西安科技大学

学 位 论 文 诚 信 声 明 书

本人郑重声明：所呈交的学位论文（设计）是我个人在导师指导下进行的研究（设计）工作及取得的研究（设计）成果。除了文中加以标注和致谢的地方外，论文（设计）中不包含其他人或集体已经公开发表或撰写过的研究（设计）成果，也不包含本人或其他人在其它单位已申请学位或为其他用途使用过的成果。与我一同工作的同志对本研究（设计）所做的任何贡献均已在论文中做了明确的说明并表示了致谢。

申请学位论文（设计）与资料若有不实之处，本人愿承担一切相关责任。

学位论文（设计）作者签名：　　日期：

学 位 论 文 知 识 产 权 声 明 书

本人完全了解学校有关保护知识产权的规定，即：在校期间所做论文（设计）工作的知识产权属西安科技大学所有。学校有权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。本人允许论文（设计）被查阅和借阅；学校可以公布本学位论文（设计）的全部或部分内容并将有关内容编入有关数据库进行检索，可以采用影印、缩印或其它复制手段保存和汇编本学位论文。

保密论文待解密后适用本声明。

学位论文（设计）作者签名：　　指导教师签名：

年　　月　　日

分 类 号　　密 级

学校代码 10704　　学 号 16408070603

西安科技大学

学 士 学 位 论 文

题目：基于 COAL 语言 Android 程序的 ICC 建模与分析

作者：高瑜瑜

指导教师：刘晓建　　专业技术职称：副教授

学科专业：软件工程　　申请学位日期：2020 年 6 月

摘　要

随着 Android 系统的飞速发展，Android 应用程序在国内的开发和使用数量迅猛增长，据统计 2020 年仅华为应用商店 app 分发量就已达到 20100 次。然而追求松耦合特点进行 ICC 通信的 Android 程序也给恶意程序的设计与应用带来了可乘之机。近年来，大量的恶意程序威胁着 Android 用户的隐私，在互联网时代盛行之时，海量的私人数据通过组件进行信息交流时面临着泄露，并且是在用户未知情的情况下或未同意的情况下。因此，对恶意程序进行检测并采取相应措施刻不容缓，其中对 ICC 通信机制进行分析研究是对 app 进行精确检测的必要前提。

本课题学习 Android 系统及组件通信特征、字符串分析方法以及组件间数据泄露检测工具的相关知识，通过 COAL 语言对组件间 ICC 通信方式进行建模，并对 ICC 通信机制的特点进行分析和总结，随后通过恶意程序案例对 ICC 模型进行分析，获得相关实验数据，并通过分析数据得到有关 ICC 通信和信息流的一些基本结论，有助于 Android 程序的信息安全问题解决。

关键词： Android组件；COAL语言；ICC通信；程序恶意性；隐私泄露

ABSTRACT

With the rapid development of Android system in China, the number of Android applications developed and used in China is growing rapidly. According to statistics, in 2020, the app distribution volume of Huawei app store, which only performs strongly, has reached 20100 times. In recent years, a large number of malicious programs threaten the privacy of Android users. When the Internet age prevails, a large number of private data are exposed when they communicate through components, and in the case of user location or without consent In this case. Therefore, it is urgent to detect malicious programs and take corresponding measures. Among them, accurate analysis and Research on ICC communication mechanism is the necessary prerequisite for the accuracy design of detection system.

In this topic, the ICC communication mode between components will be modeled through the coal language, and the characteristics of ICC communication mechanism will be analyzed and summarized. Then, the ICC model will be analyzed through malicious program cases to obtain relevant experimental conclusions, analyze the communication characteristics between components, provide a model for the research of ICC characteristics, and make a theoretical summary of malicious application detection to promote Android Solve the information security problem of the program.

Through this research and analysis, students can learn about Android system and component communication features, string analysis methods and data leakage detection tools between components. The ICC model proposed in this paper is a general model, which focuses on the reusability of programs. It can be used to accurately detect malicious programs, analyze the information exchange mechanism of malicious programs, and play a great role in the research and measures of malicious programs.

Key Words: Android compnents COAL luanguage ICC Program malice Privacy leaks

目 录

1 绪论.....1

1.1 选题背景与研究意义.....1

1.1.1 研究现状.....1

1.1.2 发展趋势.....2

1.2 研究内容.....2

1.2.1 研究内容.....2

1.2.2 研究思路.....2

1.2.3 工作流程.....3

1.2.4 通信污点分析实现方案.....3

2 实验工具.....6

2.1 Eclipse.....6

2.2 FlowDroid+IccTa.....6

2.2.1 FlowDroid 简介.....6

2.2.2 IccTa 简介.....8

2.3 IC3.....9

3 Android 系统与组件及其通信机制.....10

3.1 Android 系统及其组件.....10

3.1.1 Android 系统概述.....10

3.1.2 组件.....10

3.1.3 组件的 LifeCycle.....11

3.1.4 回调函数.....12

3.2 通信机制.....12

3.2.1 Intent 对象.....13

3.2.2 ICC 方法.....13

3.2.3 ICC 泄漏.....14

4 基于 COAL 语言的 ICC 机制分析与设计.....16

4.1 COAL 语言及 COAL 规范.....16
4.1.1 COAL 语言.....16
4.1.2 COAL 求解器.....18
4.2 字符串分析.....21
4.2.1 约束生成.....21
4.2.2 约束求解.....21
4.3 设计.....22
4.3.1 工具整合.....22
4.3.2 COAL 模型设计.....22
5 案例测试与实验分析.....24
5.1 案例.....24
5.1.1 案例描述.....24
5.1.2 COAL 模型建立.....25
5.1.3 生命周期及回调函数模型的建立.....26
5.1.4 静态污点分析.....27
5.1.5 字符串分析.....28
5.2 案例总结.....29
6 结论及展望.....30
致谢.....31
参考文献.....32

1 绪论

1.1 选题背景与研究意义

随着互联网及移动通讯技术的飞速发展，以智能手机为代表的 Android 移动设备也快速发展，各种 Android 程序也以其开源、开发容易、市场价格低、适应性强等特点迅速涌入人们视野，人们也越来越依赖它，而且在设备上绑定的私人数据随之越来越多。然而，由于 Android 系统的开放性，使得 Android 安全问题源源不断，追其原由，一方面，Android 系统的碎片化难以遏制，自身的漏洞难以完全修复。另一方面，恶意程序利用系统漏洞被投放到 app store，导致用户隐私泄漏或病毒插入，进而对移动设备进行远程控制。像这样利用手机系统漏洞进行犯罪的行为屡禁不止。据统计，2019 年，360 安全大脑发布的《2019 年 Android 恶意软件专题报告》显示，仅 360 安全大脑全年共截获新增恶意软件样本共 180.9 万个，恶意软件在种类和数量上都是在呈增长趋势，其中消费消耗与隐私窃取类别的占比高达 46.8%和 41.9%。因而，Android 安全问题越来越威胁着我们的生活。

近些年，人们在 Android 恶意软件的检测方法、Android 安全分析、Android 恶意攻击模式、Android 网络安全等方面进行不断地深入研究，为 Android 安全问题的解决做出巨大贡献。然而，在 Android 组件间信息泄露的相关文献还是不够完善，对于 Android 程序的 ICC 模型与检测实验分析也有待完善，急需统一地进行分析研究。为此，本文将通过大量实验对 Android 程序 ICC 进行分析研究，用数据解释分析 Android 的通信机制以及恶意软件的泄露机制。

本课题在现有工作的基础上，对大量相关文献进行分析总结，并针对案例，使用相应测漏软件对其进行实验分析，根据数据分析 ICC 方法，并归纳总结其特征。同时整理相关结果进行建模，加深人们对 ICC 模型的特性和方法了解与理解，对 Android 应用程序隐私泄露检测的精确度以及恶意程序遏制研究等相关领域的研究有着一定的作用。

1.1.1 研究现状

目前，针对 Android 操作系统程序有关 ICC 通信模型而带来的信息泄露，国内外研究者主要面向恶意软件检测、安全性分析方法、组件内通讯研究等方面，国内外相关研究工作如下：

房鼎益[1]等人提出一种构建潜在泄露隐私的组件序列的方法用于检测数据泄露路径方面所遇到的难题；郑尧[2]运用一种关于矢量机制和建立 ICC 特权有向图推理的方法,对

Android 恶意应用进行检测；夏从里[5]提出了 ICC 通信机制的特点，分析工具以及理论和方法；唐娅[7]提出了基于机器语言和基于形式化的静态分析检测方法等。

如上文所示，目前大多数研究都比较倾向与检测恶意软件和利用静态污染分析工具进行检测，而对分析结果的准确性，或者结合 ICC 通信机制深入理解 ICC 特征的相关文献却很少。有些研究虽在通信特征下手来研究检测方法和检测工具，但很少返回再次根据结果来分析 ICC 特征的。还有研究比较侧重于检测手段或者通信机制，却没有将两者进行系统的统一分析。因此，本文将通过在分析了解通信机制的基础上，选择相关典型案例，用 FlowDroid+IccTa 进行实验，然后分析数据，通过检测出来的数据再次进行 ICC 通信分析。对信息泄露检测方法，ICC 通信机制研究，以及检测方法的准确性进行统一的总结与整理，对系统了解 ICC 通信及信息泄露有着重要意义。

1.1.2 发展趋势

随着互联网的流行，互联网安全问题在人们的生活中显得及其突出，其中 Android 程序所引发的数据安全性问题也越来越困扰着人们，其中对恶意程序的识别与预防是主要的方式，就此而言，对恶意应用程序的检测精确性的要求将会越来越高，因而，作为数据泄露的主要原因的组件间交流的机制，ICC 的特性研究及建模就显得尤为重要了。

1.2 研究内容

1.2.1 研究内容

Android 恶意程随着 Android 应用程序的普遍使用广泛地被下载安装，并时刻可能导致人们的隐私外泄，威胁着人们的正常生活。为此，全面的了解并理解作为通信模式的 ICC 模型是我们预防私人数据外流的前提。在已有文献和研究的基础上，在理解通信原理，了解通信方式，掌握常用的字符串分析方法之后，根据已有知识对恶意软件泄露信息的案例进行试验得出数据结果进而进行分析。最后对 ICC 建模进行分析，方便人们对 ICC 模型进行更为深入的了解或研究。

1.2.2 研究思路

(1) 查询及阅读大量相关文献，理解Android Intent通信的基本原理，学习构件间Intent通信方式；

(2) 理解字符串分析的重要意义，学习字符串分析的常用方法；

(3) 利用FlowDroid+IccTa分析三个典型案例，得到实验数据；

(4) 分析实验数据，结合数据对结果进行解释并分析。

1.2.3 工作流程

学习和理解 Android Intent 通信原理，学习组件间通信方式及字符串的分析，主要通过查阅国内外文档，归纳总结相关知识的方式。在理论知识了解熟悉的基础之上结合 FlowDroid+IccTa 工具测出的隐私泄露数据对 ICC 通信进行分析建模，然后整理总结成相关文档。

具体工作流程安排如下图：

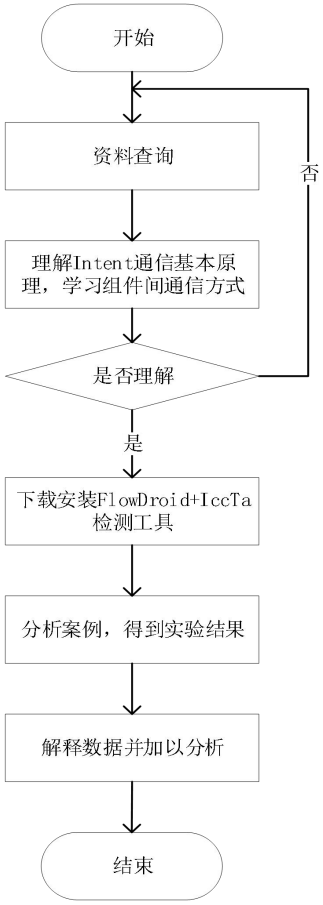

图 1-1 工作流程图

1.2.4 通信污点分析实现方案

(1)实现方案

开发环境及工具：Windows 10 x64、JDK1.8、FlowDroid+IccTa 等

实现方案：Android 操作系统良好的开放性，允许用户随意下载安装第三方软件，随着应用程序的使用数量不断增加，人们在程序上连通相关私人数据不断泄露，随时可能威胁着人们的生活。为分析恶意程序存在的隐私泄露情况，Android 应用程序分析师可以

利用 FlowDroid+IccTa 工具来识别泄漏私人数据的恶意应用程序。IccTa 是一种组件间通信污点分析工具，以合理、准确地检测 ICC 链接和泄漏。

根据查阅资料可知，对 ICC 在 Android 应用程序中的使用进行实证研究还没有发现，鉴于 ICC 在 Android 开发模型中的重要性以及它与恶意软件功能之间的潜在关联，本文必须考虑以下问题：ICC 不同方法的使用范围及所针对的组件类型；应用程序中 ICC 使用的 Intent 种类；恶意程序和良性程序中 ICC 使用的不同之处。因此，挑选 MalGenome 数据集和 Google 官方市场上下载的数据集对相关问题进行分析。

由于 Android 程序的特殊性，对其使用静态分析法有一定的困难，所以使用 IccTa 进行 ICC 泄露检测。

(2) IccTa检测ICC泄露实现方案

开发环境及工具：Windows 10 x64、JDK1.8、Maven3.6.1、FlowDroid+ICCTA 等

实现原理：IccTa 检测大致分为两个步骤：ICC 链接提取；ICC 污染流分析。其具体步骤如下图所示：

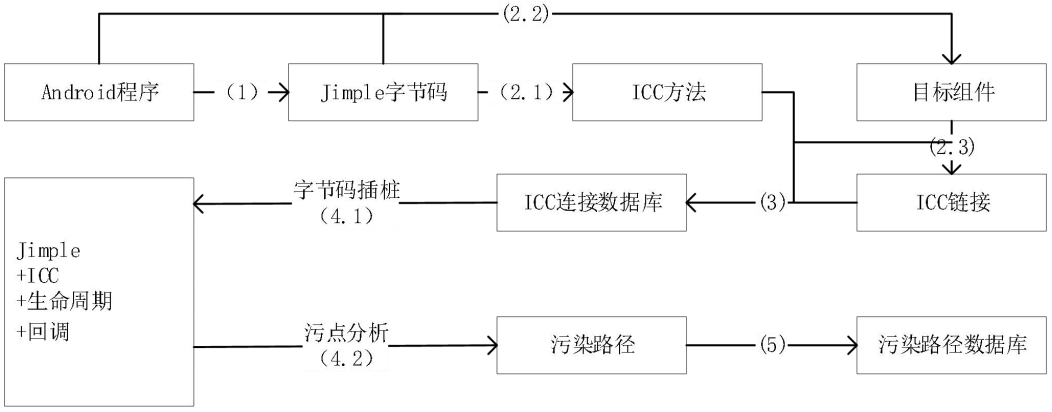

图 1-2 IccTa 概述图

在第一步中，IccTa 通过 Dexpler[16]将 Dalvik 字节码转换为 Jimple 字节码。Coothe 是用于分析基于 Java 的应用程序的流行框架。在第二步中（如箭头 2.*所示），使用 IccTa 提取 ICC 链接，并在步骤 3 中，将它们以及收集的所有数据（例如 ICC 方法调用参数或 intent 过滤值）存储到数据库中。在步骤 4.1 中，IccTa 修改了 Jimple 表示，用以直接连接组件，从而进行组件之间的数据流分析。在步骤 4.2 中，IccTa 使用一个针对 Android 程序的高精度组件内污染分析工具 FlowDroid[17]的修改版本，构建了 Android 应用程序的完整（CFG）控制流图。这样就允许在 Android 组件之间传播上下文（例如 Intents 的值或 Intent 参数传递），同时产生一个高精度的数据流分析。据我们所知，这是第一种精

确连接数据流分析组件的方法。最后（在步骤 5 中），IccTa 将生成的污染路径（泄漏）存储到数据库中。

在步骤（3）和（5）中，我们将所有结果（包括 ICC 方法及其属性值（如 URI 和 Intent）、目标组件及其 Intent 过滤器值、构建的 ICC 链接和报告的 ICC 泄漏）存储到数据库中。这只允许分析一个应用程序一次，然后重用数据库中的结果。

2 实验工具

2.1 Eclipse

FlowDroid 静态检测工具是基于 java 开发的，因此本课题中使用 JDK1.8+clipse+Maven3.6.1。实验中，首先得进行 java 环境及 Maven 的环境搭建并进行配置，然后下载 FlowDroidFlowDroid、IccTa 及 DroidBench 测试数据,最后将 soot-infoflow 以及 soot-infoflow-android 导入 eclipse 中，如下图所示：

图 2-1 eclipse 项目文件夹截图

2.2 FlowDroid+IccTa

2.2.1 FlowDroid 简介

FlowDroid 是一种高度精确的 Android 应用程序的静态污染分析工具。Android 生命周期的精确模型使分析正确处理 Android 框架调用的回调，上下文、流、字段及对象敏感性使得分析得错误警报的数量减少，此外，新型的按需算法有助于 FlowDroid 同时保持高效率和高精度。

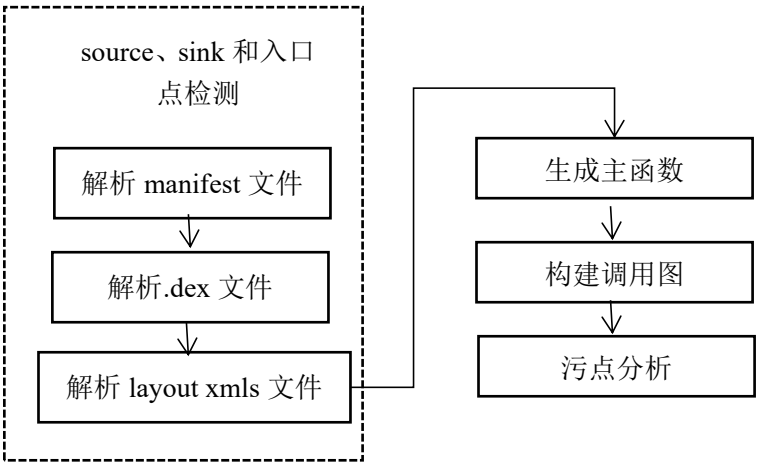

图 2-2 FlowDroid 的概述

图 2-2 显示了 FlowDroid 的架构。Android 应用程序被打包在 apk 文件中，这些文件本质上是 zip 压缩的归档文件。在解压之后，FlowDroid 进行生命周期和回调方法的搜索，以及对 Source 和 Sink 的查找。这些是通过解析各种 android 特定的文件来完成的，其中包括布局 XML 文件、可执行代码的 dex 文件以及定义应用程序中的活动、服务、广播接收器和内容提供者的 manifest.xml 文件。接着，FlowDroid 根据生命周期和回调方法生成虚拟主方法。然后在此主方法基础上生成调用图和过程间控制流图（ICFG）。检测到污染源后，污染分析就通过遍历 ICFG 来追踪污染。FlowDroid 使用了 SuSi 项目[18]推断的 Sources 和 Sinks 来查找源和汇。最后，FlowDroid 存储了所有发现的从源到汇的流。报告中包括完整的路径信息。为了获得此信息，实现将数据流的抽象对象与其前身和生成语句链接起来。这允许 FlowDroid 的报告组件完整地重构所有相关赋值语句的图，这些语句可能在给定的接收器上造成污染冲突。

FlowDroid对Android生命周期从入口点、异步执行组件和回调三个方面进行精确建模。首先，Android程序，它没有一个主方法main，对于每个应用程序都有多个入口，因此，在构建调用图时，Android分析不能从一个预定义的主方法开始，为了解决这一方法，FlowDroid对Android生命周期构建了一个模拟的主方法。其次，Android的一个应用程序包含多个组件，虽说可以按照顺序运行，但无法确定组件运行的先后，FlowDroid通过假设程序内的全部组件（活动、服务等）都能以任意顺序（包括重复）运行来模拟这种执行。然后FlowDroid使用对路径迟钝的IFDS[19]的分析框架进行处理一些静态分析的路径敏感问题，所有控制流合并点直接连接分析的结果。也就是说FlowDroid可以生成并有效地分析一个虚拟主方法，其中每个组件的生命周期和回调的顺序是任意的，它不需要遍历所有全部可能的路径。再者，Android操作系统允许应用程序为各种类型的信息注册回调，FlowDroid假定所有回调都是可以以任何可能的顺序调用，但是，回调只发生在父组件（例如activity）运行时。为了保证精确度，FlowDroid把组件（活动、服务等）与它们注册的回调关联起来。

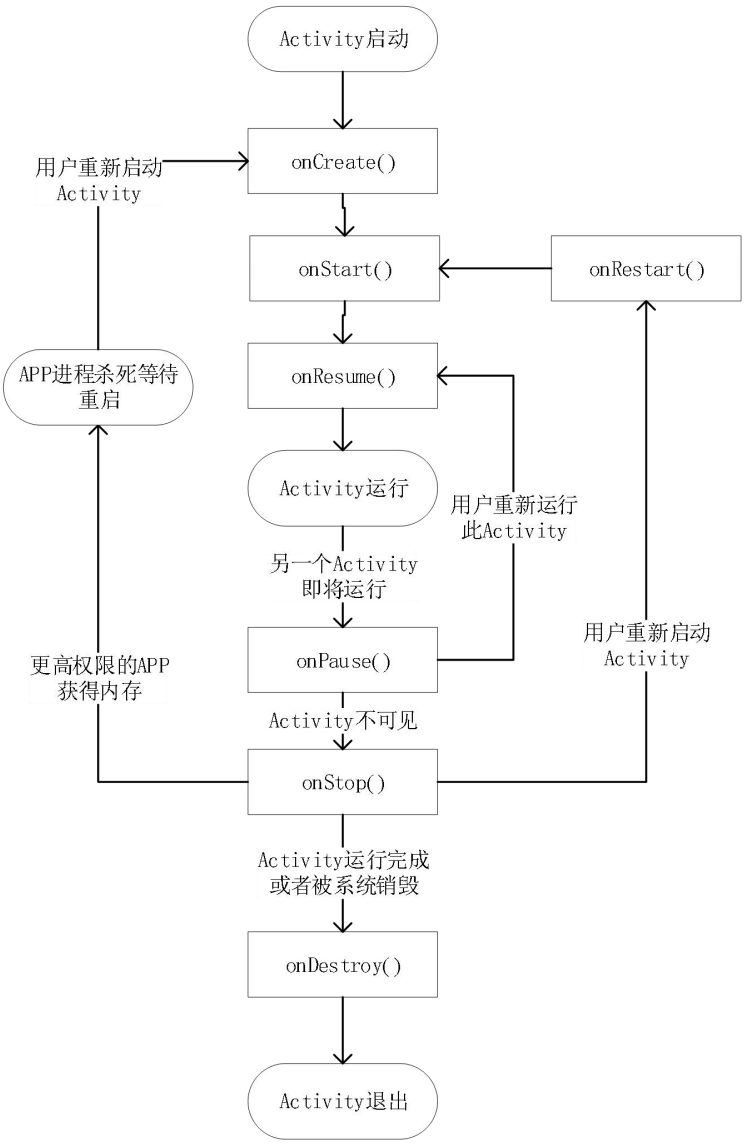

图 2-2 用于虚拟主方法的生命周期图

为了获得最大的精确度，FlowDroid为分析的每个应用程序构建一个新的虚拟主方法。根据应用程序的XML配置文件，每个主方法只涉及在运行时实际发生的生命周期的一部分。禁用的活动被自动过滤，回调方法只在它们实际所属的组件的上下文中调用。例如，按钮单击处理程序仅在其相应活动的上下文中进行分析。在图2-2中,展示了一个dummymain方法的控制流图，将在第三节中进行细致讲解。

2.2.2 IccTa 简介

尽管Android应用程序主要是用Java编程的，但Java现成的静态污点分析工具在Android应用程序上不起作用，Android的静态分析器需要进行调整。面对这一问题，本课题提出使用IccTa对ICC信息泄露进行检测。

IccTa是由Li Li、AlexandreBartel等设计提出的一个基于Android应用的静态污点流分析工具，是在Soot和IC3工具的基础上开发的。IccTa工具致力于分析Android应用中组件间通信相关的私人数据泄露，并将发现的污点流路径保存于相应数据库。

IccTa通过检测间的污点流进行实现，是Android程序污点流分析中比较先进的工具。该工具依赖于组件间传播的上下文信息，这样提高了IccTa污点流分析的精确程度。该工具主要使用插桩的方法，并考虑到Android程序中的每个组件的生命周期，从而实现了更完善的静态污点流分析。

2.3 IC3

IC3（Inter-Component Communication analysis with COAL）是基于COAL规范的ICC推理工具，它是以MVC（Multi-Valued Composite）常数作为先决条件，面向复合型常数的传播问题，用以推断在过程间、流和上下文敏感的程序中推断复杂对象的所有可能值，其中使用的语言就是COAL规范形成的COAL语言。

为了推断出组件间的交互内容，要对进行信息交流的程序点上找出组件对象的所有可能值，使用IC3比目前的ICC对象建模方法中的要更加全面，所有本文使用IC3。在使用IC3对ICC组件间时进行建模时，主要的ICC类是Intent、IntentFilter和Uri，主类类型需要一些引用参数，为此，还需对组件的相关属性进行模型化，比如，对组件的name、Bundle、Pending Intent和Uri Builder等类。

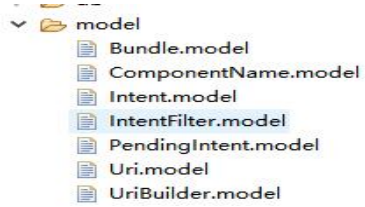

图 2-3 组件模型建立

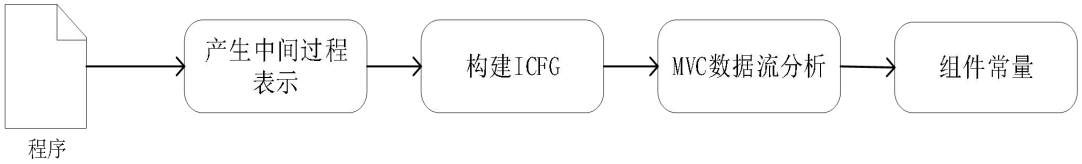

图 2-4 MVC 常量分析过程图

正是因为IC3对复合型常数问题进行分析，程序需要通过一个中间转换步骤称为中间表示IR（intermediate representation），然后方可进一步生成过程流程图（ICFG）,形成整个程序的所有过程流程图集合所构建的调用图，最后通过MVC常数数据流分析，最后生成可能的组件值，如上图所示。

3 Android 系统与组件及其通信机制

3.1 Android 系统及其组件

3.1.1 Android 系统概述

安卓系统由四层架构组成，分别为Linux内核层（Linux Kernel）、系统运行层、应用框架层（Application Framework）、应用层（Applications）。其中，Linux内核层是为Android设备的各种硬件提供了底层的驱动，应用层是安装在用户机上面的app。下来，主要讲一下系统运行层和应用框架层：

1) 系统运行层：使用一些 C/C++库来为 Android 系统提供了主要的功能支持，如提供浏览器内核的支持 Webkit 库、提供数据库的支持 SQLite 库、提供 3D 绘图的支持 OpenGL|ES 库等。在这一层还有 Android 运行时库，它提供了一些核心库来支持开发人员通过 Java 来编写 Android 应用程序。其中，关键是 Dalvik 虚拟机，使用此虚拟机不仅保证每一个 Android 应用都能独立运行，而且优化了手机内存和 CPU 性能，可以说是为安卓量身定制的虚拟机。
2) 应用框架层：主要提供了构建应用时可能用到的 API，使用这些 API 完成 Android 自带的一些核心应用程序，开发者还可以通过使用这些 API 构建自己的应用程序。比如有 View 系统、活动管理器、通知管理器、内容提供器等

3.1.2 组件

四大组件是Android应用的一大特色，它们分别是活动（activity）、服务（Service）、广播接收器（BroadCast Receiver）和内容提供器。下来对它们进行一一列举：

- 活动：是用户可以看到的界面，是打开一个程序的门面，为可视化的；
- 服务：在后台运行，对于用户不可见，例如广播收听、I/O 文件、内容提供者（Content Provider）；
- 广播接收器：用于接收并发送广播信息，并对得到的数据进行相关处理解决；
- 内容提供器：用于 app 之间进行数据的共享，比如，要短信发送时可以进行手机号码的共享。

四大组件都必须注册才能使用，其中活动、服务、内容提供者需要在AndroidManifest文件中进行声明才能被系统看见，需要注意的是广播接收者在AndroidManifest中配置后就会一直处于激活状态等待感兴趣的信息触发。对于激活方式除了内容提供者需要接到ContentReasolver请求外，其余三种组件通过intent的异步消息进行激活。其次，在Android

任务中中所扮演的角色。Android任务本质上就是activity栈，任务的所有活动是作为整体进行入栈、出栈。Android是多任务（Multi-Task）系统,因此，随着任务执行得多起来，内存消耗就会加快，为了解决这一问题，Android引入生命周期（LifeCycle）这一概念。

3.1.3 组件的 LifeCycle

Android系统通过组件生命周期来解决多任务内存不足问题，有系统框架来统一管理安排，当内存紧张时，Android系统就会根据进程优先级对组件进行释放，而这一过程依赖于组件生命周期函数。由此可见，Android组件生命周期对分析组件有着重大意义，要想对Android通信机制进行建模，储备组件交流的控制流程是必要的，而生命周期便可看作是组件的一部分CFG（Control Flow Graph）。

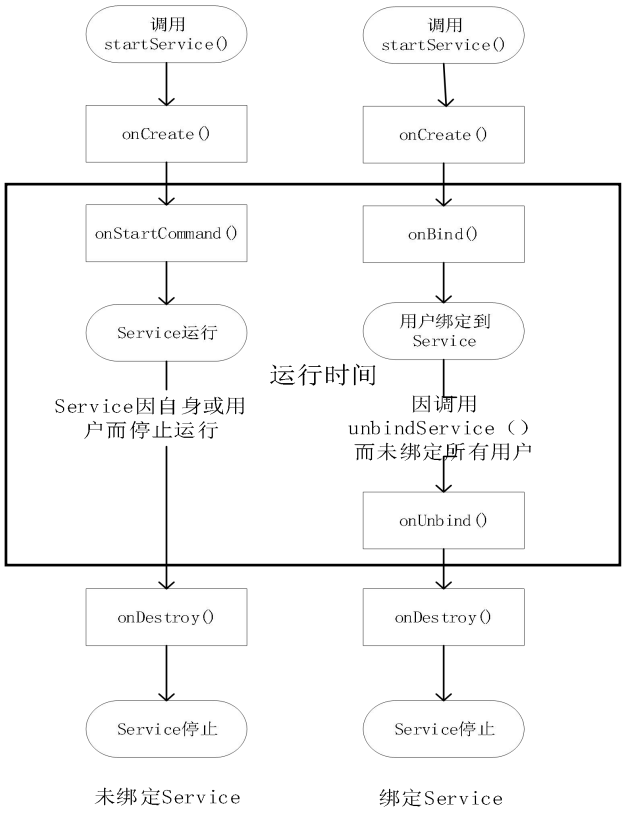

图 3-2 Service 组件的生命周期图

图3-2显示了Service组件的生命周期，图中的矩形表示生命周期函数，椭圆形表示服务的状态。从图中可以看出，Service生命周期有两条不同的路径：（1）非绑定Service在其他组件调用startService( )函数时创建，然后无期限运行，而且必须通过调用stopSelf( )函数来停止，该种Service停止运行后，系统就会把它销毁。（2）绑定Service在组件（客户端）调用bindService( )函数时创建。实际上，这种Service 有一个计数器，每当客户端与其建立连接时，计数器增加1，当客户端关闭连接时，相应的计数器就进行减少一个。

服务上的所有的绑定取消后，即当计数器为0时，系统自动销毁Service。值得注意的是，上述两条路径并不完全独立，可以选择绑定已使用startService( )启动的服务。

如图2-2所示就可表示为一个Activity的生命周期。图中矩形代表回调方法(生命周期函数),Activity在各个状态切换时调用这些方法;椭圆形则代表Activity所处的某些状态（比如运行、暂停和死亡等）。首先，Activity过调用函数onCreate( )开始整个生命周期的生存时间（entire lifetime），调用onDestroy( )数结束。在调用onCreate( )函数时对组件进行所有的状态进行设置，而在调用onDestroy( )函数时释放该组件占用的所有资源。其次，Activity的可见生存时间（visible lifetimne）发生在调用onStart( )函数和对应的onStop( )函数之间。在这期间，虽然在屏幕上该组件对于使用者来说是可见的，但是Activity可能运行在后台并且不能与用户交互。最后，Activity调用onResume( )和对应的onPause( )期间被称为前台生存时间（foreground lifetime）。在这期间，此Activity在所有活动组件最前面，而且能够与用户进行交互。

广播接收器生命周期很短暂，整个过程大约10，是调用轻量级函数onReceive( )的过程。

内容提供者组件在安装后会提供一个URI以支持相关操作。该组件没有运行状态。

3.1.4 回调函数

一般java静态分析工具对于Android应用来说并不适用，部分原因就是要考虑Android的回调函数这一机制。回调函数就是在A类中定义一个方法，该方法调用某抽象方法c，此方法在A中调用，在B类中实现，称此机制为回调。

Android系统中使用回调的情况大致分为两种。一种就是用户触发事件与应用进行交互时，用户输入管理这一过程就需要回调方法来实现，比如onClick( )方法，就是在用户点击按钮就会触发点击事件，操作系统就会调用onClick( )方法。另外一种就是管理生命周期的生命周期函数。

因为在收集回调时，不仅要考虑UI触发事件，而且要考虑生命周期函数，只是生命周期函数在Android程序中没有显式声明，所以就有了挑战，所以在分析ICC机制时，需要将组件的生命周期考虑进去，从而得到完整的控制流。

3.2 通信机制

Android应用程序由称为组件的基本单元组成，这些基本单元在应用程序包中包含的特殊文件Manifest中进行了描述。在进行数据交流时，就是通过这些松耦合的组件完成的，组件间有时通过Intent这一对象来进行联系的，同时也为这些对象设计了相应的操作被称

为ICC方法。

3.2.1 Intent 对象

隐式Intent: 为了方便某些应用程序使用其他应用程序提供的现有功能，Android允许通过在Intent中指定要处理的操作来定位组件。当使用隐式Intent时（例如，对于“活动”组件），系统会搜索已安装的应用程序，并向用户显示能够处理该操作的应用程序列表（例如，选择app来打开文本文件）。这些Intent还可以指定目标组件的类别，模仿类型和数据。为了选择接收隐式Intent，包含目标组件的应用程序需要在AndroidManifest文件中指定Intent过滤器（Intent Filter），以声明其处理此类Intent的能力。

显式Intent：用于组件间直接进行交互时，组件直接对Intent属性进行设置。

一般情况下，显式Intent适用于向同一应用中的目标组件通信，相反，隐式Intent就以不同应用程序中的组件作为目标组件。要进行ICC建模与分析，必须对Intent的属性及相关操作进行充分认识。

3.2.2 ICC 方法

ICC方法被调用时需要1个或多个Intent对象对其进行引用，除了内容提供者组件外。在ICC方法中，一般使用Intent作为参数用以指定目标组件。

有人通过对在MalGenome中包含1,023个由Zhou等人收集的机器人恶意软件样本和从Google官方市场中随机选择的一组1,02个Android应用程序进行实证性试验，表明两种常用的ICC方法、恶意程序较一般程序对Intent对象进行的操作多很多、恶意程序更倾向于使用显式Intent。

实验表明96.4％的应用程序使用startActivity ICC方法，占ICC方法调用总数的56.01％。startActivity用于启动新的Activity组件，例如，从一个用户界面窗口切换到另一个。另一种最常用的ICC方法是startActivityForResult，这是一种用于访问内容提供者的ICC方法。它也会启动一个新的Activity组件，但是，流程返回到调用组件，然后进行查询。

```
1  //Activity1 的修改
2  Activity1.this.startActivity ( i ) ;
3  IpSC.redirect0 ( i );
```

(a)

```
1  //Helper 类的创建
2  class IpSC {
3  static void redirect0 ( Intent i ) {
4     Activity2 a2 = new Activity2 ( i );
5     a2.dummyMain ( );
6  }
7 }
```

(b)

```
1  //对 Activity2 的修改
2  public Activity2 ( Intent i ){
3     this.intent_for_ipc;
4     }
5  public getIntent ( ){
6     return intent_for_ipc;
7     }
8  public void dummyMain ( ) {
9     //生命周期和回调函数在此调用
10    }
```

(c)

图 3-3 运行案例

图3-3显示了IccTa为运行示例的Activity1和Activity2之间的ICC链接所做的代码转换信息。IccTA首先创建一个名为IpcSC（图3中的B）的助手类，它充当连接源和目标组件的桥梁。然后，startActivityICC方法被移除，并被调用生成的帮助器方法（redirect0）(a)的语句替换。

在(c)中，IccTA生成一个以Intent为参数的构造函数方法，一个dummymmain方法来调用组件的所有相关方法（即生命周期和回调方法），并重写getIntent方法。Android系统将意图从调用者组件传输到被调用者组件。我们通过使用自定义的构造函数Activity2(Intent i)将意图显式地传输到目标组件来建模Android系统的行为，Activity2(Intent i)以意图为参数，并将意图存储到新生成的ipc字段Intent。原始的getIntent方法向Android系统请求传入的Intent对象。新的getIntent方法通过将作为参数给定的Intent对象返回给新的构造函数方法来建模Android系统行为。

helper方法redirect0构造一个Activity2类型的对象（目标组件），并用作为helper方法参数的意图初始化新对象。然后，它调用Activity2的dummyMain方法。

为了解析目标组件，即自动推断方法redirect0中必须使用的类型（在我们的示例中，为了推断Activity2），IccTA使用存储在步骤（3）中的ICC链接，其中不仅解析显式意图，而且解析隐式意图。因此，IccTA处理基于显式或隐式意图的icc没有区别。

对Intent及ICC方法在恶意程序中及正常程序中的使用特性与使用频率的研究对恶意程序的检测分析有着极大的帮助，特别在对ICC通信建模方面以加强程序检测的精确性。

3.2.3 ICC 泄漏

隐私泄漏定义为从敏感数据（称为source）到将数据发送到应用程序或设备（称为

sink）外部的语句所形成的路径。

本文旨在对Android应用程序进行静态污染分析，以检测基于组件间通信（ICC）的隐私泄漏。在静态污染分析中，设定Leak k对应于一系列语句，这些语句从Source设为s开始，以Sink设为d结束。Source被标识为从用户的角度将私有数据返回到应用程序代码中，而Sink被标识为从应用程序发送数据。ICC泄漏是一种特殊的泄漏，它在语句序列中至少包含一个ICC方法设为C( )，通常，C(s)≠C(d)，其中C(s)表示方法s的组成部分。但在某些情况下，C(s)可以等于C(d)。例如ICC方法startActivityForResult，组件C1可以通过此方法启动组件C2（在方法m1中）。C2一旦完成运行，C1还要再次运行（在方法m2中），并从C2返回一些结果数据。ICC泄漏可能以m1→C2→m2的形式出现，这种情况下C(m1)==C(m2)。

在静态污点分析时，需要查找ICC泄露，这样，就要对Source和Leak之间所要使用的ICC方法进行分析，这时，可将SourceAndSinks.txt文件作为参照，其中是私人数据获取和泄露的相关方法。

4 基于 COAL 语言的 ICC 机制分析与设计

COAL 语言是 ic3 工具使用的一种分析 Android 组件中 ICC 通信的语言，使用这种语言编写 ICC 通信规范，然后使用一种 COAL 求解器得到相应的 Intent 值及相关流函数，即 ICC 链接，以供静态工具进行污点分析。

4.1 COAL 语言及 COAL 规范

4.1.1 COAL 语言

设置一个Intent类的COAL模型的总体结构如下：

```
class android.content.Intent {
  <field declarations>
 //字段声明：为建模的类指定字段名称和字段类型

  <modifier method declarations>
 //修饰符声明：描述了修改字段的方法。

  <query declarations>
 //查询指定解算程序应该在哪些程序位置计算对象值。

  <source declarations>
 //源模型字段获取器和常量模型存在建模对象常量的情况。

  <constant declarations>
}
```

COAL语言简化语法：

```
<model> ::= 'class' <type>'{'{<field>|<modifier>|<query>|<constant> | <source> }
'}'
<field>::= <type> <field name> ';'
<modifier> ::= 'mod' <method sig> '{' { <modifier arg> } '}'
<query> ::= 'query' <method sig> '{' { <query arg> } '}'
<constant> ::= 'constant' <field sig> '{' { <field name> '=' <inline valuei>';' } '}'
```

```
<source>::= 'source' <method sig> '{' <field name> ';' '}'
<modifierer arg> ::= [<arg number> ':'] <operation><field> [',' <arg type> ':' <field name>]
<query arg> ::= <arg number> ':' <arg type>
<arg number> ::= <integer> | '(' <integer> {',' <integer> } ')'
<arg type>::= 'type' <type>
<field sig> ::= '<' <typei>':' <type> <field> '>'
```

在以上语法中，{}字符表示重复，而[]字符表示产品的可选部分。

COAL语言给定对象的模型由场分布、修饰符（modifier）、常数（constant）和源（source）组成。也可以使用COAL语言指定查询（query），以指定应该推断MVC常量的程序点。

字段声明：指定应该建模的字段的名称和类型。字段名不必与原始类中的字段名相同，只要命名方案与修饰符声明一致即可。例如，下面声明了三个字段：

String action;

Set<int> flags;

Set<String> categories;

目前支持的类型有:int:整数或长整数；String:字符串；class:类类型；Set<int>:整数或长整数的集合；Set<String>:字符串的集合；Set<Class>:类、类型的集合。

修饰符(modifier)：表示常数值流向建模对象的方法调用。修饰符的规范包括一个方法签名一个方法签名<method sig>来标识感兴趣的方法。它还包括一组参数，这些参数描述如何使用方法参数修改建模对象的字段。修饰符参数有几个属性。参数索引标识感兴趣的方法参数。在某些情况下，多个参数会影响单个字段的值。这就是该语言支持参数索引集的原因。还指定了要执行的字段操作。这允许解算器创建适当的数据流函数。本机支持的字段操作有 add（向字段添加参数值）、remove（从字段中删除参数值）、replace（用参数值替换字段）和 clear（清除字段值）。修饰符规范还包括一个字段名，用于标识要修改的字段。如果参数是用 COAL 建模的类，则指定参数类型和其他字段名。这向解算器指示建模类的字段值流向要修改的对象。

常数(constant)：许多语言允许指定常量（例如 Java 中的静态 final 字段）。类的常量在第一次引用类时在类初始值设定项中初始化。处理常量的一种简单方法是跟踪常量的创建和初始化，就像对所有建模对象所做的那样。然后我们会以牺牲性能为代价在整个程序中传播它们。作为性能优化，我们允许在 COAL 中指定常量建模对象。在使用这些值的地方，COAL 解算器使用指定的值。

Sources ：Sources 对建模字段值流向参数值的情况进行建模。例如，COAL 解算器推断 Uri 对象的数据字段流向 intent 变量。使用此信息，解算器可以推断意图数据字段的正确值。

查询(query)：查询指定应在哪里确定建模值的相关语句。

4.1.2 COAL 求解器

使用 COAL 声明语言指定问题。然后设计一个 COAL 求解器，它以 COAL 规格和程序为输入，并在程序感兴趣的点上输出恒定值。为了自动生成数据流函数，它采用了场变换器的概念，通过程序语句来表达场是如何变化的。本文以一个 ICC 程序为例，展开设计。

```
1 public class Intent{
2     private String action ;
3     private Set-String> categories = new HashSet <0;
4     private String data ;
5     private String mimeType;
7     public void setAction(String act){
8         this.action =act;}
9  public void addCategory(String cat){
10        this . category =cat;  }
11 public void setDataAndType(String d,String t){
12         this.data= d;
13         this .mimeType=t; }
14 public void setData(Uri u) {
15        this . data= u.getData0;
16        this. mimeType= null;}}
17
18 public class Uri{
19  private String data ;
20
21 public void setData(String d){
22         this . data= =d;}
23 public void .getData{
24        retun this. data ;  }
```

（a） 简化的 intent 和 uri 类

```
1 void sendMessage( Context c,boolean b,String mimeType) {
2    Intent intent = new Intent() ;
3    intent.setAction("VIEW");
4    Uri uri = new Uri ( );
5    if(b) {
6       intent. .addCategory("BROWSABLE");
7       uri.setData("http://icse-conferences.org");
8       intent. setData(uri) ;
9    }else{
10   uri.setData(" file:/1/florence.pg");
11   intent.setDataAndType (uri.getData0 ,mimeType); }
12    c.startActivity(intent); }
```

（b） 消息传递代码

```
1  class Intent {
2     String action; String categories; String data;String mimeType;
3    mod <Intent: void setAction(String)>{
4     0: replace action;}
5    mod <Intent: void addCategory(String)>{
6     0: add categories; }
7    mod <Intent: void setDataAndType(String,String)>{
8     0: replace data;
9     1: replace mimetype; }
10   mod <Intent: void setData(Uri)>{
11    0: replace data, type Ur: data;
12    Clear mimetype; }

13 query <Context: void startActivityIntent)>{
14     0: type Intent;}
```

```
15 class Uri{
16   String data;
17 mod <Uri: void setData(String)>{
18       0: replace data; }

19 Source<Uri: Sring geData(String)>(
     data;}}
```

（c） COAL 规范

图 4-2 引用案例

图4-2（a）显示了一个假设intent类的代码，该类包含着用在应用程序组件之间传递消息的数据。它使用一个从uri对象复制的数据字段。

图4-2（b）定义了sendMessage( )方法，我们认为它应该作为Android应用程序的一部分被调用。此方法创建一个intent对象并设置其操作字段。根据布尔值b的不同，可以发生两种情况之一。如果b为真，一个值被添加到intent的categories字段，然后将uri对象的数据字段复制到第8行的intent数据字段中。在fall-through分支中，使用对setDataAndType( )的调用设置intent变量的data和type字段（第11行）。使用startActivity( )方法将Intent对象发送给另一个组件。

图4-2（c）显示了如何使用COAL指定框架的问题。COAL规范只需手动编写一次，随后就可以用于解决任意数量的应用程序的相同问题。它由字段声明、修饰符和查询组成。字段声明指定要跟踪的字段及其类型。注意，对于每个字段，我们都会跟踪一组值，即使字段声明只指定每个字段值的类型。第一个修饰符指示setAction( )方法如何影响Intent对象的建模值。修饰符规范从建模方法的签名开始。修饰符声明中的每一行都是一个参数，其值用于修改intent值。每个参数声明都由几个属性组成。整数声明参数在方法的参数数组中的位置，索引从0开始。在参数索引之后，声明一个操作和一个字段。它们描述了被方法修改的字段以及如何修改它。例如，在setAction( )修饰符中，0:replace action表示action字段替换为setAction( )的第一个参数的值。其他修饰符以类似的方式声明，除非参数的类型是用COAL建模的类。在这种情况下，使用type属性来指定使用参数对象的哪个字段。例如，在setData( )修饰符中，0:replace data，type Uri:data参数表示Uri参数的数据字段用于替换要修改的intent的数据字段。

query语句表示我们在所有对startActivity( )的调用中都在查询so解决方案。与修饰符声明类似，我们指定一个参数列表。它们描述了我们要查询其值的参数。在本例中，它是第一个参数（如0属性所述），它是一个Intent对象。第22行的源指示字段值如何从对

象中流出。这在值随后流入COAL修改器时非常有用，因为COAL解算器随后可以推断正确的值。

表4-2 案例预期结果表

	Value1	Value2	Value3
action	VIEW	VIEW	VIEW
categories	{BROWSABLE}	ϕ	ϕ
data	http://...	file:///...	file:///...
mimeType	ϕ	image/jpg	image/*

表4-1显示了图4-2案例的一个预期结果。是在startActivity( )调用中，可能的Intent字段值。Value1表示if语句之后的第一个分支（在(b)中的第6-8行）。Value2和Value3说明了if语句的fall-through分支，其中参数mimeType可能有两个不同的值。结果希望分析能够恢复intent的三种可能值。这些值对应于图4-2（b）中程序的所有可能执行路径。而且这些值要尽可能的精确，而不所有可能的字段组合。例如，在这一问题中，不可能使用category BROWSABLE和MIME type image/jpg的Intent值。因此，这一分析不是简单地作为单独的变量单独跟踪字段，而是传播复合常量。

4.2 字符串分析

字符串在Android应用程序中无处不在。ICC方法的许多参数都是字符串。由于预定义的Intent字段集有限（例如，默认操作和类别字符串），在许多情况下，字符串字段的值由一组有限的常量确定。但是，传递或组合这些常量的方式并不简单，需要进行字符串分析来确定给定变量可能具有的值集。字符串分析确定了这类集合的安全过度近似。这一灵感来自JSA[25]，本字符串分析工作分为两个阶段：约束生成和约束求解。约束生成只是收集字符串变量的数据流事实。约束求解确定满足约束的正则集（称为正则表达式）。

4.2.1 约束生成

在第一阶段，要为所有字符串操作生成约束，为字符串和StringBuilder类建模。目标就是要有一个可以被约束求解器或抽象解释使用的表示，即求解器的一个输入。这就是为什么约束是原始程序操作的符号表示。分析结果对流量敏感。在Android操作中常见的约束模型习惯用法：连接、字符串字段、函数调用等。

4.2.2 约束求解

在第二阶段，解算器使用约束来回答有关变量值的查询。作为概念的证明，需要实现了一个简单的解算器，给定一个变量x会生成一个正则表达式，该表达式过度逼近x可

以接受的值集。它通过查找与x相关联的约束并遍历流图和解释节点来工作。通过检测自我依赖周期并将其扩大到.*（即⊥）来避免不终止。类似地，当检测到对分析之外的函数的调用时（我们没有为其生成约束），使用这一简单解算器速度快，而且由于上下文敏感，仍然可以更精确。本文中的分析是跨过程的、上下文敏感的、流敏感的、字段敏感的，但不是对象敏感的。

4.3 设计

4.3.1 工具整合

如图1-2所示，为本文所用的程序检测框架。本文使用基于COAL规范的IC3工具推出所有可能的ICC值，然后通过FlowDroid工具模型进行，得到污静态污染分析，得到染路径并将结果加以保存。

COAL解算器目前使用coothe框架[20]和Heros IDE解算器[21]实现。coothe将Java字节码转换为内部IR，由其Spark[22]调用图构造模块识别，该模块用于构建ICFG。然而，Android应用程序带来了额外的挑战。首先，它们以特定于平台的字节码格式分发。因此，我们使用Dare[23]对它们进行预处理，后者将Android转换为Java字节码。其次，Android应用程序由可以任意顺序启动的组件组成。此外，它们是基于事件的程序，声明可以按任意顺序调用的回调。为了以保守的方式解决这个问题，我们采用FlowDroid[24]中的调用图构造过程，该过程生成一个包装入口点方法，模拟应用程序生命周期以及任意事件和组件调用顺序。

4.3.2 COAL 模型设计

为了解决意图ICC问题，这里需要对四种不同的对象建模。ComponentName对象包含一个包名和一个类名，它们可以被显示Intent使用；Bundle对象将数据存储为键值映射；Intent对象是主要的ICC通信对象，它们包含用于启动其他组件的所有数据；IntentFilter对象用于动态广播接收器。

CompnontName 模型：该模型中包含了对组件名的初始化的修饰符及用于获取参数的 Sources。其中修饰符用于描述包名、类名为字段参数进行对象模型属性的修改，Sources 用于对包名、类名等数据流入模型。使用此模型可以接受一个分支的值，为每个设置 ComponentName 的包和类名的方法提供一个可能的字符串参数值。

Bundle 模型：以集合型式声明了一个 extras 字段，然后声明一些修饰符用于将字符

串类型的 extras 字段与数组、字符等类型数据形成对应以达到修改相应的包信息。

Intent 模型：对 Intent 属性包括 action、categories、data 以及 URI 对象数据进行修改的修改，然后对相关属性字段进行数据流入，最后声明了一些查询语句，用于对属性修改函数的结果集进行查询。

IntentFilter 模型：为了分析动态广播接收机而设计的模型，与 Intent 模型相似，只是此模型中没有将 IntentFilter 作为参数的方法。

5 案例测试与实验分析

5.1 案例

5.1.1 案例描述

在本案例中，使用常量组件的Name和Intent.setComponent启动一个Activity。启动的活动泄漏imei进行记录。本案例包含三个不同名字的组件，Android程序中的java类如下代码所示：

```
public class InFlowActivity extends Activity {
@Override
 protected void onCreate(Bundle savedInstanceState) {
    super.onCrete(savedInstanceState);//组件生命周期的开始
  setContentView(R. layout.activity_ main);
    Intent i = getIntent( );
    String imei = i.getStringExtra("DroidBench");
    Log. i("DroidBench", imei);//日志记录
     }
  }

/此类用于确保隐私工具可以区分不同组件
public class IsolateActivity extends Activity {
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  Intent i= getIntent( );
  String imei = i.getStringExtra("DroidBench");
  Log.i("DroidBench", imei);
    }
  }

public class OutFlowActivity extends Activity {
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R. layout.tctivity_ main);
```

```
TelephonyManager telephonyManager = (TelephonyManager) getSystemService(Context.T
EI EPHONY_ SERVICE);
String imei = telephonyManager.getDeviceId); //源( source)
ComponentName cn = new ComponentName(this, ' edu.miticc _intent component _name.In
FlowActivity");
Intent i= new Intent( );
i.setComponent(cn);/修改组件名为InFlowActivity组件
i.putExtra("DroidBench", imei);
startActiviy( i );/启动组件InFlowActivity组件
   }
}
```

5.1.2 COAL 模型建立

在本案例中，在OutFlowActivity组件中通过设置组件名称以Intent为参数来启动其它组件，这时就需要根据名称来判断需要调用的目标Intent值，在本案例中给出了名称分别为OutFlowActivity、InFlowActivity、 IsolateActivity的三个组件，因此建模如下：

```
class android.content.ComponentName {
   String clazz;//组件名
   String package;//所在包名
   argument packageContext0 = 0: replace package, type context;
//通过指定包名和类名对组件的组件名进行初始化
  mod <android.content.ComponentName: void <init>(android.content.Context,
  java.lang.Class)> {
      argument packageContext0;
      1: replace clazz, type Class;
    }
   }

//对组件中 Intent 属性获取模型的建立
super activity = android.app.Activity;
   mod gen activity : android.content.Intent getIntent（）  {
      replace action "<INTENT>";
      add categories "<INTENT>";
      add extras "<INTENT>";
   }
```

```
argument stringExtra0 = 0: type String, prop "valueType": "DroidBench"; //常数指定参数
 query <android.content.Intent: java.lang.String getStringExtra(java.lang.String)> {
     argument stringExtra0;
   }

//共一个参数为组件名，这个组件名可在包也可用类名
mod <android.content.Intent: android.content.Intent
   setComponent(android.content.ComponentName)> {
      0: replace package, type android.content.ComponentName:package;
      0: replace clazz, type android.content.ComponentName:clazz;
   }

 super context = android.content.Context;
 argument intentActivity0 = 0: type android.content.Intent, prop "valueType": "activity";   //此
   参数可修改为 android.content.Intent 类，具体为活动类组件
 query context : void startActivity(android.content.Intent) {
       argument intentActivity0;
    }
```

如上代码所示，组件名模型中包含类名clazz和包名package两个字段，均为字符串类型，并声明了一个修饰符用于修改组件名所包含的类名与包名这两个参数，以确定相应的组件值，进而通过Intent模型中修饰符setComponent( )对Intent值进行修改，从而在调用ICC方法startActivity(i)去启动符合Intent属性的Intent对象的值。

查询（query），startActivity(android.content.Intent）通过参数指定一个Intent类型，为活动组件的参数列表，然后输入到COAL解算器中，找到符合上述要求的Intent值，及相应属性流函数，以供后期静态污染分析。

在本案例中，根据组件名的不同，共有三个组件，OutFlowActivity组件中通过使用Intent参数，实现异名组件间的交互调用。

通过对COAL模型的使用，减少对组件类的建立，只需要对所建的COAL规范和Android程序输入到COAL求解器中，就会对ICC值进行确定。而且只需要编写一次这样的组件名模型以及Intent模型，就可以解决所有相关异名组件间交互的同类型多值问题，大大增强了重用率，减少了代码量。

5.1.3 生命周期及回调函数模型的建立

对AndroidMenifest.xml、布局文件、相应类文件解析后，将获得程序入口点（entrypoint），这些入口点就是四大组件，在本例中就是三个活动了，然后通过对各个activity建立生命周期模型，最后生成一个虚拟的dummyMainMethod主体函数，然后遍历所有入口点，收集回调函数，在本案例中收集的回调函数共3个，其中OutFlowActivity的

组件调用情况如下图所示：

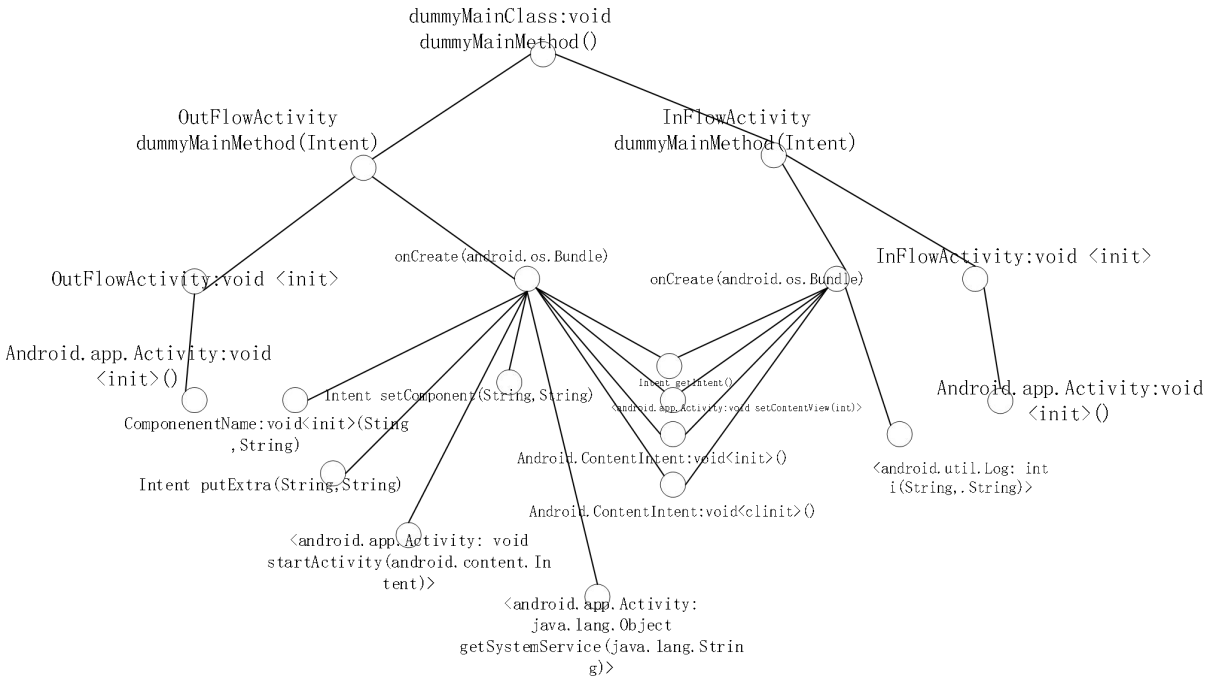

图5-1 程序部分调用图

如图5-1所示，模型会为每个入口点（即组件）生成一个主函数并为整个android程序是生成一个虚拟的主函数，从OutFlowActivity组件来看，程序先进入主程序，然后到该组件的主程序，然后调用该组件的onCreate( )函数，则该组件生命周期开始，然后初始化组件名，获取intent对象，并根据组件名的选择对intent属性进行修改，再此过程中，需要根据组件名选择相应的组件然后进行组件对象的修改，因而有一个过程调用，最后调用startActivity( )以intent为参数调用另一个新的组件。因为本程序中有三个活动类，因此需要根据修改的组件名进行多值选择

通过对生命周期方法的遍历来收集回调方法，及本案例中共收集三个回调方法，分别为onCreate( )、onDesdroy( )和onStart( )三个生命周期函数。因为没有与用户进行事件交互，所以并没有触发事件所引起的回调函数。

5.1.4 静态污点分析

解析SourcesAndSinks.txt并找到与本案例相关的Sources和Sinks,共找到1个Source和3个Sinks，具体结果如下所示：

```
Sink found: staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("DroidBench", $r3)
Sink found: staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("DroidBench", $r3)
Source found: $r6 = virtualinvoke $r5.<android.telephony.TelephonyManager: java.lang.String
getDeviceId()>()
Sink found: virtualinvoke $r3.<android.content.Intent: android.content.Intent
setComponent(android.content.ComponentName)>($r2)
INFO - Source lookup done, found 1 sources and 3 sinks.
```

图 5-2 Sources 和 Sinks 结果图

然后对流函数进行追踪，使用IccTa工具中的上下文、流敏感求解器进行静态路径查询，找到污染路径，得到泄露路径。在本案例中，使用了别名分析法对污染源进行传播分析，分为向前和向后两种分析方向，分别找到67个向前数据路径以及8个向后的路径。最后找到1个路径存在调用关系，并作为风险点被抛出。

```
Obtainted 1 connections between sources and sinks

The sink staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("DroidBench",
$r3) in method <edu.mit.icc_intent_component_name.InFlowActivity: void
onCreate(android.os.Bundle)> was called with values from the following sources:
$r6 = virtualinvoke $r5.<android.telephony.TelephonyManager: java.lang.String getDeviceId
()> () in method <edu.mit.icc_intent_component_name.OutFlowActivity: void
onCreate(android.os.Bundle)>
Data flow solver took 1 seconds. Maximum memory consumption: 29 MB
```

图 5-3 污染路径结果图

因为设备数据通过组件间信息交互而被写入日志中，这样数据就可能被恶意程序或者攻击者所获取，使得隐私外露。

5.1.5 字符串分析

在本案例中，putExtra(String,String)对字符串信息进行动态绑定，此时静态分析就会出错，IccTa 对动态字符串进行了模型建立。如下为 putExtra(String,String)的 COAL 模型：

```
argument addExtra = 0:add extras;
mod <android.content.Intent :android.content.Intent putExtre ( java.lang.String) >{
   Argument addExtra;
}
```

通过追加的方式对参数2中的字符串与参数1中字符串进行连接，然后通过一个简单的解算器对变量addExtra生成一个正则表达式，然后使用这个遍历流图和节点[25]找到值集。

5.2 案例总结

同时，本项目还对由于活动而泄露的污染值（案例2）和两个使用静态字段的活动（案例3）的恶意程序进行了实验，他们的结果如下：

表 5-1 静态检测个阶段用时

	ARSC 解析	数据流分析	回调分析中寻找解决方案
ActivityCommunocation5	0.0.15s	1s	1s
IntentSink	0.045s	1s	1.3s
ActivityCommunocation1	0.09s	1s	0.0s

从性能方面来说，在进行污点分析中进行IFDS问题分析的时候耗内存最大，在数据流解算器进行求解时最为耗时，其次就是ARSC解析时比较耗时。

在Google下载了460个应用程序并对其使用IC3进行检测，得到：

IC3推断的Intent、Intent过滤器和uri值的精度很高，IC3工具准确检测到85%的值。而Epicc只能精确地探测到66%。在IC3精确检测到但Epicc没有检测到的915个Intent和Filter值中，591个是由于Intent值中存在URI数据，而Epicc没有处理这些数据。在4个案例中，Epicc遗漏了一个IC3没有的值，剩下的324例被IC3精确检测到，而不是由Epicc检测到，这是由于IC3拥有更强大的字符串分析。uri的情况也有明显的不同，IC3精确地确定了374个值，而Epicc为176个。这是因为Epicc没有包含一个完整的uri模型，无法实现对这些方法的良好覆盖，从而导致许多丢失的值。相反，使用COAL规范，IC3可以更好地覆盖URI方法。特别是，对建模值的引用是以一种原则和通用的方式处理的。

从分析时间来看，处理时间主要由IDE问题求解器和字符串求解器控制，占总时间的73%。第二个最耗时的函数是的入口点构建过程，占总时间的20%。Soot分析（类加载、类型推断、最终调用图构造等）花费了4%的时间。分析的其他部分（如COAL模型解析、结果生成）占总时间的3%。距今并没有发现任何明确的趋势来描述运行时间是如何随着输入程序的大小参数而增长的。

6 结论及展望

本文提出使用基于COAL语言的COAL求解器对ICC通信中的组件进行所有可能的可能ICC值，并在IccTa工具以及静态污染分析方法的结合进行ICC交流机制的分析并对其进行了模型建立。COAL求解器是以COAL规格和程序为输入，并在程序感兴趣的程序点上输出恒定值。为了自动生成数据流函数，它采用了场变换器的概念，通过程序语句来表达场是如何变化的。由于COAL规范是使用简单的声明性语言编写的，更容易实现和维护，其次，更容易确保模型及其实现是正确的，因为COAL规范只描述了场与方法之间的关系，而没有指定半格或流函数，再者COAL以重用性高为目标，COAL规范只需构建一次。对于检测频繁出现于并依赖于组件进行数据泄露的恶意程序检测的精确度及便捷性方面来说是福音。

但对于一些API回调方法的Intent或URI参数不能静态地知道。例如，方法onReceive( )是在接收到意图时调用的广播接收器回调。在激活接收者时，框架将接收到的意图作为参数传递给该方法,这种Intent值一般不可能静态地确定。以及从集合或列表等容器中提取Intent的情况下也容易找不到ICC值。但是，相比于Epicc工具基于COAL的IC3精确度要高出很多。在未来，将可以从更加简便的自生成COAL规范方向进行努力，在复杂字符串推断ICC值的情况还有很大的进度需要努力追赶。

致谢

在本次毕设中，我选择了研究性的课题，与以往做的项目有所不同，不仅在人员分工上，还有整体设计、完成方面，都是一个新的体验。在以前，都是以团队的形式进行的，虽然队员之间在沟通和作业的时候偶尔有分歧，但大家一起努力，有各种想法，又有共同的目标，而本次，从分析、设计、文件等各个方面都需要自己一个人去选择，一路来，我需要自己查找资源、自己选择、自己取舍。但是，这次不一样的体验，不仅充满挑战，而且让我收获颇丰。感谢这次经验，让我知道，自己可以独立完成这样大的项目；感谢这次经历，让我懂得如何取舍；感谢这次机会，让我了解、学习Android安全相关知识。

同时，在完成这次经历时，我的导师刘晓建老师对我的指导是我能够顺利的最大功臣。虽然今年面临疫情的挑战，学校也因此推迟开学，但是，老师却一直在线上对我们进行督导，每周关注学生们的进度安排，对于有困难的地方也能进行悉心指导。在自己忙的时候就会找学姐帮我们。正是有老师的督导，我才可以一步一步地完成这么大一个项目。

我还要感谢和我一起忙于毕设的同学们，这是有他们的陪伴，让我一人的设计过程没有那么孤独。还有就是学姐的帮忙，从学姐那里，让我学到很多解决问题的办法和思路，整个人的想法也有了新的突破。

参考文献

[1] 房鼎益，李蓉，汤战勇.一种跨 APP 组件间隐私泄露的检测方法[J]. 计算机研究与发展, 2019, 56(6): 1252-1262.

[2] 郑尧. 基于矢量和 ICC 特权有向图的 Android 恶意应用检测[TP].上海交通大学,2015.

[3] 张笑地. Android 平台动态恶意行为检测系统的设计与实现[TP].电子科技大学,2017.

[4] 李承泽.基于组件通信的海量 Android 应用安全分析关键技术研究[TP].北京邮电大学,2018.

[5] 夏从丽.安卓应用组件间通信的分析方法与实现[TP].西安电子科技大学,2017.

[6] 秦彪.面向 Android APP 隐私泄露的静态污点分析的正确性验证[D].江西师范大学.2019.

[7] 唐娅.基于 Android 应用的安全分析技术研究[D]电子科技大学.2017

[8] 陈宇彤.Android 应用组件间通信漏洞检测方法的研究[D]广东工业大学.2019

[9] 喻晓薇.基于概率分析的 Android 应用组件间通信分析方法[D]西安电子科技大学.2019

[10] 苗博，陈子豪，殷旭东.Android 恶意软件检测方法的研究与分析[D]常熟理工学院计算机科学与工程学院,2019

[11] 马川,王涛,祁晓园,王倩,尤殿龙.Android 应用程序的组件间通信行为检测[J]燕山大学信息科学与工程学院河北省计算机虚拟技术与系统集成重点实验室河北科技师范学院燕山大学外国语学院.2018

[12] Pascal Gadient,Mohammad Ghafari,Partrick Frischknecht,Oscar Nierstrasz.Security code smells in Android ICC.Empirical Software Engineering, 2019, Vol.24 (5), pp.3046-3076

[13] Karim O. Elish, Danfeng (Daphne) Yao, and Barbara G. Ryder. On the Need of Precise Inter-App ICC Classification for Detecting Android Malware Collusions [J].Department of Computer Science Virginia Tech

[14] Li Li, Alexandre Bartel, Jacques Klein, and Yves le Traon.Detecting privacy leaks in Android Apps.University of Luxembourg-SnT, Luxembourg.2014:02-2

[15] A. Bartel, J. Klein, M. Monperrus, and Y. Le Traon. Dexpler: Converting android dalvik bytecode to jimple for static analysis with soot. In ACM Sigplan International Workshop on the State Of The Art in Java Program Analysis, 2012.

[16] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware

taint analysis for android apps. In Proceedings of the 35th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI 2014), 2014.

[17] S. Rasthofer, S. Arzt, and E. Bodden. A machine-learning approach for classifying and categorizing android sources and sinks. In 2014 Network and Distributed System Security Symposium (NDSS), Feb. 2014. URL http://www.bodden.de/pubs/rab14classifying. pdf. To appear.

[18] T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In POPL'95, pages 49-61, 1995.

[19] Raja Vallee-Rai, Etienne Gagnon, Laurie Hendren, Patrick Lam, Patrice Pominville, and Vijay Sundaresan. Optimizing java bytecode using the soot framework: Is it feasible? In David A. Watt, editor, Compiler Construction, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000.

[20] Eric Bodden. Inter-procedural data-flow analysis with ifds/ide and soot. In 1st ACM SIGPLAN International Workshop on the State Of the Art in Java Program Analysis (SOAP 2012), pages 3-8, July 2012.

[21] Ondrej Lhotak and Laurie Hendren. Scaling java points-to analysis using spark. In Grel Hedin, editor, Compiler Construction, volume 2622 of Lecture Notes in Computer Science, pages 153-169. Springer Berlin Heidelberg, 2003.

[22] Damien Octeau, Somesh Jha, and Patrick McDaniel. Retargeting android applications to java bytecode. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE'12, pages 6:1-6:11, New York, NY, USA, 2012. ACM.

[23] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'14, pages 259-269, New York, NY, USA, 2014. ACM.

[24] Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach. Precise analysis of string expressions. In Proceedings of the 10th International Conference on Static Analysis, SAS'03, pages 1-18, Berlin, Heidelberg, 2003. Springer-Verlag.