的一般特征进行分析，我们通过实例分析证明方法是高效和可靠的；其次，设计了一套以数据为中心的 Android 系统安全增强方案，其主要思路是通过提供可信数据库、可信内核和数据实时加密等技术，对系统执行强制访问控制策略。我们提供的评估结果显示该方案的开销是可以接受的。

关键词： Android，恶意软件，逆向分析，电子取证，安全方案

Securiy Analysis and Enhancement for Android System and Applications

ABSTRACT

Google's Android is an operating system design for mobile device. As the rapid evolution of mobile device, Android is becoming the most widely used operating system for mobile platform. While deployed world-wide, there exists a huge number of security deficiency for Android. Even the Android OS has already enforced many security policies such as permission control for apps, users often expose privacy information to attacker due to the system's defensive privacy protecting policy, and it is still difficult for existing Android users to setup and control this security mechanism.

In this paper we study the Android security problem, especially Android malware and Android security enhancement, in depth. We first propose a systematic approach for analyzing current Android malware. The approach is based on basic features of Android malware such as information leakage and misuse of Android permissions. A real world sample of malicious app detection and analysis is given to prove that using our approach to analyze suspicious Android app and detect the malicious one is efficient and effective. Then we design and implement a a data-centric security enhancement scheme to actively restrict privacy violation on Android. The main idea is to first build trusted database by introducing secure enhanced kernel and data-at-rest encryption. Then, the system enforces an isolation of applications with

privacy data access privilege mode. We compared our scheme with other heavyweight isolation scheme and the result shows that the overhead is also controlled into an acceptable range.

Keywords: Android, security, reverse engineering, system enhancement

目 录

第一章 绪论

1.1 研究背景及意义

随着技术的进步，移动智能设备在人们的日常生活中承担的任务变得更加复杂与艰巨。除了处理个人日事务和私密信息之外，越来越多的大型企业和政府部门希望能够将具有高便携性及强大计算能力的移动智能设备引入企业工作和政务处理之中。而将移动智能设备引入企业及政府工作体系中，不但要求移动智能设备能够正常、高效、稳定的工作，还要求移动智能设备在接入内部专有网络时，能够保证移动智能终端设备中以及在专有网络中传输的数据的安全性。这就要求使用的移动智能设备本身具有极高的安全性能。

随着移动智能技术的高速发展，Android 已经成为世界上使用最广泛的移动智能操作系统。2012 年智能手机市场占有率的报告显示，从 2011 到 2012 年，Android 在全球智能手机市场上的份额从 49.2%上升到 68.8%，2012 年全球激活总共约 4.97 亿台终端设备。Android 应用平均每月下载量高达 10 亿次，Android 应用市场里的应用数量已经超过 45 万，Android 已经成为一个高速增长的生态系统。

在 Android 系统高速发展的同时，其安全问题也越来越引起人们的关注。企业级的移动智能设备通常需要极高的安全性能，而 Android 系统作为一个开源的新型操作系统，存在很多安全问题，当用户对通信内容、应用程序、位置信息和数据加密算法等方面提出更严格的要求时，现有的 Android 系统并不能够满足这些安全标准。为了满足高安全标准下的使用，美国、俄罗斯等国都基于 Android 系统开发了安全加固的移动智能设备，例如，美国国家安全局（NSA）正在测试其设计的超级安全 Android 智能手机，手机代号为 Fishbowl，已发给政府雇员测试。该系统能符合 NSA 严格的信息安全政策，同时又具有易于使用和造价低廉的优点。俄罗斯由副总理 Dmitry Rogozin 牵头设计的安全改进版 Android，不仅拥有 Android 全功能，且绝对不会把私人数据返回给 Google。该项目除了防止 Google 获取数据，还拥有更好的抵御常见漏洞的安全指标。该项目虽然是纯粹的国防部项目，但是也可以运营在普通智能手机上。

Android 系统的安全问题极大地影响了用户对 Android 系统的信任和使用，尤其是对安全需求高的领域。无论是恶意病毒木马的传播，还是应用程序中的种种隐私窃

取问题，都威胁着用户的安全。Android 智能终端管理的资源中涉及到用户切身利益，承载了大量存储和处理用户隐私数据的任务。由于 Android 系统独特的架构与特性，Android 平台下的恶意程序有着隐蔽性更强、破坏力更大、传播速度更快等特点。目前已知的 Android 恶意程序中，最为主要的恶意行为是恶意扣费和窃取用户隐私这两种。除了对已有恶意程序的分析评估之外，我们还要关注的就是针对将来可能出现的恶意程序的预知防护。

1.2 国内外研究现状

智能手机平台近年来迅速发展，其安全问题也越来越引人关注，对它的安全问题进行研究主要集中在最近几年。

在 Android 已有的安全增强方面，最主要的安全增强工作包括：第一，手机硬件保护功能。通常该功能是通过将存储分区中的一些关键目录（如/system）强制加锁，禁止修改，以保护系统安全，但此功能并非所有设备均具备；第二，基于 root 权限的保护，利用 Linux 系统的权限管理机制进行保护；第三，基于 SEAndroid 的安全增强方案（思想最初在 IEEE S&P 杂志上由文章 Securing Android-powered mobile devices using SELinux 提出），利用强制访问控制（Mandatory Access Control）来应对可能出现的未知 0-day 漏洞的攻击威胁。上述增强工作中，第一点需要硬件厂商配合，第二点基本上已经成熟，也无法对一些漏洞进行有效防范，而第三点工作是对第二点的有效改进，同时，这项工作从 2012 年 1 月刚刚开始通过美国国家安全局 NSA 下属 SELinux project 网站进行开放源代码和实际支持，并未完全成熟，也没有对真实的系统进行移植。

文献[2]设计了一套经过修改的 Android 系统，能够在仅造成 14%额外开销的情况下对用户隐私数据的流向进行实时的跟踪，经过测试发现超过 50%的应用程序具有对用户隐私数据的可疑操作。文献[3]通过对 Android 代码进行修改，让系统虚拟删除用户信息，来测试应用程序是否能容忍用户信息的缺失。文献[4]实现了一个对 iOS 上应用程序自动的静态分析工具：PiOS。工具中运用了从 Objective-C 二进制代码重建控制流图的方法，克服了静态分析 Objective-C 无法正确还原消息传递部分逻辑的困难。文献[5][6][7]分别提出了 Android 和 Windows Mobile 手机平台的安全方案，控制应用程序对敏感数据的访问，但是并不能跟踪程序对数据的具体处理过程。文献[8][9]通过黑盒的方式对应用程序进行安全分析，但是不能应对程序将隐私数据加密后泄露的情况。文献[20]通过监视手机连接网络 HTTP 包的内容分析了其中隐私泄露的问题。

文献[21]给出了一种对 iOS 手机平台进行深入取证的方法。

其他平台的软件动态分析方法研究，国内外也有着不少研究成果。

文献[22]设计了一套基于 Bochs 的动态指令流分析系统，可以动态记录 x86 平台下应用程序的运行轨迹。文献[23]推出了名为 BitBlaze 的二进制代码分析平台，它基于 QEMU 虚拟机，通过对二进制代码的行为分析，把低抽象的二进制代码还原为较易理解的高抽象层语义模型，便于分析程序的安全性。文献[24]提出了一套精简二进制代码的分析方法。文献[25]提出了一套基于 Taint 分析技术的自动化漏洞检测方法。

总结国内外研究的先状，国外对智能手机平台的安全问题研究主要集中在用户隐私泄露问题，而国内对智能手机平台，尤其是 Android 平台的安全问题研究还较为欠缺。而对于其他平台的软件动态分析方法研究，国内外也已拥有一定的研究成果。

1.3 主要研究内容

本论文的主要目的是，提出一套 Android 平台上的软件恶意行为安全分析方法，该方法包括如何快速的鉴定恶意软件，如何去除软件代码的混淆保护，如何对代码的结构模块进行分析，以及如何结合分析进行恶意行为提取重建。对一个实际的恶意软件入侵案例（来自于国际安全组织 honeynet 挑战赛[19]），进行了深入彻底的分析研究，提取还原了软件中的全部恶意行为和相关的数据。包括恶意软件的工作原理，通讯协议，窃取的隐私数据内容等。通过该实例的分析，证明我们提出的方法是高效和可靠的。针对 Android 平台的系统安全，提出了一套系统安全增强方案。该方案建立了可信的数据库，并对应用程序进行权限隔离。该方案的设计集中在数据保护上。和同类研究相比，具有轻量级的特点。

具体如下：

1. 提出了一套系统化的分析 Android 软件恶意行为的安全分析方法，给出了 Android 恶意代码的一般特征，归纳了一套通用的逆向分析的方法流程。包括如何快速的鉴定恶意软件，如何对抗恶意软件的反分析代码，如何根据恶意代码进行行为重建等。该方法提取恶意行为的一般特征进行分析，包含鉴别，代码反保护，代码分析，行为提取等步骤。

在提取了恶意软件，去除了恶意软件的代码保护手段之后，为了最终还原出恶意软件的行为，分析者还需要对恶意软件的代码进行大量的分析理解工作。这是整个逆向分析过程最关键的步骤。

我们总结了一些恶意软件常有的代码特征模块，可以帮助分析者更好更快的进行

分析理解恶意软件：

(a) 后台服务 Core 循环

恶意软件需要有一个在后台持续运行的服务，从代码结构上看，需要一个循环结构，控制程序总的运行流程，找到了这个循环，就找到了恶意代码的总入口。

(b) 通讯协议

恶意软件通常使用自定义的特殊协议和远程服务器进行通讯。那么必然能够从它的代码中找到相关的协议实现代码。这些代码模块需要把发送的数据打包并加密，把从服务器接收的数据解密和解包。

(c) 密码学算法

为了实现加密解密相关步骤，恶意软件需要调用系统的密码算法库，以实现如对本机特征码计算消息摘要，解码内置字符串常量，和服务器进行密钥交换，和服务器进行加密通讯等功能。因此，密码算法 API 调用，密码学计算等相关代码，也是恶意程序代码的重要模块。

(d) 隐私数据访问

窃取隐私数据是恶意软件的核心功能之一，为了实现这个功能，它需要访问系统中的关键数据

2．本文选取了一个 Android 系统受到恶意软件入侵的实例进行分析，以一个实际的恶意软件为例（来自于国际安全组织 honeynet 挑战赛），进行了深入彻底的分析研究，我们有效的提取出了该恶意软件与秘密服务器进行加密通讯，窃取用户隐私数据等全部的恶意行为。还原提取了软件中的恶意行为和相关的数据。包括恶意软件的通讯协议，通讯过程中发送的隐私数据等。通过该实例的分析，证明我们提出的方法是高效和可靠的。

3．提出针对 Android 的系统安全增强方案，基于 Android 已有的安全机制进行了安全增强。首先，我们吸收了 Kernel 层的 SEAndroid 强制访问方案作为访问基础，其次，我们通过轻量级数据保护方案保护了敏感数据的安全，最后，我们在应用层面上对应用程序进行了权限和信息的严格监管。

本文就 Android 系统的内核保护、数据加密和应用程序保护这三个具体内容进行讨论，详细描述了本方案是如何设计从而保护这些内容。该方案建立了可信的数据库，并对应用程序进行权限隔离，该方案的设计集中在数据保护上。

1.4 本文组织结构

本文由以下章节组成：

第一章：总体介绍 Android 软件安全分析和系统安全增强研究的背景和意义，国内外研究进展，并阐明本文的研究意义与主要研究内容。

第二章，重点介绍 Android 系统的结构和安全架构，总结 Android 现有安全机制，当前学术界对 Android 平台软件安全分析的各项研究结果以及现有安全增强方案的研究的缺陷与不足。

第三章，主要对 Android 恶意软件的分析的方法研究。提出一套 Android 平台上的软件恶意行为安全分析方法，该方法包括如何快速的鉴定恶意软件，如何去除软件代码的混淆保护，如何对代码的结构模块进行分析，以及如何结合分析进行恶意行为提取重建。对一个实际的恶意软件入侵案例，进行了深入彻底的分析研究，提取还原了软件中的全部恶意行为和相关的数据。

第四章，提出针对 Android 的系统安全增强方案。提出了一套以数据为中心的 Android 的系统安全增强方案，对 Android 已有的安全机制进行了安全增强。首先，我们吸收了 Kernel 层的 SEAndroid 强制访问方案作为访问基础，其次，我们通过轻量级数据保护方案保护了敏感数据的安全，最后，我们在应用层面上对应用程序进行了权限和信息的严格监管。

第五章，对全文的工作进行总结，并对今后 Android 软件安全分析和系统安全增强研究进行了展望。

第二章 Android 系统和软件安全问题

2.1 Android 系统结构

Android 作为一个移动设备的平台，其软件层次结构包括了一个操作系统，中间件和应用程序。根据 Android 的软件框图，其软件层次结构自下而上分为以下几个层次（图 2-1）：

- 操作系统层（OS）
- 各种库（Libraries）和 Android 运行环境（RunTime）
- 应用程序框架（Application Framework）
- 应用程序（Application）

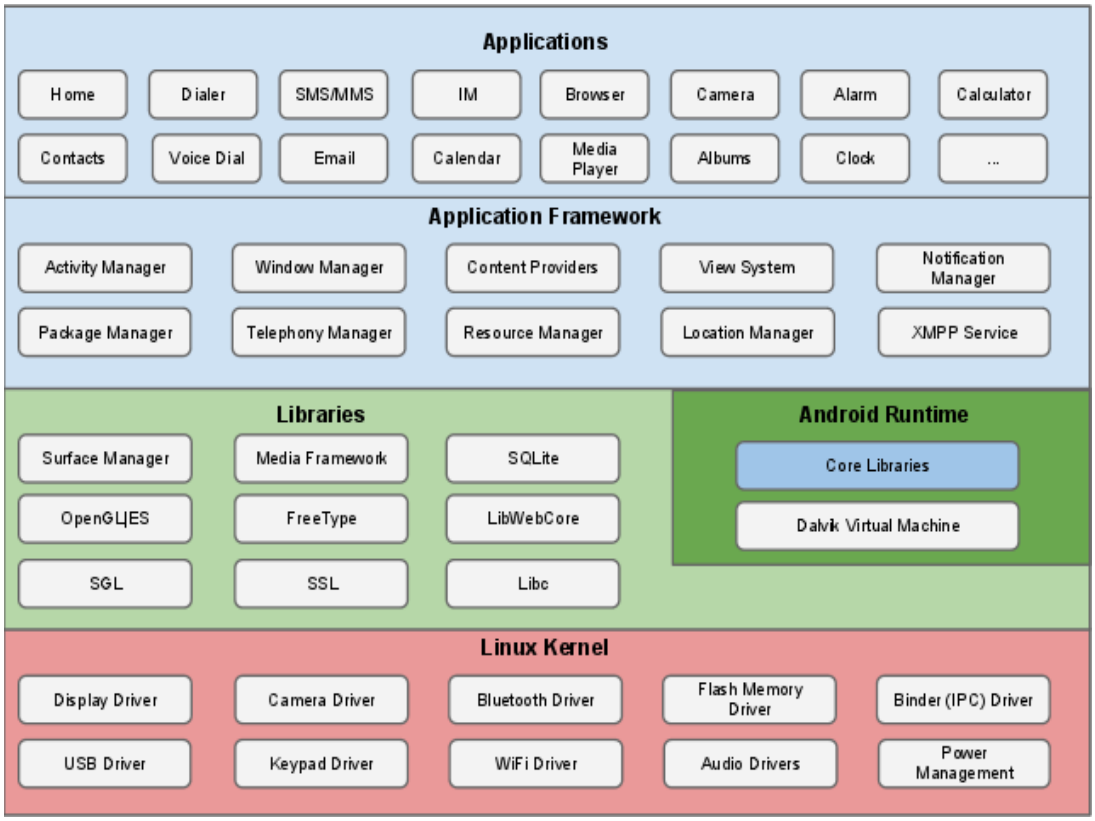


图 2-1　Android 分层结构[1]

以下分别介绍 Andoid 各个层次的软件的重点及其相关技术。

2.1.1 操作系统层

Android 使用 Linux2.6 作为操作系统，Linux2.6 是一种标准的技术，Linux 也是一个开放的操作系统。Android 对操作系统的使用包括核心和驱动程序两部分，Android 的 Linux 核心为标准的 Linux2.6 内核，Android 还包含了一些与移动设备相关的驱动程序。主要的驱动如下所示：

- 显示驱动：常用基于 Linux 的帧缓冲（Frame Buffer）驱动
- Flash 内存驱动
- 照相机驱动：常用基于 Linux 的 V4L 驱动
- 音频驱动：常用基于 ALSA（Advanced Linux Sound Architecture，高级 Linux 声音体系）驱动
- WiFi 驱动：基于 IEEE 802.11 标准的驱动程序
- 键盘驱动
- 蓝牙驱动
- Binder IPC 驱动：Andoid 一个特殊的驱动程序，具有单独的设备节点，提供进程间通讯的功能
- 能源管理

2.1.2 库和 Android 运行环境

本层次对应一般嵌入式系统，相当于中间件层次。Android 的本层次分成两个部分，一部分是各种库，另一部分是 Android 运行环境。本层的内容大多是使用 C++ 实现的。其中的库包括：

- C 库：C 语言的标准库，这也是系统中一个最为底层的库，C 库是通过 Linux 的系统调用来实现。
- 多媒体框架：这部分内容是 Android 多媒体的核心部分，基于 PacketVideo 的 OpenCORE，从功能上本库一共分为两大部分，一个部分是音频、视频的回放，另一部分是则是音视频的纪录。
- SGL：2D 图像引擎。
- SSL：即 Secure Socket Layer 位于 TCP/IP 协议与各种应用层协议之间，为数据通讯提供安全支持。
- OpenGL ES 1.0 ：本部分提供了对 3D 的支持。

- 界面管理工具：本部分提供了对管理显示子系统等功能。
- SQLite：一个通用的嵌入式数据库。
- WebKit：网络浏览器的核心。
- FreeType：位图和矢量字体的功能。

Android 的库一般是以系统中间件的形式提供的，它们均有的一个显著特点就是与移动设备的平台的应用密切相关。Android 运行环境主要指的虚拟机技术——Dalvik。Dalvik 虚拟机和一般 JAVA 虚拟机（Java VM）不同，它执行的不是 JAVA 标准的字节码（bytecode）而是 Dalvik 可执行格式（.dex）中执行文件。在执行的过程中，每一个应用程序即一个进程（也就是 Linux 的一个进程）。二者最大的区别在于 Java VM 是以基于栈的虚拟机（Stack-based），而 Dalvik 是基于寄存器的虚拟机（Register-based）。显然，后者最大的好处在于可以根据硬件实现更大的优化，这更适合移动设备的特点。

2.1.3 应用程序框架

Android 的应用程序框架（Application Framework）为应用程序层的开发者提供 APIs，它实际上是一个应用程序的框架。由于上层的应用程序是以 JAVA 构建的，因此本层次提供的首先包含了 UI 程序中所需要的各种控件， 例如： Views（视图组件），包括 lists（列表），grids（栅格），text boxes（文本框），buttons（按钮）等，甚至一个嵌入式的 Web 浏览器。一个 Android 的应用程序可以利用应用程序框架中的以下几个部分：Activity（活动）、Broadcast Intent Receiver（广播意图接收者）、Service（服务）、Content Provider（内容提供者）。

2.1.4 应用程序

Android 的应用程序主要是用户界面，通常以 JAVA 程序编写，其中还可以包含各种资源文件，这些资源文件放置在 res 目录中。JAVA 程序及相关资源经过编译后，将生成一个 APK 包。Android 本身提供了主屏幕（Home），联系人（Contact），电话（Phone），浏览器（Browsers）等众多的核心应用。同时应用程序的开发者还可以使用应用程序框架层的 API 实现自己的程序。

2.2 Android 现有安全机制

Android 的安全模型主要分为数据和权限两大部分：

2.2.1 数据

Android 是一个基于 linux 的多进程系统，在这个系统中，应用程序（或者系统的部分）会在自己的进程中运行。系统和应用之间的安全性是通过 Linux 在进程级别来强制实现的，比如会给应用程序分配 user ID 和 Group ID。Linux 的基础安全机制能防止进程之间的互相访问，也能对一些特殊的数据的访问权限进行限制，所以应用程序之间的一般是不可以互相访问的，但是 Android 提供了一种权限（permission）机制，用于应用程序之间数据和功能的安全访问。

每一个 Android 应用程序（*.apk 文件）都会在安装时就分配一个独有的 Linux UID，程序运行在沙盒中，不能与其他应用程序进行接触。这个用户 ID 会在安装时分配给它，并在该设备上一直保持同一个数值。由于安全性措施是发生在进程级，所以两个 package 中的代码不会运行在同一个进程当中，他们要作为不同的 Linux 用户出现。默认情况下他们是无法相互访问的。但是我们可以通过修改 AndroidManifest.xml 文件中的 manifest 标签中的 sharedUserId 属性，来使不同的 package 的程序共用同一个用户 ID。通过这种方式，这两个 package 就会被认为是同一个应用程序，拥有同一个用户 ID（只有当两个应用程序被同一个签名签署的时候才能够申请拥有同一个用户 ID），并且拥有同样的文件存取权限。有关 app 的签名会在下面进行说明。

2.2.2 权限

所有的 Android 应用程序（*.apk 文件）必须用证书进行签名认证，而这个证书的私钥是由开发者保有的。该证书可以用以识别应用程序的作者。该证书也不需要 CA 签名认证。Android 应用程序使用的是自签名证书。证书是用于在应用程序之间建立信任关系，而不是用于控制程序是否可以安装。开发程序默认生成的 apk 文件的签名是 debug 版本的签名。

Android 的权限分为两种，一种是系统提供的权限，主要是针对手机设备的操作和对系统数据的读写。还有一种是开发者自己声明的权限，主要是对开发者所开发的应用中一些接口和功能的使用。系统提供的权限其实也是开发者声明的权限，不过开发者是 system 权限。

权限的级别有四种，也可以分为两大类，一类包括了普通级别（Normal），危险级别（dangerous）两种，这两种权限是只要应用程序申请之后就能够使用的。还有一类是签名级别（signature）和系统/签名级别（signature or system），这一类需要申请的程序和声明权限的程序具有相同的签名。系统应用可以使用任何权限。权限的声明者可无条件使用该权限。开发者可以通过修改 Android 源码然后 make 的方式来生成带有 system 签名的 app，也可以从源码中取出系统的签名密钥，然后用这个密钥给自己写的 app 签名，这样自己开发的 app 也同样获得全部的系统权限。

当一个应用需要使用某个权限的时候，通过在 AndroidManifest.xml 文件中增加<permission>标签来申请。应用程序安装的时候，应用程序请求的 permissions 通过 package installer 来批准获取。Package installer 通过检查该应用程序的签名和所申请权限的级别来判断是否给予该程序要求的权限。在用户使用过程中不会去检查权限，也就是说要么在安装的时候就批准该权限，使其按照设计可以使用该权限；要么就不批准，这样用户也就根本无法使用该功能，也不会有任何提示告知用户尝试失败。

2.3 Android 系统安全方案研究

国内外针对 Android 系统方面的安全研究主要分为安全分析和安全加固两个方面的工作。我们首先介绍已有的 Android 系统安全分析，然后对已有的 Android 安全加固方案进行总结，最后，对现有研究的缺陷与不足进行分析。

2.3.1 TISSA

TISSA[27]是一个针对 Android 系统中用户隐私数据的保护系统。该系统会监控整个系统中对用户隐私数据的访问并根据用户配置的不同来实施访问控制。用户可以自行决定向哪些应用程序开放哪些隐私数据的访问权限，并且用户能够制定当有应用程序访问未授权的隐私数据时系统所做的应对方式，例如拒绝访问、返回虚假信息、返回空白信息等。

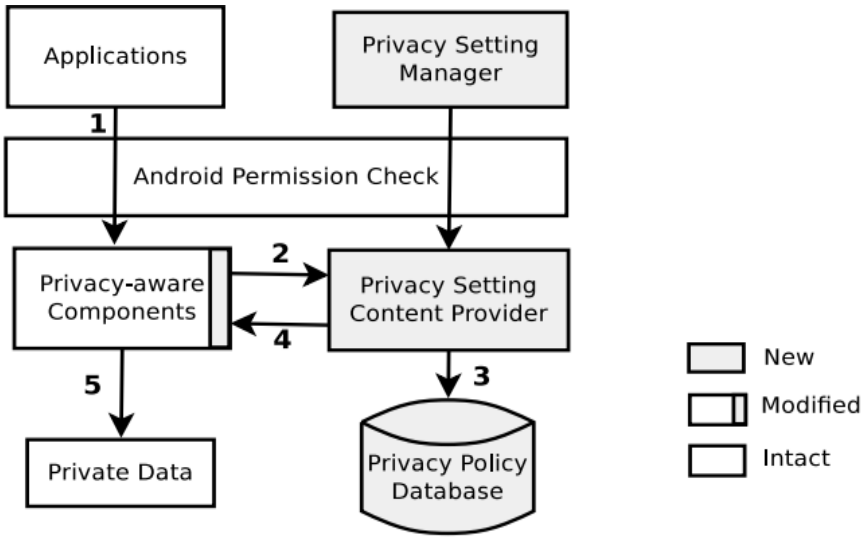


图 2-2 TISSA 系统基本架构[27]

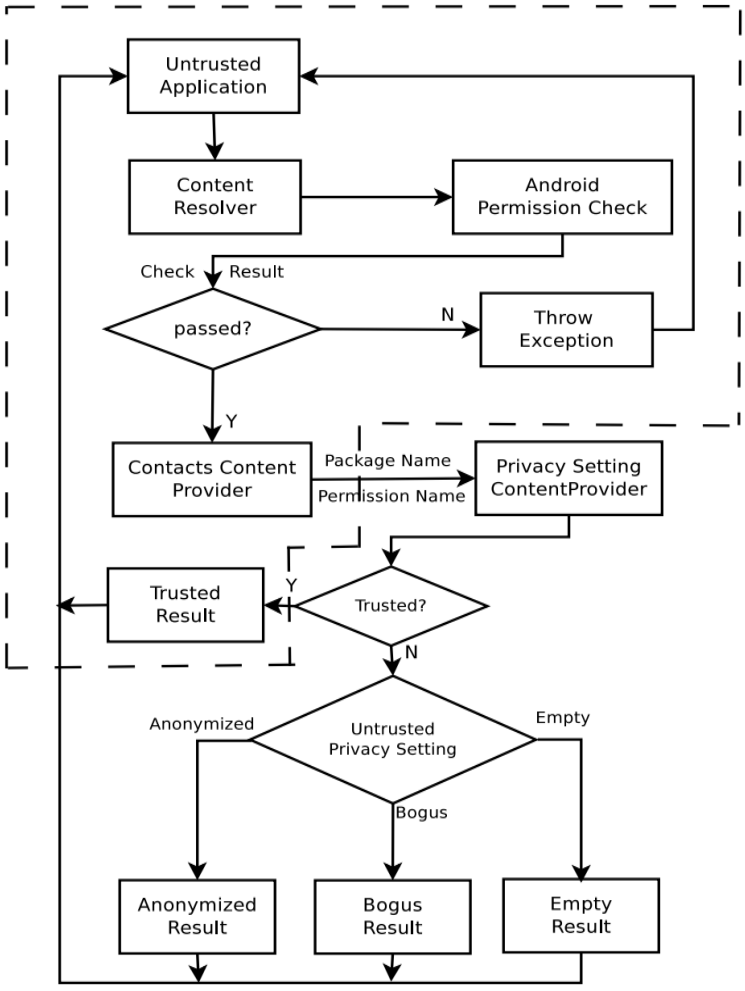


图 2-3 TISSA 实施一次针对 Contact 的访问控制流程[27]

2.3.2 ComDroid

在 Android 系统中，应用程序之间的通信由 Intent 组件实现，而由于系统设计原因，使用 Intent 进行应用程序之间的信息传递会导致潜在的安全漏洞。Intent 所携带的内容能够被攻击者嗅探、修改、盗取、置换。另一个方面，由于 Intent 的特性，恶意软件能够伪造带有恶意信息的 Intent，当这些 Intent 被普通应用程序接收时，就有可能造成用户隐私数据的泄漏。ComDroid[42]就是一款针对这种漏洞的 Android Apk 静态分析工具，ComDroid 通过对 Apk 文件的静态扫描来检测 Apk 中是否包含了存在 Intent 漏洞隐患的代码实现。

Exposed Communication
ComDroid has found exposed communication.

- Possible Activity Hijacking: fred/share/AppProtectService$1/run()@305, Source Line: 331, hasExtras=false, hasRead=false, hasWrite=false
- Possible Activity Hijacking: fred/share/ConfirmPassword$2/onClick(Landroid/view/View;)@55, Source Line: 155, hasExtras=false, hasRead=false, hasWrite=false
- Possible Activity Hijacking: fred/share/ConfirmPassword/backHome()@40, Source Line: 262, hasExtras=false, hasRead=false, hasWrite=false
- Possible Malicious Activity Launch: fred.share.GestureAppProtect, 0
- Possible Malicious Activity Launch: fred.share.ReadSms, 1
- Possible Malicious Activity Launch: fred.share.contact, 0
- Possible Malicious Activity Launch: fred.share.showSms, 0
- Possible Malicious Service Launch: fred.share.AppProtectService, 0

图 2-4　使用 ComDroid 分析 Apk 结果示例

2.3.3 TrustDroid

TrustDroid[43]是一个提供轻量级域隔离功能的 Android 安全增强框架。该框架在从内核到网络的多个系统级别上实现了资源分类、自主访问控制、强制访问控制等功能。该框架首先对内核级别的 ICC(Inter-Component Communication)以及网络层使用强制访问控制来保证基本安全，之后允许用户对文件系统实现自主访问控制。用户能够对系统中的基本资源（应用程序、硬件设备、敏感数据等）进行分类，并对所属不同分类的资源赋予不同的访问权限。通过使用该框架，能够满足类似可信计算的安全需求。

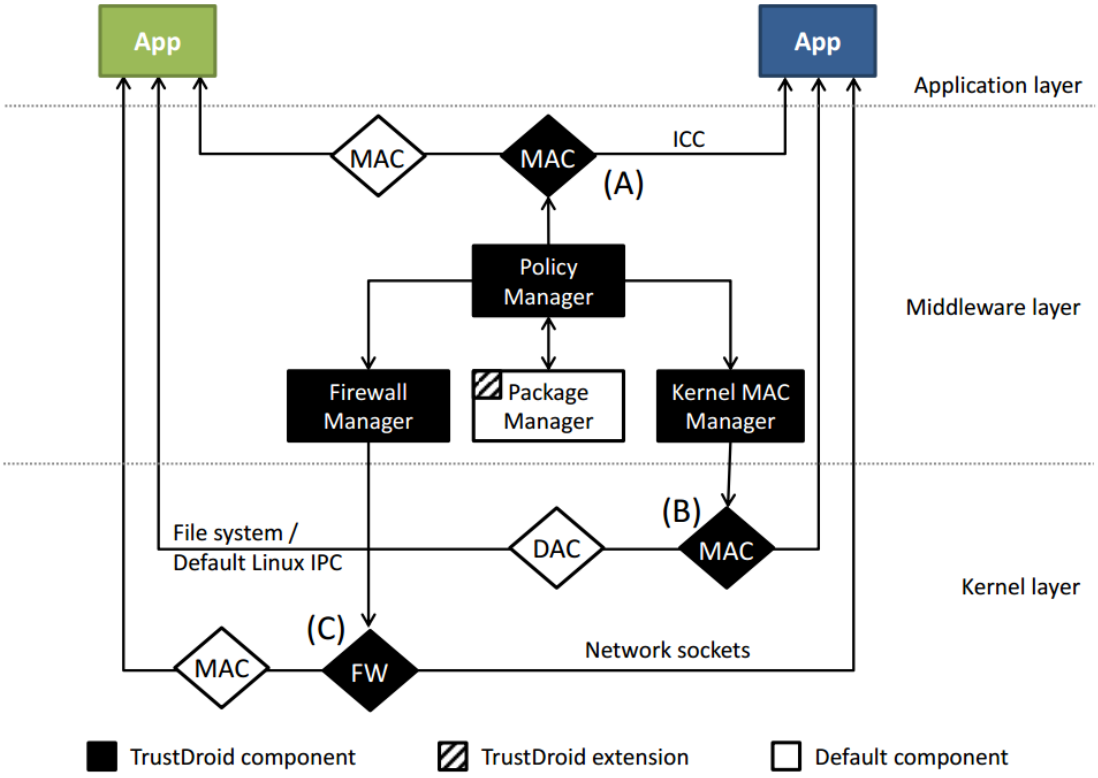


图 2-5　TrustDroid 系统基本架构[43]

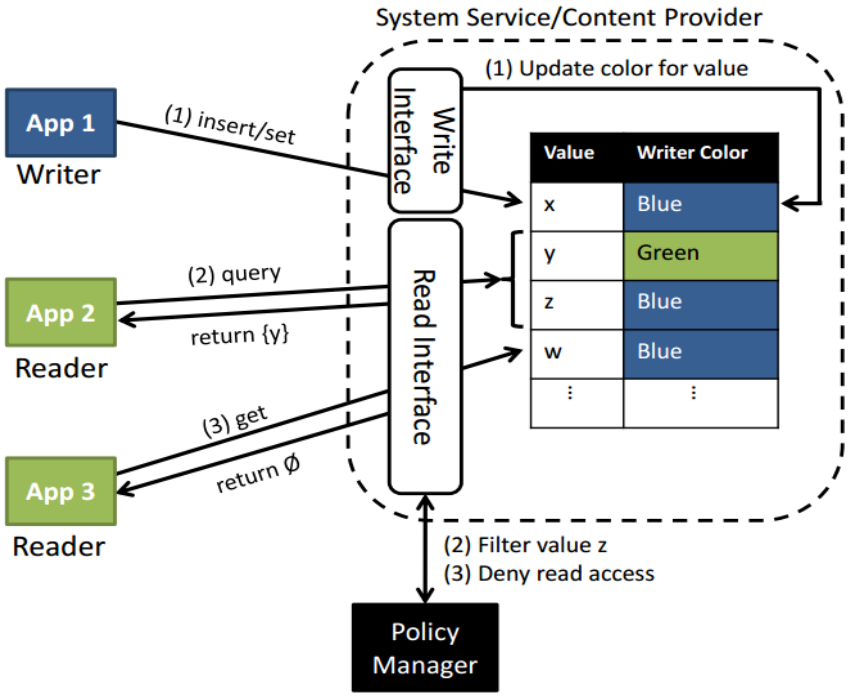


图 2-6　Content Provider 自主访问控制示例[43]

2.3.4 SELinux

SElinux[28]是一种强制访问控制（mandatory access control）的实现。它的做法是以最小权限原则（principle of least privilege）为基础，在 Linux 核心中使用 Linux 安全模块（Linux Security Modules）。它并非一个 Linux 发布版，而是一组可以套用在类 Unix 操作系统（如 Linux、BSD 等）的修改。

SELinux 是 2.6 版本的 Linux 内核中提供的强制访问控制(MAC）系统。对于目前可用的 Linux 安全模块来说，SELinux 是功能最全面，而且测试最充分的，它是在 20 年的 MAC 研究基础上建立的。SELinux 在类型强制服务器中合并了多级安全性或一种可选的多类策略，并采用了基于角色的访问控制概念。

大部分使用 SELinux 的人使用的都是 SELinux 就绪的发行版，例如 Fedora、Red Hat Enterprise Linux (RHEL)、Debian 或 Centos。它们都是在内核中启用 SELinux 的，并且提供一个可定制的安全策略，还提供很多用户层的库和工具，它们都可以使用 SELinux 的功能。

SELinux 是一种基于“域-类型”模型（domain-type）的强制访问控制（MAC）安全系统，它由 NSA 编写并设计成内核模块包含到内核中，相应的某些安全相关的应用也被打了 SELinux 的补丁，最后还有一个相应的安全策略。

众所周知，标准的 UNIX 安全模型是 DAC 模型，即“任意的访问控制”。也就是说，任何程序对其资源享有完全的控制权。假设某个程序打算把含有潜在重要信息的文件放在/tmp 目录下，那么在 DAC 情况下是不会阻止它的。

而 MAC 情况下的安全策略完全控制着对所有资源的访问。这是 MAC 和 DAC 本质的区别。

SELinux 提供了比传统的 UNIX 权限更好的访问控制。

优点：

- 优点 1：MAC(Mandatory Access Control）－－对访问的控制彻底化

对于所有的文件，目录，端口这类的资源的访问，都可以是基于策略设定的，这些策略是由管理员定制的、一般用户是没有权限更改的。

- 优点 2：TE（Type Enforcement）－－ 对于进程只赋予最小的权限

TE 概念在 SELinux 里非常的重要。它的特点是对所有的文件都赋予一个叫 type 的文件类型标签，对于所有的进程也赋予各自的一个叫 domain 的标签。Domain 标签能够执行的操作也是由 access vector 在策略里定好的。

以 apache 服务器为例，httpd 进程只能在 httpd_t 里运行，这个 httpd_t 的 domain

能执行的操作，比如能读网页内容文件赋予 httpd_sys_content_t，密码文件赋予 shadow_t，TCP 的 80 端口赋予 http_port_t 等等。如果在 access vector 里我们不允许 http_t 来对 http_port_t 进行操作的话，Apache 启动都不能启动。反过来说，我们只允许 80 端口，只允许读取被标为 httpd_sys_content_t 的文件，httpd_t 就不能使用别的端口，也不能更改那些被标为 httpd_sys_content_t 的文件，它们是只读的。

- 优点 3：domain 迁移 －－ 防止权限升级

在用户环境里运行点对点下载软件 azureus，比如，假设当前的 domain 是 fu_t，但是，考虑到安全问题，我们打算让他在 azureus_t 里运行，如果在 terminal 里用命令启动 azureus 的话，它的进程的 domain 就会默认继承你实行的 shell 的 fu_t。

有了 domain 迁移的话，我们就可以让 azureus 在我们指定的 azureus_t 里运行，在安全上面，这种做法更可取，它不会影响到 fu_t 这个 domain。

- 优点 4：RBAC（role base access control）－－ 对于用户只赋予最小的权限

对于用户来说，被划分成不同的 ROLE，即使是 ROOT 用户，如果不在 sysadm_r 里，也还是不能实行 sysadm_t 管理操作的。因为，哪些 ROLE 可以执行哪些 domain 也是在策略里设定的。ROLE 是可以迁移的，但是也只能按策略规定的进行迁移。

缺点：

- 缺点 1：存在特权用户 root

任何人只要得到 root 的权限，对于整个系统都可以为所欲为。这一点和 Windows 系统一样。

- 缺点 2：对于文件的访问权的划分不够细

在 linux 系统里，对于文件的操作，只有“所有者”，“所有组”，“其他”这 3 类的划分。对于“其他”这一类里的用户要进行更细的划分的话是没有办法的。

- 缺点 3：SUID 程序的权限升级

如果设置了 SUID 权限的程序有了漏洞的话，很容易被攻击者所利用。

- 缺点 4：DAC（Discretionary Access Control）问题

文件目录的所有者可以对文件进行所有的操作，这给系统整体的管理带来不便。

对于以上这些的不足，防火墙，入侵检测系统都是无能为力的。

2.3.5 XmanDroid

权限提升攻击是一种利用 Android 系统进程间通信机制缺陷所完成的恶意攻击行为，通过权限提升攻击，攻击者能够使一个应用程序获得其自身未申请的系统权限。

XmanDroid[44]就是一个用于防范此种攻击的 Android 安全增强框架。

XmanDroid 能够检测系统之中应用程序之间的通信，并验证此次通信是否为潜在的权限提升攻击，完成验证之后通过查询实现存储在系统中策略配置来决定此次通信是否有效。

(1) An application that is notified about *incoming or outgoing calls* and can *record audio* must not communicate to an application with *network access*.
(2) An application that can obtain *location information* must not communicate to an application that has *network access*.
(3) An application that has *read access to the user contact database* must not communicate to an application that has *network access*.
(4) An application that has *read access to user SMS database* must not communicate to an application that has *network access*.
(5) An application that has no explicit *network access* must not be able to *download archived files* and *install packages*.
(6) An application that has no explicit permission to perform *outgoing voice calls* can perform such calls only upon *user confirmation*.
(7) An application that has no explicit permission to send *text messages* can send them only upon *user confirmation*.

图 2-7　XmanDroid 用于判断权限提升攻击的规则示例

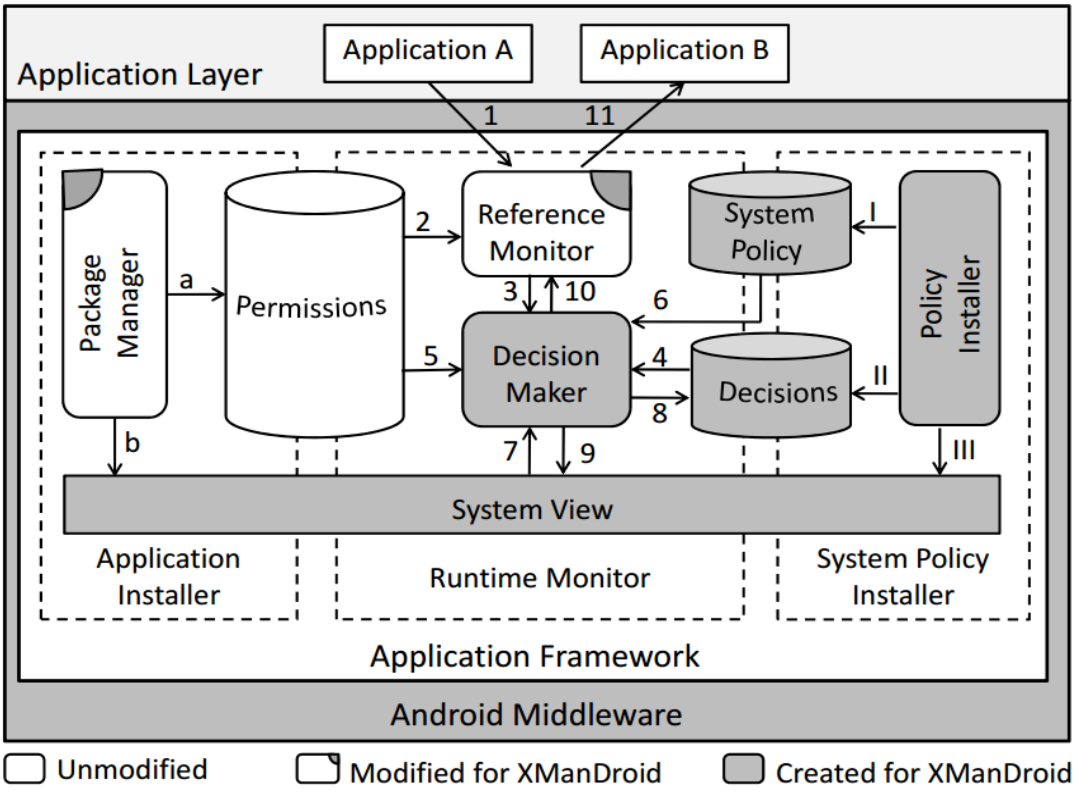


图 2-8　XmanDriod 系统基本架构[44]

2.3.6 Aurasium

Aurasium[45]是一款实现 Android 系统细粒度权限控制的工具。Aurasium 通过对 Apk 文件进行修改来实现对应用程序行为的实时监控。首先 Aurasium 在 Apk 中添加

自行修改过的 C/C++基本库，这些代码的功能是对上层的 java 代码进行重定向，当上层的java代码需要调用底层的C/C++库，会被重定向至Aurasium自己实现的C/C++基本库，这样做的好处在于，无论上层的 java 代码用何种方式实现，当 java 代码需要调用底层 C/C++基本库时，都会被 Aurasium 截获。之后 Aurasium 会修改应用程序的dex文件，dex文件由应用程序的java代码编译而成，在反编译dex文件后，Aurasium会对代码中实现各种敏感行为（打电话、发短信、获取敏感信息等）的具体代码进行修改，修改后的代码在被执行时会进行额外的策略匹配，当符合 Aurasium 预设的策略时，将会向用户弹窗告警（图 2-10）。

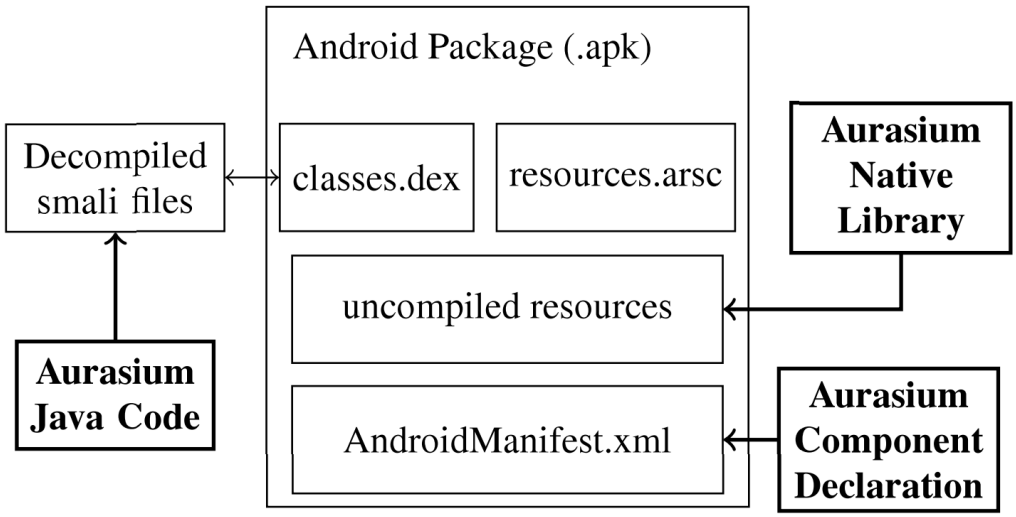


图 2-9　Aurasium 基本架构[45]

图 2-10　程序申请 IMEI（敏感数据）时告警

2.3.7 Dr. Android and Mr. Hide

Dr. Android and Mr. Hide[46]是一个和 Aurasium 类似的 Android 安全增强工具。该工套具通过修改 Apk 中的 dex 文件实现了一套相较于 Android 原生更为细粒度的权限机制。MR.Hide 是常驻 Android 系统后台的一个服务，该服务的主要功能是根据重新定制的细粒度权限机制提供一整套的全新系统 API 共其他程序使用。DR.Android 是一个 Apk 修改工具，其功能是修改 Apk 文件中的 dex 文件，对其中调用系统 API 的代码进行替换，通过分析原有代码的意图，将原有代码替换为调用 MR.Hide 提供的细粒度 API。由此解决了 Android 原有权限机制过于粗糙的问题。

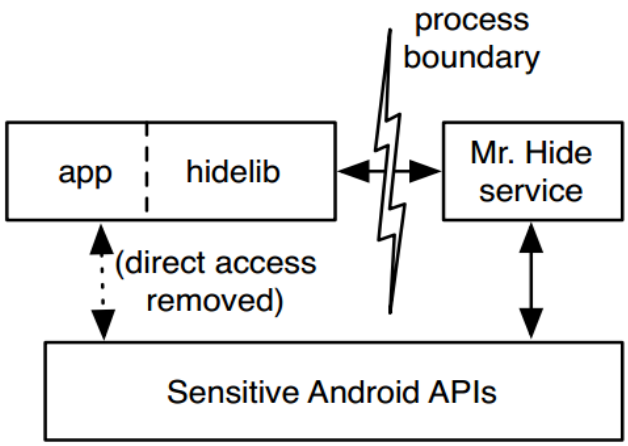


图 2-11 MR.Hide 基本架构[46]

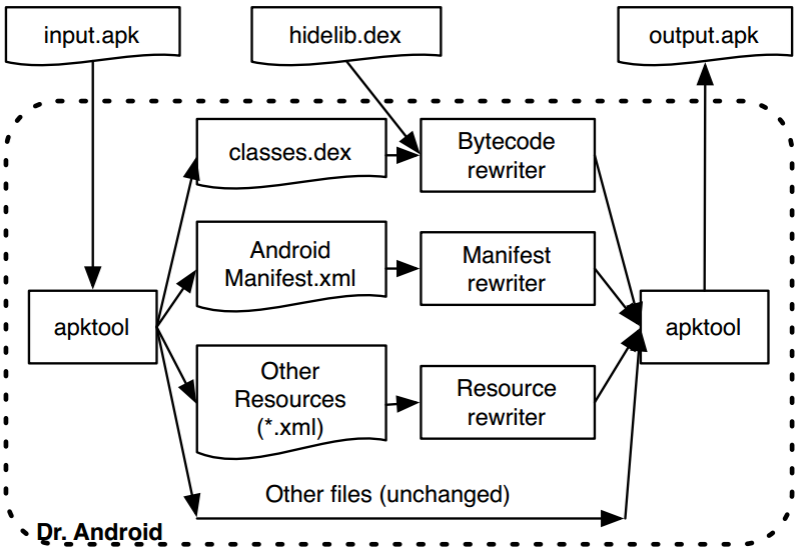


图 2-12 Dr.Android 基本架构[46]

2.4 现有安全增强方案的研究的缺陷与不足

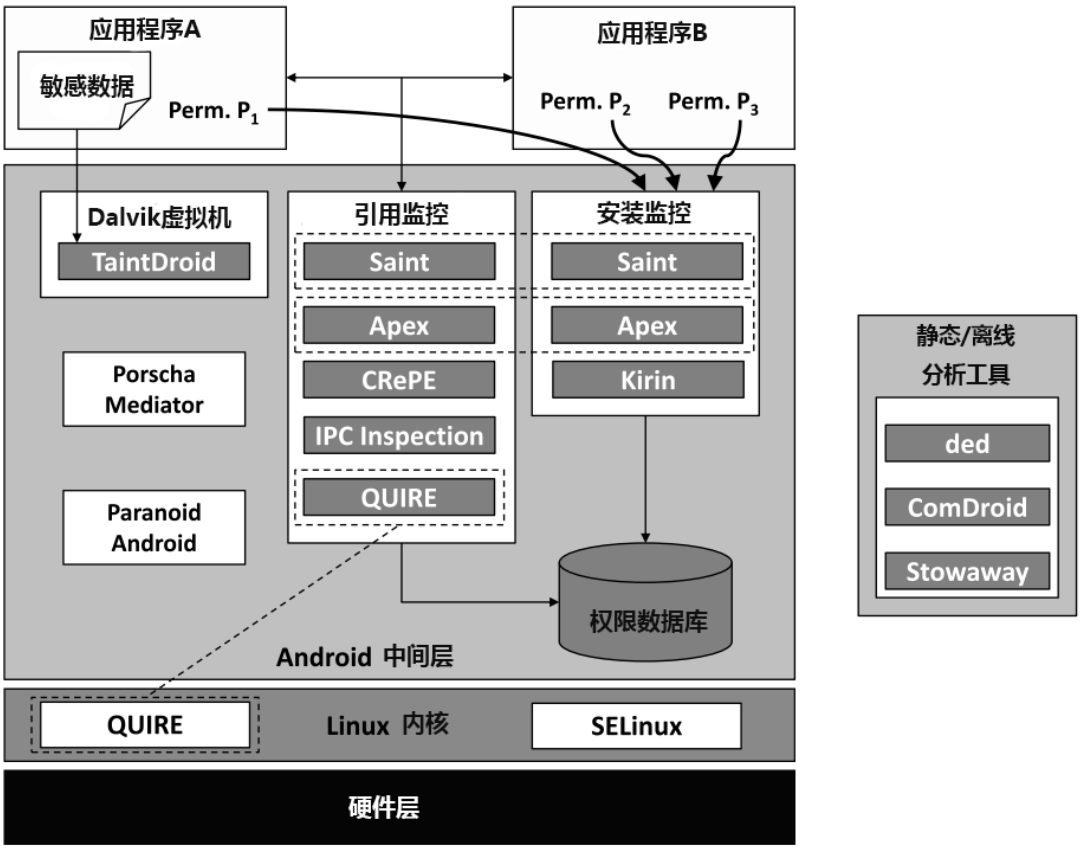


图 2-13 当前 Android 安全工具的层次分布

图 2-13（当前 Android 安全工具的层次分布）展示了当前各类安全分析与增强工具在 Android 系统层面上的分布情况，可以看到现有各类 Android 安全研究对各类安全问题进行了比较广泛的讨论，但应用于实际环境中，还缺乏一个整体性的解决方案。图 2-14（当前主要的安全增强方案功能）展示了当前一些主要的安全增强方案的优点与不足之处。

可以看到，对于 Android 系统的安全增强方案，主要需要考虑六个方面的安全特性，它们是：

- 防御权限暴露攻击（Permission re-delegation Attack）
- 用户可以配置策略（Based on user/developer policy）
- 保护敏感数据（Sensitive data protection）

- 减少误报率（false positives/false negatives）
- 不损害原有功能（Does not reduce devices functionality)
- 没有明显性能损耗（No substantial performance overhead)

	防御权限暴露攻击	没有明显性能损耗	用户可以配置策略	保护敏感数据	减少误报率	不损害原有功能
XManDroid	✓	✗	✓	✗	✓	✗
CRePE	✗	✗	✓	✗	✗	✗
QUIRE	✓	✓	✗	✗	✗	✓
Kirin	✗	?	✗	✗	✗	✓
IPC Inspection	✓	✗	✗	✗	✓	✗
ComDroid	✗	✓	✗	✗	✓	✓
TrustDroid	✗	✗	✗	✗	✗	✗
Saint	✗	?	✓	✗	✓	✗
Stowaway	✗	✓	✗	✗	✗	✓
TaintDroid	✗	✓	✗	✓	✗	✓
Apex	✗	✗	✓	✗	✗	✗
SELinux	✓	✗	✓	✗	✗	✓
Porscha	✗	✗	✓	✗	✗	✗

图 2-14　当前主要的安全增强方案功能

而当前的安全增强方案，基本上仅能做到 2-4 点，很难兼顾全局，因此很难被应用于实际系统中。

2.5 本章小结

本章重点介绍 Android 系统的结构和安全架构，总结 Android 现有安全机制，当前学术界对 Android 平台软件安全分析的各项研究结果以及现有安全增强方案的研究的缺陷与不足。

第三章 Android 软件安全分析和恶意行为提取

本章主要介绍 Android 软件安全分析的工作，我们提出了一套系统化的分析 Android 软件恶意行为的安全分析方法，该方法提取恶意行为的一般特征进行分析，包含鉴别，代码反保护，代码分析，行为提取等步骤。同时，我们选取了一个 Android 系统受到恶意软件入侵的实例进行分析，我们有效的提取出了该恶意软件与秘密服务器进行加密通讯，窃取用户隐私数据等全部的恶意行为。

3.1 Android 软件结构分析

Android 系统上的软件大都使用 Java 语言开发，并且运行在 Dalvik 虚拟机上。虽然 Android 是基于 Linux 的操作系统，但是恶意软件一般也是伪装成正常的 Android 应用程序，通过安装的方式来植入，完成植入后，以 APK 包的形式，存储在设备中。因此，为了能够进行 Android 软件安全分析，首先要清楚 Android 系统上一般的应用程序的结构特点。

Android 系统的应用程序在发布的时候，会把所有的部分，包括代码和资源文件，全部打包进一个单独的 APK 包文件里面。完成安装后，这个 APK 文件会被拷贝到系统的特定目录下。系统预装的程序，通常是放在/system/app 目录下，对于用户安装的应用程序，一般是放在/data/app 目录下。

APK 文件实际上是一个 ZIP 压缩包，其中包含了代码，资源，签名，以及 manifest 配置文件，这个文件的格式是符合 JAR 文件格式标准的。

从软件安全分析的角度看，我们需要从 APK 文件中提取的信息，主要分为以下三个方面。

（1）哈希值

这里指的是整个 APK 文件的 MD5 和 SHA-1 哈希值。由于任何对 APK 文件的修改，都会导致哈希值的变化。所以，通过哈希值的比对校验，可以快速的判断，一个 APK 文件是否已经被感染或者破坏。通常来说，分析者可以收集大量正常应用程序的哈希值以供比对，从而快速排查出可疑的 APK 文件。

（2）字节码

字节码是应用程序的可执行代码。对应的是 APK 包里面的 classes.dex 文件。一

般来说，JAVA 应用程序会有很多个类构成，经过编译后会转换为 JAVA 字节码。而 Android 应用程序要运行于 Dalvik 虚拟机上，Dalvik 虚拟机和 JVM 是完全不一样的，Dalvik 是基于寄存器的而 JVM 是基于堆栈。所以代码会再进一步的转化为 Dalvik 虚拟机所支持的 Dalvik 字节码，最后打包压缩放入 classes.dex 文件中。

（3）资源

资源指的是应用程序的非可执行部分，它包含了程序运行所需的额外数据。大部分应用程序的资源主要是用户界面组件，比如图片、菜单，布局，小部件等。而通常来说，恶意代码都是在后台运行，不需要用户界面。所以用户界面资源，通常不是逆向分析的重点。

但是，APK 包中存在一个特殊的资源文件，AndroidManifest.xml。通常这个文件会经过编码并以二进制的形式存储在 APK 包中，需要通过解码来提取。这个文件在恶意软件逆向分析的过程中非常关键，因为他包含了两大重要信息：权限和组件的声明。

(a) 权限声明

为了访问一些受保护的 API，应用程序需要在 AndroidManifest.xml 文件中，声明相应的权限，比如读取短信、读取通讯录的权限等。权限声明是每个程序独一无二的重要分析特征之一。在分析程序的恶意行为时，分析它的权限声明是非常重要的[11]。举例来说，一个正常的计算器程序，如果声明了一个 READ_CONTACT（读取通讯录）的权限的话，那么这个程序就变得非常可疑，因为为了正常的计算器功能，是不需要这样特殊的权限的。

(b) 组件声明

Android 应用程序是由多个组件结合而成的。应用程序组件共有四类：activities，services，broadcast receivers 和 content providers。一般来说，带有恶意行为的软件由于需要在后台运行，都会具有至少一个 service 组件。而对于会随着系统启动而自启动的恶意软件，通常会带有一个 broadcast receiver 组件，用来接收系统引导的 Intent。分析者通过检查程序包含那些组件，也可以对程序潜在的行为进行推测。

3.2 提取和鉴别可疑软件

通常情况下，一个智能手机设备上，都安装有大量的应用程序，包括系统预装以及用户安装的程序在内，可能有上百个之多。而恶意软件只是其中的一小部分，混杂在大部分正常的软件之中。要进行软件安全分析，首先就需要把有恶意行为嫌疑的软

件和正常的软件，区分开来。也就是说，需要从大量的应用程序中，找出可疑的应用，来进行更进一步的分析。为了降低后续分析的工作量，这个鉴别的过程应该尽量的精确，并降低误检率。

关于恶意软件的自动化检测方法，已经有一些相关的研究工作，也有一些辅助工具[12]。但是要实现自动化的检测仍存在困难。一方面，自动化工具通常需要大量恶意软件样本来建立数据库，而恶意软件更新频繁，自动化的检测工具始终是滞后于它的更新脚步的。更特殊的情况是，一些的恶意软件并非面向普通用户而是专门为了入侵特定的设备来设计的，此类恶意软件很难提前收集到它的信息。另一方面，逆向分析者在手动鉴别的过程中，还能同时收集许多有价值的信息，对于后续的深入分析工作，是有帮助的。再者，即使有自动检测工具，也只是作为辅助，分析者最终还是需要手动的鉴定来确认检测结果的正确性。

如图 3-1 所示，我们鉴别恶意软件主要通过消息摘要，权限请求，和组件，三个方面。

图 3-1　软件的可疑特征

（1）哈希值

通常从 Android 应用市场上下载的应用，都会带有一个哈希值，这个哈希值是应用市场提供的，用来保证用户下载到的是未经过第三方篡改的应用程序。理论上，只需要收集所有正规 Android 应用市场上的软件的哈希值，就可以构建一个正常的哈希值数据库。那么分析者在排查恶意软件的时候，只需要把软件的哈希值到正常哈希值数据库中查询，如果能查到，则可以直接排除相应软件的嫌疑。在实际的分析中，如果没有事先维护一个这样的数据库，也可以通过搜索引擎近似的完成这项工作，通过 *google* 搜索引擎搜索软件的哈希值，如果能搜索到正规应用市场的链接，那么就排除了这个软件的嫌疑。

通过消息摘要的检查，通常能够排除大部分正常的软件，但是还远远不能断定剩

下的软件都有恶意行为，还需要更进一步的深入分析。

（2）权限请求

权限请求也是 Android 应用程序的一个重要分析特征。根据 Android 系统的设计原则，Android 应用程序只需要在安装的时候向用户请求一次，就可以永久的拥有对应的权限。而安装过程中向用户请求权限的界面繁琐而不友好，许多用户，特别是中国的用户，为了能够正常的安装软件，常常并不会因为权限原因而拒绝安装。
通常恶意软件为了进行恶意行为的需要，会请求一些比较敏感的权限，如对通讯录、短信、电话的访问控制权限等，因此，可疑的权限请求，常常是发现 Android 恶意软件的首要线索[13][14]。通过检查 APK 包中的 AndroidManifest.xml 文件，分析者可以得到应用程序的所有权限请求信息，从而筛选出可疑的请求。

（3）组件

从抽象层面上来讲，Android 应用程序是由多个组件构成的。同时，组件的结构信息，可以用来判断程序的行为特征。3.1 节中提到，service 组件和 receiver 组件都是恶意软件非常敏感的构成部分。所以通过检查组件，以及组件声明中接收的 intent，分析者可以推测软件的大体行为，因此也是把恶意软件区分鉴别出来的重要手段之一。

3.3 软件代码反保护

在完成恶意软件的提取鉴别之后，理论上来说，借助反编译工具，反编译恶意软件并分析它的代码，分析者可以还原出恶意软件的所有行为。但是，实际情况下恶意软件制作者往往会采用各种自我保护手段来加大分析的难度。

常见的做法是，对软件代码进行混淆处理以及植入一些保护代码。植入保护代码的做法在 PC 平台的恶意软件制作中，是非常常见的，比如，很多恶意软件在执行的过程中，会先执行一段代码，检测是否自身是在被调试的环境中执行，如果是则直接关闭程序，从而阻止分析者对恶意软件进行动态调试。Android 恶意软件也常常使用类似的自我保护手段以对抗逆向分析。

本节将介绍三种常见的代码保护手段，并提出对抗这些手段的方法。

3.3.1 代码保护手段

常见的 Android 代码保护手段有以下几类：

（1）混淆

由于是基于 JAVA 语言编写，Android 恶意软件的混淆技术，和通常的 JAVA 混淆技术[15]是类似的。重命名是一种很常见的混淆技术，被混淆的程序代码中，所有的包名、类名、变量名、方法名，全部被无意义的，理解困难的字符串替代，例如 a, c, b.a(), f, b.e, g.a, c.b()。对于分析者来说，经过混淆后的代码，无法根据包、类以及方法的名称来进行模块划分和含义理解，分析难度显著提升。

（2）环境检测

某些移动平台的恶意软件，是专门设计成攻击特定的设备的。它们会在启动的时候，读取一些系统的属性值（android.os.BUILD），从而保证它只会在特定类型的设备上进行恶意行为。通过读取和判断设备的用户标识符（IMSI），恶意软件可以做到只在某一台或者某几台设备上运行恶意代码。而分析者如果不具有该类型的设备，或是在模拟器中运行该软件，恶意软件则佯装正常的应用程序。对于这种类型的恶意软件，分析者无法直接通过黑盒分析的手段，找出它的恶意行为。

（3）字符串加密

通常的逆向分析过程中，代码中的字符串常量，是一类非常重要的特征信息，通过跟踪特殊的字符串，可以快速的在程序代码中定位出要重点分析的关键部分。例如，如果发现代码某处使用了类似"GET /..."的字符串，则可以推测这部分代码是在和服务器进行 HTTP 通讯，进一步的，还可以寻找符合 IP 地址结构的字符串，从而推测出与它通讯的服务器的地址。

正因为字符串常量中常常包含这些关键信息，所以恶意软件也会使用字符串加密的手段，来保护明文的字符串常量。通过使用 DES 或者 AES 这样的对称加密算法和一个内置的固定密钥，可以对所有的字符串常量进行加密保护。这样的加密，给代码静态分析的过程，带来了很多的困难。当然，由于字符串在运行时终究会被解密出来使用，那么在动态分析时，依然是可以得到明文字符串的。

3.3.2 反保护方法

针对这些代码保护手段，我们提出了一些反保护的方法。

（1）反编译和反混淆

在对 Android 应用程序进行逆向分析时，借助 JAVA 反编译工具，能够得到一份 JAVA 源代码。基于源代码的分析，会比基于字节码或是汇编码容易的多。然而，现有的反编译工具，都存在种种缺陷，不能完全正确的反编译 Android 应用程序。通常

反编译出来的代码都带有大量的错误和内容缺失。而且实际分析中，恶意软件常常经过混淆处理，从而导致反编译更加困难，反编译出来的代码都包含大量的错误和缺失。另一方面，通过反汇编工具，可以得到软件的 smali 汇编码。由于汇编码与字节码是一一对应的，所以得到的汇编码是准确无误的。然而，汇编码也更加难以理解和分析。因此，分析者常常两者结合使用，取长补短。

实际分析中，反编译反混淆主要包括这几个步骤：首先，分析者使用 *apktool*[16]，解压 APK 包，把字节码（.dex 文件）提取出来。然后，通过配合使用 *dex2jar*[17]和 *jd-gui*[18] 这两个工具，可以把字节码反编译成 JAVA 源码。通常反编译出来的源码都包含大量的错误，需要通过大量的工作来进行纠正。

代码的纠正一般会包括以下这些过程：

- 移除空类
- 重命名
- 修正反编译错误
- 修正控制流错误
- 名字冲突修正
- 缺失代码补充

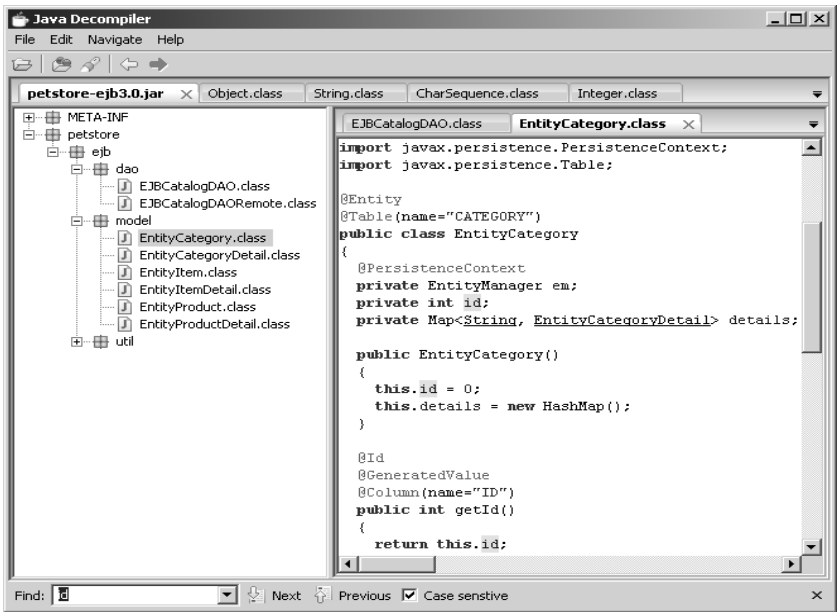


图 3-2　使用 jd-gui 进行反编译分析

（2）字符串解密

字符串是恶意软件中非常重要的信息（例如远程服务器的 URL 地址）。在恶意软件中，这样的字符串通常是加密形式存储的，因此需要通过一个解密的步骤把明文字

符串提取出来。解密的过程包含以下几个步骤：识别加密算法，提取内置密钥，和解密字符串。由于恶意软件的加密算法通常使用系统提供的密码学算法 API，通过跟踪这些 API 的调用，分析者可以找出它使用的密码算法，提取它使用的密钥。

（3）保护代码去除

上一节中提到，为了对抗动态分析，一些恶意软件会在启动的时候检测设备标识符等系统信息。为了能够正确的进行动态调试分析，分析者可以找到这些相关的代码，并手动修改代码，去掉这些校验逻辑。

3.4 恶意代码的分析理解

在提取了恶意软件，去除了恶意软件的代码保护手段之后，为了最终还原出恶意软件的行为，分析者还需要对恶意软件的代码进行大量的分析理解工作。这是整个逆向分析过程最关键的步骤，由于即使是简单的软件代码往往也有数万行之多，这个分析的过程也非常耗时的。分析代码并没有一个标准的流程，而是相当程度的依赖于分析者自身的逆向分析经验，以及对 Android 系统，API，程序结构等的熟悉程度。

分析理解恶意软件代码的过程，如图在迷雾中寻找线索，发现一些关键的部分，既可以顺藤摸瓜，解开一部分迷雾，不断的进行这个探索的过程，花费足够的时间即可理清整个程序的结构。

一些 Android 恶意软件窃取用户的隐私信息，并这些信息发送到远端服务器上。具体来说，恶意程序会伪装成一个正常的应用，同时，在后台启动一个服务，专门用来收集隐私信息。后台服务通过自己的秘密通讯协议和远程服务器进行通讯，向服务器发送隐私数据，同时接受远程服务器的发来的指令，并根据指令来控制手机的行为。为了实现这样的功能，可以在代码里找到这样的关键特征。

如图 3-3 所示，我们总结了一些恶意软件常有的代码特征模块，可以帮助分析者更好更快的进行分析理解恶意软件。

（1）后台服务 Core 循环

恶意软件需要有一个在后台持续运行的服务，从代码结构上看，需要一个循环结构，控制程序总的运行流程，找到了这个循环，就找到了恶意代码的总入口。

图 3-3　恶意软件常有的代码模块

（2）通讯协议

恶意软件通常使用自定义的特殊协议和远程服务器进行通讯。那么必然能够从它的代码中找到相关的协议实现代码。这些代码模块需要把发送的数据打包并加密，把从服务器接收的数据解密和解包。在 Android 系统上，为了进行网络通讯，它还需要申请网络相关的权限，如 android.permission.INTERNET 和 android.permission.ACCESS_NETWORK_STATE。

（3）密码学算法

为了实现加密解密相关步骤，恶意软件需要调用系统的密码算法库，以实现如对本机特征码计算消息摘要，解码内置字符串常量，和服务器进行密钥交换，和服务器进行加密通讯等功能。因此，密码算法 API 调用，密码学计算等相关代码，也是恶意程序代码的重要模块。

（4）隐私数据访问

窃取隐私数据是恶意软件的核心功能之一，为了实现这个功能，它需要访问系统中的关键数据。首先，它会请求如 android.permission.READ_SMS，android.permission.READCONTACTS 之类的数据访问权限。然后，通过特定的 API 和 content provider，来从数据库取得隐私数据。

3.5 恶意行为的提取重建

软件安全分析的一个重要应用是服务于安全取证工作，在取证过程中，最终目的是通过恶意程序样本以及网络通讯记录等数据，还原重建出当时的恶意行为。通常，分析者并不能提前预知恶意行为的发生并做记录，往往是在恶意行为发生之后，才得到恶意软件样本等数据。Android 恶意软件使用 JAVA 语言编写，恶意软件的所有字节码，就包含了它自身的所有功能行为。所以，分析者需要通过代码反推逻辑，同时

结合恶意行为发生时留下的通讯记录等数据，来重建恶意行为发生时的情况。

经过重建后的恶意行为，包含这几个部分：恶意代码的工作流，恶意软件取得的敏感数据内容，加密算法，恶意软件的通讯协议。为了实现恶意行为的重建，不仅要理解恶意代码的各个功能模块，还需要理清楚各个功能模块的逻辑顺序。Android 提供了 logcat 机制，用来捕获记录如 API 调用、服务启动停止等高层操作。在条件允许的情况下，分析者应该重新运行恶意软件，记录这些关键的操作，并绘制事件流程图。

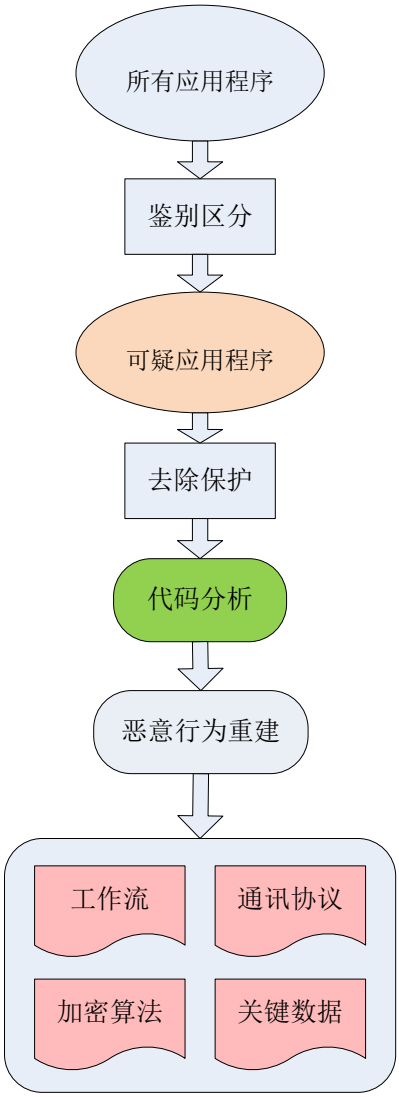


图 3-4　软件分析和恶意行为提取完整工作流程

3.6 案例分析

在本节中，我们将以一个实际的恶意软件分析案例，来展现整个软件分析和恶意行为提取的具体过程。这个恶意软件样本，来自于国际安全研究机构 *honeynet* 组织的取证分析挑战赛[19]。具体竞赛文档请参见[10]。

3.6.1 案例背景

该案例基于一个真实的 Android 智能手机被恶意软件劫持的事件，案例提供的原始数据文件包括：受到入侵后，从该手机中提取出来的文件系统镜像（受损的），以及相关的网络通讯记录片段（PCAP 文件）。分析者需要从中提取可疑信息，寻找并分析恶意软件，重现恶意行为。要求分析者具备这些技术：受损文件系统的恢复，恶意软件逆向分析，PCAP 网络数据包分析等。

图 3-5 展示了该案例中的原始数据文件，其中 data.bin 是从被入侵的手机中提取的内存储器分区镜像，traffic.pcap 文件是在恶意软件通讯过程中，捕获的网络数据包。

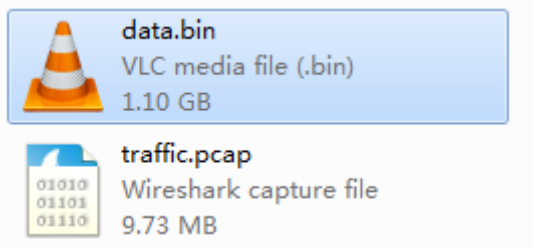


图 3-5　分析案例的原始数据文件

通过快速的检查这两个文件的内容，可以发现 data.bin 是一个损坏的镜像文件，无法直接加载，但是可以在其中搜索到一些有意义的字符串内容（如图 3-6），可以初步猜测可以从这个镜像文件中还原出一些 apk 文件出来。而在 traffic.pcap 文件中，则发现了一些可疑的加密通讯数据（如图 3-7）。

```
020D6000  81 DC 00 00 0C 00 01 02  2E 00 00 00 02 00 00 00  .Ü..............
020D6010  24 00 02 02 2E 2E 00 00  96 DC 00 00 18 00 0D 01  $.......–Ü......
020D6020  76 6D 64 6C 33 33 30 37  31 2E 74 6D 70 00 00 00  vmdl33071.tmp...
020D6030  8C DC 00 00 48 00 19 01  63 6F 6D 2E 61 6E 64 72  ŒÜ..H...com.andr
020D6040  6F 69 64 2E 76 65 6E 64  69 6E 67 2D 31 2E 61 70  oid.vending-1.ap
020D6050  6B 70 6B 00 8D DC 00 00  24 00 1B 01 63 6F 6D 2E  kpk..Ü..$...com.
020D6060  66 63 39 2E 63 75 72 72  65 6E 63 79 67 75 69 64  fc9.currencyguid
020D6070  65 2D 31 2E 61 70 6B 00  8E DC 00 00 20 00 16 01  e-1.apk.ŽÜ.. ...
020D6080  63 6F 6D 2E 67 6F 6F 67  6C 65 2E 65 61 72 74 68  com.google.earth
020D6090  2D 31 2E 61 70 6B 32 2E  8F DC 00 00 20 00 17 01  -1.apk2..Ü.. ...
020D60A0  63 6F 6D 2E 6F 70 65 72  61 2E 62 72 6F 77 73 65  com.opera.browse
020D60B0  72 2D 31 2E 61 70 6B 00  90 DC 00 00 2C 00 22 01  r-1.apk..Ü..,."
020D60C0  63 6F 6D 2E 67 6F 6F 67  6C 65 2E 61 6E 64 72 6F  com.google.andro
020D60D0  69 64 2E 73 74 61 72 64  72 6F 69 64 2D 31 2E 61  id.stardroid-1.a
020D60E0  70 6B 00 00 91 DC 00 00  28 00 20 01 63 6F 6D 2E  pk..‘Ü..(. .com.
```

图 3-6　数据文件 data.bin 片段

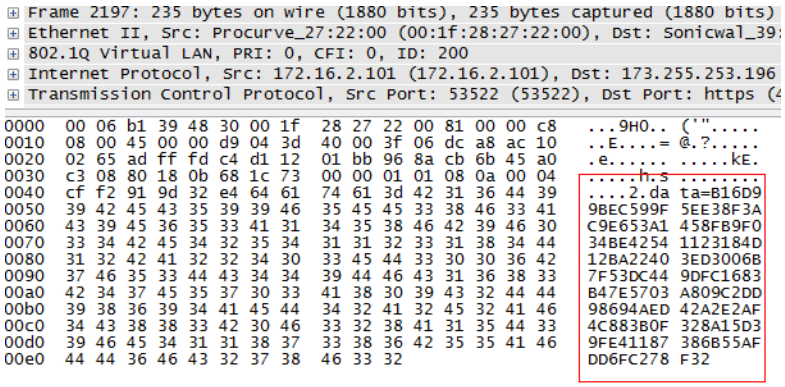


图 3-6　通讯数据 traffic.pcap 片段

3.6.2 鉴别可疑软件

分析过程的第一步要做的是，提取并鉴别出可疑的应用程序。

从 data.bin 镜像文件中，通过文件系统修复还原，我们提取出了其中包含的文件，文件目录如图 3-7 所示。

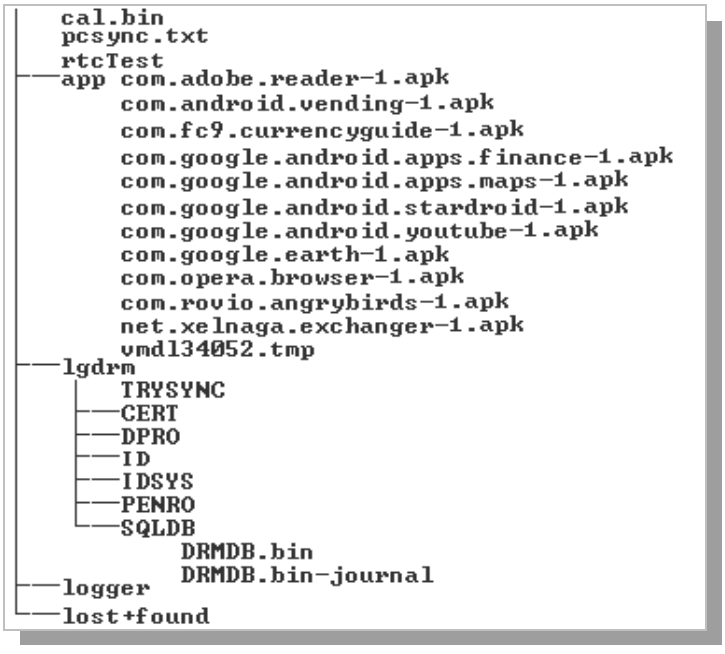


图 3-7　从 data.bin 中恢复出来的文件目录

检查这些恢复出来的文件，总共可以找到 10 有效的应用 APK 文件，和两个应用临时文件。通过计算和比对这些文件的 MD5 和 SHA-1 消息摘要（图 3-8），发现其中

7 个应用的哈希值，可以在网上正规的应用市场中找到。这 7 个应用是：com.adobe.reader-1.apk,com.google.android.stardroid-1.apk,com.rovio.angrybirds-1.apk,com.android.vending-1.apk,com.google.android.apps.maps-1.apk,com.google.earth-1.apk,com.opera.browser-1.apk。

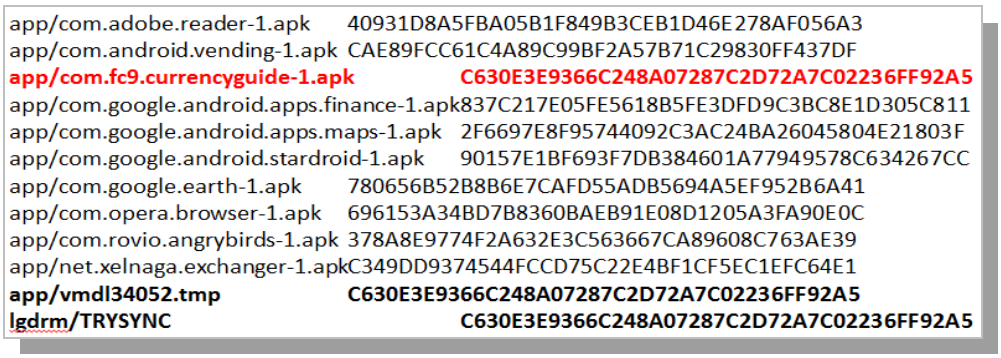


图 3-8　所有 APK 文件的 SHA1 值

剩下的三个应用是无法通过哈希值来排除它的嫌疑的。因此，我们做了进一步的分析，通过解压 APK 包，检查它的权限申请和组件声明信息，我们发现，其中的 com.google.android.apps.finance-1.apk 和 net.xelnaga.exchanger-1.apk 两个应用，并无任何嫌疑特征。最后一个应用文件 app/com.fc9.currencyguide-1.apk，我们经过检查发现，该 APK 文件和另外两个临时文件 app/vmdl34052.tmp 和 lgdrm/TRYSYNC 完全相同，猜测这可能是因为，导出文件系统镜像时，该应用还在执行，尚有临时文件存储在手机内存中。

进一步的分析发现，com.fc9.currencyguide-1.apk 这个应用，请求了以下的应用程序权限：

- android.permission.INTERNET
- android.permission.READ_PHONE_STATE
- android.permission.ACCESS_WIFI_STATE
- android.permission.WAKE_LOCK
- android.permission.ACCESS_NETWORK_STATE
- android.permission.RECEIVE_BOOT_COMPLETED
- android.permission.CAMERA
- android.permission.VIBRATE
- android.permission.ACCESS_COARSE_LOCATION
- android.permission.ACCESS_FINE_LOCATION
- android.permission.CALL_PHONE
- android.permission.SEND_SMS

- android.permission.READ_CONTACTS
- android.permission.RECEIVE_SMS
- android.permission.READ_SMS

然而实际上，观察这个程序的执行时的界面（图 3-9），可以发现，它表面上只是一个汇率计算器，并不应该需要这么多的权限。所以它请求了这么多数量的敏感权限，可以推测这个应用具有重大的嫌疑。

在确定了 com.fc9.currencyguide-1.apk 这个应用为主要可疑软件之后，我们接着对它进行专门的深度分析。

图 3-9　com.fc9.currencyguide-1.apk 的运行界面

3.6.3 代码反保护和重建

这个恶意软件使用了我们 3.3 节中提到的所有 3 种反分析手段。由于反编译本身的困难和反编译技术的不成熟，再加上代码混淆的干扰，直接从反编译器输出的代码有着大量的缺失和错误。为了纠正这些错误，我们结合了汇编代码进行分析，重建了正确的软件源代码。

代码重建的过程，包括移除空类，重命名，反编译错误纠正，字符串解密等步骤。

（1）移除空类

在分析的过程中，我们找到了一些空类和空函数。这种情况有两种可能，第一种是，混淆器为了使得代码更加混乱，刻意添加的一些空类。第二种可能是，由于反编译器本身的功能局限，一些特殊的 JAVA 类（例如枚举类），往往无法被正确的反编译出，反编译器只输出一个代码空壳，看起来就像是个空类，但实际上是代码在反编

译时缺失了。为了区分这两种情况，我们逐一检查了每个类的汇编代码，排除掉符合前者的混淆空类，并且补完后者的缺失内容。图 3-10 展示了一个典型的空类和它的汇编代码。

```
1   package com.fc9.currencyguide.daemon.b;
2
3   public final class a
4   {
5   }
6
7   /* Location:           D:\puzzle\decompiled_apps\com.fc9.
    currencyguide-1_dex2jar.jar
8    * Qualified Name:     com.fc9.currencyguide.daemon.b.a
9    * JD-Core Version:    0.6.0
10  */
```

a) 一个空类的源代码

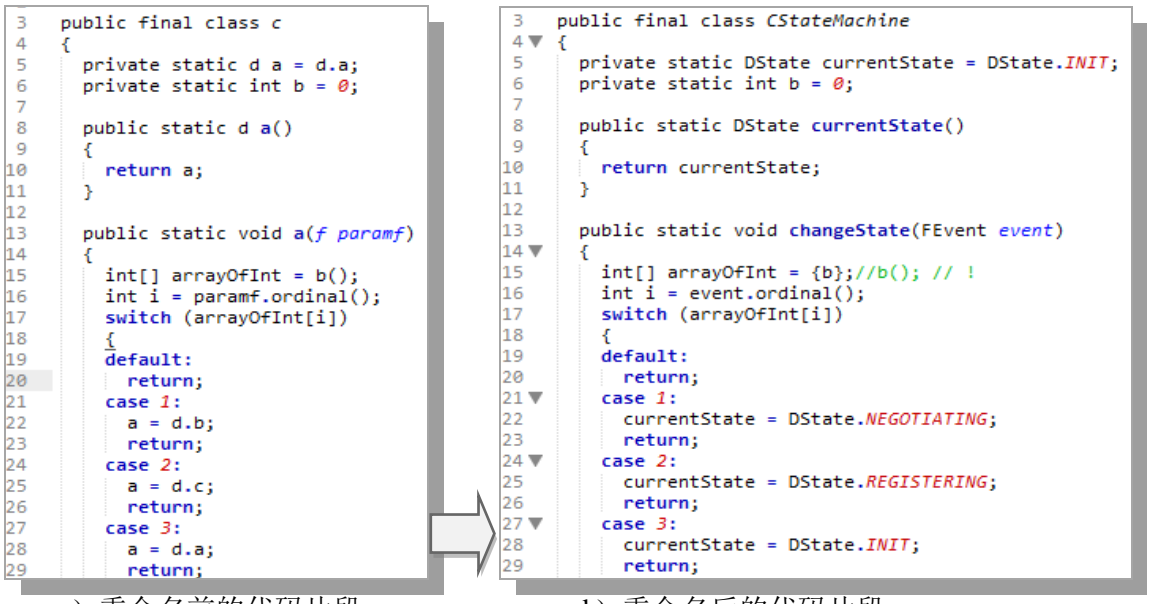
```
1   .class public final Lcom/fc9/currencyguide/daemon/b/a;
2   .super Ljava/lang/Object;
3
4
5   # direct methods
6   .method public constructor <init>()V
7       .locals 0
8
9       invoke-direct {p0}, Ljava/lang/Object;-><init>()V
10
11      return-void
12  .end method
13
```

b) 一个空类的汇编代码

图 3-10　混淆代码中的空类源码和汇编码

（2）代码重命名

由于该恶意软件的源代码经过混淆处理，所有的包名，类名，方法名，变量名，都被重命名成了 a, b, c, d 等无意义的字符，难以对代码进行模块划分和分析理解，并且重命名后的代码也存在着大量的名字冲突错误。因此需要对代码进行一层一层的分析重建。重建所有源代码的过程，看起来就像是个深度优先搜索的过程一样。首先我们检查代码中位于最低层次的原子功能函数，随着原子函数的功能逐步摸清楚，给原子函数，原子类，重命名，添加注释，然后再来分析理解它的上一层的代码逻辑，也就是再分析调用这个原子函数的上层函数。图 3-11、3-12、3-13 展示了一些重命名前后的代码和文件名对比。图 3-11 中的代码，是恶意软件代码的通讯模块中，负责状态转换的类代码，图 3-12 是通讯模块的几个主要的代码文件，图 3-13 是恶意软件代码中几个主要的包。



```
public final class c
{
  private static d a = d.a;
  private static int b = 0;

  public static d a()
  {
    return a;
  }

  public static void a(f paramf)
  {
    int[] arrayOfInt = b();
    int i = paramf.ordinal();
    switch (arrayOfInt[i])
    {
    default:
      return;
    case 1:
      a = d.b;
      return;
    case 2:
      a = d.c;
      return;
    case 3:
      a = d.a;
      return;
```

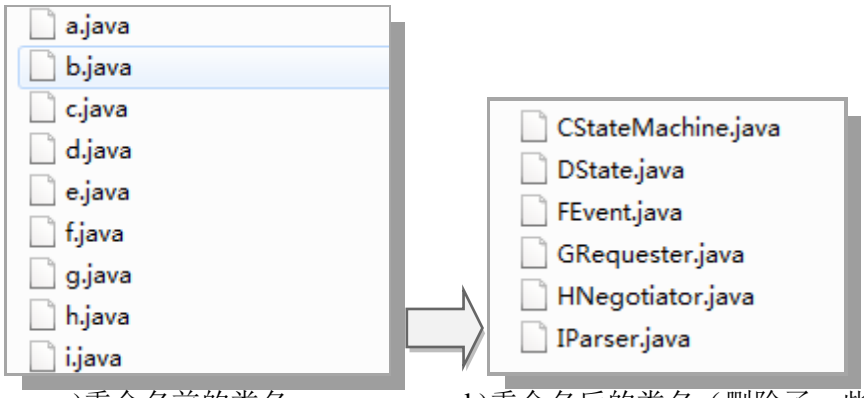
```
public final class CStateMachine
{
  private static DState currentState = DState.INIT;
  private static int b = 0;

  public static DState currentState()
  {
    return currentState;
  }

  public static void changeState(FEvent event)
  {
    int[] arrayOfInt = {b};//b(); // !
    int i = event.ordinal();
    switch (arrayOfInt[i])
    {
    default:
      return;
    case 1:
      currentState = DState.NEGOTIATING;
      return;
    case 2:
      currentState = DState.REGISTERING;
      return;
    case 3:
      currentState = DState.INIT;
      return;
```

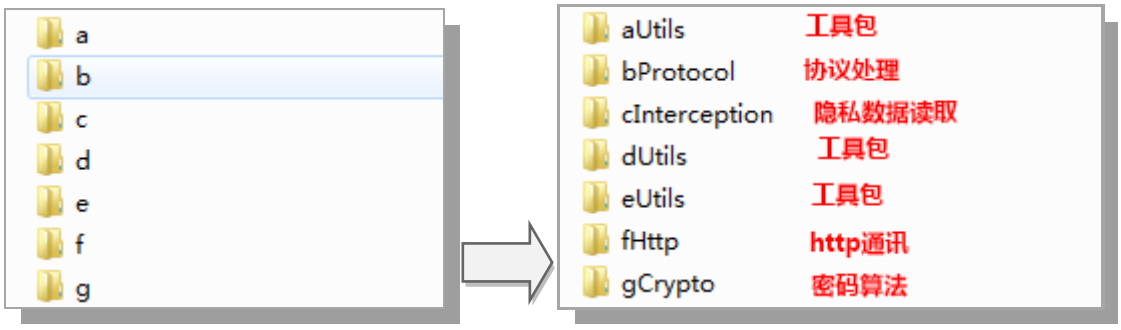
a) 重命名前的代码片段　　b) 重命名后的代码片段

图 3-11　重命名前后的源代码片段



a)重命名前的类名　　b)重命名后的类名（删除了一些空类）

图 3-12　重命名前后的类文件



a)重命名前的包　　b)重命名后的包

图 3-13　重命名前后的包

（3）反编译错误修正

除了存在空类，名称混乱之外，我们发现反编译出来的代码，还包含有很多逻辑错误。一方面是因为反编译器本身的缺陷导致，另一方面由于代码经过混淆处理，也使得反编译出来的逻辑含有许多错误。主要的错误包括以下几个方面

- 控制流错误

图 3-14 展示了代码中的一个典型的控制流错误，这段代码是恶意软件启动时检查设备 ID 的代码，由于原始代码较长，这里采用的是伪代码来进行说明。图 3-14a 中，很显然第六行将直接返回，而后面的判断语句将完全没有作用。经过对汇编代码的查验和结合上下文对代码进行理解分析，我们还原了正确的逻辑（图 3-14b）。

```
Boolean a()
    Boolean ret;
    if (sha1(device) == "xxx")
        ret = true;
    while (true) {
        return ret;
        if (sha1(model) == "xxxx") {
        continue;
        if (sha1(product) == "xxxx") {
        continue;
        ...
    }
}
```

a) 错误的控制流（伪代码）

```
Boolean a()
    Boolean ret;
    if (sha1(device) == "xxx")
        ret = true;
    if (sha1(model) == "xxxx") {
        ret = true;
    if (sha1(product) == "xxxx") {
        ret = true;
    ...
    return ret.
}
```

b) 经过修正的控制流（伪代码）

图 3-14　代码控制流错误修正

- 名字冲突

由于代码经过了重命名混淆处理，所有的类和方法名都变成了 a,b,c,d,e…，所以，在实际的代码逻辑中，就产生了大量的重名错误。例如，类 com.fc9.currencyguide.daemon.CCom_Service 中，导入了两个名字叫 a 的类，g.a 和 b.a，并且 CCom_Service 中还有一个方法的名字也是"a"，导致反编译器处理这个类的代码时出现了许多的混乱和错误，我们分析时，根据环境上下文，仔细检查每一个像"a"这样重复的名字的实际含义，纠正了名字冲突带来的逻辑错误。

- 信息缺失

由于反编译器本身的局限性，一些类在反编译之后的代码里面，会缺失了很多的信息。例如，com.fc9.currencyguide.daemon.b.f 这个类，它是一个常量类，用以定义通讯过程中每一个状态常量。反编译器对这样的常量类，输出的是完全错误的不包含任何实质信息的代码。这时候，就需要结合 smali 汇编码，手动的补充这个类里面的

缺失信息（图 3-15）。

```
public enum f {
  static   {
    f[] arrayOff = new f[11];
    f localf1 = a;
    arrayOff[0] = localf1;
    f localf2 = b;
    arrayOff[1] = localf2;
    f localf3 = c;
    ....
    arrayOff[10] = localf11;
    l = arrayOff;
  }
}
```

```
public enum FEvent {
    INIT_OK("INIT_OK"),
    NEGOTIATE_OK("NEGOTIATE_OK"),
    NEGOTIATE_ERROR("NEGOTIATE_ERROR"),
    ...
    EXECUTE_OK("EXECUTE_OK"),
    EXECUTE_ERROR("EXECUTE_ERROR");
    ...
}
```

a) 一个常量类的错误反编译结果　　b) 经过补充信息后这个类的代码

图 3-15　一个常量类的信息缺失修正

（4）常量字符串解密

该恶意程序内部最重要的数据，是以 DES 加密的形式存储的，包括，服务器地址，协议中的指令名称等。我们从代码中，提取出了 DES 加密的密钥：0x63B252F6FAF4167F。然后编写了一个小工具，解密了程序中所有的加密字符串，包括远程服务器的地址：*http://faeacdeadbeefada.zonbi.org:443*。

图 3-16 展示了一段含有加密字符串的代码（来自 com.fc9.currencyguide.daemon.a 这个类，其中长串的数字部分就是加密后的字符串），和对应的字符串解密结果。

```
String[] arrayOfString1 = new String[2];
String[] arrayOfString2 = new String[2];
String str1 = com.fc9.currencyguide.daemon.gCrypto.ACryptoUtils.decryptStringFromBytes(
    new byte[] { 42, 167,
        73, 214, 71, 27, 136, 254, 50, 90, 90, 213,
        193, 87, 209, 28, 239, 154, 72, 51, 186,
        50, 238, 242, 32, 214, 41, 139, 73, 230, 181, 92 });
arrayOfString2[0] = str1;
String str2 = com.fc9.currencyguide.daemon.gCrypto.ACryptoUtils.decryptStringFromBytes(
    new byte[] { 87, 138,
        6, 47, 250, 190, 198, 254 });
arrayOfString2[1] = str2;
String[] arrayOfString3 = new String[2];
String str3 = com.fc9.currencyguide.daemon.gCrypto.ACryptoUtils.decryptStringFromBytes(
    new byte[] { 42, 167, 73,
        214, 71, 27, 136, 254, 50, 90, 90, 213, 193,
        87, 209, 28, 239, 154, 72, 51, 186, 50, 238,
        242, 32, 214, 41, 139, 73, 230, 181, 92 });
arrayOfString3[0] = str3;
String str4 = com.fc9.currencyguide.daemon.gCrypto.ACryptoUtils.decryptStringFromBytes(
    new byte[] { 212,
        62, 160, 45, 66, 134, 28, 236 });
arrayOfString3[1] = str4;
```

a) com.fc9.currencyguide.daemon.a 中含有加密字符串的代码片段

```
{42,167,73,214,71,27,136,254,50,90,90,213,193,87,209,28,
239,154,72,51,186,50,238,242,32,214,41,139,73,230,181,92
} = "faeacdeadbeefada.zonbi.org"

{87,138,6,47,250,190,198,254} = "443"

{212,62,160,45,66,134,28,236} = "80"
```

b) 字符串解密结果

图 3-16　含有加密字符串的代码片段和解密结果

3.6.4 代码分析

在 3.6.3 的代码重建过程中，我们已经对所有的恶意代码进行了还原重建，重建过程中不断修正代码中反编译错误的过程，也就对每一行代码的功能进行理解的过程。在此基础之上，我们进一步的对恶意软件的代码进行分析。

（1）代码入口

从 com.fc9.currencyguide-1.apk 里的 AndroidManifest.xml 文件中，我们发现了 6 个应用程序组件：

- *Main_Activity*
- *Converter_Service*
- *PrefMenu_Activity*
- *com.fc9.currencyguide.daemon.fc9*
- *com.fc9.currencyguide.daemon.CCcomService*
- *com.fc9.currencyguide.daemon.BootReceiver*

从每个组件的入口函数代码开始仔细分析，我们发现前三个组件：*Main_Activity*，*Converter_Service* 和 *PrefMenu_Activity* 执行的是普通的汇率转换功能，也就是这个恶意软件伪装成的功能，而另外三个组件，则是用来执行恶意功能的。

组件 *com.fc9.currencyguide.daemon.fc9* 是用来在程序启动时执行的 activity 组件；*com.fc9.currencyguide.daemon.BootReceiver* 是用来接收 *BOOT_COMPLETED* 这个 intent 的组件。这两个组件都会启动最终的 service 组件：*com.fc9.currencyguide.daemon.CCcomService*，这个 service 是所有恶意代码的总入口。通过这样的设计，最终可以达到这样的效果：无论是应用启动的时候，还是系统启动完成引导步骤时，都会自动的启动后台的 service 服务，执行恶意代码。这个 service

的源代码是一个无限循环结构，从而使得它可以常驻在后台运行。

（2）模块划分

经过分析后，我们得出了恶意代码的以下几个主要的模块：

- 服务 Core 循环

类 com.fc9.currencyguide.daemon.CComService，包含了恶意软件的核心 Core 循环，它内部是一个无限循环的结构，并通过一个状态机来总控所有的恶意行为。

- 通讯协议

这个模块对应的是包 com.fc9.currencyguide.daemon.b，它实现了恶意软件和服务器的通讯协议，处理发送数据时的加密打包，和接收数据时的解密解包功能。

- 隐私数据读取

这个模块对应包 com.fc9.currencyguide.daemon.c，它处理获取隐私数据，发送短信，等关键功能。这个模块里面涉及到许多 Android 系统的敏感 API 调用。

- 密码学算法

这个模块对应包 com.fc9.currencyguide.daemon.g，它包含了一些常用的密码学算法，为各个模块的功能提供支持，如生成设备 ID 的消息摘要，字符串解密，密钥交换，加密通讯等。

- HTTP 客户端

这个模块对应包 com.fc9.currencyguide.daemon.f，它主要是处理 HTTP 协议的基本功能，为建立在 HTTP 之上的通讯提供支持。

（3）核心状态机分析

Core 循环和它包含的状态机是恶意代码的中心，它控制着整个恶意软件的行为流程，我们着重的分析了它的状态机转换逻辑。

它的有限状态机总共包含 6 个状态：INIT，NEGOTIATING，REGISTERING，REQUESTING，PARSING 和 EXECUTING。状态机启动后，会首先从 INIT 状态开始，然后依次进入 NEGOTIATING 和 REGISTERING 状态，最后在 REQUESTING，PARSING 和 EXECUTING 三个状态间不断循环，每次循环间隔 15 秒。

- INIT 状态

该状态下，程序对一些基本的变量进行初始化，包括解密字符串用的密钥，和远程服务器的地址等。初始化完成后，进入 NEGOTIATING 状态。

- NEGOTIATING 状态

这个状态下，程序向服务器发起密钥协商，完成协商后，就建立起了双方的一个秘密信道，之后所有的通讯过程，都将是加密的。完成协商后，进入 REGISTERING

状态。

- REGISTERING 状态

该状态下，程序将设备的 ID 提交给服务器，进行注册。注册完成后，进入 REQUESTING 状态。

- REQUESTING 状态

这个状态下，程序将设备信息发送给服务器，向服务器请求指令。如果顺利请求则进入 PARSING 状态，如果请求超时，则重试三次后，进入 INIT 状态。

- PARSING 状态

这个状态下，程序解码分析服务器发来的指令，分析成功则进入 EXCUTING 状态，分析失败则重新转入 REQUESTING 状态重新请求。

- EXECUTING 状态

这个状态下，程序将执行服务器的指令，如读取通讯录，发送短信，进入短信监视模式等。

执行过程完成后，状态机将等待 15 秒，然后再次进入 REQUESTING 状态。实际上，REQUESTING，PARSING 和 EXCUTING，是一个完整的请求指令，分析指令，执行指令的循环。恶意软件绝大部分运行时间都是处在这个循环中。

3.6.5 恶意行为提取结果

我们对代码分析的结果进行了整理，还原出了这个恶意软件的完整行为。

这个恶意软件表面上伪装成一个正常的汇率转换软件，于此同时，在后台运行了一个服务来读取用户的私人数据（短信，联系人，ID 等）。这个后台服务会和一个远程服务器建立连接，并且等待服务器发来的指令，然后通过自定义的加密通讯协议（基于 Diffie-Hellman 密钥交换和 DES 加密），把取得的数据加密发送给服务器。同时恶意软件还带有短信监视功能，可以在手机收到短信时立即通报给服务器。

图 3-17 展示了恶意软件和服务器通讯的完整流程。整个流程可以划分为：密钥交换，隐私数据加密发送，服务器指令接受执行，短信监视四个部分。

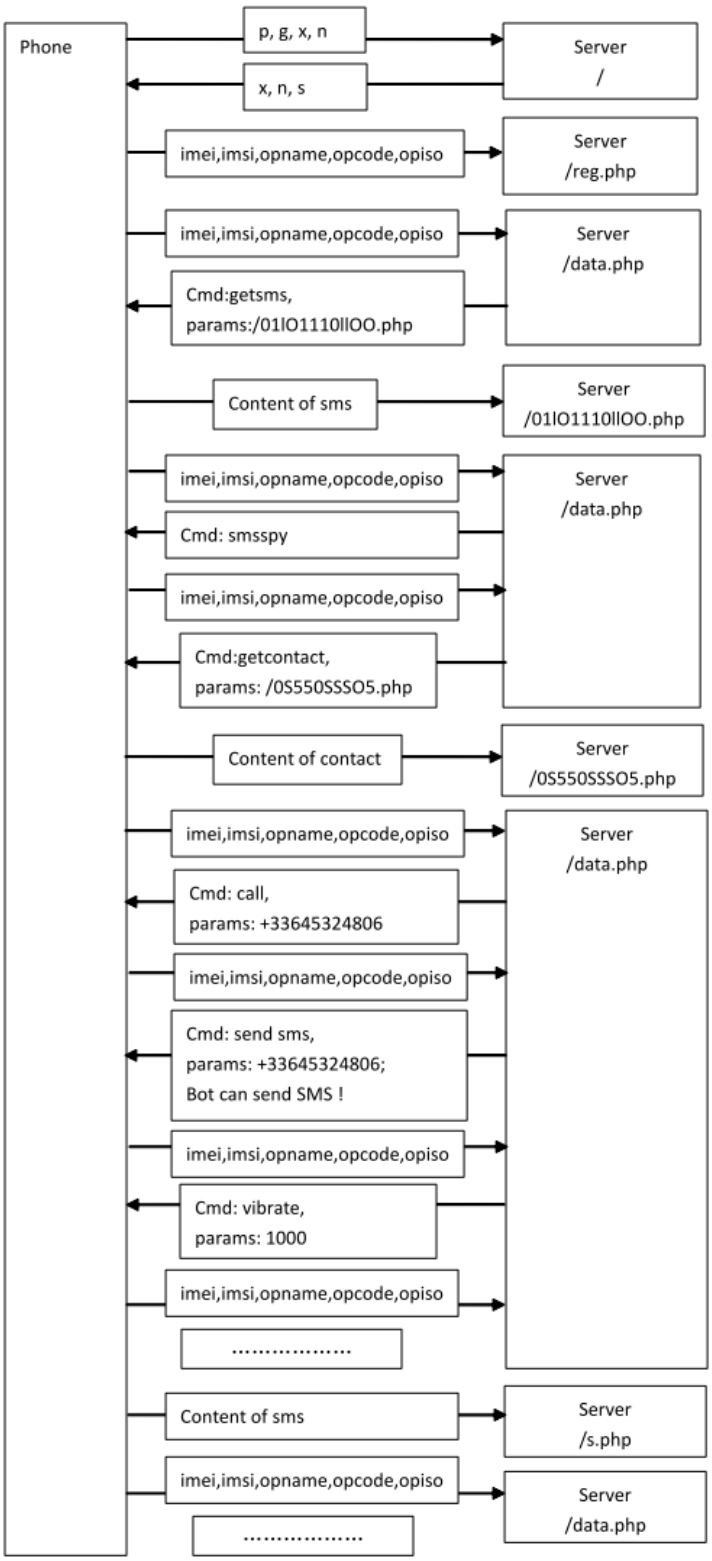


图 3-17　恶意软件的完整通讯流程

（1）密钥交换

恶意软件和服务器进行通讯的第一个步骤，是通过一个自定义的 Diffie-Hellman 密钥交换过程，建立一个共享密钥。在协商过程中，使用了典型的 Diffie-Hellman 密钥交换算法，然后在加密通讯过程中，使用了 DES 加密算法。根据代码分析以及抓取的数据包，我们推算出了，使用的 DES 密钥为 0xc4c9973a45c7007d。推算过程如下：

恶意软件扮演的是 Diffie-Hellman 算法中的 Alice 的角色，它最终通过公式 s=B^a mod p 计算出密钥，我们通过分析还原了这个过程：在图 3-17 中服务器发来的第二个数据包中的 x 就是 B，我们在 PCAP 文件里面找到了它的值=9915D4E8B2F342BEFC3E70C352D78F49；p 是由恶意软件在通讯的过程中随机生成并发送给服务器的，我们在 PCAP 文件中也找到了 p=00EABBE22F3A27C63780C932C76B351199；a 是一个固定的常量，在恶意软件代码里面找到 a=640963485269741EF69AE45D69F23AA9。

最后，我们计算出：s=B^a mod p=17355c6874cba653c4c9973a45c7007d。恶意软件截取后 8 个字节用来作为 DES 密钥。

（2）隐私数据加密发送

恶意软件和服务器进行加密通讯的过程中，不断的将各种隐私数据发送到服务器。通过解密网络数据包，我们还原了以下信息。

- 设备信息。

包括设备 ID（IMEI），用户识别码（IMSI），网络提供商名称，网络 ISO 国家代码，参见图 3-18a。

- 联系人数据。

包括联系人的 ID 和名字，参见图 3-18b。

- 短信数据

包括每条短信的内容，地址，部分短信中含有关键的密码数据，参见图 3-18c（内容是法语，图中方框标出的是用户在短信中泄露的密码）。

```
imei=356772040481677&imsi=208013002954000&opname=Orange
F&opcode=20801&opiso=fr
```

a) 设备信息数据

```
Niobe=5553&Agent
Smith=5551&Merovingian=5559&Trinity=5558&Seraph=5
557&Neo=5555&Morpheus=5554&
```

b) 联系人数据

21080=Votre mot de passe est strictement confidentiel : conservez-le précieusement et ne le communiquez pas à un tiers. Votre mot de passe est 3B6AT4&+33666186296=Le 0645324806 remporte 1 chèque en euro!Composez le 0899650923 pr le retirer immédiatement. Jeu sous Controle d huissier (1E35/ap+0E34/mn)&+33644066241=VocalMessenger: Vous avez recu 1 nouveau message vocal a 11:59. Pour le consulter, composez le 0899230625
Code confidentiel: 1701
(1e35+0e34-noSmsEnvStop)&20904=Mobicarte - Compte Principal. Attention, il vous reste moins d'une journée pour utiliser votre crédit de 3.93 EUR.&20904=Mobicarte - Compte Principal. Attention, il ne vous reste plus qu'une journée pour utiliser votre crédit de 4.05 EUR.&20904=Mobicarte - Compte Principal. Attention, il vous reste moins d'une semaine pour utiliser votre crédit de 4.17 EUR.&20904=Mobicarte - Compte Principal. Attention, il ne vous reste plus qu'une semaine pour utiliser votre crédit de 4.29 EUR.&20904=mobicarte: votre ligne est identifiée, vous pouvez maintenant choisir votre Bonus et recharger votre compte au #123# (appel gratuit).&20904=Mobicarte: Votre numero de telephone est le 0645324806, valide jusqu'au 05/01/12.&20904=Bienvenue chez Orange,votre numéro mobicarte est le 0645324806. Vous bénéficiez d'un crédit de 5E de communications valable jusqu'au 05/01/12 en Fce métro.&

c) 短信数据

图 3-18　从通讯数据包中解密出的隐私数据

（3）服务器指令接收执行

恶意软件每隔 15 秒向服务器请求一次指令并执行。部分指令的执行过程中还需要多次跟服务器进行通讯。例如，收到指令"getsms"、"getcontacts"时，恶意软件会把当前取得的短信和联系人数据发送给服务器。

服务器发来的指令是一个 xml 格式的文件，其中包含了关键信息是指令名 cmd 和参数 params（图 3-19）。

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<rootElem>
<cmd>getcontacts</cmd>
<params>/0S550SSSO5.php</params>
</rootElem>
```

图 3-19　服务器发送给恶意软件的指令内容

我们提取出了恶意软件能够接受并执行的 8 种有效指令。详细内容见表 3-1。

表 3-1　恶意软件的所有有效指令

指令名	处理类	行为	参数含义
vibrate	c.g	使手机震动	震动时间长度（单位：微秒）
call	c.b	拨打电话	对方的电话号码
sendsms	c.j	发送短信	对方的电话号码和短信内容（分号分隔）
getcontacts	c.i	把手机上所有联系人数据发送给服务器	服务器 URI 地址
goto	c.a	打开浏览器	要打开的 URL
smsspy	c.h	开启短信监视模式	
smsunspy	c.d	关闭短信监视模式	

getsms	c.c	把手机上所有短信数据发送给服务器	服务器 URI 地址

（4）短信监视模式

服务器发送给恶意软件的指令中，包含一条特殊的指令——“smsspy”。收到这条指令时，恶意软件会进入”smsspy”监听模式，在此模式下，会监听每一条短信的收发，并且在收到短信时，立即把短信数据通报给服务器。

3.7 本章小结

本章提出了一套 Android 平台上的软件恶意行为安全分析方法，该方法包括如何快速的鉴定恶意软件，如何去除软件代码的混淆保护，如何对代码的结构模块进行分析，以及如何结合分析进行恶意行为提取重建。

我们对一个实际的恶意软件入侵案例（来自于国际安全组织 honeynet 挑战赛[19]），进行了深入彻底的分析研究，提取还原了软件中的全部恶意行为和相关的数据。包括恶意软件的工作原理，通讯协议，窃取的隐私数据内容等。通过该实例的分析，证明我们提出的方法是高效和可靠的。

第四章 Android 系统安全增强设计

本章主要介绍 Android 系统安全增强设计相关的工作。针对 Android 系统用户隐私数据容易被泄露等安全问题，我们围绕着安全性、运行效率和易用性，设计了一套以数据为中心的 Android 系统安全增强方案，该方案对系统执行强制访问控制策略, 提供了可信内核、可信数据库和数据实时加密等技术。同时我们也对该方案与其他方案相比所具有的的特点，以及安全性和性能开销进行了评估。

4.1Android 安全增强问题

Android 系统提供了多种机制，来提高数据的安全性。它本身提供的安全保护有：屏幕锁定，数据加密，以及应用程序权限声明等。然而，这些机制仍然其有不足之处，它们们并不能对用户的隐私数据进行充分的保护，而且对于不熟悉 Android 安全问题的用户来说，是难以正确的使用的。

以下三个问题，是本文提出 Android 安全增强方案的设计动机。

（1）国内第三方市场大量处于”灰色”地带的应用

目前在许多第三方 Android 应用市场上，存在大量处于“灰色地带”的应用程序，一方面这些应用程序带有用户需要的常规功能，另一方面，他们为了达到收集用户数据的目的，申请了大量的权限，任意的去读取用户的隐私数据，如通讯录等。而对于大部分普通手机用户来说，他们都并没有充分的安全意识以及对安全问题的理解，并不能够正确的对系统进行安全的配置和管理。

（2）没有仅针对隐私数据保护的轻量级方案

一些想要增强自己的系统安全性的用户，可能会选择安装杀毒软件[12][33]，还有一些用户可能会选择磁盘加密[34]。然而，这些安全增强技术，并不能提供单独对于隐私数据的针对性保护。此外，由于常见的杀毒软件都需要在后台常驻服务，从而导致系统资源的大量开销。

（3）普通用户并不能很好的管理应用的权限，现有安全方案过于复杂

关于提升 Android 系统的安全性，也有不少的学术研究。TISSA[27]给用户提供了一个灵活并且高精度的机制，用户可以控制哪些个人信息可以被应用程序访问到。同时，为了满足用户在不同情境下的使用需要，相关的控制权限也可以在运行时由用户

动态的进行调整。Saint[7]是另一个相关的研究工作，它允许开发者提供一个增强的权限策略，该策略既能控制 Android 应用程序在安装时的权限授予，也能控制他们在运行时的权限使用。Apex[35]是一个 Android 安全策略增强的框架，它不但允许用户选择授予应用程序权限，也允许用户授予应用程序对于系统资源的使用权。L^4Android[36]是一个有代表性的基于虚拟机的系统，他使用用户权限隔离和系统管理程序来使得权限控制更加严格。然而他们并没有考虑到应用程序层面的安全性，并且，这些学术研究提供的方案，对于最终用户来说都过于复杂，并不适合于一般用户的使用。

即使受到许多攻击的威胁，移动设备的用户通常并不会去通过设置安全策略来保护自己的设备。因为大部分现有的安全增强机制，都并不适合不懂安全的一般用户来手动的进行安全策略配置。TrustDroid[37]是一个主动的安全框架，它在包括内核和应用层的 IPC 通信渠道在内的多个层面，进行了域隔离，来实现安全性的增强。尽管它自称是一个轻量级的策略，然而它需要监控每一个 IPC 过程，并且采用 MAC 强访问控制策略，这带来了相当的复杂性。

针对上述安全问题，在本文的设计中，我们提出了一个主动的轻量级安全增强方案。方案的核心思想是以数据为中心的安全模型。我们仅专注在数据源的保护，并且对于数据流只做相对松散的监控。尽管存在大量的“不可信任”的 Android 应用程序，但是其中的大部分并不是专门的恶意软件或者病毒程序，他们收集许多用户使用数据是为了用户统计之用。这些应用程序在安全增强系统中应当是允许运行的，然而我们却需要禁止它们对于敏感数据的访问。所以，在我们的设计中，通过强制的安全限制，在用户层控制了应用程序对于隐私数据库的访问，通过对数据库进行加密，保护了数据的访问安全性。通过可信内核中使用的严格权限控制策略来保护文件系统，从而为上层的安全模块提供可靠的支持。

4.2 设计目标

在 Android 安全增强方案的设计过程中，我们着重针对三个方面的特性：

- 安全性
- 运行效率
- 易用性

为了提升安全方案的效率和易用性，我们在设计时遵循了这三条基本思路：

一、大部分隐私泄露都是来源于常用的应用程序中的广告和恶意行为，而不是来

源于那些利用系统漏洞进行攻击的破坏性程序。这些应用程序通常具有用户所需的正常功能，但是同时又会滥用用户的隐私数据。所以我们选择限制这些处于灰色地带的应用，而不是直接删除卸载他们。也就是说，对于这些带有隐私泄露行为的应用程序，应当采用一个宽松且智能的过滤手段，使得它只能访问到有限的隐私信息。

二、设计方案在达到保护安全的同时，也应当考虑到功率消耗以及用户的使用体验。毋庸置疑，对于移动设备来说，功耗和用户体验都是极其重要的；如果一个安全增强系统设计得极其复杂，那么它不但会降低系统的相应速度，也会提高设备的耗电量。所以说，数据加密和访问控制，需要做到足够的轻量级，使得由它带来的额外开销是可以被用户接受的。

三、作为安全增强方案，很显然安全性是设计时最着重考虑的目标。我们的基本设计思想是：建立一个以数据为中心的安全模型。

因为用户希望得到一个隐私数据保护和管理机制，这个机制是主动的灵活的并且拥有健壮性。它应当包含限制应用程序行为的主动策略，因为对于没有经验的用户，可能不会手动对恶意程序采取任何措施。我们期待建立的系统是这样的，它基于以数据中心的安全存储，并且提供主动的、完善定义的隐私管理策略。为了达到提升效率和易用性的目的，这些新的特性，需要能够正常的运行，同时又只对原有的系统和用户体验带来很小的干扰。也就是说，最好的选择是，采用对原有系统模型改变最小的方案。

为了达到我们的三个主要设计目标：安全性，运行效率，以及易用性，我们提出了一套以数据为中心的安全模型，并且，从内核保护，隐私数据库加密，以及应用程序管理三个方面给出了我们的设计方案框图（图 4-1）。我们采用了静态数据加密（data-at-rest encryption），以及安全增强内核的技术，来保护隐私数据不被违规访问。同时，我们使用隔离不同类型的数据，来处理不同应用程序之间的隐私访问管理问题。本章接下来的部分，将会详细介绍安全模型和相关的设计细节。

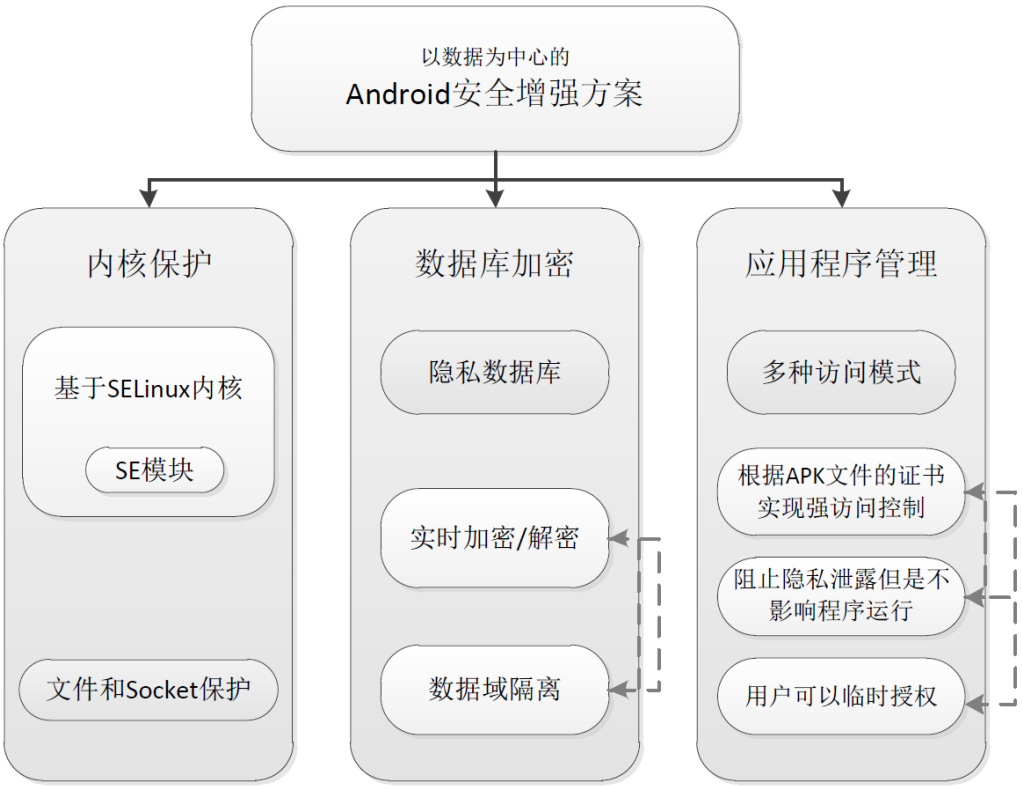


图 4-1 安全增强方案结构图

4.3 以数据为中心的安全模型

4.3.1 安全模型概述

Android 系统提供了一套以基于权限的安全模型，来确保它对应用程序进行安全限制。然而在实际应用程序开发时，常常会为了实现某个功能而申请所有相关的权限，导致相应的控制策略倾向于模糊化。我们很难判断一个应用程序申请的权限是用于正当用途还是恶意用途。例如，一个地图程序可能会要求申请一个获取 GPS 信息的权限，申请这个权限的基本目的是用来进行位置定位，然而，它也有可能滥用用户的位置信息，把它泄露出去用作其他不正当的用途。

为了解决这个问题，我们提出了一个以数据为中心的安全模型。这个模型的核心思路就是，建立一个可信的隐私数据库，然后，建立一个定义完善的数据隔离和访问策略，使它作用在隐私数据的访问过程上，从而限制隐私数据被滥用的行为。在这个模型中，在绝大部分情景下，应用程序的操作是在会在一个很低的数据访问特权环境

下进行，此时隐私数据是被完善的保护起来的。

该安全模型包含以下特性：

- 基于内核的文件访问控制
- 静态数据加密
- 身份认证增强
- 数据域管理

我们把系统中的数据根据敏感程度进行划分。应用程序在默认状态下，只能够获取非隐私数据和非敏感数据。如果一个应用程序试图访问它权限之外的数据，那么将会得到空数据的或是伪造的数据。如果它想要得到真实的敏感数据，则需要通过用户的密码认证，才能够访问到。在这样的限制之下，即使用户为了使用需要而不得不安装一些不充分可信的“灰色”应用程序，它也会在预先定义好的隐私权限之内运行，它所读取到的隐私数据也是空的或者伪造的，所以用户的真实隐私数据不会受到危害。

4.3.2 可信数据库建立

目前 Android 系统的一个重要的不安全因素，就是所有的数据库内容，都是以明文存储的。由于数据以明文存储，那么恶意程序，或是能够物理接触到智能手机的攻击者，就可以很容易的通过访问明文数据库来获取到隐私信息，而不用担心任何权限限制。然而，如果要建立一个完整的可信加密体系，则又会带来许多的新问题。首先，Android 系统中现有的数据加密保护技术，都非常的重量级[34]。其次，一些恶意软件可以通过特权扩大攻击的手段，在没有通过用户许可的情况下访问这些数据。更严重的情况是，一旦用户不小心安装了一个恶意软件，这个软件的权限就已经是被授予并且一直固定不变了，它可以轻易的获取用户的隐私数据或是使用系统资源来做任何的事情，而不需要再次获得用户的许可。

为了改进加密体系并且建立一个可信的数据库，我们采用了 MAC（强制访问控制）内核和轻量级的静态数据加密方案。为了建立这样一个可信的数据库，MAC 内核和轻量级的静态数据加密方案可以抵御绝大部分的物理接触攻击，从而保证数据库是安全可靠的。MAC 内核彻底杜绝了获取系统 Root 权限的可能性，无论是通过物理恢复手段，还是通过 adb（Android Debug Bridge）接口。由于没有了 Root 权限，那么数据加密的密钥将不可能被攻击者获取到，数据库是被完善保护的。即使攻击者能够物理接触到手机，他仍然不能获得任何隐私相关的数据。

（1）轻量级静态数据加密

我们认为，相比起全文件系统加密方案，一个轻量级的数据库加密方案，就已经是足够达到设计目的。在我们的系统设计中，通过加密只和用户敏感数据相关的数据库，来消除数据库的不安全性。这些包含敏感信息的数据库包括：通讯录，短信，通话记录，备忘录等。在我们的安全方案中，这些敏感的数据文件，只有在合法用户使用手机时，才会被解密。而当 Android 系统启动时，加载数个加密的数据库文件，显然要比加载整个加密的文件系统要快的多。由于重新加载这些数据库的开销是非常小的，所以，为了不再内存中留下明文数据，每次手机屏幕锁定时，系统都可以把内存中已经解密出来的明文数据抹除，只要在下次解锁屏幕时，再重新读取加密数据库就可以了。这样就没有任何敏感数据是以明文存储的，所有的隐私数据，得到了充分的保护。

（2）MAC 内核

Android 系统是一个非常开放的操作系统，然而正是由于他的开放特性，也使得它经常遭受来自系统底层的攻击。Root 是一种允许用于获取 Android 系统完整控制特权的过程。由于 Android 是从 Linux 内核发展而来，Root 一个 Android 设备实际上跟在 Linux 上获取管理员权限的过程，是类似的。有一些应用程序，会利用 Root 权限，来获取关键的系统资源和数据，从而导致敏感数据泄露。另一个潜在的威胁是 Android 中具有很高控制权的 adb 模式。Adb 是 PC 机上许多手机管理工具用来进行手机访问的接口和桥梁。但是通过电脑连接 adb 接口却不需要通过严格的身份认证，这样导致它也容易被攻击者利用。

为了防范这些潜在的对于系统特权的滥用，在我们的安全设计中，把 Android 系统原有的 DAC 自主访问控制策略，替换了 MAC 强制访问策略，后者的控制精度，从应用程序精确到了进程。在一般的 DAC 模型中，由用户来控制资源的访问。而为了方便，一些没有安全意识的用户，会打开比较危险的文件访问权限。由一个用户启动的进程，可以修改或者删除这个用户所能访问到的所有文件。与 DAC 模型不同，MAC 模型基于一个组织化的安全策略来控制用户和进程的资源访问，能够更对权限进行更高层面的控制，并且可以防范许多已知的 Android 漏洞。

4.3.3 身份鉴定与数据域隔离

Android 系统并不在应用层提供多级数据访问控制。所有的应用程序，都是在 Dalvik VM[38]虚拟机中的同一个特权级别下运行的。为了提升应用程序层的隐私安全

性，我们需要设计一个比现有模型更好的 MAC 数据访问模型。我们的设计主要包括两个方面：身份鉴定增强，以及数据域隔离。

（1）身份鉴定增强

我们借鉴了通常浏览器的对于网站身份认证做法，来运用到 Android 应用程序的身份认证上。

系统维护一个证书数据库，数据库中保存所有可信的证书，基于这些可信证书来区分鉴别哪些应用程序是可信的。安全限制策略检查每个应用程序的证书，基于他的证书可信度来给予它相应的特权级别。也就是说，每个应用程序，都会在安装的时候，基于数据库中的可信证书，来进行应用程序的签名验证，最后根据验证结果来进行特权级别分类。通过增强应用程序的身份鉴定，我们能在给予第三方应用程序正常安装权限的同时，控制他对于数据的访问，从而防止敏感数据泄露。一个通过可信任的证书签名的应用程序，将会无阻碍的正常运行，而没有通过可信证书签名的应用，将会在访问数据的时候，受到严格的审查和限制。

（2）数据域隔离

为了实现动态的隐私数据保护，仅仅使用可信数据库，是不够的。我们在设计中采用了多级别的数据隔离手段，基于数据的隐私程度构建了一个敏感数据的层次结构。

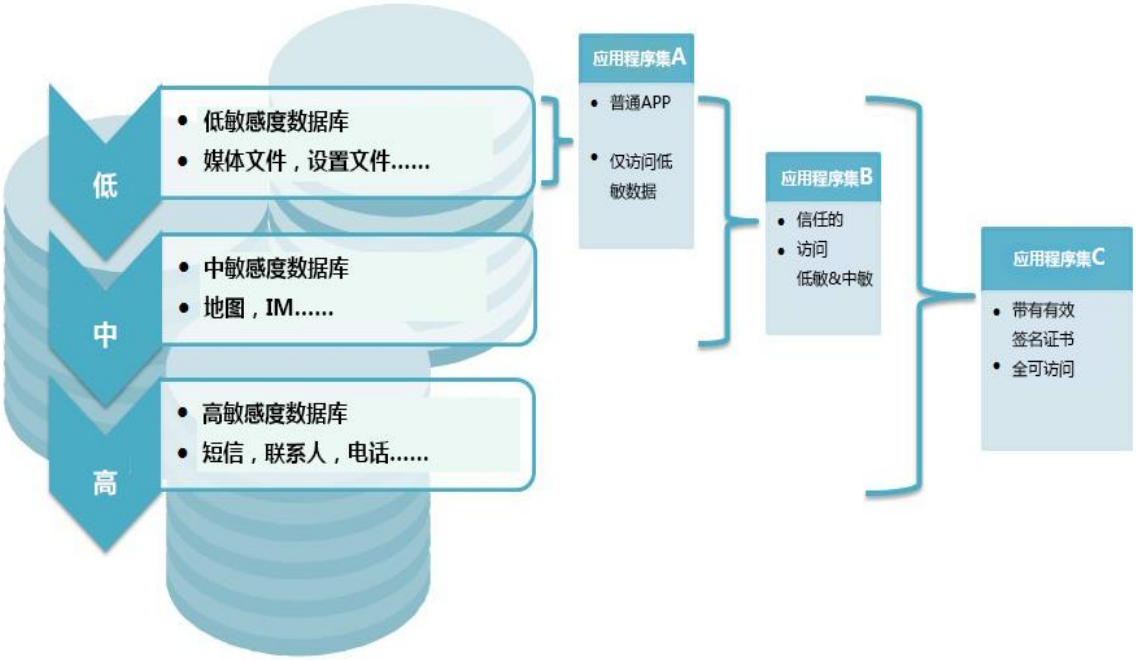


图 4-2 数据域隔离

在默认状态下安装的应用程序，都会在一个相对较低的特权模式下运行，并且只能访问部分敏感数据。我们的设计提供了一个应用程序层面的隔离，并且把经过分类的应用程序，映射到不同的数据域中（图 4-2）。对于可信的应用程序，可以访问所有它需要的敏感数据。对于不可信的应用程序，访问敏感数据时，将会得到空数据或者伪造的数据。也就是说，如果一个应用程序没有可信的证书签名，将会无法获得敏感数据的访问权限，无法获得用户的隐私数据。作为这个机制的补充，在特殊的情境下，应用程序可以通过用户界面交互来获得更高的特权。用户如果手动提供了密码，可以临时提升应用程序的特权，从而解锁对于敏感数据库的访问。

4.4 实现细节与评估

这一小节中，我们将描述安全增强方案的具体实现。包括我们为了达到设计目标和安全模型，所采用的技术手段。

4.4.1 内核保护

在我们的安全增强方案中，是基于 SEAndroid 来进行内核保护的[28]。SE Android 建立了一个内核级的保护机制，从而确保了文件系统的安全性，并且提供了内核级的防注入保护。它保护了密钥，为后续的数据库加密模块提供了基础。

表 4-1 测试 root 工具对于 SEAndroid 内核保护的有效性

Root 工具	能否 root
GingerBreak	无法 Root
Exploid	无法 Root
RageAgainstTheCage	无法 Root
Zimperlich	无法 Root
KillingInTheNameOf	无法 Root
Psneute	无法 Root

表 4-1 的测试结果显示，在基于 SEAndroid 的内核下，这些 Root 工具都无法获取 Root 权限，这证明了，基于 SE Android 来实现安全增强，可以克服 Android 的基于 DAC 模型的缺陷，并且防御大部分的系统漏洞。

SE Android 内核使用 MAC 强访问控制模型来取代 Android 系统本身的 DAC 自主访问控制模型，从而使得控制精度从应用程序级精确到了进程级。所有的应用程序都是无法读取其他应用程序创建的文件的。它还建立了一个固定的策略，把权限检查改为域管理和应用程序隔离，通过沙盒和分类的机制。所有这些手段，都可以增强灵活性和授权的精度。

4.4.2 隐私数据库加密

为了配合 SE Android 内核，同时保证系统的轻量级，我们选择了仅加密隐私数据库，而不是加密整个文件系统。数据库是在屏幕解锁时，用户输入自己的私钥之后解锁的。具体技术实现时，我们选择使用 SQLite 的加密 API。这样的做法，阻止了非法用户通过物理接触的方式，来获取信息。敏感数据是在系统运行的时候被频繁访问的，所以我们选择了运行效率很高的 RC4 算法来进行数据库加密。加密数据的整个访问过程参见图 4-3。

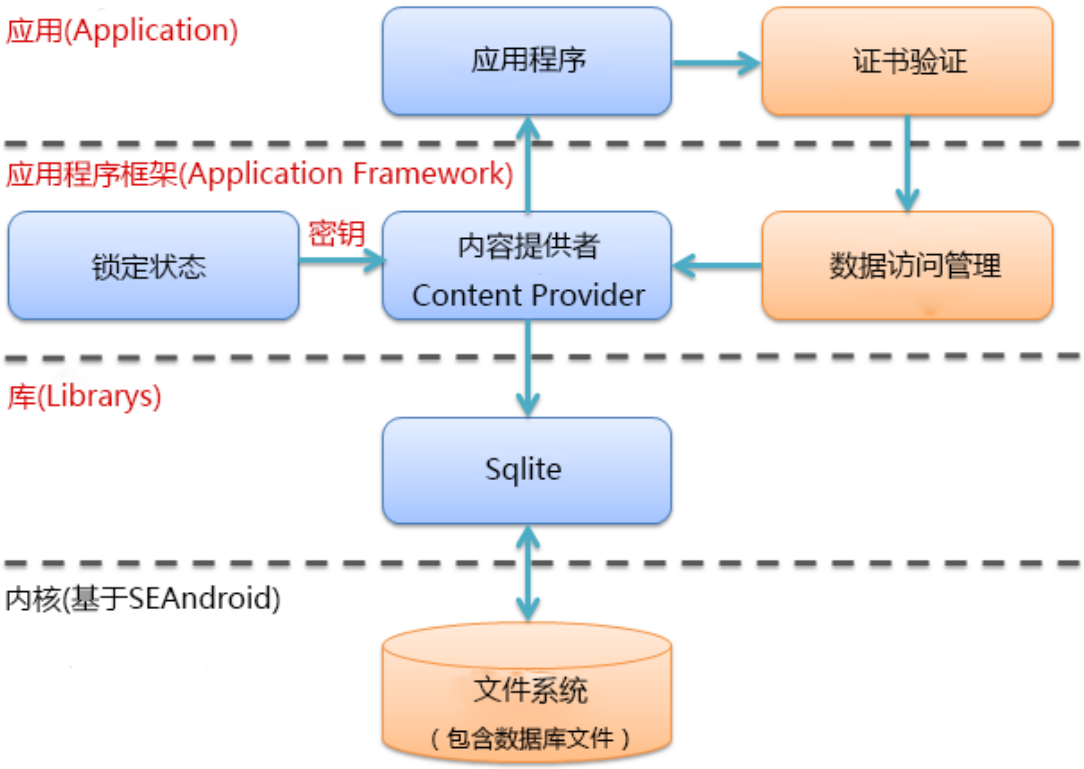


图 4-3 加密数据访问过程

（1）数据库加密过程

在我们所设计方案的强制的安全限制和策略下，会在用户层检验每个应用程序的证书，在应用程序需要访问用户隐私数据时，检查它的信任等级。如果是可信的应用，它就可以访问这些数据，否则就会收到一个空的或者伪造的数据，通过返回空值或伪造数据，可以保证该应用在能正常运行的同时，又不泄露真实的隐私数据。我们会对所有核心数据进行加密。也就是说，我们只会对那些包含用户隐私数据的数据库文件进行加密，这样可以显著的减少系统资源的开销。

当使用加密数据库的手机开机时，系统会需要用户提供密钥来解锁屏幕。如果用户提供的密钥正确解锁成功，那么用户可以进入完全正常的使用状态，运行在后台的服务会每隔一段时间自动写入加密数据库。实际上，解密后的数据库会被存储在手机的内存中，这样用户在解锁屏幕后进入系统然后正常的运行应用程序。因为只有合法的用户才拥有密钥来解锁屏幕，所以系统在合法用户登录之后会把数据库进行解密。当用户锁定屏幕时，解密出来的明文数据库会被再次加密并写回，同时，内存中存储的明文数据库也会被抹除。

（2）数据库加密测试评估

我们针对联系人数据库和短信数据库，这两个 Android 系统中关键的隐私数据库，测试了对隐私数据库进行加密的效果和性能开销。

在 Android 系统中，联系人数据，是由 ContactsProvider 来维护的，而 ContactsProvider 是通过 SQLite 来进行数据库存取。最终，联系人数据的实际数据库文件，存放于 Android 系统的一个文件中，文件路径为：

/data/data/com.android.providers.contacts/databases/contacts2.db。

类似的，短信数据由 TelephonyProvider 进行维护，短信数据的最终数据库文件路径为：

/data/data/com.android.providers.telephony/databases/mmssms.db

这两个数据库在默认状态下都是以明文存储在存储器中的，我们选择了一台实际的 Android 设备，通过 adb 接口，下载出其中的 contacts2.db 文件，使用 sqlite database browser 打开数据库文件，可以看到，contacts2.db 里面包含了所有的明文存储的联系人数据，包括电话号码，姓名，等在内的所有数据都是可以读取到的（图 4-4）。

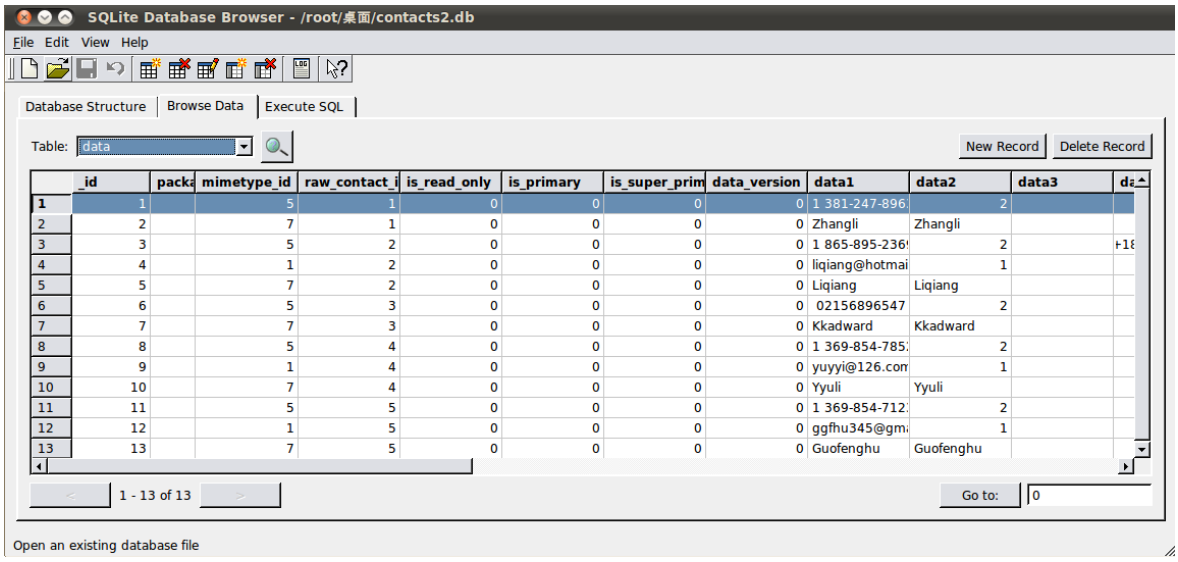


图 4-4 未加密状态下的联系人数据库文件

为了对这些数据进行加密，我们分析了系统中 SQLite 模块的实现。SQLite 是开放源码的数据库模块，它的源码中含有预留的加密函数接口，我们通过实现这些加密函数接口，从而实现对于隐私数据库的加密。

我们对 Android 源码中的 Sqlite 模块进行了以下修改：

（a）开启 Sqlite 编译开关

在 external/sqlite/dist/Android.mk 文件中，增加编译开关：

-DSQLITE_HAS_CODEC=1

（b）实现加密的接口函数

在 sqlite.c 文件中，有以下接口函数，需要实现：

void sqlite3_activate_see(const char* right) ;

void sqlite3CodecGetKey(sqlite3* db, int nDB, void** Key, int* nKey) ;

int sqlite3CodecAttach(sqlite3 *db, int nDb, const void *pKey, int nKeyLen);

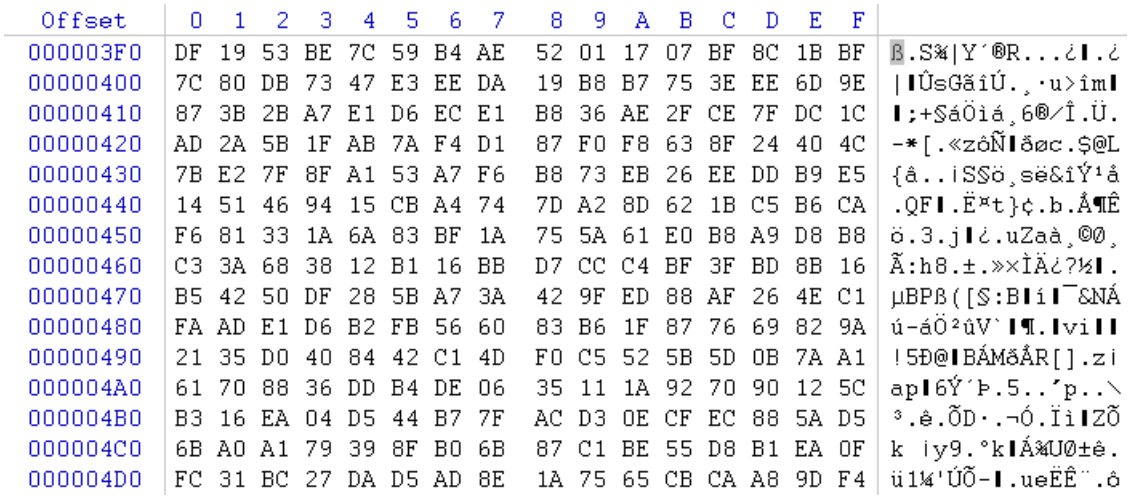
int sqlite3_key(sqlite3 *db, const void *pKey, int nKey);

int sqlite3_rekey(sqlite3 *db, const void *pKey, int nKey);

我们实现了这些函数的具体逻辑，其中包括加解密算法函数，以及数据库页面编码、密钥提供、密钥扩展，等加密辅助逻辑。

通过实现这些函数接口，我们采用 RC4 加密算法，使用预置的加密密钥对联系人数据库和短信数据库进行了加密。植入这些加密代码之后重新编译生成 Android 系统镜像，即可得到自动对隐私数据库进行加密的 Android 系统。

实现了数据库加密之后我们再次打开 contacts2.db 文件，显示内容如图 4-5 所示，数据库的内容已经以密文存储无法直接读取。



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
000003F0	DF	19	53	BE	7C	59	B4	AE	52	01	17	07	BF	8C	1B	BF	ß.S¾|Y´®R...¿▌.¿
00000400	7C	80	DB	73	47	E3	EE	DA	19	B8	B7	75	3E	EE	6D	9E	|▌ÛsGãîÚ.¸·u>îm▌
00000410	87	3B	2B	A7	E1	D6	EC	E1	B8	36	AE	2F	CE	7F	DC	1C	▌;+§áÖìá¸6®/Î.Ü.
00000420	AD	2A	5B	1F	AB	7A	F4	D1	87	F0	F8	63	8F	24	40	4C	-*[.«zôÑ▌ðøc.$@L
00000430	7B	E2	7F	8F	A1	53	A7	F6	B8	73	EB	26	EE	DD	B9	E5	{â..¡S§ö¸së&îÝ¹å
00000440	14	51	46	94	15	CB	A4	74	7D	A2	8D	62	1B	C5	B6	CA	.QF▌.Ë¤t}¢.b.Å¶Ê
00000450	F6	81	33	1A	6A	83	BF	1A	75	5A	61	E0	B8	A9	D8	B8	ö.3.j▌¿.uZaà¸©Ø¸
00000460	C3	3A	68	38	12	B1	16	BB	D7	CC	C4	BF	3F	BD	8B	16	Ã:h8.±.»×ÌÄ¿?½▌.
00000470	B5	42	50	DF	28	5B	A7	3A	42	9F	ED	88	AF	26	4E	C1	µBPß([§:B▌í▌¯&NÁ
00000480	FA	AD	E1	D6	B2	FB	56	60	83	B6	1F	87	76	69	82	9A	ú-áÖ²ûV`▌¶.▌vi▌▌
00000490	21	35	D0	40	84	42	C1	4D	F0	C5	52	5B	5D	0B	7A	A1	!5Ð@▌BÁMðÅR[].z¡
000004A0	61	70	88	36	DD	B4	DE	06	35	11	1A	92	70	90	12	5C	ap▌6Ý´Þ.5..'p..\
000004B0	B3	16	EA	04	D5	44	B7	7F	AC	D3	0E	CF	EC	88	5A	D5	³.ê.ÕD·.¬Ó.Ïì▌ZÕ
000004C0	6B	A0	A1	79	39	8F	B0	6B	87	C1	BE	55	D8	B1	EA	0F	k ¡y9.°k▌Á¾UØ±ê.
000004D0	FC	31	BC	27	DA	D5	AD	8E	1A	75	65	CB	CA	A8	9D	F4	ü1¼'ÚÕ-▌.ueËÊ¨.ô

图 4-5 加密后的联系人数据库文件片段

我们在对数据库解密的时间进行了测试，测试使用设备型号 Xiaomi 1S，改设备是一台实际使用超过 1 年的手机，手机中包含了大量的联系人数据和短信数据，这样的数据容量是具有现实意义的。

表 4-2 各个隐私数据库解密时间

数据库文件	内容	文件容量	解密时间
/data/data/com.android.providers.contacts/databases/contacts2.db	联系人数据，通话记录等	209 个联系人和对应的通话记录，文件大小 1.87MB	45 毫秒
/data/data/com.android.providers.telephony/databases/mmssms.db	短信数据	1546 条短信数据，文件大小 6.92MB	167 毫秒

通过测试结果可以看出，在包含了一定相当容量的数据的情况下，对隐私数据库进行加密单个数据库的解密时间在百毫秒级，因此，在解锁屏幕的时候，解密隐私数据库并读入内存，所带来的的额外开销是较少的，并不会影响到用户的使用体验。

4.4.3 应用程序管理

（1）应用程序的特权划分

每当一个应用程序安装时，系统会自动完成它的授权工作，这个过程不需要用户的参与。在安装结束之后，每个应用程序，都已经归类到一个应用程序集合里面去了，

每个集合对应了一个特定的数据域。

应用程序将被划分到三个集合中：可信应用，半可信应用，不可信应用。只有系统的应用程序，才会被添加到可信应用的集合中。我们把来自 Google Android 电子市场上带有签名的应用，添加到半可信应用的集合。而来自其他第三方市场或渠道的应用，添加到不可信应用集合。

于此同时，敏感数据也被划分为三个级别：低敏感度，中敏感度，和高敏感度。举例来说，低敏感度的数据包括，settings.db，media.db 等与隐私的关联很小的数据。而高敏感度的数据包括 SMS.db，contacts.db 等和隐私关联较大的数据。

（2）特权控制下的应用程序运行模式

通过在应用程序安装期间就进行分类，实现了应用程序的特权划分。在不同的特权下应用程序运行时，从系统中读取到的隐私数据，是来源不同的。对于可信应用，所有的数据库，都是可以直接访问到的。从逻辑上说，半可信应用，只能访问到中敏感度和低敏感度的数据，而不可信应用，只能访问到低敏感度数据。

为了所有的应用程序都能正常运行，并确保用户体验良好。在应用程序尝试访问超过他的特权的数据时，将会收到一份空的或伪造的数据，同时系统也将把此次访问作为一个警告信息，在日志中记录下来。

当日志中的警告超过特定的数量时，那么说明这个应用程序是一个恶意程序，或是没有得到合适的授权。此时系统会向用户弹出提示，由用户决定，是删除这个程序，还是忽略这些警告，亦或是提升这个应用的权限，如果选择提升权限，则需要用户输入密码进行认证。

（3）测试评估

我们再次使用了 3.6 小结中的恶意程序案例进行测试，该恶意软件会在运行的过程中，会将本机的设备 ID 值等信息发送出去给自己的远程服务器。然而，如果彻底禁止该软件请求获取设备 ID 的权限，则又完全无法正常使用该程序。

在我们的应用程序管理方案下，会将该软件判定为不可信程序，不可信程序具有最低的数据访问权限，当它尝试越权访问数据时，我们会给它返回一份伪造的数据。

为了对该方案进行模拟测试，我们定位到了 Android 系统中获取设备 ID 的 API 实现源码，该 API 位于 Android 源码应用程序框架（Application Framework）层内的 TelephonyManager.java 文件中。

我们发现，包括 getDeviceId，getDeviceSoftwareVersion，getSimSerialNumber，getSubscriberId，getIsimDomain，getIsimImpu 等 API 函数，均会通过 getSubscriberInfo()

这个函数来最终获得设备的实际信息。改函数的原型为：

private IPhoneSubInfo getSubscriberInfo();

因此，我们修改了该函数的实现，在其中植入返回伪造数据的逻辑，针对不可信的应用程序，该函数将返回一组随机生成的设备 ID 等信息。

修改该模块源码后，重新编译 Android，然后在新的系统中再次运行原来的恶意软件，进行测试。

```
imei=356772040481677&imsi=208013002954000&opname=Orange
F&opcode=20801&opiso=fr
```

a) 原先恶意软件发送出的用户信息

```
imei=569714365478523&imsi=2364125485654223&opname=FakOp
&opcode=23651&opiso=af
```

b) 经过安全增强后，恶意软件泄露的用户信息（伪造值）

图 4-6 测试对不可信程序返回伪造的信息

测试结果（图 4-6）表明，该软件的仍然可以正常运行，但是它对后台服务器发送的设备 ID 等数据，变成了伪造的信息。在不影响程序正常运行、不影响用户体验的同时，本设备的真实信息得到了保护，没有泄露出去。

4.5 讨论

本小节中我们就本文提出的安全增强设计方案的进行讨论。

4.5.1 调用时验证与监控运行的比较

我们的系统安全增强方案之所以是轻量级的，是因为它采用的是调用时验证模式，而大部分安全软件和杀毒软件，都采用的是不间断的监控运行模式。

在调用时验证模式下，所有的安全验证代码，都是只有在需要的时候才会被调用执行。而监控运行模式下，则需要开启一个监控服务，监控服务驻留在系统中，会带来较大的开销，从而导致电量消耗过快，系统资源被占用过多等。而我们的方案设计中相关的安全代码，只有在需要的时候，才会被调用，被调用的时候，进行证书验证等操作，带来的开销也是较小的。此外大部分的监控程序，都需要一个复杂的配置过程，并且把监控结果输出成一个复杂而专业的报告。这对于大部分用户来说，也是不友好，难以使用的。

4.5.2 数据库加密与全系统加密的比较

从 Android 系统 3.0 版本开始，就提供了一个全系统的静态数据加密选项，在这个选项开启时，系统将会加密整个文件系统。而在我们的方案设计中，我们只挑选出来与用户隐私数据相关的数据库来进行加密。

全系统加密，需要花费数小时才能完成，并且在加密的过程中是不能取消的。每次用户需要修改密码时，都需要花费等同于全部重新加密的时间才能完成；并且，由于全系统加密作用于整个运行中的系统，如果用户使用不当，容易导致系统出现麻烦的问题。在我们的方案设计中，我们只加密预先选中的几个隐私相关数据库，如 sms.db 等，这样就可以在保护用户隐私数据的同时，又不影响到原有系统的运行，也不会带来过多的时间和资源开销。Android 提供的全文件系统加密本身并不会带来明显的使用延迟或是频繁的打扰用户输入密码，而我们设计的系统方案，会带来更小的使用延迟，以及更少的对于用户参与和操作的要求。

4.5.3 对于常见攻击的抵御能力

即使加密了整个文件系统，Android 依然不能抵御内核级的攻击。而在我们的设计中，通过安全增强内核，以及三层的安全策略，增强了用户在不同情境下的隐私数据安全性。

通过采用 SE Android 内核，我们可以抵御内核注入攻击，以此为基础来保证数据库的加密过程不在底层受到攻击。加密后的数据库可以抵御物理层面的攻击，也能抵御用户层面的应用程序导致的数据泄露。在用户层面的权限控制上，我们提供了一个灵活的策略，用来保护用户的隐私数据，同时又不影响应用程序的正常运行。

4.5.4 数据安全性

数据的安全性，是依赖于加密和授权的。我们的设计目的是建立一个以数据为中心的安全系统，以此来保护用户的私有数据，同时能抵御远程攻击和近距离的物理攻击。

通常来说，隐私数据最容易泄露的是通过两个渠道，文件系统，以及 Content Provider。对于通过文件系统的攻击，如果攻击者可以访问到数据库文件，它将可以获取到所有的数据。而我们设计的数据加密解密的方案保证了所有的明文数据仅会在

内存中进行存储，而一旦数据被写入文件系统时，就已经是加密后的形式了。我们的策略保证了数据一定是通过标准的加密算法进行加密后存储的，所有攻击者唯一可能获取数据的方式，只能通过 dump 内存，为了防止他这么做，我们在设备屏幕锁定时，会把内存中的数据，都清除掉。

对于通过 Content Provider 的攻击，攻击者如果需要通过 Content Provider 来获取数据，则需要通过两次身份认证的过程。第一个过程是，应用程序安装时的证书认证。第二个是解锁数据时的密码认证。这对于数据访问的安全性控制来说，是非常严格的。

4.5.5 性能开销

我们通过对比 Android 原有的全系统加密和我们的隐私数据库加密方案，评估了安全增强方案的加密过程的性能开销。

在我们的设计方案中，当设备解锁时，数据库是需要解密并存储在内存中的。基于我们在 Android 手机中的测试，这个解密数据的过程，并没有带来用户明显察觉的性能变化，是可以接受的。

另一个方面是数据的可靠性，我们的设计方案中，用户数据只要发生改变，就会被写回加密数据库。因为修改用户数据的操作频率远远小于读取用户数据的频率，所以在实际使用中也是可接受的。

此外，首次加密隐私数据库只需要数秒时间，远远短于加密整个文件系统的时间。总之，加密解密数据库的过程，只带来了很小的性能开销，是可以接受的。

4.6 本章小结

本章提出针对 Android 的系统安全增强方案，基于 Android 已有的安全机制进行了安全增强。首先，我们吸收了 Kernel 层的 SEAndroid 强制访问方案作为内核保护的基础，其次，我们通过轻量级数据保护方案保护了敏感数据的安全，最后，我们在应用层面上对应用程序进行了权限和信息的严格监管。本文就 Android 系统的内核保护、数据加密和应用程序保护这三个具体内容进行讨论，详细描述了本方案是如何设计从而保护这些内容。该方案建立了可信的数据库，并对应用程序进行权限隔离，该方案的设计集中在数据保护上，它具有轻量级的特点，在保证安全性的同时，也提升了运行效率和易用性。

第五章 全文总结

本章将对全文进行总结，归纳本文研究的 Android 软件安全分析和系统安全增强研究的主要工作和创新点，并对今后工作进行展望。

5.1 本论文的主要结论与创新点

本文主要有以下结论和创新点：

本论文的主要目的是，提出一套 Android 平台上的软件恶意行为安全分析方法，该方法包括如何快速的鉴定恶意软件，如何去除软件代码的混淆保护，如何对代码的结构模块进行分析，以及如何结合分析进行恶意行为提取重建。对一个实际的恶意软件入侵案例，进行了深入彻底的分析研究，提取还原了软件中的全部恶意行为和相关的数据。包括恶意软件的工作原理，通讯协议，窃取的隐私数据内容等。通过该实例的分析，证明我们提出的方法是高效和可靠的。针对 Android 平台的系统安全，提出了一套系统安全增强方案。该方案基于以数据为中心的安全模型，建立了可信的数据库，并对应用程序进行权限隔离。和同类研究相比，具有轻量级的特点。

具体如下：

1. 提出了一套系统化的分析 Android 软件恶意行为的安全分析方法，给出了 Android 恶意代码的一般特征，归纳了一套通用的逆向分析的方法流程。包括如何快速的鉴定恶意软件，如何对抗恶意软件的反分析代码，如何根据恶意代码进行行为重建等。该方法提取恶意行为的一般特征进行分析，包含鉴别，代码反保护，代码分析，行为提取等步骤。本文总结了一些恶意软件常有的代码特征模块，可以帮助分析者更好更快的进行分析理解恶意软件。

2. 同时，本文选取了一个 Android 系统受到恶意软件入侵的实例进行分析，以一个实际的恶意软件为例（来自于国际安全组织 honeynet 挑战赛），进行了深入彻底的分析研究，我们有效的提取出了该恶意软件与秘密服务器进行加密通讯，窃取用户隐私数据等全部的恶意行为。还原提取了软件中的恶意行为和相关的数据。包括恶意软件的通讯协议，通讯过程中发送的隐私数据等。通过该实例的分析，证明我们提出的方法是高效和可靠的。

3．本文提出了一套以数据为中心的 Android 系统安全增强方案，对 Android 已有

的安全机制进行了安全增强。安全方案的设计包含内核层，数据保护方案，以及应用程序管理三个层面，我们吸收了 Kernel 层的 SEAndroid 强制访问方案作为访问基础，通过轻量级数据保护方案保护了敏感数据的安全，并且在应用层面上对应用程序进行了权限和信息的严格监管。我们对方案的可行性和运行效率进行了评估，评估结果显示该方案的开销是可以接受的。

本文就 Android 系统的内核保护、数据加密和应用程序保护这三个具体内容进行讨论，详细描述了本方案是如何设计从而保护这些内容。该方案建立了可信的数据库，并对应用程序进行权限隔离，该方案的设计集中在数据保护上。

5.2 未来工作展望

本文的后续工作主要可以往以下几个方面开展：

1. 本位提出的恶意软件分析方法是针对当前存在的恶意软件提出的，实际上，软件安全分析的研究是永无止境的，Android 系统将来会不断的更新，恶意软件也会不断的更新，将来会有采用新的方法，更加复杂，更加难以分析的恶意软件出现，将来需要对恶意软件的更新换代进行不断的跟踪分析，不断的完善分析方法。

2. 本文提出的恶意软件分析方法，依然很大程度依赖于人力分析，为了减轻人工负担，实现同时分析大量恶意软件的的能力，可以依据其中的一些方法，提取自动化分析策略，进行自动化分析工具的研究，例如恶意软件的自动化鉴别，自动化反保护等，如果能够研究开发出此类自动化工具，将可以大大提升恶意软件分析的效率。

3．本文提出了一套轻量级的，以数据为中心的 Android 系统安全增强方案，并且针对该方案的几个关键设计模块进行了测试评估。后续可以依据该方案的设计进行更加完善的系统开发，制作出能够给一般用户使用的安全增强系统。

参 考 文 献

[1] Google, “Android” URL http://www.android.com/ accessed March, 2012.

[2] W. Enck, P. Gilbert, B. G. Chun, L. P. Cox, J. Jung, P. McDaniel, “TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones,” in OSDI, 2010.

[3] A. R. Beresford, A. Rice, N. Skehin, R. Sohan. “MockDroid: trading privacy for application functionality on smartphones.” HotMobile 2011.

[4] M. Egele, C. Kruegel, E. Kirda, G. Vigna. “PiOS: Detecting Privacy Leaks in iOS Applications,” in NDSS, 2011.

[5] W. Enck, M. Ongtang, P. McDaniel. “On Lightweight Mobile Phone Application Certification,” in Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS) 2009.

[6] L. Desmet, W. Joosen, F. Massacci, P. Philippaerts, F. Piessens, I. Siahaan, D. Vanoverberghe, “Security-by-contract on the .NET platform.” Information Security Technical Report 13, 1 (January 2008), 25–32.

[7] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, “Semantically rich application-centric security in android,” in Computer Security Applications Conference, 2009. ACSAC’09. Annual. Ieee, 2009, pp.340–349.

[8] J. Jung, A. Sheth, B. Greenstein, D. Wetherall, G. Maganis, T. Kohno, “Privacy Oracle: A System for Finding Application Leaks with Black Box Differential Testing,” in Proceedings of ACM CCS 2008.

[9] A. R. Yumerefendi, B. Mickle, L. P. Cox, “TightLip: Keeping Applications from Spilling the Beans,” in Proceedings of the 4th USENIX Symposium on Network Systems Design & Implementation (NSDI) 2007.

[10] Y. Luo, W. Yang and J. Li, “Submission of the honeynet forensic challenge 9.” URL http://www.honeynet.org/files/1317348062_lyh62771@gmail.com_ForensicChallenge2011-Challenge9-Submission.doc accessed March, 2013.

[11] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011, pp. 627–638.

[12] L. M. Security, "Android security for mobile." URL https://www.mylookout.com/ accessed March, 2013.

[13] F. Di Cerbo, A. Girardello, F. Michahelles, and S. Voronkova, "Detection of malicious applications on android os," Computational Forensics, pp. 138－149, 2011.

[14] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," in Proceedings of the 19th Annual Network and Distributed System Security Symposium, 2012.

[15] D. Low, "Protecting java code via code obfuscation," Crossroads, vol. 4, no. 3, pp. 21–23, 1998.

[16] Brut.alll, "Android-apktool." URL http://code.google.com/p/android-apktool/ accessed March, 2013.

[17] pxb1988, "dex2jar." URL http://code.google.com/p/dex2jar/ accessed March, 2013.

[18] E. Dupuy, "Jd-gui: Yet another fast java decompiler." URL http://java.decompiler.free.fr/?q=jdgui/ accessed March, 2013.

[19] Honeynet, "Forensic challenge 9 - mobile malware." URL http://www.honeynet.org/node/751/ accessed March, 2013.

[20] C. Mulliner, "Privacy leaks in mobile phone internet access," in ICIN 2010.

[21] S. Morrissey, "iOS Forensic Analysis: for iPhone, iPad, and iPod touch." Apress, 2010.

[22] 邓超国, 谷大武, 胡维奇. 一种基于动态指令流的恶意程序检测方法. 全国计算机安全学术交流会论文集•第二十五卷, 杭州, 2010. P173-179

[23] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, P. Saxena. BitBlaze, "A new approach to computer security via binary analysis," in Proc. of the 4th International Conference on Information Systems Security, Hyderabad, India, 2008.

[24] L. Harris, B. Miller. "Practical analysis of stripped binary code." ACM SIGARCH Computer Architecture News, v.33, 2005

[25] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In Proc. of NDSS, San Diego, USA, 2005.

[26] J. Lessard and G. Kessler, "Android forensics: Simplifying cell phone examinations." Small Scale Digital Device Forensics Journal, 2010.

[27] Y. Zhou, X. Zhang, X. Jiang, and V. Freeh, "Taming information stealing smartphone applications (on android)," Trust and Trustworthy Computing, pp. 93－107, 2011.

[28] A. Shabtai, Y. Fledel, and Y. Elovici, "Securing android-powered mobile devices using selinux," Security & Privacy, IEEE, vol. 8, no. 3,pp. 36－44, 2010.

[29] W. Enck, M. Ongtang, and P. McDaniel, "Understanding android security," Security & Privacy, IEEE, vol. 7, no. 1, pp. 50－57, 2009.

[30] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, “Google android: A comprehensive security assessment,” Security & Privacy, IEEE, vol. 8, no. 2, pp. 35 – 44, 2010.

[31] RSA Inc., “Securing data at rest: Developing a database encryption strategy,” URL: http://www.rsa.com/products/bsafe/whitepapers/DDES WP 0702.pdf, 2012.

[32] Lookout Inc., “Lookout mobile security,” URL: https://www. mylookout.com, 2012.

[33] NetQin Inc., “Netqin mobile security,” URL: http://www.netqin.com, 2012.

[34] Google Inc., “Notes on the implementation of encryption in android 3.0,” URL: http://source.android.com/tech/encryption/android crypto implementation.html, 2012.

[35] M. Nauman, S. Khan, and X. Zhang, “Apex: Extending android permission model and enforcement with user-defined runtime constraints,” in Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security. ACM, 2010, pp. 328 – 332.

[36] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter, “L4android: a generic operating system framework for secure smartphones,” in Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. ACM, 2011, pp. 39 – 50.

[37] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A. Sadeghi, and B. Shastry, “Practical and lightweight domain isolation on android,” in Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. ACM, 2011, pp. 51 – 62.

[38] D. Bornstein, “Dalvik vm internals,” in Google I/O Developer Conference, vol. 23, 2008, pp. 17 – 30.

[39] Google, “Android forensics.” URL http://code.google.com/p/android-forensics/ accessed March, 2012.

[40] Google, “Code and documentation from android’s vm team.”
URL http://code.google.com/p/dalvik/ accessed March, 2012.

[41] Ophonesdn, “The structure of android package (apk) files.”
URL http://en.ophonesdn.com/article/show/354/ accessed March, 2012.

[42] E. Chin, A. P. Felt, K. Greenwood, D. Wagner, “Analyzing Inter-Application Communication in Android.” MobiSys '11 Proceedings of the 9th international conference on Mobile systems, applications, and services, ACM , 2011, pp. 239-252.

[43] S Bugiel, L Davi, A Dmitrienko, S Heuser, A. R. Sadeghi, B. Shastry, “Practical and lightweight domain isolation on Android.” SPSM '11 Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, ACM , 2011, pp. 51-62.

[44] S Bugiel, L Davi, A Dmitrienko, T. Fischer, A. R. Sadeghi, “Xmandroid: A new android evolution

to mitigate privilege escalation attacks.” Technical report TR-2011-04, Technische Universität Darmstadt, 2011.

[45] R Xu, H Saïdi, R Anderson, “Aurasium: Practical policy enforcement for android applications.” Proceedings of the 21st USENIX Security Symposium. 2012.

[46] J. Jinseong, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, T. Millstein. "Dr. Android and Mr. Hide: fine-grained permissions in android applications." Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices, ACM, 2012. pp. 3-14.

致谢

我想要首先感谢我的导师谷大武教授对我的悉心指导。回顾这些年的求学生涯，在谷老师的指导和带领下，我经历了许多的事情，成长了许多，进步了许多。我一直以来都有着诸多的毛病，也犯过不少的错误，而谷老师却给予了非常多的耐心和容忍。即使在百忙之中，谷老师依然占用休息时间来帮助我评阅和修改论文，这让我非常感动。谷老师有着严谨务实的治学态度和宽厚谦虚的为人品格，一直是我的精神榜样和楷模。

我想要特别感谢李卷孺师兄长期以来对我的进行科研工作和完成论文的大力帮助。自我开始硕士研究生的学习之时，李卷孺师兄就一直对我的科研工作给予了大量具体的帮助和指导，直到我要毕业之时，李卷孺师兄依然不遗余力的帮助我完成这篇论文。李卷孺师兄一直是我在学习和工作中的榜样。

感谢 LoCCS 实验室所有老师和同学对我的关心和帮助。感谢实验室提供的良好的学习氛围和友爱的生活环境。感谢刘军荣老师，刘亚师姐，张媛媛师姐，刘志强师兄，对我完成论文提供的鼓励和支持。感谢邓超国师兄，李升同学，赵若旭同学，对我完成论文提供的帮助。

感谢李柏岚同学，孙明同学，曲博同学，成为同学，柳妃妃同学对我的帮助。很有幸我们能够在求学生涯中共同进步和成长，你们都是我最优秀的同学，你们身上的闪光的优点，不断的鼓励着我奋斗和前行。

感谢研究生教务处的蔡幼玲老师，感谢蔡老师一直在帮助我完成论文和毕业相关的事务。

最后，深深感谢给予我生命的父亲和母亲，感谢你们把我养大，感谢你们在我的求学之路上给予的一贯的支持和鼓励，你们是我在前进之路上的坚实后盾，我将会努力奋斗，报答你们的养育之恩。

攻读硕士学位期间已发表或录用的论文

[1] Y. Luo, D. Gu, and J. Li, “Toward Active and Efficient Privacy Protection for Android” in Third IEEE International Conference on Information Science and Technology (ICIST 2013)