2015 年全国大学生信息安全竞赛

作品报告

作品名称：面向 Android 系统内部数据传递过程的敏感信息保护系统

电子邮箱：1242750990@qq.com

提交日期：2015 年 6 月 7 日

填写说明

1. 所有参赛项目必须为一个基本完整的设计。作品报告书旨在能够清晰准确地阐述（或图示）该参赛队的参赛项目（或方案）。
2. 作品报告采用 A4 纸撰写。除标题外，所有内容必需为宋体、小四号字、1.5 倍行距。
3. 作品报告中各项目说明文字部分仅供参考，作品报告书撰写完毕后，请删除所有说明文字。(本页不删除)
4. 作品报告模板里已经列的内容仅供参考，作者也可以多加内容。

目　　录

第一章 摘要

Android 是由 Google 公司领导、开发的一种基于 Linux 的智能设备操作系统，由于其开放性和易用性，得到了众多手机厂商和广大消费者的青睐，现已成为全球最大的智能手机操作系统。 然而，由于 Android 系统自身的安全设计缺陷和手机中各类高价值隐私信息的存在，使得各种针对 Android 系统的隐私窃取为主的恶意软件层出不穷。据网秦云安全数据分析中心统计：2014 年第一季度查杀到手机恶意软件共计 41199 款，同比增长 63.9%；感染智能手机共计 1784 万部，同比增长 71.5%， 其中隐私窃取类恶意软件以超过 43.5%的比例高居榜首。因此，研究如何保护用户敏感数据不被窃取成为了保证 Android 系统安全的重中之重，而其中**针对用户敏感数据在 Android 系统内部传递过程中极易被恶意软件窃取与篡改的安全威胁目前尚无相关研究，**这给用户的隐私和财产安全无疑带来了巨大的威胁。

针对此类威胁，本作品创新性的展开研究，给出了实用性较强的解决方案和原型系统。作品首先详细分析了用户敏感数据在 Android 系统内部产生、传递和使用的整个流程：为了实现不同应用程序、不同进程间的资源共享、数据传输，Android 系统提供了基于 Binder 的进程间通信的机制。敏感数据在系统内以 Binder 为通道进行传输。但是敏感数据在 Binder 传输过程中是以明文形式进行的，这就使得恶意软件能够轻易地截获和篡改基于 Binder 通信的敏感数据，如短信内容、GPS 位置信息等。针对这一问题，在上述研究的基础上，**本作品创新性地提出并实现了针对 Android 系统中敏感数据从产生到使用整个生命周期的自适应性透明加密的保护方案**，有效防止了敏感数据被恶意第三方窃取与篡改的威胁，**保障了用户敏感数据在系统内部传输过程中的私密性及完整性**。此外，本系统还向用户提供了简单、灵活的操作体验，使得用户可独立自主地对指定应用程序中特定类型的敏感数据进行保护，增强了本系统的易用性和实用性。

本作品的成功完成，不仅创新性的为 Android 系统中基于 Binder 的敏感数据传递提供了一种可选的防护方法；同时也为企业、政府等安全性要求较高的机构构建了一种强制性的安全增强型策略，促进了 Android 系统在企业中的应用，推动了 Android 生态系统的健康良性的发展，具有较大意义。

关键词：Android，系统安全，Binder 机制，进程间通信，隐私保护

第二章 作品介绍

2.1 背景与意义

随着近年来移动互联网的迅猛发展，移动智能终端已成为与人们的生活和工作关系最为密切的电子设备。与传统的个人用户计算机相比，移动智能终端由于其丰富的传感器和良好的便携性，使得其中包含更多的用户隐私信息和日常敏感数据。然而，针对智能终端的安全性研究和智能设备中用户隐私数据的保护尚处于初级阶段，因此研究智能设备中用户隐私的保护具有重要意义。下面将详细分析。

2.1.1 作品背景

一、时代背景

市场分析机构 Strategy Analytics 公布了 2014 年第二季度智能手机操作系统全球分布情况，报告显示，Android 操作系统的全球市场份额已达 84.6%（有史以来最高比重），成为最主要的智能手机系统。Android 操作系统自身安全性一旦出现问题，其影响范围则十分巨大；此外，卡巴斯基实验室的调查显示，98.05%的恶意软件将其目标锁定在 Android 操作系统中。综上，研究 Android 系统及整个 Android 生态系统的安全性具有十分重大的意义。

二、Android 安全机制缺陷

Android 系统以进程为单位分配和管理资源，当进程需要某种服务时，必需要向相应的管理者申请服务，管理者对请求进行应答，其中，对于服务的请求以及应答是通过 Binder 进行的。在请求服务的过程中，程序将请求的服务类型及数据，传递给 Binder，Binder 将其交给相应的服务管理者，服务管理者再通过 Binder 传回应答。因此，Binder 就好像是服务端与客户端之间沟通的邮差，然而，这个“邮差”并不是安全的，因为数据在 Binder 的传输过程中是以明文传输的，就相当于邮差传送的书信是没有信封的，只要通过对 Binder 的攻击，就可以拦截到“邮差”携带的书信。之后便可通过多种方式如：Broadcast ,Intent ,ContentProvider 等，将数据发送出去。

而这其中存在如下的安全缺陷：整个通信过程中，Binder 数据包内容都是以明文的形式存放的，这就意味着一旦攻击者截获了数据包，也就意味着用户的隐私已经泄露，这会给用户的安全带来极大的危害。而对于 Android 系统而言，由于其生态系统的碎片化严重，不少设备都提供或者可以轻易获得 Root 权限，那么攻击者或恶意软件就可以很容易的进行 Binder 通信数据中敏感信息的窃取和篡改。

因此，对于安全性要求较高的用户和特定应用场景，Android 这种基于 Binder 的 IPC 通信机制在安全方面存在一些不足，急需一种灵活的安全增强方式来保证用户的隐私和系统安全。

2.1.2 作品意义

基于以上背景，本小组开发了一套面向 Android 系统内部敏感信息传递过程的安全保护系统。该系统从 Android 系统的 Binder 机制出发，在不对系统进行修改的前提下，设计并实现了从敏感信息的产生到使用整个过程的透明加密方案，能够有效地保护用户的数据不被窃取、篡改。并且，本作品能以友好、灵活的方式向用户提供系统当前的保护状况，保护时间，达到良好的用户体验。

2.2 作品特色

当今的 Android 恶意软件防护技术已经难以应对日益严重的安全威胁，需要开拓思路，寻找更实用的技术手段，来保护用户的隐私数据。基于上述考虑，我们深入研究了 Android 系统中敏感数据的传递和处理流程，总结如下：

在 Binder 通信过程中，必然有对敏感数据进行处理并发送的功能模块，通过保护此模块中的敏感数据，就能够实现对整个 Binder 通信流程中用户隐私的保护，而这也是本作品关注的重点。

因此，本作品的核心设计思想主要有以下几点：

1) 在敏感数据的产生位置和接收位置分别进行数据的加密和解密，从而保证敏感数据在传输过程中的安全性。
2) 整个保护过程对用户透明。无需用户进行额外的操作、不会影响用户正常的使用。
3) 无需对已有应用程序和 Android 系统进行更改，最大程度的保证了作品的可用性和应用范围。

与现有技术和产品相比，本作品的特色表现在以下几个方面：

1) 暂无针对 Android 系统内部敏感信息传递过程的保护方案，本作品具有创新性。
2) 着眼于系统 IPC 通信机制的防护方式，加强了 Binder 敏感数据的安全性，有效的抵御了不法分子基于 Android 系统 IPC 通信的攻击。
3) 为用户提供了一种全新的安全防护方式，也弥补了当今市场上在 Android 系统中隐私保护方法上的缺陷。

2.3 应用前景

本作品创新性的为 Android 系统中基于 Binder 的敏感信息传递提供了一种可选的防护方法，拥有广阔的应用市场和良好的应用前景，适用于企业、政府相关部门以及个人用户：

1) 大型企业和政府部门分别需要保护商业机密和政府机密信息，需要保证工作人员智能手机的安全性。企业、政府等安全性要求较高的机构可以通过本作品构建一种强制性的安全增强型策略，减少因恶意软件造成的机密信息外泄。
2) 本作品向用户提供了简单、灵活的操作体验，使得用户可独立自主地对指定应用程序中特定类型的敏感数据进行保护，通过本作品，用户可以根据自身需求

打造个性化的保护方案，减少因恶意软件造成的隐私泄露。并且整个保护过程对用户透明,无需用户进行额外的操作,不会影响用户正常的使用。

第三章 实现方案

为了灵活地保护敏感信息在 Android 系统内部传递过程，本小组设计了一款应用软件，其核心思想是在敏感信息产生的地方进行加密，只允许拥有解密权限的应用解密，并且让用户能够自由选择要保护的敏感信息类型以及选择授予应用软件权限，以保证用户隐私。本章将对软件系统的详细设计和实现进行介绍。

3.1系统总体介绍

3.1.1 软件功能

本程序界面如图 3-1 所示，目前本程序主要功能有：

1） 将当前的保护信息以图表的形式展现给用户。

2） 根据用户的需求独立自主地对指定应用程序中特定类型的敏感数据进行保护。

图 3-1 程序主界面图

其中点击下拉菜单可以设定开启或关闭程序防护功能。

3.1.2 基本模块

本作品程序负责监控 Android 手机的基本安全情况，根据用户输入开启相应的保护模块，并对整个系统的保护状态进行反馈与评估。整个系统分如图 3-1 所示，分为

四个模块：**数据呈现模块，基本信息获取模块，命令接收模块，敏感数据保护模块**。其中，敏感数据保护模块又分为**敏感数据加密模块，敏感数据解密模块**两个子模块。

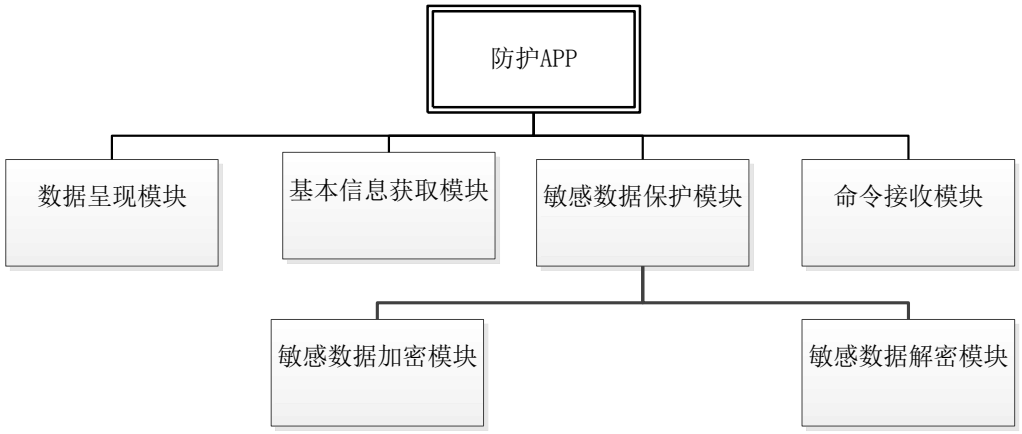


图 3-1 系统基本模块图

程序各模块功能的详细说明分别如表 1 所示。

表 1 程序各模块功能说明

模块名称		功能说明
基本信息获取模块		读取手机的软硬件信息和当前系统状态信息，如程序的权限，Android 系统的版本，程序的图标等
敏感数据保护模块	敏感数据解密模块	拦截并解密相应的数据包
	敏感数据加密模块	拦截并加密相应的数据包
命令接收模块		接收并处理用户所发送的命令
数据呈现模块		将程序的保护记录以图表的方式呈现给用户

3.2系统原理与总体设计

3.2.1 工作流程

3.2.1.1 工作流程结构图

图 3-2 描述了本软件系统的总体工作流程。

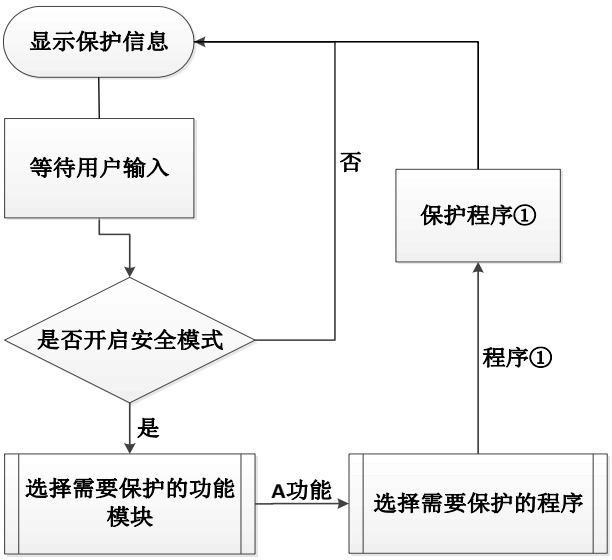


图 3-2 系统的工作流程图

具体流程说明如下：

1) 程序启动，获取手机基本状态并显示当前保护信息。
2) 等待用户输入。
3) 若已经开启保护模式，判断当前需要保护的功能模块。
4) 当需要保护的功能模块确定后，判断需要保护的 APP。
5) 判断成功后，对手机的相应程序功能实施保护方案并反馈保护结果。

3.2.1.2 保护相应的隐私数据

图 3-3 描述了数据保护模块的工作流程，其中功能 A，功能 B 代表了 Android 系统中的服务，例如短信服务，GPS 定位服务等。而相应的程序①，程序②代表的是调用相应服务的程序，例如信息 APP，百度地图等。

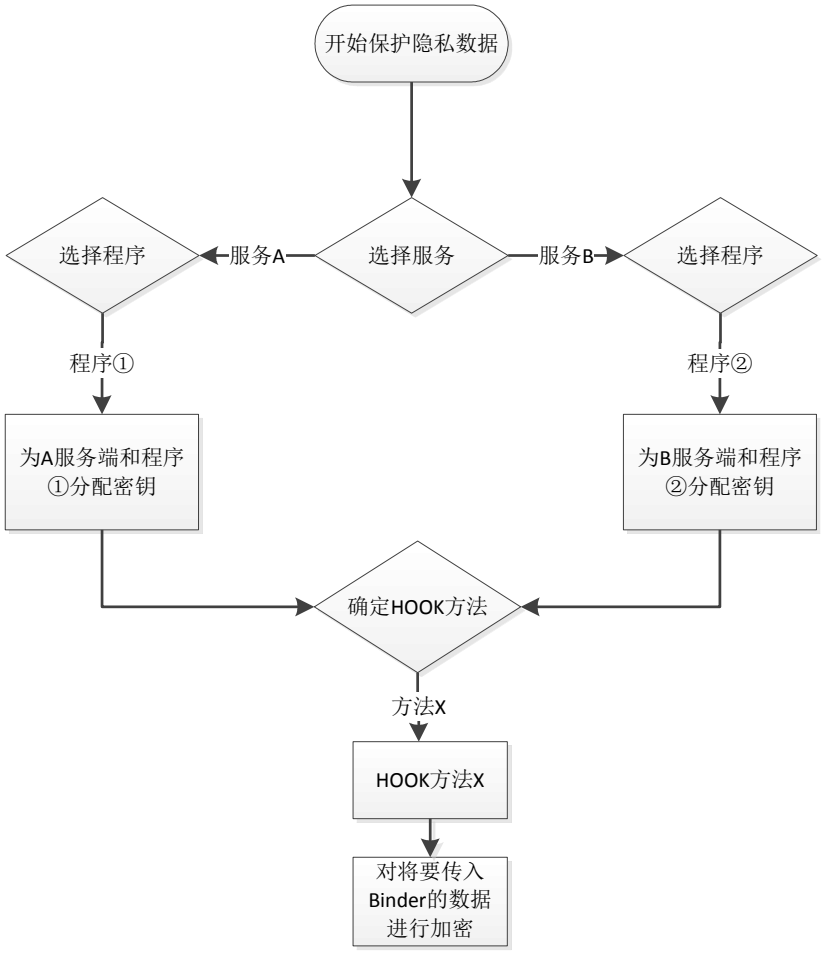


图 3-3 数据保护模块的工作流程图

本作品保护隐私数据并反馈结果的相关步骤如下：

1) 开启保护功能，判断需要保护的功能模块。
2) 判断需要保护的 APP。
3) 为相应服务的 Server 端和需要保护的 APP 分配密钥。
4) 根据保护的功能模块选择相应的 HOOK 方法保护 Binder 数据包中的敏感数据。
5) 调用 SMS4 加解密模块对敏感数据进行处理。

3.2.2 工作原理

本作品的系统原理如下：在 Binder 通信过程中，可以将通信双方分为服务端和客户端，服务端由各个服务管理者（service manager）构成，提供各种基本服务，如短信、GPS 等，而客户端是需要通过 Binder 向服务端请求服务的应用软件或进程。下图中最左侧的方框表示服务端，最右侧表示客户端。本作品的设计的核心思想，即为在服务端产生数据的时候加密，在有权限的客户端获取数据时进行解密，如图 3-4 所示，用户通过选择模块确定将要保护的服务类型，并通知加密模块对相应服务端产生的敏感数据进行加密，并且，用户通过选择模块选择可以信赖的应用软件，并通知解密模块对相应的客户端拿到数据时进行解密。作品原理图见下图。

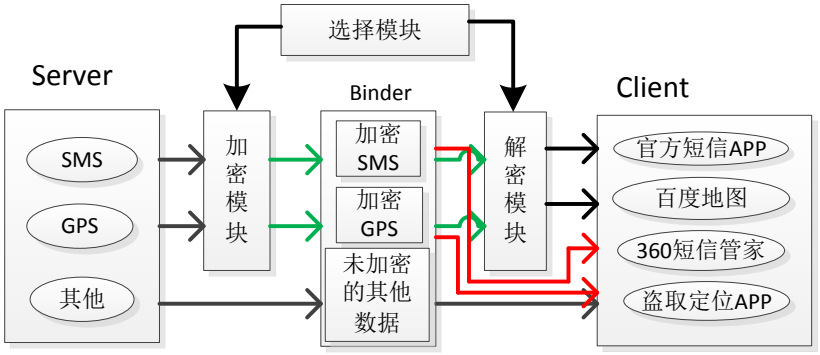


图 3-4 系统工作原理图

图 3-4 中，以短信、GPS 为例，短信和 GPS 服务作为被保护的服务类型，加密模块拦截了短信和 GPS 服务端发出的数据，进行加密后，再将加密数据给 Binder ，在 Binder 中传递的即为加密的数据。当具有权限的应用软件（如上图中的官方短信、百度地图）请求 Binder 数据传输时，对其进行解密，保证有权限的应用软件可以得到明文信息。而不具有权限的应用即使向 Binder 请求，或者恶意攻击 Binder 的应用软件（如盗取信息定位 APP）得到的将是密文。上图的模型可以适用于：应用软件向 Binder 请求有关敏感数据的过程，例如，请求备忘录，查看通话记录等。SMS 和 GPS 的信息保护只是冰山一角。

在通过 Binder 传输过程中，还有一种数据流向是从应用软件传递给相应的服务管理者，如利用短信应用编辑短信发送的过程，在这种信息传递模型中，服务端是应用软件，客户端是服务管理者。如下图所示：

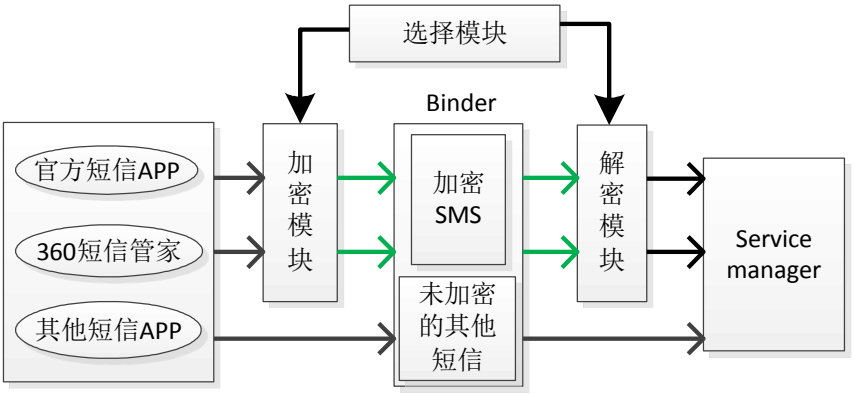


图 3-5 短信保护方案工作原理图

图 3-5 中，官方短信 APP 和 360 手机管家是受保护的应用软件，因此，加密模块对这两种应用软件所发送的敏感数据进行加密，使之在 Binder 中以密文传输，最后，在 ServiceManager 获取发送时解密。上图中的模型还适用于应用软件拨打电话以及备忘录写入等其他由应用软件向 Binder 传递隐秘信息的过程。

3.3作品实现的技术基础分析

经过对Android系统各层次能够完成的功能进行评估发现，整个系统的功能实现均能在JAVA层实现。在整个系统中，数据的保护是通过API Hook劫持方法调用实现的，其本质是对目标方法进行重定向，在执行目标方法前进行一些额外的操作。虽然Android并没有提供与Hook操作相关的方法，但是通过Xposed框架能共完成相同的功能。

本作品使用的关键技术有：对相关函数的拦截技术采用API Hook技术；数据加密解密采用SMS4加解密算法；攻击测试方案中对Binder IPC数据拦截采用注入和HOOK技术。

3.3.1 Binder 数据传输过程安全性分析

通过分析 Binder 的数据流程，找出了 Binder 的脆弱点并且论证脆弱点的易被攻击性。

一、 Binder 机制研究

Binder 机制抽象的模型如图 3-6 所示：

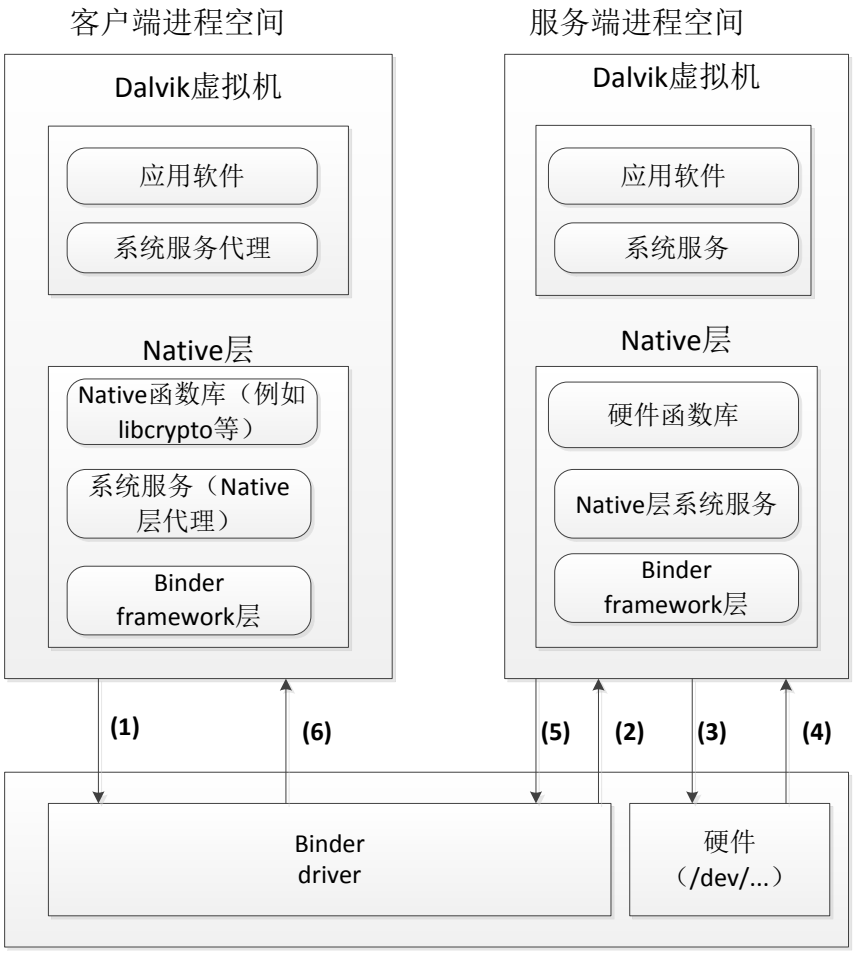


图 3-6 Binder 基本结构图

具体说明如下：

1) binder 框架层创建一个 ioctl syscall， 其中有文件描述作为参数， 以及相关数据传入内核 。
2) binder driver 寻找对应的 服务，把数据复制到 server 的空间，创立线程等待处理 。
3) server 响应服务，向相关 硬件提交请求 。
4) 硬件回应请求 。
5) 将回复传递给 driver。
6) 将回复传递给客户端进程。

以上的模型是客户端的数据流向 Binder，例如用短信 APP 编写短信这种情况，在 1)过程中，短信 APP 进程会把创建一个 ioctl syscall，将有关的数据（发送地址，短信信息）传入 BinderDriver。这一过程中，数据是以原始的明文传输，即恶意软件只要拦截 1)过程就可以获得相关信息。并且这一恶意行为可以针对几乎所有的 Binder 通信数据，因为在上图模型中可以看到，每个进程都是通过 ioctl 与 BinderDriver 进行通信，只要拦截了 ioctl，短信发送，获取联系人信息，获取电话通话记录等窃取用户敏感信息的目的便可以轻松达到。

3.3.2 API Hook

通过 API Hook 可以改变方法的行为，进而影响进程的行为，其本质是将目标方法重定向到其他方法，进行一些额外操作后再跳回目标方法。在课题中，通过使用 API Hook 技术对关键的函数调用拦截，从而实现对敏感数据的加密、解密。考虑到 Java 中函数和共享库中函数的拦截方法完全不同，接下来将分别对这两类 API 的拦截方法进行介绍。

3.3.2.1 基于 Xposed 框架的 HOOK 方式

Xposed 框架介绍

Xposed 框架是一款 Android 系统辅助工具，可以在不修改 APK 的情况下影响程序运行（修改系统）的框架服务，基于它可以制作出许多功能强大的模块，完成 Android 系统权限管理、内存清理、电池控制、界面修改显示等功能，并且能够互不冲突地运作。

Xposed 模块及相关方法介绍

在 Android 系统环境中对 Xposed 框架进行安装后，开发者可以通过编写 Xposed 模块的方式使用 Xposed 框架的接口。Xposed 模块设定了部分特殊数据标志位，因此需要在应用程序的工程中进行相应的配置。

1) Xposed 模块编写配置

具体应用实现中，需要在 AndroidManifest.xml 文件中添加如下内容：

```
<meta-data    Android:name="xposedmodule"    Android:value="true" />
```

同时需要将 XposedBridge.jar 包导入工程，并在 assets 目录下新建“xposed_init”文件，在文件中说明需要加载到 XposedInstaller 的入口类。完成相应的配置后，即可按照开发一般 Android 应用程序的方式进行开发，开发完成后在有 Xposed 框架的

Android 环境中安装，安装完成后激活相应的应用并重启，此模块就会依赖于 Xposed 框架正常运行。

2) Xposed 接口方法介绍

Xposed提供了丰富的接口方法以便开发者调用，下面对这些较常用的方法进行说明。

IXposedHookLoadPackage中的handleLoadPackage方法主要用于加载应用程序包时执行用户的操作。类似于一个统一调度的Dispatch例程，在Xposed源码中，其对应的C++方法是xposedCallHandler。其方法原型如下：

```
public void handleHookedMethod(final LoadPackageParam lpparam)
```

其中参数lpparam是所加载的应用程序的基本信息，包括packageName、processName、APPInfo等。

XposedHelpers类中findAndHookedMethod方法主要用于寻找需要进行hook的类及其相应的方法，并在原方法调用前或调用后运行自定义的替换方法。其方法原型如下：

```
findAndHookMethod(
String className,//需要进行 hook 的类名
ClassLoader classLoader,//类加载器，可以为空
String methodName,// 需要进行 hook 的方法名
Object... parameterTypesAndCallback//参数集，目标方法的参数类及回调
)
```

其中，回调分为如下两种：

XC_MethodHook:在目标方法执行前/后运行相应的替换方法；

XC_MethodReplace:完全替换目标方法，执行用户自定义的新方法。

在findAndHookedMethod方法的XC_MethodHook回调中，有beforeHookedMethod和afterHookedMethod两种方法，其方法原型如下：

```
beforeHookedMethod(
MethodHookParam param
)
```

该方法在hook目标方法执行前调用，其中，参数param指的是目标方法的相关参数、回调、方法等信息。

```
afterHookedMethod(
MethodHookParam param
)
```

该方法在hook目标方法执行后调用，其中，参数param指的是目标方法的相关参数、回调、方法等信息。

Xposed运行多个模块对同一个方法进行hook时，框架就会根据Xposed模块的优先级来排序，在具有a.before、a.after、b.before、b.after的情况下，运行的先后顺序如下：

```
a.before -> b.before ->originalMethod -> b.after -> a.after
```

XposedBridge类中hookAllMethods和log方法主要用于一次hook每个类的所有方法或构造方法。其原型如下：

```
hookAllMethods(
Class<?> hookClass,//需要进行 hook 的类
String methodName,//需要进行 hook 的方法名
XC_MethodHook callback//回调方法
)
```

3.3.2.2 基于 ptrace 系统调用的 SO 库注入的 HOOK 方式

一、HOOK 的原理介绍

图 3-7 和图 3-8 以 Ioctl()方法为例，对比了 Hook 前后方法的调用流程，展示了 Hook 的基本原理。

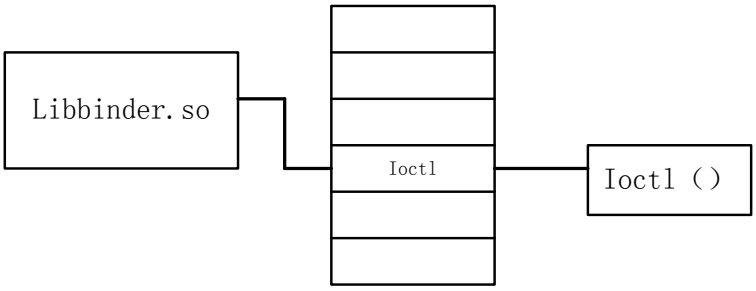


图 3-7 Ioctl 正常的调用流程

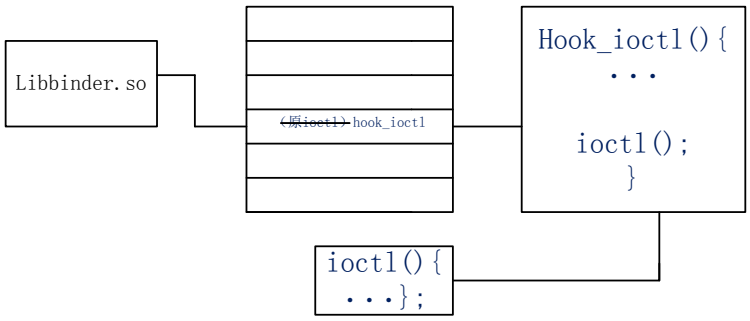


图 3-8 Hook 后 Ioctl 的调用流程

如图 3-7 所示，正常调用流程是，Libbinder.so 通过其 GOT 表中的 Ioctl 地址调用 Ioctl 方法，被 Hook 之后，GOT 表中填入新的 Ioctl 地址，Libbinder.so 会在 ioctl()执行前执行 hooked_ioctl 方法。

二、SO 库介绍

SO库是Linux下的一种共享库，提供各种功能方法的集合，对外提供标准的接口。可动态加载。类似与Windows下的dll文件。进程在运行时加载相应的so库。

基于shellcode针对SO库进行注入的原理

1) 编写 shellcode，shellcode 是使用汇编语言写一段汇编程序，该程序实现 SO 库的加载、SO 库方法查找以及执行库中的方法。

2) 通过远程进程 pid，ATTACH 到远程进程。

3) 获取远程进程寄存器值，并保存，以便注入完成后恢复进程原有状态。

4) 获取远程进程系统调用 mmap、dlopen、dlsym 调用地址。

5) 调用远程进程 mmap 分配一段存储空间，并在空间中写入 shellcode、SO 库路径以及方法调用参数。

6) 执行远程进程 shellcode 代码，实现 SO 库的加载、SO 库方法查找以及执行库中的方法。（在此方法库中可以实现想要的方法功能）。

7) 恢复远程进程寄存器。

8) detach 远程进程。

3.3.3 SMS4 加解密算法

在本作品中采用 SMS4 的加解密方法对敏感数据进行保护，需要注意的是在数据处理过程中，需要对 Android 系统的服务端和保护的应用程序端分别发放密钥，其方式描述如下：

在本作品中，是通过Xposed框架HOOK特定方法来实现对用户数据的保护的，因此，在HOOK的同时，可以将密钥以参数的形式传入需要HOOK的方法中，以此来实现密钥的发放与管理。

另外，对于在数据保护过程的密钥的选择，方式如下：

在本作品中，我们通过对敏感数据进行 hash 运算，以运算结果作为密钥，这种方式可以保证密钥的随机性，抵御攻击者字典攻击等基于统计学的攻击方式。

3.3.4 短信保护方案的实现的技术基础分析

3.3.4.1 短信有关类的介绍

对于 Android 操作系统中，关于短信应用程序，从软件的功能角度来讲，短信分为**对话列表，消息列表，短信编辑，彩信编辑，短信显示**和**配置**。从实现的角度来看，它分为 **GUI 展示层，发送/接收，信息数据等**，这些分类对应着源码中的各种包。在 Android 操作系统中，与短信相关的源码放在 com.android.mms 包中。下面主要介绍在 com.android.mms 包中与短信有关的类和包的功能。

对于 com.android.mms.ui 包，主要负责系统直接与用户交互，用于 GUI 的显示，如展示对话列表，消息列表，消息编辑页，等。

对于 com.android.mms.data 包，主要用于操作当前正在编辑的信息的相关数据，比如联系人列表，当前对话，当前消息等。该包中的类都是在编辑信息的时候使用，短信产生的最源头的类也在这个包中。部分类的功能的介绍如下表：

类	功能
WorkingMessage.java	用来管理当前正在编辑的消息，它从创建，草稿到发送完成后一直存在，只要打开了编辑信息的页面就会创建一个 WorkingMessage，直到退出编辑页面。
Conversation.java	用来管理对话 Threads，通常用来管理当前的对话，也就是进入的对话和正在进行操作的对话，它也用来管理对

	话列表，比如查询对话列表。
Contact.java	用来代表一个联系人的信息，和管理联系人，加载联系人信息，其中还有相应的 Cache 信息。
ContactList.java	Contact 的 List 列表，继承自 ArrayList<Contact>，用来管理一个 Contact 列表，或管理多个 Contact。
RecipientIdCache.java	用于保存所用到的 Contact 的 Id 和地址（电话）。每次 WorkingMessage 会更新这个 Cache，然后 ContactList 会优先从这个 Cache 中查询联络人。

在上表中需要重点关注的类是 WorkingMessage，该类中封装了代表短信内容的属性变量和相关的处理方法，可以认为是 Framework 层上短信产生的源头。

对于包 com.android.mms.dom，是解析短信内容的工具包；

对于包 com.android.mms.drm，是用于处理 DRM 媒体文件的工具包；

对于包 com.android.mms.layout，包含了短信布局方式的布局元素；

对于包 com.android.mms.util，包含了整个 MMS 共享的工具类。

对于包 com.android.mms.transaction，是有关 MMS 最底层的一个包，用户不可见，主要是负责发送信息和接收信息，用于发送信息的最后处理和接收信息的最初处理。从物理层次上来讲，并不是真正的发送和接收信息，而是由 Android 系统 Framework 层来负责接收和发送信息。

在系统接收端，部分类的功能介绍如下表：

类	功能
MmsMessageSender.java	继承自 MessageSender，用于发送彩信。它并不是做发送的事情，而是做一些错误检查和前期准备工作，然后启动 TransactionService 来做发送相关的事情。
MessageSender.java	为了发送信息而封装的一个接口，它里面只有一个方法 sendMessage()，UI 层只需要调用实现了这个接口的类即可发送信息。
SmsMessageSender.java	发送短信的封装，继承自 MessageSender，它会启动 SmsReceiverService 来发送短信。
SmsSingleRecipientSender.java	继承自 SmsMessageSender，它针对一个收信人，调用 Frameworks 层的接口发送信息，对于 Mms 应用来说，这是发送短信的最后一站，对就是说对于应用来说，它会把短信发送出去。
Transaction.java	各种 Transaction 的基类，定义了两个方法，getPdu() 和 sendPdu()，从 MMSC 取数据，和向 MMSC 发送数据。

在上表中需要重点关注的类是 SmsSingleRecipientSender 和 SmsMessageSender，SmsSingleRecipientSender 是短信发送的最后一站，对于 Android 系统的应用来说，可以会利用该类来实现短信的发送，因此要想在 Server 端进行拦截，应该考虑 HOOK 该类本身或者与该类有关的类。

3.3.4.2 短信的发送流程

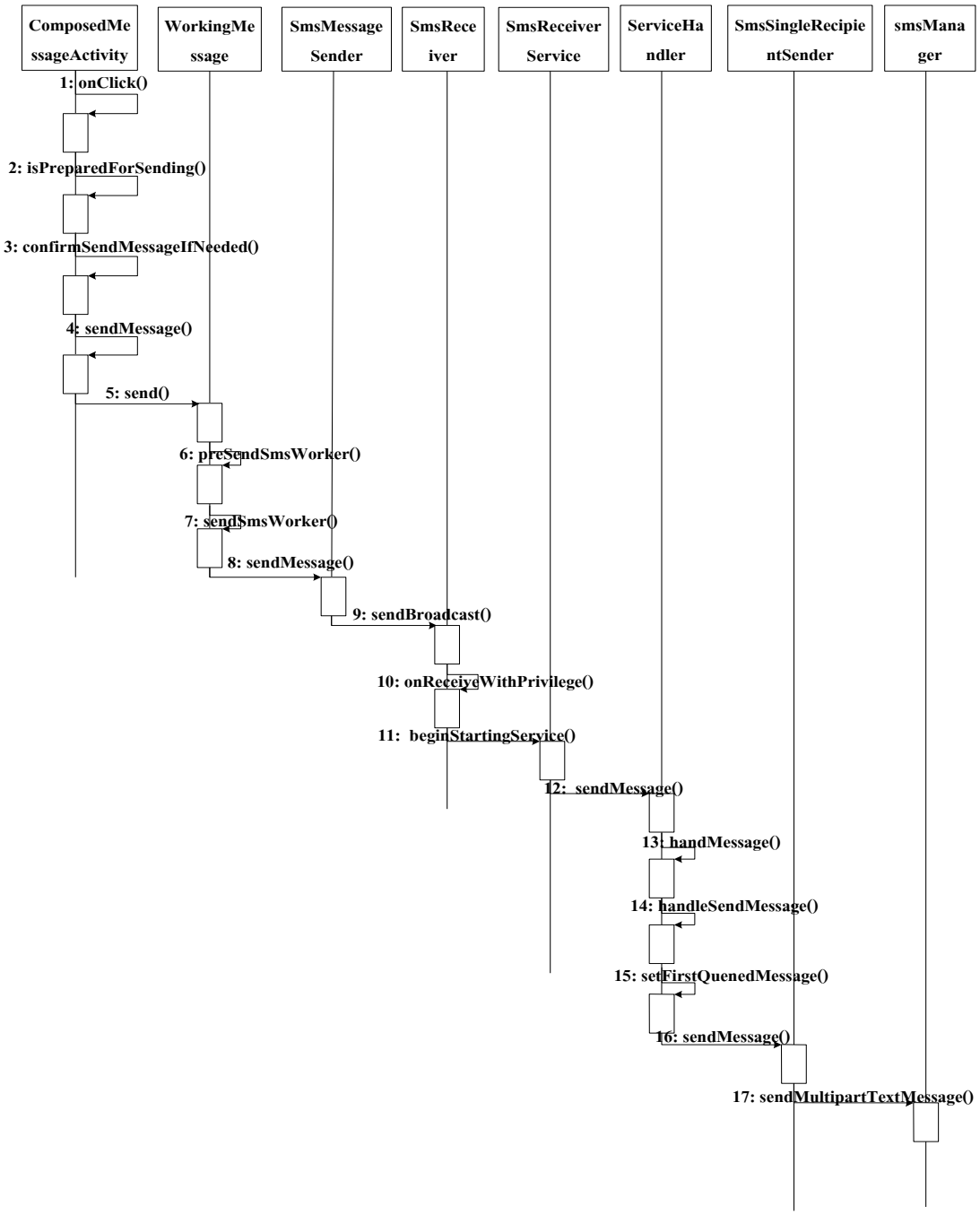


图 3-9 短信的发送流程

Android 系统下短信的发送流程如图 3-9 所示，具体如下：

1) 当用户点击发送后，系统会调用 WorkingMessage 对需要发送的短信进行处理。

2) 在刷新联系人之后，它就会创建一个 SmsMessageSender 对收件人地址进行分析（判断是否为群发短信）。

3) 根据发送地址的不同将短信以不同方式插入一个待发送的队列(也就是写进一个数据库）。

4) 发送 Intent 来唤起 SmsReceiverService。注意到这里的 SmsRecevierService 就是短信（SMS）处理的 Service。

通过以上的分析，不难得出结论：需要保护的部分就在 WorkingMessage 和 SmsReceveierService 之间。

3.3.4.3 短信的接收流程

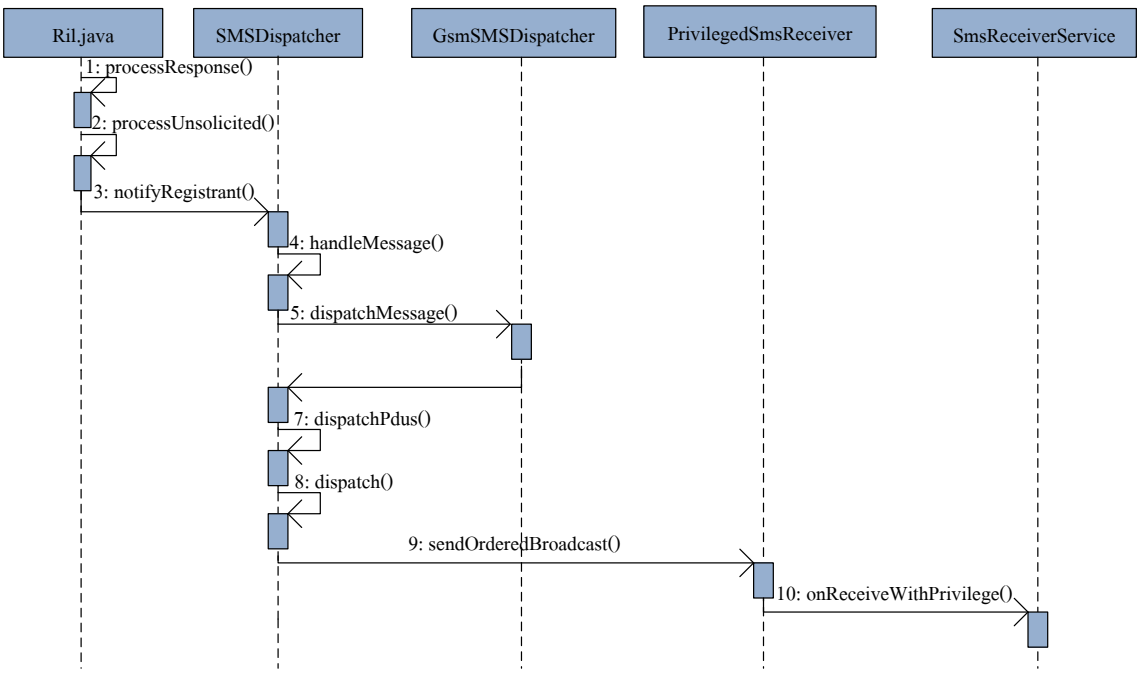


图 3-10 短信的接收流程

通过阅读上图将 Android 系统下短信发送流程总结如下：

1) 在系统的底层，会启动一个 RILReceiver 的线程来对本地 Socket 进行监听。
2) 当读取到数据后，就会调用 processResponse (Parcel p)对读取的数据进行判断（判断是为短信数据、电话数据或是其他）。
3) 在经过一系列数据处理之后，交给 GsmSMSDispatcher 的 dispatchPdus(Pdus)方法进行处理。注意到这里的 Pdus 字段即为此时的短信数据，而这个方法会将短信数据通过 Broadcast 的方式传送出去。
4) 上层的应用程序通过接收器 PrivilegedSmsReceiver 就能获取到这个广播并进行相应的处理。

通过以上的分析，不难得出结论：需要保护的部分就在 GsmSMSDispatcher 和应用程序的 PrivilegedSmsReceiver 之间。之后将针对具体的代码分析保护方案。

3.3.5 GPS 保护方案实现的技术基础分析

3.3.5.1 GPS 有关类的介绍

LocationMangerService：该类它就是 Android 系统中的 Server 所在的类，在系统中有关 GPS 的服务都是通过它来完成的。

LocationManager：应用层获取 Server 的接口，的应用程序正是通过它才能向服务端提出各种请求。

Location：该类中封装了地址信息，也就是用户隐私数据所在的地方，这一点在后文中还会继续提到。

LocationProvider：它代表的是位置提供者，可以为应用程序提供位置信息。在 Android 系统中一共有两种，分别是 GpsLocationProvider 和 NetworkLocationProvider 它们的区别如下：

1) **GpsLocationProvider**：GPS 模式，精度比较高，但是慢而且消耗电力，而且可能因为天气原因或者障碍物而无法获取卫星信息，另外设备可能没有 GPS 模块。
2) **NetworkLocationProvider**：通过网络获取定位信息，精度低，耗电少，获取信息速度较快，不依赖 GPS 模块。

3.3.5.2 GPS 位置信息的存放方式

在 Android 系统中，系统通过位置提供者（LocationProvider）的回调方法，当 LocationProvider 接收到位置信息的变化时就会调用 LocationMangerService 的 reportLocation()，这个方法会促使 Server 端更新位置信息，并将其存放在一个 HashMap 中。

3.3.5.3 GPS 的工作流程

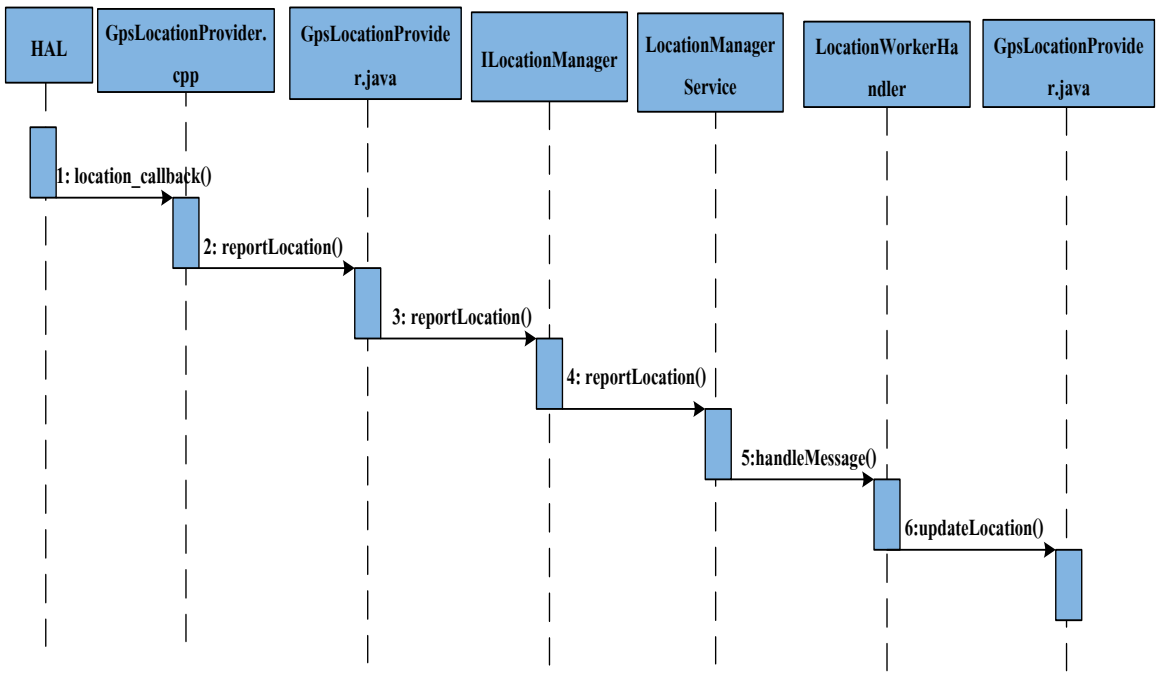


图 3-11 GPS 的工作流程

1) 当硬件检测到有位置更新之后，最初调用的是 GpsLocationProvider.cpp 中的 location_callback 函数。
2) location_callback 函数中对应的是调用 GpsLocationProvider.java 中的 reportLocation 方法。
3) GpsLocationProvider.java 中的 reportLocation 方法会调用 ILocationManager 的 reportLocation 方法，然后是调用 LocationManagerService 的 reportLocation 方法。
4) LocationManagerService 的 reportLocation 方法中会对 LocationWorkerHandler 发送消息 MESSAGE_LOCATION_CHANGED。该消息在 LocationWorkerHandler 的 handleMessage 方法中被处理。处理方法中调用 GpsLocationProvider 的 updateLocation 方法。

3.4短信的保护方案

3.4.1 短信的发送保护方案

通过 3.3.3 的分析，已经了解到了在短信发送保护方案中需要保护的部分就在 WorkingMessage 和 SmsReceveierService 之间，现在将针对具体的代码进行分析。

3.4.1.1 短信发送保护模块方法分析

方法：WorkingMessage.send()

```
public void send(final String recipientsInUI) {
        ...省略代码
    final Conversation conv = mConversation;\\建立会话
    String msgTxt = mText.toString(); \\此处的 msgTxt 中的内容即为短信内容
    if (requiresMms() || addressContainsEmailToMms(conv, msgTxt)) {
        ...省略代码}
```

图 3-12 发送加密的关键代码

从图 3-12 可以看出，对于短信的内容，系统将它赋值给了一个 String 类型的变量 msgText 之后将他作为参数传向下一个方法。分析以上代码可知，此方法将当前短信文本窗中的内容转化成 string 并赋值给 msgTxt，并将其作为参数传递给下一个方法,因此，需要将此方法中的 msgText 值加密，就可以保证短信内容在传递过程中以密文形式传递。

3.4.1.2 短信发送还原模块方法分析

在短信发送服务端 smsReceiverService 类中，在初始化的时候就创建了一个线程用于监听广播。

```
Public void onCreate(){
...省略代码
    thread.start();
    mServiceLooper = thread.getLooper();   \\创建线程用于监听广播
    mServiceHandler = new ServiceHandler(mServiceLooper);
}
```

当接收到广播过来的 Intent 时，线程将会对 Intent 的 Action 进行判断。

当监听到一个 Action 为 SmsReceiverService.ACTION_SEND_MESSAGE 的 Intent 时就代表着系统将要发送短信最后将会调用 ServiceHandler 的 handleSendMessage（）方法。

```
Public void handleMessage(Message msg) {
        Int serviceId = msg.arg1;
        Intent intent = (Intent)msg.obj;
        ...省略代码
```

```
                If (MESSAGE_SENT_ACTION.equals(intent.getAction()))
                    handleSmsSent(intent, error);
                    ...省略代码
Else if (ACTION_SEND_MESSAGE.endsWith(action)) {{\\通过此处判断进入发送短信处理模块

                handleSendMessage();
                    ...省略代码
}
```

hanleSendMessage()代码如下：

```
Private void handleSendMessage(){
            If (!mSending){
              sendFirstQueuedMessage();
}
}
```

如上所示，handleSendMessage 中调用了 sendFirstQueuedMessage()方法。其核心代码是创建一个 SmsSingleRecipientSender 对象，将 address(短信发送地址)，msgTest（短信内容），threadId（线程标号），status（是否需要发送报告），等作为参数。然后执行此对象的 sendMessage 方法，如下图所示。

```
Public synchronized void sendFirstQueuedMessage() {
                    ...省略代码
                    SmsMessageSender sender = new SmsSingleRecipientSender(this,
      address, msgText, threadId, status == Sms.STATUS_PENDING,  msgUri); \\以短信内容作为参数构建 Serder
{
if (LogTag.DEBUG_SEND ||  LogTag.VERBOSE ||  Log.isLoggable(LogTag.TRANSACTION, Log.VERBOSE)) {
                Log.v(TAG, "sendFirstQueuedMessage " + msgUri +
                  ", address: " + address +  ", threadId: " + threadId);
                  }
                    try{
sender.sendMessage(SendingProgressTokenManager.NO_TOKEN) \\调用 Sender 的 sendMessage 方法发送短信
                        ...省略代码}
```

整个流程最终的核心功能代码如下：

方法：SmsSingleRecipientSender.sendMessage()

```
SmsManager smsManager = SmsManager.getDefault();
        ArrayList<String> messages = null;
        if ((MmsConfig.getEmailGateway() != null) &&
                (Mms.isEmailAddress(mDest) || MessageUtils.isAlias(mDest))) {
            String msgText;
            msgText = mDest + " " + mMessageText;
            mDest = MmsConfig.getEmailGateway();
           messages = smsManager.divideMessage(msgText); \\如果短信内容大于 70 字节就将其分割
        } else
```

```
...(省略代码)
    smsManager.sendMultipartTextMessage(mDest, mServiceCenter, messages, sentIntents,
deliveryIntents); \\发送短信的最终方法
```

图 3-13 发送加密的关键代码

从图 3-13 可知，SmsSingleRecipientSender 在 sendMessage()方法中构造了一个 smsManager 并将 msgtext 作为参数传给了 SmsManager 的 sendMultipartTextMessage()从而调用发送短信功能，要保证能够正常的发送短信，需要做的就是拦截此处的 SmsSingleRecipientSender.sendMessage()并且对其中的 msgText 进行解密。这样，就能保证传递过程中是密文，发送时是明文，既能保证 Binder 传递过程用户隐私的安全，又保证了短信的正常工作。

3.4.2 短信的接收保护方案

通过 3.3.3 的分析，已经了解到了在短信接收保护方案中需要保护的部分就在 GsmSMSDispatcher 和应用程序的 PrivilegedSmsReceiver 之间，现在将针对具体的代码进行分析。

3.4.2.1 短信接收保护模块方法分析

在底层文件 RIL.java 开启的监听器通过 socket 监听获取到 Parcel 后，会对数据包进行判断，分辨它是短信，电话还是其他数据包，根据数据类型的不同，系统会将数据包向不同的上层处理函数传递，而对于接受短信来说，数据包类型为 RESPONSE_UNSOLICITED 类型，请求类型为 RIL_UNSOL_RESPONSE_NEW_SMS。具体的判断方法分别是 RIL.java 类的 processResponse(Parcel p)和 processUnsolicited (p)方法代码分别如下：

```
Private void
    processResponse (Parcel p) {
        Int type;
        Type = p.readInt();
        If (type == RESPONSE_UNSOLICITED){ \\通过此处判断，进入无应答服务的处理模块，
如短信等
            processUnsolicited(p);
}else if(type==RESPONSE_SOLICITED {
processSolicited(p);
}
releaseWakeLockIfDone();
}
```

由以上代码可知，接收短信过程中，由于数据包类型为 RESPONSE_UNSOLICITE，会转入方法 processUnsolicited(p)来处理，具体代码如下

```
processUnsolicited (Parcel p) {
```

```
            int response;
            Object ret;
            response = p.readInt();\\读取数据包的类型
          try {switch(response) {
libs/telephony/ril_unsol_commands.h \
                        ...省略代码
                case RIL_UNSOL_RESPONSE_NEW_SMS: ret =    responseString(p) ; \\判断数据是否为
短信，如果是则调用相应的数据处理方法
 break;
                        ...省略代码
case RIL_UNSOL_RESPONSE_NEW_SMS: {
                    if (RILJ_LOGD) unsljLog(response);
                    // FIXME this should move up a layer
                   String a[] = new String[2];
                    a[1] = (String)ret;
                    SmsMessage sms;
                    sms = SmsMessage.newFromCMT(a);
                    if (mGsmSmsRegistrant != null) {
                         mGsmSmsRegistrant
                              .notifyRegistrant(new AsyncResult(null, sms, null)); \\处理短信数据包完成
后，调用此方法进行下一步操作
                }
                break;
8                }
```

分析以上代码可知，短信 sms 对象传递给下一个方法为：mGsmSmsRegistrant.notifyRegistrant(new AsyncResult(null, sms, null))。追溯该方法可以发现该方法设置 handler 的源头是在 GsmSMSDispatcher 类中，因此，以上方法就等价于调用 GsmSMSDispatcher. handleMessage(Message msg) ，代码如下：

```
@Override
Public void handleMessage(Message msg) {
AsyncResult ar;
switch (msg.what) {
case EVENT_NEW_SMS:  \\判断短信为新接收到的短信
// A new SMS has been received by the device
if (Config.LOGD) {  \\判断此短信之前是否已经接受过
Log.d(TAG, "New SMS Message Received");
 }
SmsMessage sms;
 ar = (AsyncResult) msg.obj;
if (ar.exception != null) {
Log.e(TAG, "Exception processing incoming SMS. Exception:"
 + ar.exception);
 return;
```

```
}
sms = (SmsMessage) ar.result;
try {
 int result = dispatchMessage(sms.mWrappedSmsMessage); \\发送短信方法
                    ...省略代码
}
```

可以发现在这个方法中，通过 GsmSMSDispatcher.dispatchMessage(sms.mWrappedSmsMessage)对数据进行处理，而这个方法最终会调用它的 dispatchMessage 方法处理，在经过一系列判断之后，会将普通短信的数据交给 SMSDispatcher.dispatchpdus(byte[][] pdus)处理。

整个流程最终的核心功能代码如下。

方法：SMSDispatcher.dispatchpdus(byte[][] pdus)

```
protected void dispatchPdus(byte[][]pdus){
Intent intent = new Intent(Intents.SMS_RECEIVED_ACTION);
intent.putExtra("pdus", pdus); \\将短信数据加入 intent 里面
dispatch(intent, "android.permission.RECEIVE_SMS");
}
void dispatch(Intent intent, String permission) {
// Hold a wake lock for WAKE_LOCK_TIMEOUT seconds, enough to give any
// receivers time to take their own wake locks.
mWakeLock.acquire(WAKE_LOCK_TIMEOUT);
mContext.sendOrderedBroadcast(intent, permission, mResultReceiver,
this, Activity.RESULT_OK, null, null);  \\将 Intent 通过广播发送出去，等待上层应用程序处理
}
```

图 3-1 短信接收加密的关键代码

从图 3-14 可以看出，对于短信的内容，系统将它赋值给了一个 byte 类型的数组 pdus 之后通过 intent.putExtra("pdus”,pdus)将它赋值给了 intent，并且将其广播出去，为了保护接收到的短信需要在赋值给 intent 之前对 pdus 进行加密。

3.4.2.2 短信接收还原模块方法分析

当底层的方法对短信的数据处理完毕后，他就会发送一个 Intents.SMS_RECEIVED_ACTION 的广播，而对于应用层的 APP 来说要获取短信就必须创建一个 PrivilegedSmsReceiver 的接收器来接收广播，这个类继承自 SmsReceiver，其具体的接收代码在 SmsReceiver 中：

方法：SmsReceiver.onReceiveWithPrivilege（）

```
protected void onReceiveWithPrivilege(Context context, Intent intent, boolean privileged) {
        // If 'privileged' is false, it means that the intent was delivered to the base
        // no-permissions receiver class.    If we get an SMS_RECEIVED message that way, it
        // means someone has tried to spoof the message by delivering it outside the normal
        // permission-checked route, so we just ignore it.
        if (!privileged && intent.getAction().equals(Intents.SMS_RECEIVED_ACTION)) {\\判断广播类
```

```
型，是否为新接受的短信
                return;
            }
            intent.setClass(context, SmsReceiverService.class);
            intent.putExtra("result", getResultCode());
            beginStartingService(context, intent); \\启动 Service
        }
```

图 3-2 短信接收的解密部分

从图 3-15 可以看出，SmsReceiver 在 onReceiveWithPrivilege（）方法中接收到了 disPatchPdu()中的 Intent，也就是说此时的 Intent 的“pdus”字段及为短信数据，为保证接收到时能显示正常的短信，因此需要对整个 pdus 字段的数据进行解密。

3.5GPS 的保护方案

之前已经提到过在 Android 系统中，APP 是通过 LocationMangerService 来获取 GPS 服务的接口的，从而达到使用 GPS 服务的目的。之后通过 LocationProvider 来获得调用 Server 端的方法的接口，而 Server 端通过类 LocationMangerService 则会存储以及处理底层传来的 GPS 位置信息，进而为 GPS 程序提供位置信息。

从以上的分析，不难发现需要保护的方法就在 LocationMangerService 和应用程序 LocationMangerService 之间。

3.5.1 GPS 保护模块

之前已经提到了，在 Android 系统中，位置信息是存放在 HashMap<String,Location>中的，因此只需要在 Server 端将位置信息存放入 HashMap 之前将数据加密即可。而在 LocationManagerService 的 run()方法中就初始化了一个 LocationWorkerHandler 变量用来对位置信息进行处理，具体代码如下：

```
public void run()
{
Process.setThreadPriority(Process.THREAD_PRIORITY_BACKGROUND);
Looper.prepare();
mLocationHandler = new LocationWorkerHandler();\\创建一个 Handler 来处理 GPS 数据
initialize();
Looper.loop();
}
```

这里的 LocationWorkerHandler 是在 LocationManagerService 的一个内部类，它继承自 Handler 类，在它的 handleMessage()定义了将位置信息存放如 HashMap 的操作，具体代码如下：

方法： LocationWorkerHandler. handleMessage()

```
 public void handleMessage(Message msg) {
try {
if (msg.what == MESSAGE_LOCATION_CHANGED) {\\判断设备的地理位置是否发生变化
```

```
// log("LocationWorkerHandler: MESSAGE_LOCATION_CHANGED!");
 synchronized (mLock) {
Location location = (Location) msg.obj;
String provider = location.getProvider();  \\根据设备的情况获取 Provider
boolean passive = (msg.arg1 == 1);
  if (!passive) {\\判断此地址是否已经更新了
// notify other providers of the new location
for (int i = mProviders.size() - 1; i >= 0; i--) {
LocationProviderInterface p = mProviders.get(i);
if (!provider.equals(p.getName())) {
 p.updateLocation(location);  \\更新 HashMap 中的地理位置
}
```

图 3-16 GPS 加密的关键代码

从图 3-16 可知，GPS 的内容被赋值给了一个 Location 类型的变量 “location”，之后将 location 作为参数传向 provider.updateLocation()方法。要保护位置信息，需要对此处的 location 进行处理。这里的 LocationWorkerHandler 是在 LocationManagerService 的一个内部类，它继承自 Handler 类，其中更新位置信息的关键语句为 p.updateLocation(location)。整个方法的意义是在监听到位置信息变化时就会调用此方法，更新 HashMap 中的位置信息。需要做的也就是对此处的 Location 进行加密了。

3.5.2 GPS 还原模块

正如之前所说，在客户端中，是通过 LocationManager.类来获取 Server 端的 GPS 信息，其中用来更新位置信息的方法为 getLastKnownLocation(provider)其中的参数 provider 就是之前提到的 LocationProvider 类，通过它就能获取获得 Server 端方法的接口，进而获得位置信息。具体代码如下：

方法：LocationManager.getLastKnownLocation(provider)

```
    public Location getLastKnownLocation(String provider) {
        checkProvider(provider); \\检查是否有相应的 Provider
        String packageName = mContext.getPackageName();
        LocationRequest request = LocationRequest.createFromDeprecatedProvider(
                provider, 0, 0, true);
        try {
            return mService.getLastLocation(request, packageName); \\更新位置信息
        } catch (RemoteException e) {
            Log.e(TAG, "RemoteException", e);
            return null;
        }
    }
```

图 3-17 GPS 解密的关键代码

从图 3-17 可以看出，这个方法实现的功能很简单，就是根据 Provider 去从 Server 端的 HashMap 中提取出位置信息。因此需要做的也就是在此处完成解密功能即可。

第四章 功能和性能测试

4.1测试概述

本作品设计并实现了一个针对 Android 的 Binder 机制的敏感数据保护系统。

为了测试其是否满足本章将对如下三个方面进行测试：

1) **防护的有效性**：程序能有效保护敏感数据在系统内部传输过程中不被窃取与篡改，并以直观灵活的方式反馈保护结果，记录程序运行的各项信息。
2) **易用性**：作品的界面程序能够向用户提供简单、灵活的操作体验，使得用户可独立自主地对指定应用程序中特定类型的敏感数据进行保护,打造个性化防护方案
3) **程序稳定性**：程序运行稳定，性能良好。

4.2测试方案

根据以上测试目的，设计了如下测试方案。

4.2.1 防护功能测试

4.2.1.1 测试 1-1 GPS 保护方案的 HOOK

一、 **测试方法**

使用手机 A 运行本作品，开启 GPS 保护模块，显示保护效果并通过 Log 信息来反馈 HOOK 结果。

二、 **测试目的**

测试程序是否能完成注入，实现预期的 GPS 保护方案，从而保障了用户敏感数据在系统内部传输过程中的私密性及完整性。

三、 **预期结果**

程序成功 HOOK 了预期方法，完成了对 GPS 敏感数据在系统内部传输过程中的加密解密功能。另外，在开启 GPS 保护后，用户无需进行额外的操作即可正常使用 GPS 服务。

4.2.1.2 测试 1-2 短信保护方案的 HOOK

一、 **测试方法**

使用手机 A 运行本作品，开启短信保护模块，发送短信给 B 手机，显示发送效果并通过 log 信息反馈 HOOK 结果。

使用手机 A 运行本作品，开启短信保护模块，从 B 接受短信，显示接受效果并通过 Log 信息反馈 HOOK 结果。

二、 测试目的

测试程序是否能完成注入，实现预期的短信保护方案，从而保障了用户敏感数据在系统内部传输过程中的私密性及完整性。

三、 预期结果

程序成功 HOOK 了预期方法，完成了对短信敏感数据在系统内部传输过程中的加密解密功能。另外，在开启短信保护后，用户无需进行额外的操作即可正常使用短信服务。

4.2.1.3 测试 1-3 Binder 数据拦截测试(以短信为例)

一、 测试方法

使用基于 SO 库的注入方式在程序开启和关闭防护之后分别拦截相同的 Binder 数据包，比较防护开启前后的攻击效果。

二、 测试目的

判断防护方案能否有效的抵御了攻击者于 Android 系统 IPC 通信的攻击，保障用户敏感数据在系统内部传输过程中的私密性及完整性。

三、 预期结果

在程序对手机进行保护之前，不法分子可以获取到 Binder 敏感数据，但是在程序对手机进行保护之后，攻击者就无法获取 Binder 敏感数据。

4.2.2 基本功能测试

4.2.2.1 测试 1-4 应用管理功能测试

一、 测试方法

使用手机 A 安装本程序，点击按钮进入应用管理功能，尝试将程序添加/删除信任，并获取系统基本信息。

二、 测试目的

测试程序能否正确实现应用管理功能，引导用户完成防护方案的基本配置。

三、 预期结果

程序能实现预期的应用管理功能。包括显示，筛选系统程序。显示程序基本信息，以及将程序添加或剔除信任。

4.2.2.2 测试 1-5 日志记录功能测试

一、 测试方法

使用手机 A 安装本程序，点击按钮进入日志记录功能，查看当前的保护记录。

二、 测试目的

测试程序是否能够正确实现日志记录功能，记录用户的历史保护方案和基本保护信息。

三、 预期结果

程序能实现预期的日志记录功能,包括记录用户的历史保护方案和基本保护信息。

4.2.2.3 测试 1-6 计划任务计划任务功能测试

一、 测试方法

使用手机 A 安装本程序，点击按钮进入计划任务功能，设置系统定时计划。并查看相应日志记录。

二、 测试目的

测试程序是否能够正确实现计划任务功能。

三、 预期结果

程序能实现预期的计划任务功能，包括设定定时保护的功能。

4.2.2.4 测试 1-7 安全配置功能测试

一、 测试方法

使用手机 A 安装本程序，点击按钮进入安全配置功能，设置系统定时计划。并查看相应日志记录。

二、 测试目的

测试程序是否能够争取实现安全配置功能，管理当前的防护方案。

三、 预期结果

程序能实现预期的计划任务功能，包括管理当前已防护程序。

4.3测试环境

本作品的测试环境如表 4，表 5 所示

表 4 作品测试必要代码与程序

名称	主要功能
作品 APP	为用户提供保护
官方短信 APP	用于测试短信保护方案
百度地图 APP	用于测试 GPS 保护方案
基于 SO 库注入的恶意代码	用于检验保护方式是否真实有效

表 5 作品的测试环境

名称	主要配置
手机 A	型号：TCL S838M　　系统：Android 4.3 内核版本：3.4.0　　CPU: 1.2GHz 内存：2048M

手机 B	型号：m1 note　　系统：Android 4.4.4 内核版本：Flyme OS 4.20.5C　　CPU：1.7GHz 内存：4096M

4.4测试过程

4.4.1　防护功能测试

4.4.1.1 测试 1-1　GPS 保护方案的 HOOK

使用手机A运行本作品，开启GPS保护模块并对程序“百度地图”添加信赖如图4-1所示。

图 4-1 开启 GPS 保护

从上图可以看出，在这里已经将百度地图添加信任，这代表着保护程序之后会对百度地图的GPS功能进行实时的保护。

之后，打开此程序，发现它依旧能正常运行，运行截图如图4-2所示。

图 4-2GPS 程序正常运行截图

可以发现，程序被保护后依然能正常运行，整个保护过程实现了对用户的透明化。

最后使用 Xposed 的日志功能，查看 HOOK 结果。发现成功打印相应 Log 信息，如图 4-3 所示。

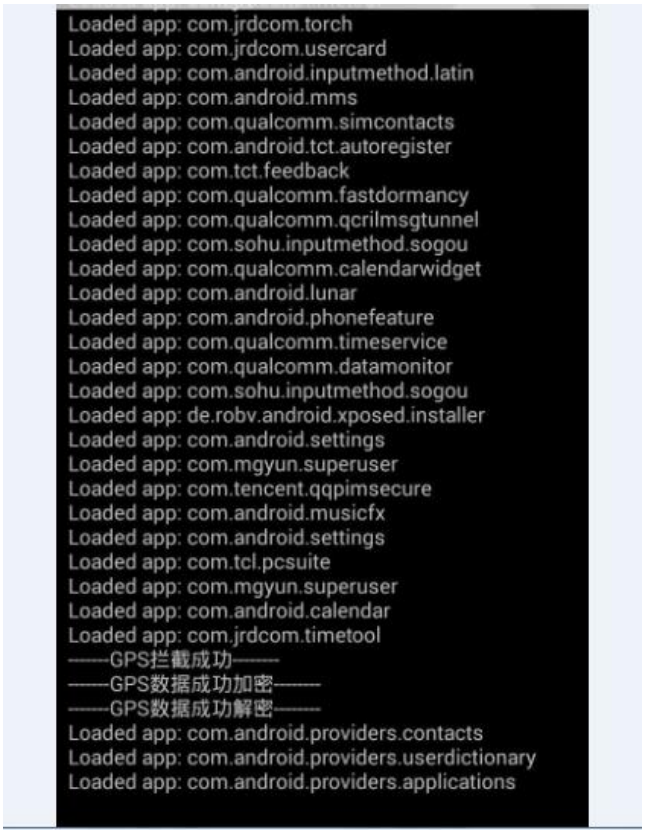


图 4-3 保护短信服务并打印 log 信息

从上图不难发现，对于程序的保护已经生效。而这一点将在最后的攻击测试中证明。

本项测试结果如表 6 所示。

表 6 测试 1-1 测试结果

编号	测试方法	测试目的	测试结果
测试 1-1	使用手机 A 运行本作品，开启 GPS 保护模块，通过 Log 信息来反馈 HOOK 结果。	测试程序是否能完成注入，实现预期的 GPS 保护方案，从而保护用户的隐私不被窃取与篡改。	程序成功 HOOK 了预期方法，完成了对 GPS 敏感数据在系统内部传输过程中的加密解密功能。另外，在开启 GPS 保护后，用户无需进行额外的操作即可正常使用 GPS 服务。

4.4.1.2 测试 1-2　短信保护方案的 HOOK

1)　使用手机 A 运行本作品，开启短信保护模块，如图 4-4 所示。

图 4-4 开启短信保护

从上图可以看出，将手机A的官方短信APP添加信任，保护了它的短信功能。

1) 使用手机 A 官方短信 APP 发送短信给 B 手机，发现能够正确的完成加解密功能，并且手机 B 能够正常接收到手机 A 发送的短信 如图 4-5，图 4-6 所示

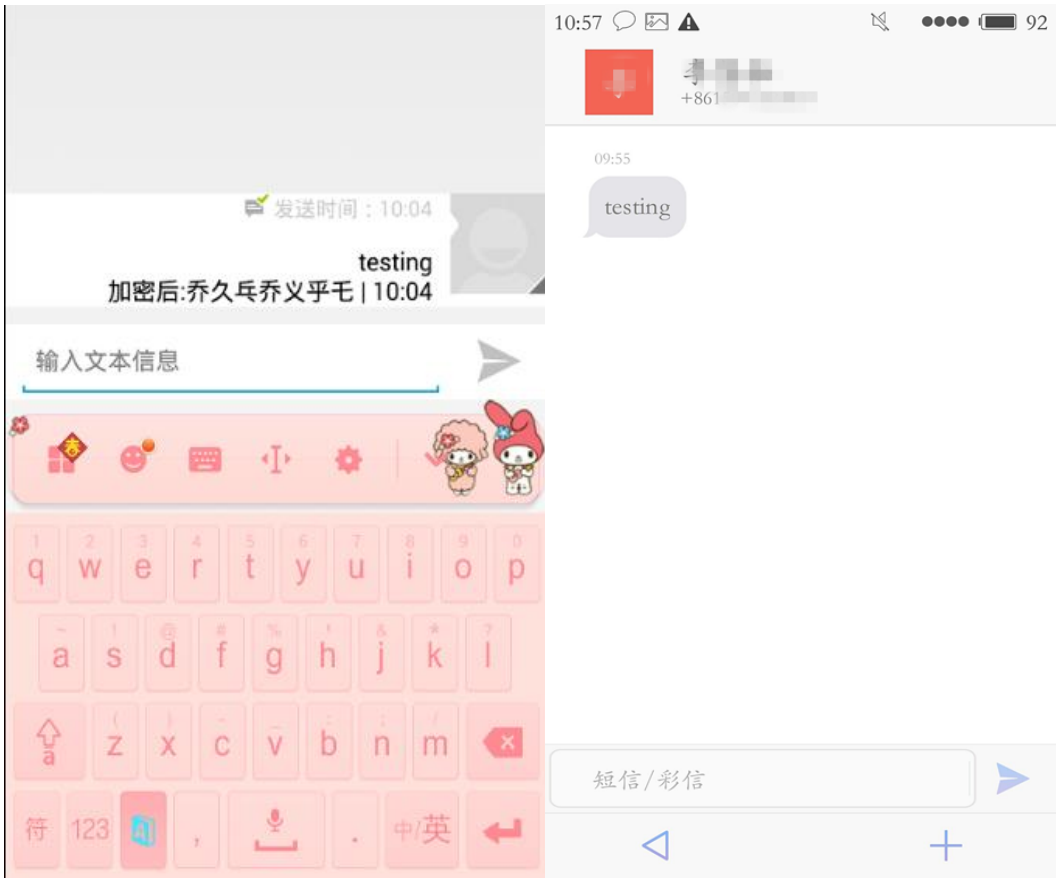
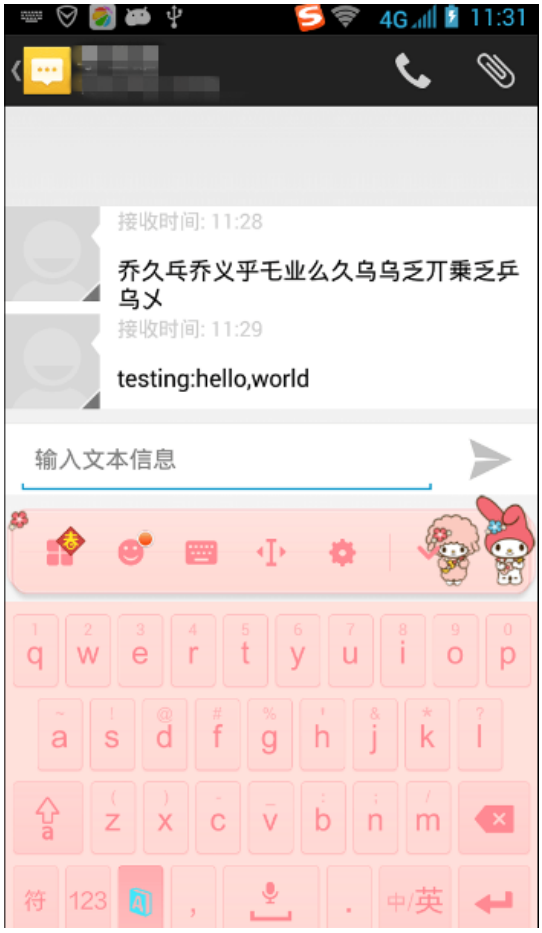


图 4-5 手机 A 在被保护后发送短信状况图　　图 4-6 手机 B 接收短信状况图

可以发现手机A在被保护后，对于用户来说发出的短信的操作依旧是正常的，并且系统还会为用户提供加密后的密文提示，而对于接收方手机B来说，接收短信的效果与接收正常短信是一模一样的。

2） 分别在对手机 A 的官方短信 APP 去除信任后和添加信任后，使用手机 B 向手机 A 发送短信，发现能够正确的完成预期功能，当开启短信保护功能时手机 A 的官方短信 APP 在没有被添加信任时接收的是密文，在被添加信任后接收的是正常短信。如图 4-7，图 4-8 所示。



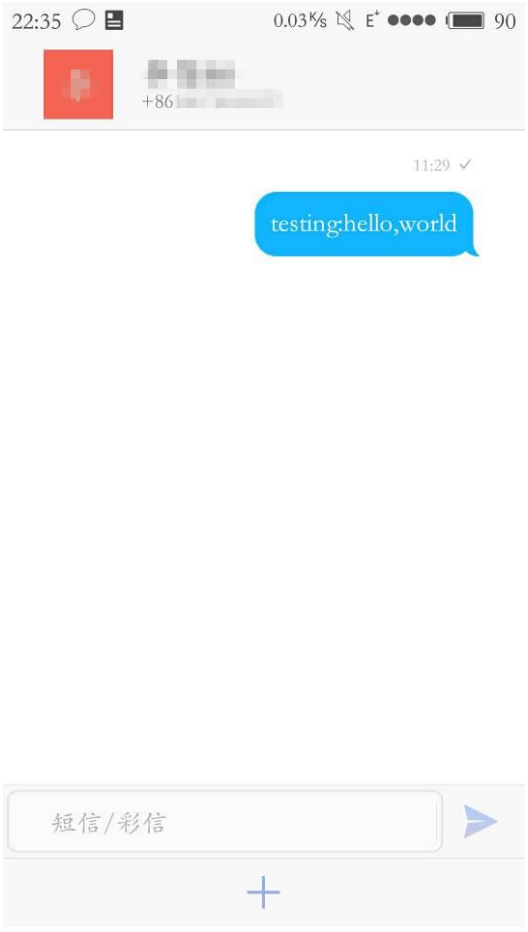


图 4-7 手机 A 官方短信 APP 保护前后短信接收状况　　图 4-8 手机 B 发送短信截图

可以发现在对手机 A 的短信去除信任和添加信任后，手机 A 收到的分别是密文和明文，需要说明的是在此处当将官方短信 APP 去除信任后并不代表程序就停止对短信服务的保护了。事实上，当将官方短信 APP 剔除信任后，程序依然在保护手机的短信服务，在这种情况下，只有添加了信任的程序才能获得正确的短信内容，这也是为什么此 APP 在第一次接收短信时收到的是密文。只有当用户在主界面关闭对短信的保护功能时，所有的短信 APP 才能收到明文。

3） 最后使用 Xposed 的日志功能，查看 HOOK 结果。发现成功打印相应 Log 信息，如图 4-9 所示。

图 4-9 保护短信服务并打印 log 信息

本项测试结果如表 7 所示。

表 7 测试 1-2 测试结果

编号	测试方法	测试目的	测试结果
测试 1-2	使用手机 A 运行本作品，开启短信保护模块，通过 log 信息反馈 HOOK 结果	测试程序是否能完成注入，实现预期的短信保护方案，从而保护用户的隐私不被窃取与篡改。	程序成功 HOOK 预期方法，并打印出相应 log 信息。另外，在开启短信保护后，用户无需进行额外的操作即可正常使用短信服务。

4.4.1.3 测试 1-3 Binder 数据拦截测试

现在以攻击短信为例，验证保护方案是否能有效的保护用户的安全。

1) 首先使用手机A的短信APP向手机B发送短信，如图4-10所示

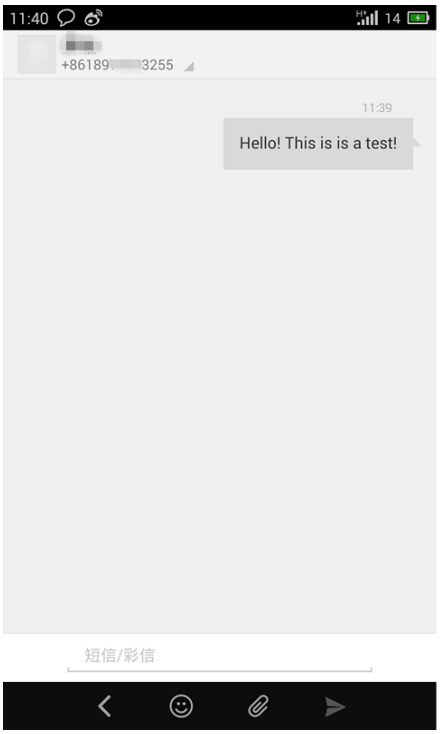


图 4-10 手机 A 向手机 B 发送短信

2) 之后使用之前介绍的基于so库注入的攻击方式对手机A进行攻击，具体操作时使用3.3.1中介绍的ptrace方式拦截短信内容，然后用hexdump读出其内容，读出结果如图4-11所示。

```
00000000  00 00 00 00 01 00 00 00  01 00 00 00 00 00 00 00  |................|
00000010  01 00 00 00 50 c3 65 b6  48 01 00 00 03 00 00 00  |....P.e.H.......|
00000020  ec 01 00 00 0e 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 03 00 00 00  fa 01 00 00 15 00 00 00  |................|
00000040  01 00 00 00 00 00 00 00  00 00 00 2b 38 36 31 38  |...........+8618|
00000050  39 [illegible] 33 32 35  35 00 48 65 6c 6c 6f 21  |9 3255.Hello!|
00000060  54 68 69 73 20 69 73 20  69 73 20 61 20 74 65 73  |This is is a tes|
00000070  74 21 00 01 00 00 00 00  00 00 00 00 00 00 00 00  |t!..............|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

图 4-11 在未进行保护的状态下拦截短信

从上图可以发现，在未对手机进行保护的情况下，可以很轻松的获取短信的内容，在此次测试中，可以看出短信内容为**“Hello! This is is a test!”**。手机 B 的号码为**“+8618****255”**。从此次测试可以看出，Android 手机 IPC 通信具有很大的安全隐患，攻击者或恶意软件可以很容易的进行 Binder 通信数据中敏感信息的窃取和篡改。

3) 之后对手机A的短信APP进行保护，并再次向手机B发送短信。效果如图4-12，图 4-13 所示。

图 4-12 对手机 A 开启短信保护

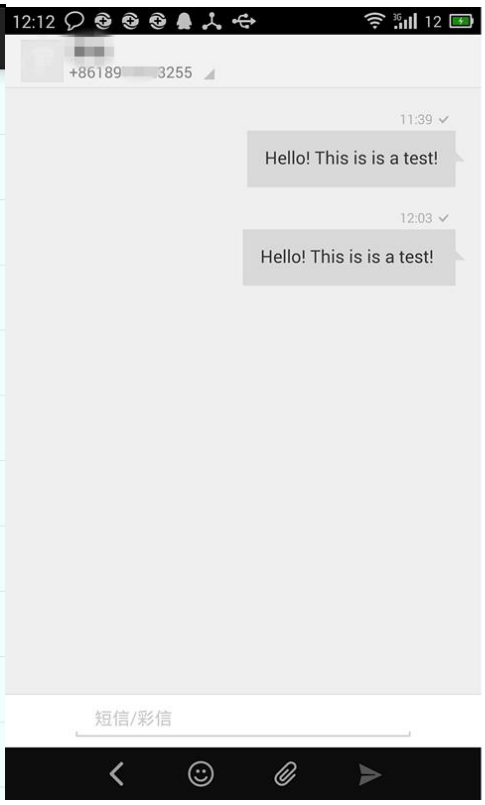


图 4-13 使用手机 A 向手机 B 发送短信

4) 之后，再次使用2)中的攻击方式，对手机A进行攻击，并查看攻击结果，如图4-15所示 。

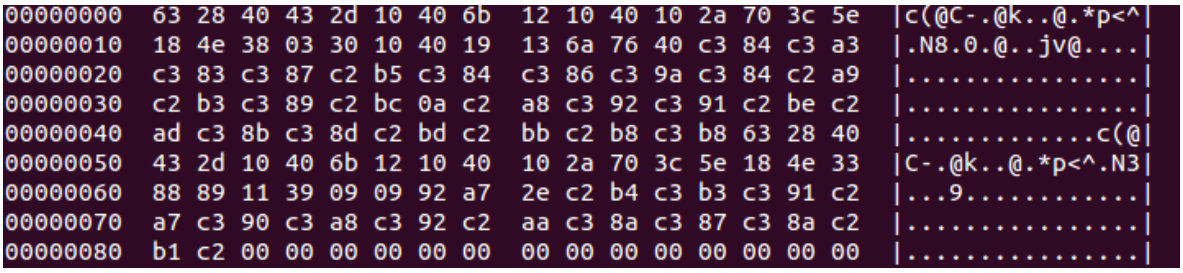


图 4-15 在进行保护后的状态下拦截短信

可以发现，在对手机进行保护后，攻击者已经无法获取正常的短信内容了，在此例中可以发现之前保存短信内容和发送号码的部分已经变成了密文，只有经过了本作品APP的授权的程序才能对密文进行正确的解密，从而获得短信内容。

经过以上测试，不难发现本作品能够**有效防止敏感数据被恶意第三方窃取与篡改的威胁，保障用户敏感数据在系统内部传输过程中的私密性及完整性。**

本项测试结果如表8所示。

表 8 测试 1-3 测试结果

编号	测试方法	测试目的	测试结果
测试 1-3	使用基于 SO 库的注入方式在程序开启和关闭防护之后分别拦截相同的 Binder 数据包，比较防护开启前后的攻击效果。	判断防护方案能否有效的抵御了不法分子基于 Android 系统 IPC 通信的攻击，加强 Binder 传输过程中敏感数据的安全性。	在程序对手机进行保护之前，不法分子可以获取到 Binder 敏感数据，但是在程序对手机进行保护之后，攻击者就无法获取 Binder 敏感数据。

4.4.2 基本功能测试

4.4.2.1 测试 1-4 应用管理功能测试

1) 使用 A 手机安装本程序，点击启动按钮。进入程序主界面，点击应用管理按钮，进入应用管理界面，如下图所示。

应用管理主界面

图4-16应用管理界面图

在上图中显示了整个手机中所有的APP，事实上，对于有不同需求的用户(例如仅仅只需要保护短信功能的用户)，程序提供了几个常见的选项，如GPS应用，短信应用，供用户筛选其需要保护的敏感程序。

2) 针对敏感软件，点击相应的应用软件，可选择添加保护或者查看详细，如图 4-17，图 4-18 所示。

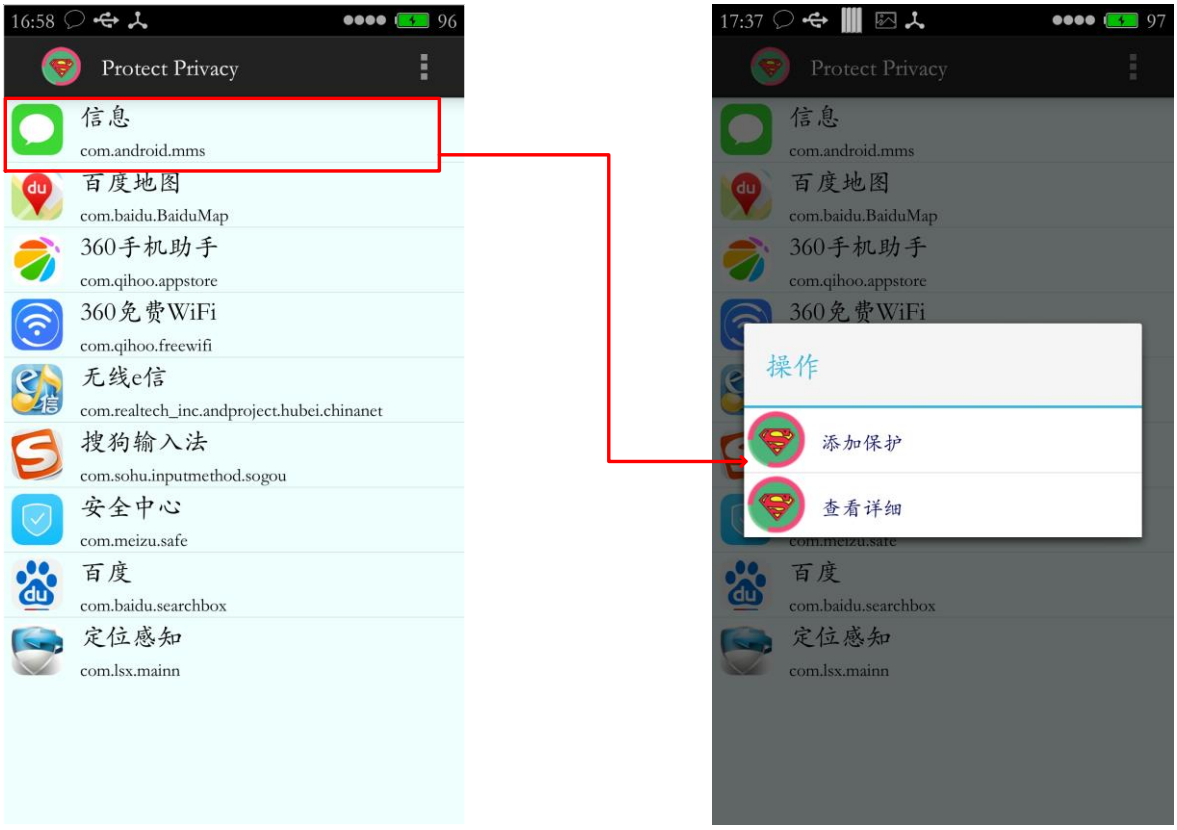


图 4-17 敏感程序列表图

图 4-18 选择添加信任或查看详情

程序的详细信息包括了程序名，版本，以及程序所申请的权限，这些信息一方面有助于用户对手机中的应用进行管理，一方面有助于技术人员对手机进行调试。另外，当用户勾选了希望保护的程序后，本作品会使用一个保护的图标对此程序进行标示，用以提醒用户此程序的运行状况。

表 9 测试 1-4 测试结果

编号	测试方法	测试目的	测试结果
测试 1-4	使用手机 A 安装本程序，点击按钮进入应用管理功能，尝试将程序添加/删除信任，并获取系统基本信息。	测试程序是否能够正确实现应用管理功能	程序能实现预期的应用管理功能。包括显示，筛选系统程序。显示程序基本信息，以及将程序添加或剔除信任

4.4.2.2 测试 1-5 日志管理功能测试

使用A手机安装本程序，点击启动按钮。进入程序主界面
点击日志记录按钮，进入日志记录界面，如图4-19

图 4-19 日志管理界面图

可以看出，在日志管理界面，记录了之前的保护信息，事实上，此日志会随着整个程序的运行进行相应的记录，记录操作行为、操作目标、操作时间，便于用户管理自己手机的安全状况。

表 10 测试 1-5 测试结果

编号	测试方法	测试目的	测试结果
测试 1-5	使用手机 A 安装本程序，点击按钮进入日志记录功能，查看当前的保护记录	测试程序是否能够正确实现日志记录功能。	程序能实现预期的日志记录功能。包括记录本作品的各种保护操作

4.4.2.3 测试 1-6 计划任务功能测试

1) 使用 A 手机安装本程序，点击启动按钮。进入程序主界面
点击计划任务按钮，进入计划任务界面，如图 4-20

图 4-20 计划任务界面图

可以看到，在计划任务的主界面中，已经集成了三个任务计划功能，分别是**系统计划**，**短信计划**，**GPS 计划**，其中系统计划包括对手机类各种服务的保护计划设定，短信计划和 GPS 计划特指对短信服务和 GPS 定位服务保护计划的设定。

2) 在这里，点击系统计划，选择对手机程序 “安全中心”进行保护计划的设定，效果如图 4-21。

图 4-21　设定对"安全中心"的保护计划

可以发现，在这里设置此次计划的开始时间是2015年5月25日0时，结束时间是2015年5月26日24时。

3) 在设定了保护计划之后，通过查看日志记录，判断是否成功设定了保护计划，效果如图 4-22

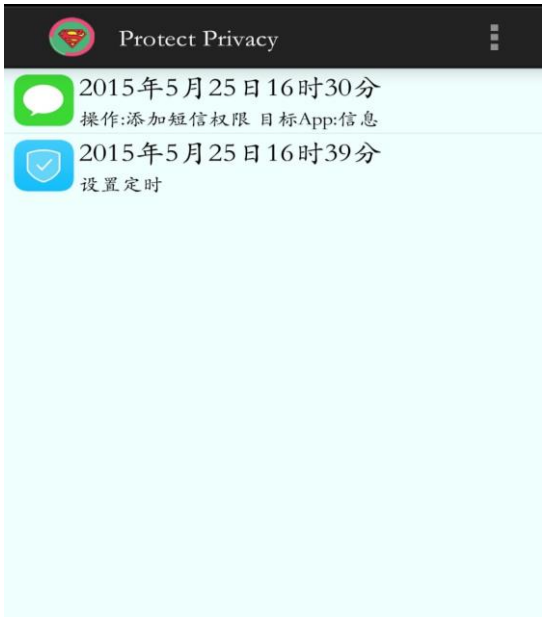


图 4-22 查看日志记录

从上图可以看出，我们已经在2015年5月26日16时39分成功的设定了计划任务。

表 11 测试 1-6 测试结果

编号	测试方法	测试目的	测试结果
测试 1-6	使用手机 A 安装本程序，点击按钮进入计划任务功能，设置系统默认定时计划。并	测试程序是否能够正确实现计划任务功能。	程序能实现预期的计划任务功能。 能够实现设定定时保护的

	查看相应日志记录		功能。

4.4.2.4 测试 1-7 安全配置功能测试

1） 使用 A 手机安全本程序，点击启动按钮，进入程序主界面，点击安全配置按钮，进入安全配置界面，如图 4-23，图 4-24 所示。

图 4-23 安全配置前

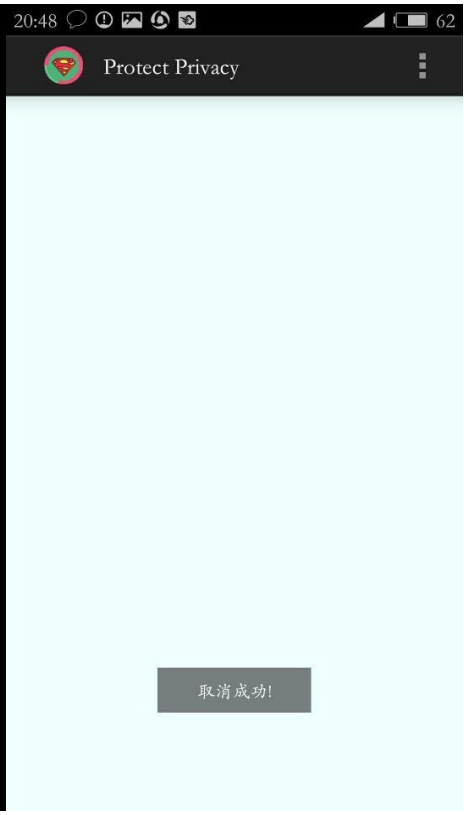


图 4-24 取消当前安全配置

在上图中显示了在测试 1-1 中配置的“百度地图”的保护方案，我们点击它，在弹出的对话框中点击取消信赖，可以发现之前开启的针对百度地图的已被取消。

表 12 测试 1-7 测试结果

测试编号	测试方法	测试目的	测试结果
测试 1-7	使用手机 A 安装本程序，点击按钮进入安全配置界面，设置定时计划。并查看相应日志记录	测试程序是否能够正确实现安全配置功能，管理当前的防护方案	程序能实现预期的计划任务功能，包括管理当前已防护程序

4.5测试总结与分析

根据以上测试工作可以总结出，本作品在功能和性能上均完全符合预期标准，归纳为以下几点：

1) 程序能有效保护敏感数据在系统内部传输过程中不被窃取与篡改，并以直观灵活的方式反馈保护结果，记录程序运行的各项信息。
2) 程序能根据用户的需求独立自主地对指定应用程序中特定类型的敏感数据进行保护
3) 程序能以直观形象的形式展现给用户，操作方式简单明了，具有良好的用户体验。
4) 整个保护过程对用户透明。无需用户进行额外的操作、不会影响用户正常的使用。

第五章 创新性与实用性

本作品为面向Android系统内部数据传递过程的敏感信息保护软件，并且此软件以用户为主，根据用户的选择为用户敏感数据提供透明化的精确的保护，在立题初衷，实现技术，应用前景方面具有重大的创新性，同时在防护加固，应用范围，实际操作等方面具有重大的实用性。

本作品的创新性具体表现在以下几个方面：

1. 立题初衷

近年来 PC 转向移动化，智能化道路，从而迎来移动与智能的春天。移动智能手持设备已成为现今人们生活，交流，工作不可缺少的一部分。然而，移动智能操作系统开始处于一方独大的局面，Android 系统 2014 年第二季度全球市场份额已达 84.6%，并且不断上升。由于漏洞与操作系统本身关联极大，多种操作系统并存的一个好处是，一个操作系统的漏洞影响范围仅仅局限该操作系统。而 Android 的巨大市场占有率本身就预示着巨大的安全隐患，而今一个 Android 的漏洞波及范围是全球 85%以上用户，可想而知，对于 Android 系统的加固与保护意义的重大性。Binder 的明文传输可以说是 Android 系统存在的一个薄弱点，前瞻性做出实际工作，防患于未然，保护 Binder，实现透明化加密解密，意义非凡。

2. 实现技术

保护 Binder 里面的敏感数据，从技术层次方面涉及的不是上层建筑，而是底层支持。在底层方面 Android 系统提出的基于 Binder 的 IPC 机制实现十分复杂，要实现这方面的保护，难度可想而知，也是十艰巨的。保护 Binder 里面的敏感数据，并实现加解密其中必要条件就是在深刻领悟源码的基础上融合加解密 SMS4 方面的知识。本小组团结合作，分工明确，在老师指导下，明确系统目标，各自攻读 Android 底层源码，同时经常交流，沟通，最终实现成果，在技术方面实现可以说是对系统层次安全性的加固，有很大的技术创新性。

3. 应用前景

Android 系统在市场上的高占有率，使得每一个针对 Android 系统的安全创新拥有广大的应用与市场，其中需要指出的是对企业，政府等安全性要求很高的部门，对安全性的追求更是永无止尽的。基于 Binder 的 IPC 机制下的敏感数据的保护在应用方面有很大的需求，本作品在技术实现的基础上，面向用户进行了界面包装美化，操作也简单易懂，不仅为广大用户提供了一种强有力保护敏感数据的解决方案，同时也为企业、政府等安全性要求较高的构建了一种新的安全增强型策略，可以促进 Android 系统在企业中的应用，推动了 Android 生态系统的健康良性的发展，具有应用创新意义。

本作品的实用性具体表现在以下几个方面：

1. 防护加固

信息时代，信息安全至关重要，尤其是手机中的大量敏感信息，因此对 Android 的加固防护必须引起十分的重视。Binder 机制中敏感数据的明文传输是一个亟待解决的安全问题。本课题组实现了针对 Binder 的加解密的解决方案，不仅有效增强了用户敏感数据的保护，而且在此基础上，增加了用户自主开启，关闭，配置，定制以及日志纪录等操作与功能，成功打造了有一个有效的，时刻保护用户敏感数据的生态体系，防护加固实用性十分强。

2. 应用范围

绝大多数的敏感数据是基于 Binder 机制的，应用的上层建筑——各种 App 尽管可能千奇百样，种类繁多，但是映射到底层，我们只需要专注于 Binder 的安全保护，在底层方面实现该机制的保护，不仅是在系统级打造了一扇安全门，截断了绝大多数的上层 App 随意获取敏感数据的通道，而且不言而喻应用范围也是极其广阔的。

3. 实际操作

本课题组在实现 Binder 机制敏感数据加解密的核心技术之上并不满足，在面向用户，尤其是普通用户，实际操作与体验实用性方面也是本课题组关注的一个地方，本课题组细致考虑，在 UI 方面尽可能友好，不仅仅提供实时动态的圆圈指示保护状态，还设身处地提供开启、关闭定制、日志记录，个性配置等操作功能，在实际操作方面十分实用。

第六章 总结

6.1项目工作总结

6.1.1 防护功能总结

本作品开发了一款 Android 系统安全强化 APP，本作品可以根据用户的需求对 Android 系统内的数据传递过程中的各种敏感数据进行保护，并以简介直观的形式将结果展现给用户。

本作品主要完成的工作归纳如下：

本作品从 Android 系统的 IPC 通信机制出发，创新性地提出并实现了针对 Android 系统中敏感数据从产生到使用整个生命周期的自适应性透明加密的保护方案，有效防止了敏感数据被恶意第三方窃取与篡改的威胁，保障了用户敏感数据在系统内部传输过程中的私密性及完整性。

6.1.2 其他功能总结

本作品在实现基本功能的基本之上，进行上层界面封装，向用户提供了简单、灵活的操作体验，使得用户可独立自主地对指定应用程序中特定类型的敏感数据进行保护，增强了本系统的易用性和实用性。

本作品的整个保护过程对用户透明。无需用户进行额外的操作、不会影响用户正常的使用。并且对为用户封装了四大功能：安全配置，日志记录，计划任务，应用管理，以满足用户的不同需求。

6.2 后续工作

在以 Android 手机为代表的移动智能终端迅猛发展的同时，针智能移动终端的恶意软件也层出不穷、花样繁多，这给保障用户隐私、财产等安全带来了极大挑战。攻击和防护作为“矛”和“盾”，则是一个长期博弈的过程，很难在一个较短的时间周期内战胜另一方。

受利益的趋势和信息安全中短板效应的决定，攻击者攻击系统的手段肯定会更加高明，不排除将来不法分子能够在 IPC 传输流程上找到一个比我们更加优先的攻击点，使得我们的安全防护失效。

除此之外，本次作品实现的防护方案还较为单一，无法满足各类用户的功能需求。

针对以上的问题，今后将继续开展如下工作：

1) 继续加强对 Android 系统的理解，尽量深入到系统的 Native C 层和内核层，彻底实现从敏感数据产生到使用整个生命周期的不完善。从最本质的部分学习 IPC 的工作过程，并研究攻击者可能的攻击入口，提供相应的保护方案并集合到我们已有的系统之上。

2) 完善程序的功能模块，提供各种行之有效的服务保护功能，争取针对用户满足其各类不同的要求。

参考文献

[1] 新浪科技. Strategy Analytics 2014-2015 年全球移动终端市场发展研究报告 http://www.199it.com/archives/343700.html

[2] 网易科技报道. IDC 2014 年智能手机出货量数据. http://tech.163.com/15/0225/07/AJ9HL4H7000915BD.html

[3] BetaNews. 2014 Half Year Security Report http://digi.163.com/14/0804/15/A2QJT0QE00162OUT.html_

[4] 腾讯移动安全实验室.腾讯移动安全实验室 2014 年 7 月手机安全报告. http://scan.qq.com/security_lab/news_detail_264.html

[5] JuniperNetworks. http://forums.juniper.net/t5/Security-Mobility-Now/Android-is-the-New-Malware-Battleground-ZeuS-GoldDream-DDLight2/ba-p/132955. 2012.

[6] 网秦. 2012 年第一季度全球 Android 手机安全报告[EB/OL]. http://wenku.baidu.com/view/50f373ea102de2bd960588c9.html, 2012.

[7] AVTest. http://www.av-test.org/fileadmin/pdf/avtest_2011-11_free_Android_virus_ scan ner_english.pdf. 2011.

[8] Schmidt A. D., Schmidt H. G., et al. Enhancing Security of Linux-based Android Devices[A]. In: Proceedings of 15th International Linux Congress. Lehmann, Oct. 2008.

[9] Chiang C. Y., Yang S. J., et al. Android-based patrol robot featuring automatic license plate recognition. In: 2012 Computing, Communications and Applications Conference, ComComAp 2012, p 117-122.

[10] Burguera L., Z. Urko, et al. Crowdroid: Behavior-Based Malware Detection System for Android[A]. In: Proceedings of the ACM Conference on Computer and Communications Security. Chicago, Dec. 2011.

[11] Egele Manuel, Scholte Theodoor et al. A Survey on Automated Dynamic Malware Analysis Techniques and Tools[J]. ACM Computing Surveys, Feb 2012, Vol 44, No. 2.

[12] CSDN 博客. Binder 设计与实现[EB/OL].

[13] http://blog.csdn.net/universus/article/details/6211589, 2011

[14] 张和君，张跃. Linux 动态连接机制研究及应用[J]. 计算机工程, 2006, Vol 32(22):

[15] Tool Interface Standard. Executable and Linkable Format[S]. 1995.

[16] ContagioMobile. http://contagiominidump.blogspot.com.