

2020 届硕士学位论文

分类号: _____ 学校代码: 10269

密 级: _____ 学 号: 51174506026



华东师范大学

East China Normal University

硕士 学位 论文
MASTER'S DISSERTATION

论文题目:

面向工业界仿冒应用的大规模实证研究

院 系: 计算机科学与技术学院

专业名称: 计算机科学与技术

研究方向: 软件方法与程序语言

指导教师: 贺樑 教授

学位申请人: 唐崇斌

2020 年 5 月

Thesis for master's degree in 2020

University Code:

10269

Student ID:

51174506026

EAST CHINA NORMAL UNIVERSITY

A LARGE-SCALE EMPIRICAL STUDY ON INDUSTRIAL FAKE APPS

Department: School of Computer Science and Technology

Major: Computer Science and Technology

Research Direction: Software Method and Programming Language

Supervisor: Prof. Liang He

Candidate: Chongbin Tang

May, 2020

华东师范大学学位论文原创性声明

郑重声明：本人呈交的学位论文《面向工业界仿冒应用的大规模实证研究》，是在华东师范大学攻读硕士/博士（请勾选）学位期间，在导师的指导下进行的研究工作及取得的研究成果。除文中已经注明引用的内容外，本论文不包含其他个人已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中作了明确说明并表示谢意。

作者签名:_____

日期: 年 月 日

华东师范大学学位论文著作权使用声明

《面向工业界仿冒应用的大规模实证研究》系本人在华东师范大学攻读学位期间在导师指导下完成的硕士/博士（请勾选）学位论文，著作权归本人所有。本人同意华东师范大学根据相关规定保留和使用此学位论文，并向主管部门和学校指定的相关机构送交学位论文的印刷版和电子版；允许学位论文进入华东师范大学图书馆及数据库被查阅、借阅；同意学校将学位论文加入全国博士、硕士学位论文共建单位数据库进行检索，将学位论文的标题和摘要汇编出版，采用影印、缩印或者其它方式合理复制学位论文。

本学位论文属于（请勾选）

- () 1. 经华东师范大学相关部门审查核定的“内部”或“涉密”学位论文 *，于年 月 日解密，解密后适用上述授权。
() 2. 不保密，适用上述授权。

导师签名:_____

本人签名:_____

年 月 日

* “涉密”学位论文应是已经华东师范大学学位评定委员会办公室或保密委员会审定过的学位论文（需附获批的《华东师范大学研究生申请学位论文“涉密”审批表》方为有效），未经上述部门审定的学位论文均为公开学位论文。此声明栏不填写的，默认为公开学位论文，均适用上述授权）。

唐崇斌 硕士学位论文答辩委员会成员名单

姓名	职称	单位	备注
A	教授	华东师范大学	主席
B	副教授	华东师范大学	
C	副教授	华东师范大学	

摘 要

作为市场占有率最高的智能手机操作系统，安卓系统吸引了无数开发者为其开发应用，也使各种各样与安卓应用及其研发相关的研究得以开展。然而，在浩如烟海的安卓应用研究中，针对仿冒应用的研究仍相当有限。有别于官方发布的正版应用，仿冒应用属于移动灰色产业的一环，其目的各异，难以一概而论，其行为特征更是不得而知。由于我们对移动灰色产业知之甚少，仿冒应用的产业链及其生态对我们仍是一个谜。

为了填补这一部分的空白，我们从现实的安卓应用市场中爬取了大量真实数据，对一批从工业界中找到的仿冒应用进行了已知的首个系统、全面的大规模实证研究。由于仿冒应用的模仿对象往往是较为热门的应用，我们按照知名数据分析机构公布的排行榜确定了全网最受欢迎的前 50 款应用，然后使用网络爬虫搜集了与这批热门应用相关的超过 150,000 个应用样本作为研究对象并收录进数据库，以进行全面的研究。

本文呈现了我们对这批样本从三个不同视角进行探究的结果，其分别为：仿冒应用的基本特征，针对仿冒样本的量化分析，及仿冒应用的发展轨迹。三个视角由浅入深，从仿冒应用的应用名、包名和 APK 包大小等基本信息特征开始测量，再将仿冒应用数据与其来源的应用市场结合分析，最后引入时间因素对数据进行挖掘。从三个不同视角的分析中，本文提供了包括仿冒应用命名倾向、仿冒应用作者对应用市场拦截的规避策略等珍贵的领域知识。本文还对数据中较为特别的样本作出了详尽的案例分析，除了可以印证上述的领域知识与发现之外，也能引起我们对现今应用市场生态环境的思考。

最后，我们又针对仿冒应用在应用市场上获得的评级、评论等用户反馈进行了一系列的分析与验证。

我们希望本文可以为读者提供一个面向仿冒应用及其生态的清晰视角，并借此抛砖引玉，吸引更多科研人员投入到对安卓灰色产业的观察研究中。

关键词: Android 应用程序, 仿冒应用, 实证研究, 数据分析

ABSTRACT

As a smart phone OS with the highest market share, Android has attracted countless developers to develop apps on it, and enabled a variety of research related to Android apps and their development. While there have been various studies towards Android apps and their development, there is limited discussion of the broader class of apps that fall in the fake area. Fake apps and their development are distinct from official apps and belong to the mobile underground industry. Due to the lack of knowledge of the mobile underground industry, fake apps, their ecosystem and nature still remain in mystery.

To fill the blank, we conduct the first systematic and comprehensive empirical study on a large-scale set of fake apps. Over 150,000 samples related to the top 50 popular apps are collected for extensive measurement.

In this paper, we present discoveries from three different perspectives, namely, fake sample characteristics, quantitative study on fake samples and fake authors' developing trend. The three perspectives follow a easy-to-complex pattern. We start from the basic pattern of fake apps, then combine the fake app data with their source app market, and lastly introduce time factor to mine the data. As a result, the three perspectives provide us with valuable domain knowledge, like fake apps' naming tendency and fake developers' evasive strategies. Moreover, we provide a number of thought-provoking case studies, confirming the findings mentioned above. Last and not least, we collect, analyze and verify a series of fake apps' feedback from the market, making our study more complete.

We hope this paper can provide the readers with a clear vision of fake apps and their ecosystem, and thus raising more researchers' interest in observing and studying the Android underground industry.

Keywords: *Android application, Fake App, Empirical Study, Data Analysis*

目 录

摘要	i
ABSTRACT	iii
第一章 绪论	1
1.1 研究背景	1
1.2 国内外研究现状	3
1.2.1 针对灰色应用的实证研究	3
1.2.2 针对恶意应用生态系统的实证研究	3
1.2.3 重打包应用检测	3
1.3 论文研究意义	4
1.4 论文研究内容	4
1.5 本研究遇到的困难与挑战	5
1.6 本文组织结构	6
第二章 Android 背景介绍	7
2.1 Android 系统介绍	7
2.2 Android 应用程序	8
2.2.1 应用程序简介	8
2.2.2 构建应用程序	9

2.3	Android 应用市场	10
2.4	第三方应用市场	12
2.5	Android App 签名机制	15
2.6	本章小结	16
第三章	研究概览	17
3.1	本文研究流程	17
3.2	数据收集	18
3.3	本章小结	20
第四章	大规模实证研究与发现	21
4.1	仿冒应用的基本特征	21
4.2	针对仿冒样本的量化分析	26
4.3	仿冒应用的发展轨迹	31
4.4	本章小结	35
第五章	总结与展望	36
5.1	总结	36
5.2	展望	37
	参考文献	38
	致谢	42
	攻读学位期间发表的学术论文	43

插 图

图 1.1 Google Play 应用商店架上应用总数变化趋势	1
图 2.1 2009 至 2020 年移动端操作系统市场份额变化图	7
图 2.2 Android App 构建流程	9
图 2.3 Google Play 应用商店首页（从桌面端浏览）	12
图 2.4 2018 中国第三方移动应用商店用户首选使用品牌分布	13
图 2.5 腾讯应用宝应用市场首页（从桌面端浏览）	14
图 3.1 本文研究流程	17
图 3.2 Janus 平台上的数据规模时序图	19
图 4.1 对 App 各项属性的统计结果	24
图 4.2 各大应用市场应用详情页（从桌面端浏览）	25
图 4.3 从不同第三方应用市场中收集到的应用数量以及各市场仿冒率	27
图 4.4 仿冒延迟总体分布	32
图 4.5 仿冒应用安全证书存活时间分布	33
图 4.6 每季度爬取到的仿冒样本数量（2015 年第四季度到 2018 年第三季度）	34

表 格

表 4.1 安全证书/仿冒应用数量对应表	22
表 4.2 目标 App 与其相关统计	29

第一章 绪论

1.1 研究背景

随着移动市场于近年逐渐兴起，Android 系统作为一个主流的移动端操作系统也在蓬勃发展。根据数据分析机构 StatCounter 的资料显示，从发布之日起，Android 的市场占有率就在逐年稳步增长。截至 2020 年，Android 系统已经占据了 74.3% 的全球移动端市场份额^[1]。与此同时，Android 应用的数量也伴随着 Android 市场的蓬勃发展节节攀高。仅看 Android 官方的应用商店 Google Play，其在 2017 年一年中就新上架了近一百万个可供下载的应用程序。虽然因为各种原因，Google Play 上的应用数量在 2018 年有所回落，但如图 1.1 所示，应用市场上目前仍有近三百万个可用的应用程序，Android 应用市场依然充满活力^[2]。

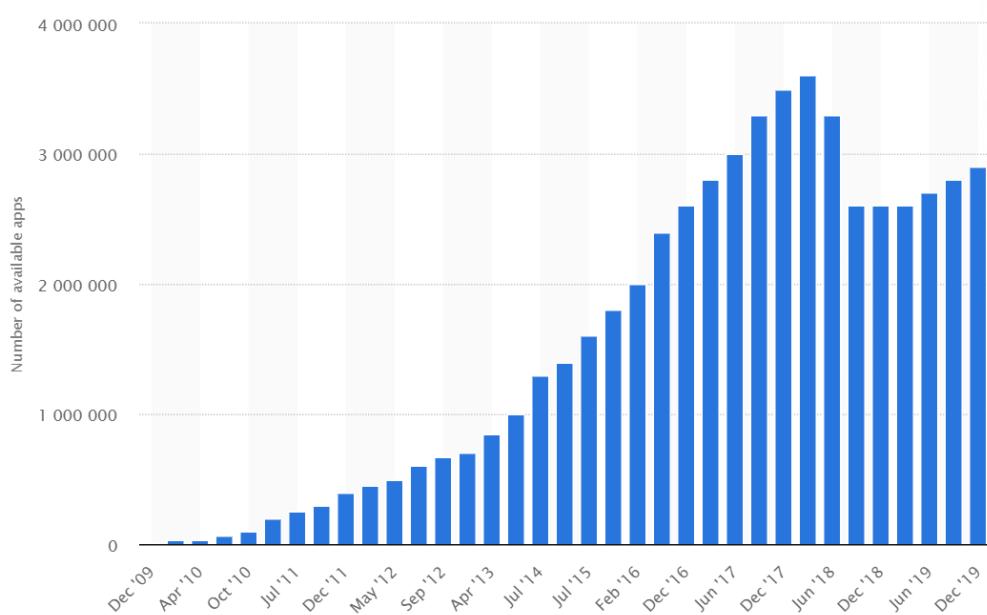


图 1.1: Google Play 应用商店架上应用总数变化趋势

伴随着应用数量爆发式增长的，还有欣欣向荣的移动黑产。仿冒应用构建了移动黑产产业链中十分重要的一环。本文提及的仿冒应用指代模仿市面上热门的应

用、甚至外观与热门应用相差无几的移动应用，其目的是诱导用户下载以赚取流量，甚至是窃取用户信息、触发有害行为从而盈利。根据的前期观察，作者发现仿冒应用以两种不同的形式出现。第一类为模仿应用，这类应用具有和原版热门应用相似的外观（比如名字、图标等），诱导用户下载。而第二类，高仿应用^[3,4]，已不单单是与原应用“相似”了，这类应用采用和原应用一模一样的外观，乃至连版本号都相同，其中的部分应用就是直接通过对原应用重打包做成的。

这些仿冒应用不仅大大损害了原应用开发者的利益，也侵犯了用户的权益。我们可以假设一个十分普遍的场景：当用户尝试通过应用市场搜索安装一个应用，应用市场往往会返回多个无论是名字或是图标都十分相像的结果，在这种情况下，用户十分有可能安装一个仿冒应用。就算用户最后通过某些途径安装上了原版应用，也浪费了时间和人力成本，遑论某些仿冒应用中潜藏着恶意行为，一旦被触发就会对用户造成更大的损害。于是，用户搜索与下载应用时的安全和体验就被仿冒应用严重影响了。

更糟糕的是，随着开发移动应用的门槛逐渐下降，开发一个仿冒应用的成本已经远低于开发一个桌面级应用所需的成本，这为地下产业涌入移动端发展提供了绝佳的温床^[5]。此外，移动应用功能在实现上的灵活性^[6]也增加了仿冒应用的复杂度，让分析移动应用变得更加困难。

尽管如今仿冒应用随处可见，我们对仿冒应用和他们的生态却依然知之甚少——他们有何共同特征、数量多少、迭代速度如何、以及他们如何规避应用市场检测等问题依然有待解答。遗憾的是，如今关于移动应用的学术研究大多关注于恶意应用的检测技术^[7-10]。据作者所知，目前还未有任何关于仿冒应用或其生态系统的研究。类似地，工业界中也鲜见关于仿冒应用的课题。多数分析与威胁报告都聚焦于恶意应用上，忽略了仿冒应用^[11]。而在另一方面，由于底层、实现等各个层面上的差异，从桌面平台上获取的关于仿冒软件的知识也不能很好地用于了解移动端仿冒应用^[12]。

1.2 国内外研究现状

1.2.1 针对灰色应用的实证研究

Andow 等人曾发表过一篇针对灰色应用的研究文献^[3]。其中，他们从 Google Play 应用商店中采集了应用样本，并将样本分类，定义出了 9 种不同的灰色应用。灰色应用即那些并非具有明显的恶意行为，但应用意图存疑、又或是会向系统申请过多权限的应用程序。本文中对高仿应用的定义参考了这篇文献中的内容。

1.2.2 针对恶意应用生态系统的实证研究

在 Felt 主导的一次研究^[13] 中，研究人员仔细剖析了来自多个不同平台的 46 个恶意程序样本以了解这些样本的激励机制。该篇文献也揭示了这些样本的运行机制和行为策略，为后人抵御此类恶意行为提供参考。另外，Zhou 和 Jiang 搜集了来自多个主要恶意应用家族的、超过 1,200 个恶意应用样本，系统性地描绘了这批样本的不同性质，包括其安装手段、激活机制和其如何执行有效负载（实现恶意行为）。这类研究帮助了从业者拓宽视野，使得从业者对恶意应用的行为更加了解。然而正如上文提及，非 Android 平台样本的相关知识并不完全适用于应对 Android 平台上的应用分析，而仿冒应用与恶意应用亦有不同点，不可一概而论。针对仿冒应用的专门研究依然是有必要的。

1.2.3 重打包应用检测

针对重打包检测的前人研究大致可划分为五个类别。第一类是基于应用指令序列的。这种方法使用模糊哈希的方法提取出应用的摘要信息，然后通过比对两两应用之间的摘要信息来获得应用之间的相似度^[14,15]。第二类凭借语义信息比对应用。如 CLANdroid^[16] 通过分析五种语义特征点（比如代码中的标识符和调用到的 AndroidAPI 等）。第三类利用了第三方库检测手段。CodeMatch^[17] 筛选出应用中使用的第三方库代码后，计算并比对剩余部分代码的哈希值。Wukong^[18] 也分两步检测重打包应用，但与 CodeMatch 相比，其第二步使用了基于计数的代码克隆检测手段，而非基于哈希的技术。ViewDroid^[19] 通过重建和比对应用的视图来筛选重打包应用，这种技术属于第四类信息可视化。第五类依赖图论衡量应用相似性。

DNADroid^[20] 基于应用的程序依赖图 (Program Dependency Graph, PDG) 来比对应用相似性，而 AnDarwin^[21] 则用从每个方法中提取的 PDG 构建出语义向量，再计算向量间相似度以检测重打包应用。Centroid^[22] 甚至为应用中的每个函数构建了三维控制流图 (3-Dimensional Control Flow Graph, 3D-CFG)，然后再将三位控制流图聚合，通过检测不同应用在控制图聚合后的质心位置判断应用间的相似程度。

检测方法林林总总，在此不一一列举，从检测的准确性和可伸缩性上看，每种都均有优缺点，其并不在本文的讨论范围之内。然而，他们全都有一个共同之处：在验证阶段，无一例外，都是基于证书系统进行的。一旦非法开发者利用具有漏洞的证书签名机制，污染了实验中的部分应用数据，即使是最先进的重打包检测机制也无用武之地。

1.3 论文研究意义

正如前文所述，大量仿冒应用就存在我们身边，蠢蠢欲动。为了保障正当开发者的利益与消费者的权益，我们需要对仿冒应用有更全面、更深入的理解，从而更好地抵御仿冒应用。然而，现有研究提供的知识仍有空缺部分。

为了填补本领域的研究空白，作者借助群众信息的移动安全威胁数据平台 Janus¹ 搜集并分析了大量数据，对仿冒应用作出了较为全面的剖析。作者亦希望能凭借此文抛砖引玉，吸引更多研究者对仿冒应用进行更多更深入的研究。

1.4 论文研究内容

简而言之，本文所作贡献可分为以下几点：

- 首篇具有较细粒度的针对 **Android** 仿冒应用的全面实证研究 据作者所知，本文是第一篇提供 Android 仿冒应用实证研究的文献内容。本文从三个不同视角分析了仿冒应用，从细粒度角度验证了他们的性质。
- 对工业界中仿冒应用的一次大规模定量分析 在本次研究中，作者搜集了超过 15 万条数据条目，从中为工业界挖掘出了有价值的建议与见解。

¹<https://www.apbscan.io/>

- **一次对现实世界中最热门应用的仿冒品的观测** 本研究基于在中国内地最受欢迎的 50 个 Android 应用的仿冒应用数据完成。鉴于这批应用的原版受众之广，作者认为其本文的研究对象，即这批仿冒应用的数据，具有足够的代表性。
- **基于真实案例发掘仿冒应用特征** 从本文测量结果中浮现的发现结果与结论都有现实世界中的案例进一步支持。本文亦会将几个具有代表性的案例一一列出并作案例分析。

1.5 本研究遇到的困难与挑战

在实证研究的过程之中，作者遇到了以下几点困难和挑战：

1) 如何确定应用是否仿冒应用？

由于研究主体是 50 个热门应用的对应仿冒应用，作者先从应用中筛选出与热门应用外观相似或是相同的样本，其后再使用 Android 本身自带的证书机制，将获得样本的证书信息与原版应用的证书信息进行比对，鉴别出仿冒的样本。

2) 如何获得针对仿冒应用的大量数据？

数据搜集是科研工作中公认的难点。本文想要提供一次全面的研究结果，除了搜集的目标应用需要由多样性之外，也必须获得不同应用市场上的数据，增加研究的代表性。前文提及群众信息的移动安全威胁数据平台 Janus 是一个数据整合平台。该平台每天从各大 Android 应用市场爬取应用样本入库，免去了我们要针对各个市场重新定制爬虫代码的麻烦。结合网络爬虫技术，作者顺利从 Janus 搜寻到了与仿冒应用相关的超过 15 万条数据条目，其中每条数据条目代表 Janus 从应用市场上获得的一个应用样本。

3) 如何对大量的数据进行有效处理？

数据规模和处理效率一直是一对矛盾。由于一条数据条目代表一个应用样本，要对超过 15 万个应用样本进行详尽分析，明显太耗费时间成本与计算成本；然而，如果只对样本进行简单处理，获得的分析结果就不够全面和深入。在尽量确保分析全面性的前提下，对于每个样本，作者只抽取关键信息进行分析，而不对整个应用的代码进行详尽的静态分析或者动态分析，以节省时间与计算成本。

1.6 本文组织结构

本文共分为七章，环绕着本次实证研究的数据搜集和分析过程、分析结果展开，各章节内容如下：

第一章 主要介绍了本文的研究背景、相关工作、研究意义、研究内容及本文遇到的困难与挑战。

第二章 介绍了 Android 系统、应用市场相关的背景知识，包括现有的 Google Play 应用商店和多个第三方市场共存的局面介绍，以及仿冒应用的背景介绍，还会阐述本文使用到的 Android 安全证书机制，如此机制有何作用、如何运作等。

第三章 提供了本篇研究的概览，解释了数据收集流程。包括收集了什么数据、收集数据的分步解释和收集到的数据概览。

第四章 从多个视角分类提出并解说针对这批仿冒应用数据得到的发现。三个视角包括仿冒应用特征、针对仿冒应用的定量分析和仿冒应用的发展轨迹，每个视角都被进一步分解成了多个不同的研究问题。

第五章 结合上一章中提出的各个发现，挑出数据中具有代表性的一些案例进行案例分析，以案例进一步佐证分析结果的正确性。

第六章 在前文基础上，从应用市场上搜集了部分仿冒应用的评论进行分析，进而了解用户对这些应用持有的态度。

第七章 对本文工作进行总结，并对下一步工作进行展望。

第二章 Android 背景介绍

本章主要介绍 Android 系统、Android 应用程序和应用市场相关的背景知识，包括现有的 Google Play 市场和多个第三方市场共存的局面介绍，同时也会介绍 Android 安全证书机制的相关背景。

2.1 Android 系统介绍

Android 系统是属于 Google 公司的开源操作系统，基于 Linux 内核研发。其最早版本于 2008 年发布（Android 1.0），起先只针对手机端发布。由于具有图形化操作界面，并且采用触屏作为交互方式，极其简单易用，Android 系统自发布起就迅速在智能手机领域抢占大量市场份额，搭载 Android 系统的平板电脑也在我们的日常生活中渐渐变得随处可见。近年来，随着 IoT（Internet of Things，物联网）的发展，家用电器趋向智能化，不少电视厂商、机顶盒厂商、甚至可穿戴设备厂商也开始为产品内嵌深度定制的 Android 系统，为用户提供更好的体验。

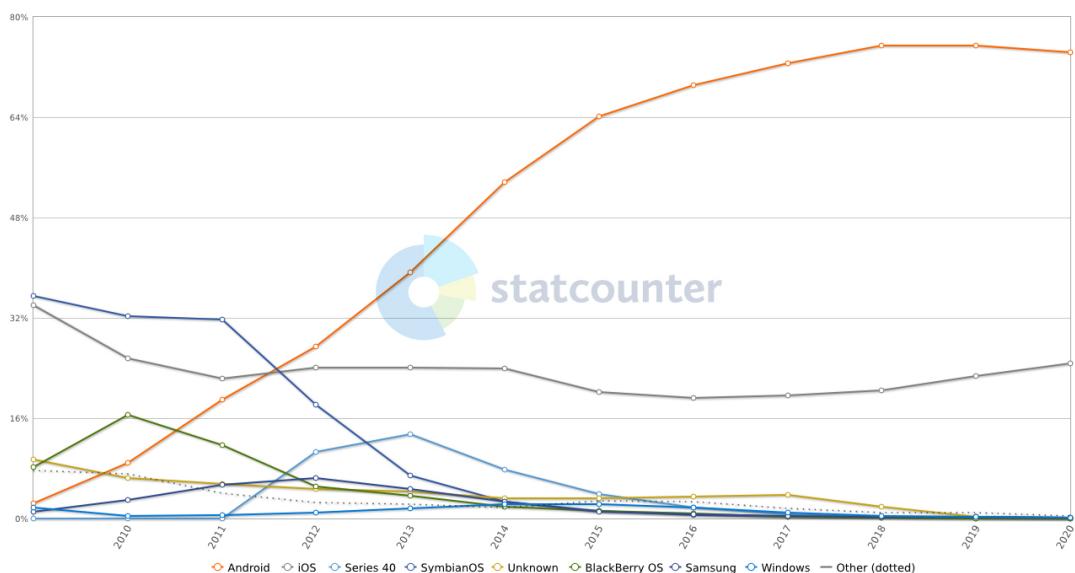


图 2.1: 2009 至 2020 年移动端操作系统市场份额变化图

图 2.1 展现了 2009 年到 2020 年间不同移动端操作系统对市场占有率的变化曲线，图表来源于数据分析机构 StatCounter^[1]，图表 X 轴为时间线，Y 轴为市场份额占总量的百分比。不同颜色的曲线代表不同操作系统，其中 Android 系统由橘红色曲线表示。从图中数据可知，自发布以来，Android 系统便一路高歌猛进，迅速占据移动端操作系统市场，从 2012 年起就获得了业界第一的市场份额占有率，直到 2020 年，其超过 70% 的市场占有率依然遥遥领先于其他操作系统。其中原因，除了对用户非常友好的操作方式以外，也在于其具有一个多元、开放又充满活力的应用程序生态环境。

2.2 Android 应用程序

2.2.1 应用程序简介

Android 应用程序（Android Application，简称 App²）是可以安装在 Android 系统上、拓展系统功能的软件，这些软件通常基于 Java 语言开发，其中可以包含用 C 语言或者 C++ 编写的库以提高性能。2014 年起，Google 宣布 Android 支持 Kotlin 编程语言，自此开发者也可以使用 Kotlin 语言进行 Android App 开发。

Android App 的开发需要使用 Google 提供的软件开发工具包（Software Development Kit，简称 SDK）和 Google 支持的集成开发环境（Integrated Development Environment，简称 IDE）。SDK 是包含了众多软件包的工具箱，其中有包括了 Android 系统应用程序接口（Application Programming Interface，简称 API）的函数库和用以编译 App 的各种工具，在编译完成之后，开发者还可以利用 SDK 的相应工具为 App 签上自己的数字签名。API 函数库提供了 Android 系统的一系列接口，开发者需要在使用 Android App 框架的前提下，调用各种 API 实现自己设计的功能。

在发布每一版本 Android 系统的同时，Google 公司也会发布一个新版本的 Android SDK 供开发者开发 App。每个人都可以从 Android 的官方网站上下载 Android SDK 和开发应用所需的 IDE，这意味着，利用官方发布的工具，任何人都可以开发出属于自己的 Android App。

²App 和应用程序/应用三者在 Android 领域中指代的是同一事物，在本文中可以相互替换，不作区分

2.2.2 构建应用程序

与大部分软件一样，开发者在发布自己的 App 之前，也需要把代码编译打包成 Android 操作系统使用的一种应用程序包格式文件 APK (Android application package)。每个 APK 文件都会包含该款 App 的一系列基本信息，包括 App 的应用名、包名 (Package name)、安全证书等。其中，包名是 Android 系统识别 App 的依据，每款 App 在不同的版本可以有不同的应用名，但其包名必须是一致的。图 2.2 展现了 APK 文件的构建流程。一般来说，一个 Android App 的构建流程会分为以下四步，整个构建流程由 Android SDK 中的 Android 插件和 Gradle 构建工具管理。

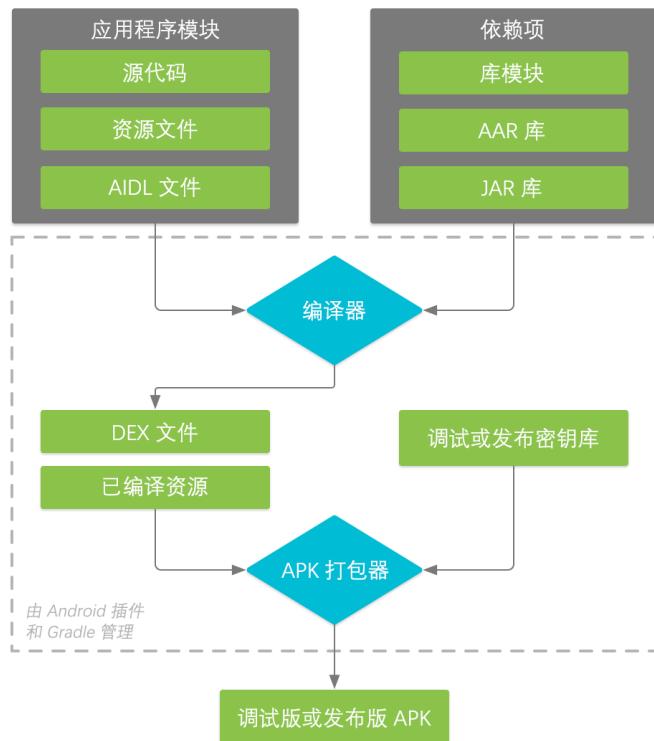


图 2.2: Android App 构建流程

首先，开发者需要编写 App 对应的源代码，然后连同一些源代码中使用到的依赖项一起输入到编译器中生成 DEX 文件。源代码可以由 Java 语言或者 Kotlin 语言编写，而 DEX 文件则是一种可执行文件，可以运行于 Dalvik 虚拟机上。Dalvik 虚拟机则是 Android 系统的核心组成部分之一，用于运行被编译为 DEX 文件的程序。此外，编译器还会将其他未被编译的资源文件转换为编译后的资源。

然后，SDK 中的 APK 打包器会将 DEX 文件和已经编译好的资源文件一起打包。APK 文件的本质是压缩文件，其中包含了被编译的代码文件、App 需要用到的资源文件（比如字符串、图片等资源）、assets 资源、App 的安全证书和 Manifest 配置文件，所以 APK 打包器的任务是将这些所有文件都压缩进一个 APK 文件里面。不过，在这个步骤，APK 打包器还未将所有文件压缩。因为在压缩之前，还需要进行下一步的签名。

在第三步，APK 打包器会使用密钥库文件对上一步中提及的资源文件和代码文件进行数字签名。这个步骤是用作校验 APK 文件是否被篡改、保证 APK 文件完整性的一个重要步骤，在本章后续内容中会有相关机制的更多介绍。

最后，APK 打包器会使用 zipalign 工具对应用进行优化，以减少 App 在设备上运行时所占用的内存。这步结束之后，整个构建流程也随之结束。开发者会获得一个编译好、签名完毕并且经过优化的 APK 压缩文件，然后就可以将这个 APK 文件安装到 Android 设备上运行使用。

2.3 Android 应用市场

由于每个人都可以开发、构建自己的 Android App，从网上发布的 App 数不胜数。这种开放性为 Android 应用生态带来开放性的同时，也会引入以下几个问题：

- 1) 难以择优 开放的环境会导致 App 数量难以胜数。由于从互联网上找到的 App 质量良莠不齐，用户有时无法判断网上的应用是否符合自己的需求；
- 2) 数据安全 智能手机与现代人的日常生活息息相关，其中也自然包含了各人的隐私数据甚至财产信息。确保用户下载、安装到的应用不会损害到他们的财产安全和数据安全至关重要；
- 3) 宣传渠道 开发者希望自己的 App 尽可能地受欢迎，但中小开发者并没有足够的资源去宣传自己的应用，可能会导致原本质量优秀的 App 因为缺少宣传渠道而无人问津；
- 4) 收入结算 开发者有从 App 中获得盈利的需求。即使是部分出于兴趣或其他非盈利目的开发手机应用的开发者，也需要从 App 中获取补贴以维持 App 的维护和正常运作。如何利用 App 变现、变现之后又要怎样将资金回流到开发者的问

题有待解决；

5) 应用更新 随着时间推进，用户的需求并非一成不变，开发者也需要针对 App 中以往出现的漏洞查漏补缺、或者更新软件功能，但要求用户定期/不定期地手动更新某个 App 甚至多个 App 并不现实。

Google 发布的应用商店 Google Play^[23] 的存在缓解了以上的问题。它是 Android 操作系统的官方应用商店，可以让用户浏览和下载商店中的 App。一方面，普通用户可以经由 Android 系统中预装好的 Google Play 服务寻找、购买和下载心仪的 App；另一方面，Google Play 也允许开发者将 App 通过开发者账户上传到 Google Play 中，在经过一系列的审查之后上架到商场上供用户下载。与上述的五个问题对应，Google Play 应用商店提供了以下几点服务：

1) 一个由官方背书的下载渠道 这确保了用户下载的 App 得到官方认可。同时，Google Play 也提供了一个关于 App 的社区条件，用户可以对 App 进行打分、评论，用户对 App 的评价经审核后可以由所有其他用户查看，直接影响其他用户对“是否要下载某款应用”的决定；

2) 上架之前的应用审核 在开发者上传 App 时对 App 作出一定的审查，以排除部分恶意开发者在商场中上架病毒和恶意应用的可能性。Google Play 也会根据用户反馈、应用本身运营数据等原因于每季度从商店中移除一部分 App，以保证商店货架上 App 的质量；

3) 作为激励机制的推荐榜单 在用户评价的基础上，Google Play 筛选出一部分质量优异的 App 制订出一些推荐榜单。推荐榜单会在应用商店首页进行展示，榜单包括“编辑精选”、“热门应用”、“年度之选”等，其中既有大公司开发的 App，也会有中小开发者独立开发的应用。此外，Google 还会通过用户的使用习惯、所在地区等因素为用户提供个性化推荐（如图 2.3 所示），在不同的 App 类别下，也有不同榜单为用户推荐，加大了优秀 App 的曝光率；

4) 集中的结算通道 Google Play 应用商店为开发者提供了一个统一的结算渠道。应用开发者可以在 App 内销售商品，然后选择 Google Play 提供的集成结算服务。用户在购买服务时，直接向 Google 付款，Google 再在每个结算周期将款项结算给开发者。这样一来，开发者尤其是独立开发者就可以更专注于 App 本身的研究和开发。

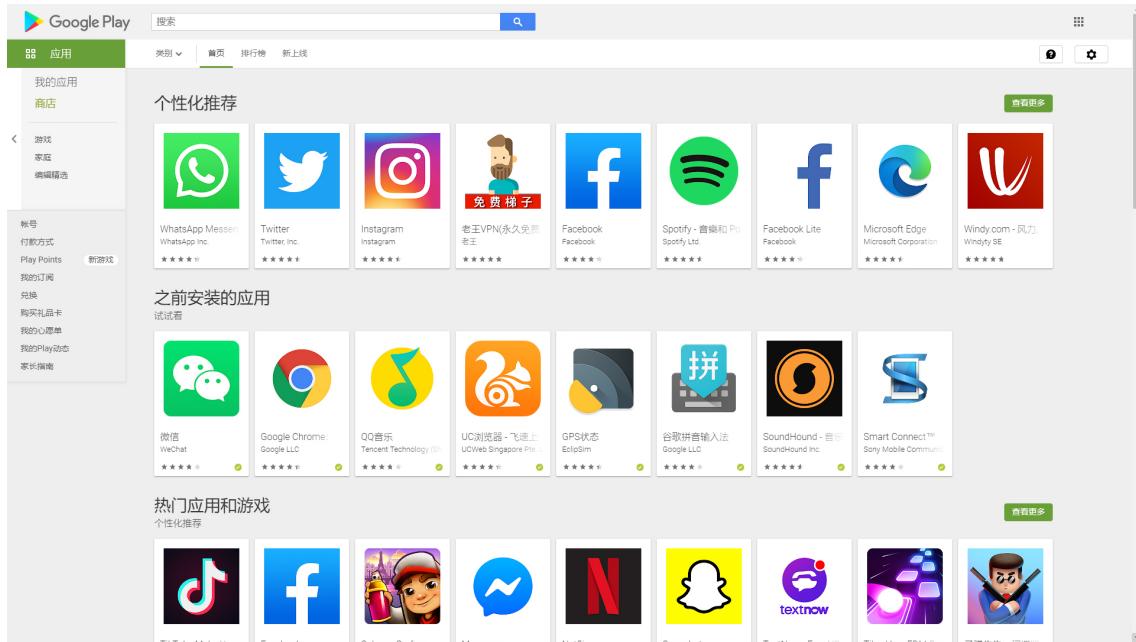


图 2.3: Google Play 应用商店首页（从桌面端浏览）

发，而不需要考虑如何利用应用变现、再将资金回收的复杂问题。Google Play 也允许开发者将自己的 App 上架为付费应用，需要用户付款后才可下载；

5) 方便快捷的更新推送 由于 Google Play 本身是个应用程序的集中平台，而且也预置到了大多搭载 Android 的设备中，所以开发者在更新应用时，只需将新版本上传到 Google Play 的后台，应用商店经过审核之后，就可以将新版本的应用推送到用户的设备中，让用户设备中的 App 实现自动更新，免去双方的麻烦。

2.4 第三方应用市场

Google Play 应用商店无疑为用户和开发者都提供了一个优良的解决方案，每个应用底下由用户评论组成的社区也促成了用户和开发者之间的交流，用户反馈直接推动了开发者对应用的改良。

遗憾的是，由于种种原因，Google Play 应用商店的服务并非对全球的所有地区和国家都开放。Google 从 2008 年开始退出中国大陆市场，因此 Google 的大部分服务，包括 Google Play 应用商店的下载服务在内，都不向中国大陆境内用户提供。换句话说，国内的大部分普通用户并不能享受到 Google Play 应用商店的便利。

为此，国内有多家厂商都推出了自己开发的应用市场服务，试图填补这一片市场空白。实际上，由于整合开发者、用户和 App 资源三者本身就有巨大的市场潜力，所以即使是在 Google Play 服务可用的其他国家和地区，也有厂商推出自己的应用市场（如三星推出的三星应用商店、Amazon 推出的 Amazon 应用市场），试着在这个市场上分一杯羹。

纵观国内的应用市场，经过几年的竞争与整合，依然未有出现像 Google Play 应用商店一样具有垄断性地位的厂商。多个厂商各据一方，主要可以分为两个类别。一类是国内 IT 行业巨头旗下的应用市场，如腾讯旗下的应用宝^[24] 和百度旗下的百度应用市场^[25]，其平台本身就具有大量基础用户，可以直接转化为应用市场中的活跃用户；另一类则是由各大手机厂商开发的应用市场，如华为的应用市场、小米的小米应用市场^[26] 等，这类的应用市场通常直接预装在手机出厂时自带的厂家定制 Android 系统中，凭借手机销量直接带动市场用户增长。

根据数据分析机构艾媒咨询于 2018 年 12 月发布的《2018-2019 中国中国移动应用商店市场监测报告》^[27] 显示，使用第三方移动应用商店的用户在手机网民中占比为 59.99%。结合国内网民数目庞大的现状，这一数字表示第三方应用市场在国内已被广泛接受。而 40.01% 未使用第三方移动应用商店的用户比例则表示这一市场还有广阔前景。该报告还提供了 2018 年国内第三方应用市场用户首选市场的分布图，如图 2.4 所示，用户对第三方移动应用市场的选择主要还是集中在国内 IT 巨头发布的应用市场上。约 40% 的手机用户会使用 360 旗下的 360 手机助手作为首选应用市场，而首选使用豌豆荚、UC 应用商店等阿里应用分发平台旗下应用商店的用户约占 11%。大部分市场份额都已被国内 IT 巨头旗下的应用市场占领。

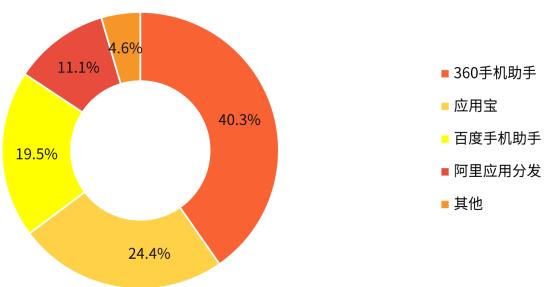


图 2.4: 2018 中国第三方移动应用商店用户首选使用品牌分布

与 Google Play 提供一个完整的 Android 生态环境相似，上述的各个厂商也致力于构建各自的 Android 应用生态链条：各大厂商本身就具有一定知名度，从其旗下的应用市场下载应用能让用户放心；开放开发者中心，让开发者注册账号后上传自己制作的 App；为每一个 App 提供评分和评论功能，构建出开发者和用户的交流平台；在应用市场首页设置应用排行榜、推荐安装软件等榜单（如图 2.5），提高优秀 App 的曝光率；应用市场自带的应用版本管理开发者在市场后台更新应用之后，将更新推送到用户的手机上；各大应用市场也会整合支付平台（如支付宝和微信支付）来应对市场内的应用购买业务，华为应用市场甚至像 Google Play 一样，为市场上的应用提供了自家的支付渠道以支持应用内的付款项目购买功能。

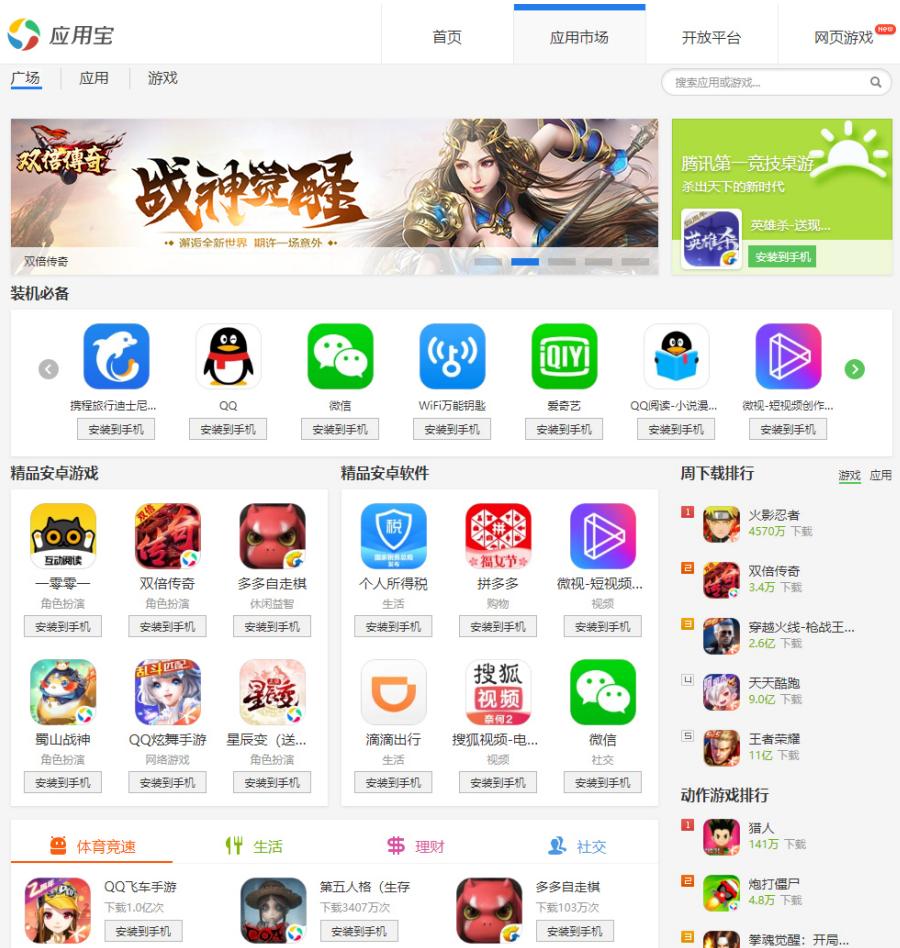


图 2.5: 腾讯应用宝应用市场首页（从桌面端浏览）

不同于在 Android 系统发布早期就存在的 Google Play 应用商店，国内的第三方应用商店是后期出现的产物，一出现就面临着激烈的市场竞争。一方面，在国内

各类第三方应用市场方兴未艾之时，国内的 Android 开发者社群尚未成熟，应用市场还未有大量开发者进驻；另一方面，在成立初期，为了抢占市场份额，各个应用商店都想方设法将商店内 App 的种类和数量最大化，以迎合市场用户各种各样的需求。作为结果，各类第三方应用市场都在各个渠道搜集 App，而非通过开发者上传的方式获得货架上的应用程序。由于在早期各种监管渠道尚未完善，各个市场在搜罗各类 App 的同时，难免会将大量的盗版应用也一并收录。

2.5 Android App 签名机制

前文提到，开发者在使用 Android SDK 构建 App 时，其中十分重要的一步是对 App 进行数字签名。实际上，Android 的数字签名和安全证书机制基于 RSA 公共密钥系统，是 Android 安全机制中不可或缺的一个部分。本章的余下内容将会对 Android App 的签名机制进行简单分析。

Android App 的签名机制是用作校验 APK 文件是否被篡改、保证 APK 文件完整性的一个重要机制，所有的应用都必须要在经过签名才能安装进 Android 系统中。在签名时，SDK 会使用一种密钥库文件，如果开发者还没有这个文件的话，SDK 会自动生成一个。密钥库中包含了开发者的各种信息，包括一对公钥和私钥。私钥用于数字签名，不可向外公布；公钥则是可以向外公布的一组密钥，用于数字前面的验证。App 中的签名也是系统用来识别开发者的重要依据，因为同一个密钥库文件会产生一致的签名，系统能根据签名中的公钥验证应用识别开发者。

签名的过程大致如下：在前文流程的第二步结束后，编译器会输出 DEX 文件和编译好的资源文件，这时，SDK 会对每个文件都扫描一次，然后对每个文件提取一次数字摘要，再把每个文件的文件名和其对应的数字摘要保存在一个名为 *MANIFEST.MF* 的文件中。之后，SDK 会再扫描一次刚才生成的 *MANIFEST.MF* 文件，再次提取一次数字摘要，把这个摘要连同刚才文件中的所有内容存入另一个新文件 *CERT.SF* 里。第三步，再计算一次 *CERT.SF* 的数字摘要，然后用密钥库中的私钥对这个摘要进行加密。加密后的结果就是数字签名。最后，SDK 将签名、公钥、计算数字摘要的哈希算法等信息写入 *CERT.RSA* 文件中，再将这整个过程中生成的四个文件放进 *META-INF* 文件夹，用 APK 打包器打包起来。至此流程结束。

而 Android 系统验证签名的方式，则是先通过 *CERT.RSA* 中的公钥验证签名是否无误，再根据文件中提供的哈希算法计算 APK 包中所有文件的数字摘要：先从 *CERT.SF* 开始，然后是 *MANIFEST.MF*，然后是 APK 中的其他所有文件... 一旦其中出现不相符的结果，就会导致验证失败。在安装 App 的过程中，验证签名失败会使得系统终止 App 的安装。

换句话说，在一个 APK 被打包签名完毕之后，如果需要更改其中的内容，就只能在更改后将 APK 重新打包签名一次，即使是一个 bit 的修改也会破坏原有的签名。这也是系统可以用数字签名识别开发者的原因：签名一致的 App，最后一定都是由同一个开发者打包的。所以，具有同样签名的 App 也可以在同一个 Android 设备上共享数据。不过这超出了本文讨论的内容，故按下不表。

目前，签名的模式共有三代，其区别主要在于构建流程第三、第四步之间的一些操作上。简单地说，越新的签名模式能更好地保障 APK 文件的完整性。实际上，第一代签名模式 V1 具有较为致命的缺陷，所以 Google 官方也在呼吁开发者在编译时采用最新的签名模式。

要注意的是，签名机制只能保证 APK 文件在被篡改之后不能凭借原有的签名被安装进 Android 系统，但恶意开发者依然可以在篡改 APK 之后，用自己的密钥库对 APK 重新签名，构建出可安装的 App。这种 App 是盗版 App 的一种，被称为重打包 App。

另外，虽然一个安全证书只能指向一名开发者，但一个开发者可以同时拥有多个安全证书。开发者和安全证书之间具有一对多的映射关系。

2.6 本章小结

本章主要介绍了 Android 系统、Android 应用和 Android 应用市场的相关背景知识，同时也阐述了 Android 应用的构建流程和简要介绍了其中使用到的签名机制，为后文实证研究中使用到的采样来源和验证方法做铺垫。

第三章 研究概览

3.1 本文研究流程

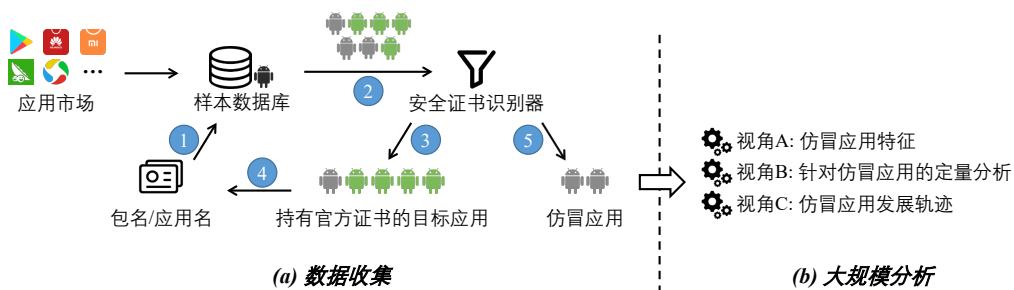


图 3.1: 本文研究流程

图 3.1 展示了本文的工作流程，其可以分为两个主要阶段：

1) 数据收集 在这一步, 我们基于搜集到的官方应用安全证书的信息, 从各大第三方应用市场收集仿冒应用和他们的相关元数据 (比如 App 的包名和应用名)。得益于与上海舜众信息技术有限公司的合作, 我们从国内的大部分主流第三方应用市场中顺利收集到了大量的原始数据。

我们的研究对象是仿冒应用，但市面上的 App 林林总总，不可一概全收，一些小型的 App 相似度极高，更是难辨其原创性。因此，我们在收集仿冒应用时，只收集了当时国内市面上最热门 50 款 App 的对应仿冒样本。在选择市面上最热门的 50 款 App 时，我们参考了当时数据平台易观千帆的月度 App 排行榜³，然后从中选出了其中的前 50 款热门 App，搜集它们的仿冒样本。

2) 大规模实证研究 在这一步，我们对搜集到的仿冒样本，从以下角度进行大规模的实证研究：仿冒的基本应用特征、针对仿冒应用的定量分析。我们也试图从收集到的信息中挖掘出仿冒应用作者的开发轨迹，以求为学术界和工业界揭示仿冒应用生态系统的更多信息。

³<https://qianfan.analysys.cn/refine/view/rankApp/rankApp.html>

在此之外，我们还随机选取了收集到的一部分仿冒应用，从其对应的第三方应用市场中提取用户对他们的评价，以期了解用户对这些应用持有的态度或是寻找更多有价值的信息。

3.2 数据收集

尽管对于学术界来说，一个从市面上 App 收集的、完整而全面的仿冒应用数据集，又或是一套能大规模又有效地收集仿冒应用的方法会对相关研究十分有裨益，但是迄今为止，都未有相关工作能填补这一空白。因此，在本文中，我们率先尝试着从工业界中系统地收集我们需要的仿冒应用数据。

收集方法 针对我们的研究主体，要获取足量的数据以组成数据集是一个十分具有挑战性的任务，难点主要有三：(1) 我们要从多个不同的应用市场中爬取 App 样本，每个第三方应用市场都有不同的网页编码，不存在一个爬虫脚本对所有应用市场数据都通用的场景；(2) 各个第三方应用市场架上的 App 数量浩如繁星，我们需要有效地找到和前述 50 个热门 App 有关的所有样本，不重不漏；(3) 我们需要一个轻量级的解决方案快速判断获得的 App 样本究竟为正版应用又或者是仿冒应用。

为了应对挑战 (1)，我们与工业界合作，利用了犇众信息公司的 Janus 平台对各大第三方应用商店进行样本爬取，然后用得到的样本构建了一个样本数据库。事实上，如图 3.2 显示，Janus 平台自 2017 年起就开始对各大第三方应用商店的 App 进行样本收集，至今已收集到上千万个 App 样本。除样本搜集外，Janus 也提供按规则搜索功能，用户可以创建自己的规则过滤平台中的应用数据，以获取自己需要的 App 样本。

为了应对挑战 (2)，我们使用了一个基于广度优先搜索（Breadth-First Search，简称 BFS）的算法，分步搜索与 50 个热门应用相关的所有 App 样本，稍后会有详细介绍。

而为了应对挑战 (3)，我们预先从官方渠道下载了 50 个热门应用的最新版本，提取出它们 APK 文件中包含的安全证书信息，构建出了一个安全证书筛选器。

更具体地说，在构建数据库的阶段，我们分别使用安全证书哈希码、App 包名和 App 应用名来聚类应用样本。当接收到查询请求时，数据库分别会返回与查询

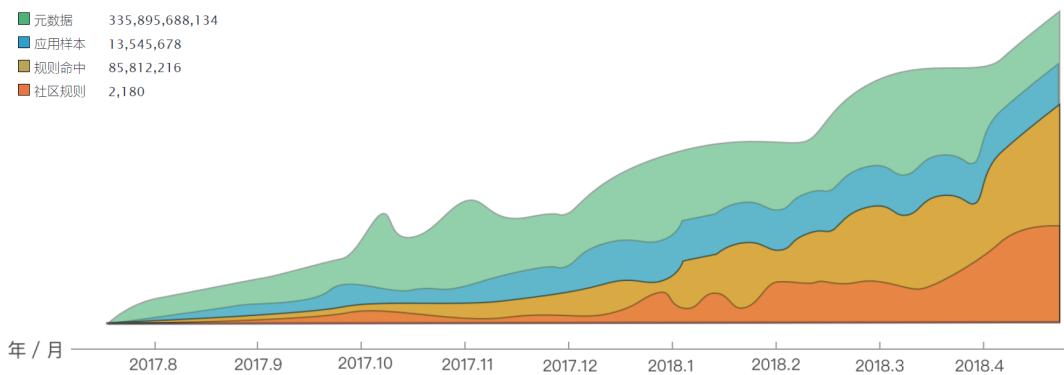


图 3.2: Janus 平台上的数据规模时序图

条目对应的包名、应用名和安全证书哈希码的聚类结果，结果以包含样本元数据的条目形式呈现。为了迷惑用户，仿冒样本往往有与正版 App 十分相似的外观，比如与正版 App 近似的名称。因此，我们以正版应用的应用名出发，搜寻仿冒应用。要做这件事，我们首先从预先下载好的正版样本 APK 文件中提取 App 的包名和应用名，放入查询队列中。这些包名和应用名接下来会被输入到样本数据库中进行查询（即图 3.1 中标识的步骤 1）。之后，数据库会返回一系列对应的样本结果（步骤 2）。对每个结果，我们会提取其中的安全证书哈希码字段，输入到安全证书识别器中，判定对应的样本是否持有正版应用的证书。如果一个样本持有正版证书，它就会被认定为正版应用（图中的步骤 3），我们会记录下它的包名和应用名（步骤 4）。如果它的包名（或应用名）还没有被用于查询过，我们会把这个包名（或应用名）加入查询队列，然后开始下一轮的迭代（回到步骤 1）。否则，这个样本会被标记成仿冒样本（图中步骤 5）。它的元数据将会被用于大规模分析。

之所以要这样操作，是因为开发者推出的 App 的应用名和包名并不是一成不变的。一些热门应用会出于商业原因频繁地更改自己的应用名（比如爱奇艺视频，会根据其近期热播的电视剧/电影变更其应用名以吸引更多用户使用）；也有个别的热门应用可能会更换自己的包名，比如 App 有重大改版、又或者是开发者安全证书有变更，开发者不得不更换包名（具体原因可参考第 2.5 节的 Android App 签名机制部分）。对于每个热门应用来说，当其对应的查询队列中的所有包名、应用名都已经被查询过，我们对这个热门应用的数据收集就结束了。

数据概览 在这里，我们先给收集到的数据提供一个数据概览：

从易观千帆提供的数据榜单中，我们选择了 50 个最热门的 App 作为研究主体，这些 App 分属 11 个不同的应用类别。由于 App 的应用名可能会在 App 更新迭代的时候随之变更，我们用近似 BFS 的策略，从 50 种 App 中一共记录了 198 个不同的应用名，来挖掘仿冒样本。在这 50 款 App 中，我们并不能在市面上找到以下三款 App 的任何样本：*OPPO* 应用商店，华为应用商店和小米应用商店。因为这三款 App 都是由手机设备厂商开发和预装在对应品牌的手机中的，仅供这些品牌的用户使用，并不在第三方应用市场上提供下载。当然，这也是这三款 App 热度高的原因——这几款 App 都被预装到了对应手机品牌厂商的每一部 Android 设备中，而 OPPO、华为和小米又是国内最大的几家手机厂商，这几款 App 自然也会有庞大的用户基数。因此，我们最后的目标 App 只有 47 款。

对这 47 款目标 App，我们总共收集到了 138,106 个应用样本。其中，69,614 个应用样本持有官方开发者证书，52,638 个应用样本并不具有官方证书。还有一部分应用样本，是某些应用的分别发布在不同第三方应用市场同一版本，在经过我们的去重筛选后被排除（共计 15,854 个）。

对于每个样本，我们会收集 8 个数据项作为元数据：样本 *SHA1* 码，安全证书 *SHA1* 码，包名，样本大小，版本号，搜集时间和 *APK* 包来源。其中，样本 *SHA1* 码是使用 *SHA1* 哈希算法对整个 *APK* 文件进行数据摘要之后获取到的编码串，每个样本都有独一无二的 *SHA1* 码；安全证书 *SHA1* 码则是对样本的安全证书采用 *SHA1* 算法提取数据摘要之后获取的编码串，用于识别不同的证书。而搜集时间则是样本从第三方应用市场被爬取到数据库的时间点，*APK* 包来源指示该 *APK* 包来源的第三方应用市场。

3.3 本章小结

本章主要对本研究进行了概览，并详细介绍了数据收集的流程和方法，然后对采集到的数据进行了简要描绘。接下来我们将针对前述的元数据，尤其是对仿冒样本中采集到的元数据，进行实证研究，以求获得对于仿冒应用生态和特征、以及对于仿冒应用开发者的行为的更全面的认知。

第四章 大规模实证研究与发现

在数据库准备好之后，我们就可以对这些仿冒应用进行大规模分析，挖掘仿冒应用的信息并了解其生态系统了。为了能更全面有效地测量这些仿冒应用的各方面，我们定义了三个不同视角，即：仿冒应用的基本特征、针对仿冒样本的量化分析和仿冒应用的发展轨迹。接下来，我们会详细描述各个不同视角的观察方法以及观测结果。

4.1 仿冒应用的基本特征

为了了解仿冒应用作者使用的仿冒策略，或者说，他们是如何绕过应用市场的监管机制的，我们有必要对仿冒应用的基本特征有所了解。为此，我们分别针对应用的安全证书和应用名称、包名和应用大小等基本信息进行了观测。

应用的安全证书就是对开发者的识别码，而仿冒应用在安全证书上的性质（也就是说，是否每个仿冒应用都有一个独一无二的安全证书），十分可能是仿冒应用规避监管技术的关键。另一方面，我们也猜想打包应用（一种高仿应用）在我们的数据集中普遍存在。对仿冒应用基本信息的测量，比如说对应用包名和大小的测量，则可以帮助我们了解重打包应用在数据集中的分布如何。因为普通的重打包技术并不会对 APK 的基本信息（比如应用名、应用版本号等）作出修改，除非仿冒作者有意为之。

为此，我们作了如下假设：

假设 1.1：绝大部分的仿冒样本有其对应的、独一无二的安全证书。换句话说，绝大部分仿冒应用和他们的安全证书呈一对一的映射关系。

假设 1.2：一大部分的仿冒样本和他们的仿冒对象（也就是原版的官方 App）有同样的应用名/包名/APK 包大小。

为了验证这些假设，同时获得更多对这些仿冒样本个体的知识，我们在本节中提出了以下研究问题（Research Question，简称 RQ），分别与上述假设一一对应：

RQ 1.1: 仿冒样本数量和他们的安全证书数量存在着什么样的关系？也就是说，一个安全证书通常会跟多少个仿冒应用样本相关联？

RQ 1.2: 在我们的数据集中，仿冒应用是怎么“山寨”正版 App 的？也就是说，仿冒应用的应用名/包名/APK 大小和他们对应的正版 App 有多相似？

表 4.1: 安全证书/仿冒应用数量对应表

仿冒样本数量	1-5	6-10	11-50	51-100	More than 100
安全证书数量	8252	525	531	71	80

RQ 1.1. 结果 在仿冒应用持有的所有安全证书中，76% 都仅仅关联到了一到两个仿冒样本，与单个安全证书相关联的仿冒应用数分布在从 1 到 1,374 的区间内。我们在表 4.1 中统计了安全证书和他们对应的仿冒样本数量。其中第一栏为仿冒样本的数量区间，第二栏为关联的仿冒样本数量该落于区间的安全证书数。大部分安全证书都只关联了 1 到 5 个应用样本，但也有少量安全证书与大量仿冒样本有关联关系。

这个发现与我们的猜想（即大多数仿冒样本都有他们对应的独一无二的安全证书）部分相符。虽然并不是大多数仿冒样本都有其单独对应的安全证书，但多数安全证书的确只对应少量样本。结合第 2.5 节 Android App 签名机制最后的说明，我们认为这是规避第三方应用市场监管机制的一个策略。如果以同一开发者的身份，用一个安全证书上传多个仿冒 App，万一其中 App 被投诉下架，其他的 App 很可能会受到牵连。然而，一个仿冒开发者其实也可以持有多个安全证书。如果使用多个安全证书分别上传仿冒 App，第三方应用市场就不容易找到这些 App 的关联，即使其中一部分被举报下架，余下的也得以被保全。另外，在整理对应多个仿冒样本的安全证书时，我们得到了一些意料之外的发现。相关内容会在第??章中加以拓展。

RQ 1.2. 结果 以包名为例，根据我们的统计结果，在所有的 52,638 个仿冒样本中，

只有 243 个（少于 0.5%）使用了正版应用的包名，余下大于 99.5% 占比的所有仿冒样本都使用了他们自定义的包名。在余下的这 52,395 个样本中，我们找到了 14,089 个不同的包名。这其实在我们的意料之中。因为每个应用在 Android 系统中都需要有独一无二的包名，如果系统在安装 App 时发现系统中已经有具有相同包名的 App，就会检查两个不同版本应用的安全证书，证书不一致会导致安装失败。因此大部分仿冒应用不会直接使用正版 App 的包名。但这是否意味着仿冒应用就会使用与正版 App 完全不同的包名呢？它们会不会使用和正版 App 相似的包名？

同理，我们对应用名和 APK 大小两个方面也有类似的疑问。下面将会就我们获得的数据，探索上述问题。

为了解决相似度问题，我们采用了编辑距离^[28]这一在自然语言处理（Natural Language Processing，简称 NLP）领域被广泛应用的距离定义作为衡量标准。

定义 1 编辑距离

给定两个字符串 a 与 b ，其间的编辑距离 $d(a, b)$ 为将 a 和 b 相互转换的最小编辑操作数。其中，每次添加、删除或将一个字符转换成另一个字符都算作一次编辑操作。

举个例子，“jingdong”和“jindeng”之间的编辑距离是 2，由前者转换为后者的其中一种编辑次数最小方法是将第一个“g”删除，然后再将“e”转成“o”。同理，字符串“fake”和“official”之间的距离是 7，其中一种方案是在“f”前添加“o”，在“f”和“a”之间添加“fici”（这里包含了 4 步操作），将“k”替换作“l”，最后删去“e”。对于从仿冒样本中获取到的每个包名，我们都计算了与其对应的官方发布 App 的正版包名的编辑距离。

图 4.1 由三个小提琴图⁴组成，分别表示了我们在应用名、包名和 APK 包大小上的统计信息。在每个“小提琴”中，中间的黑色粗条表示四分位数范围，粗条中间的小白点表示数据的中位数，而黑色细条表示 95% 置信空间。

4.1a 表示了分别在官方样本、仿冒样本的应用名和两者间编辑距离的统计数据。其中“官方”图例和“仿冒”图例中的小白点都在接近数值“6”的位置，说明官

⁴https://en.wikipedia.org/wiki/Violin_plot

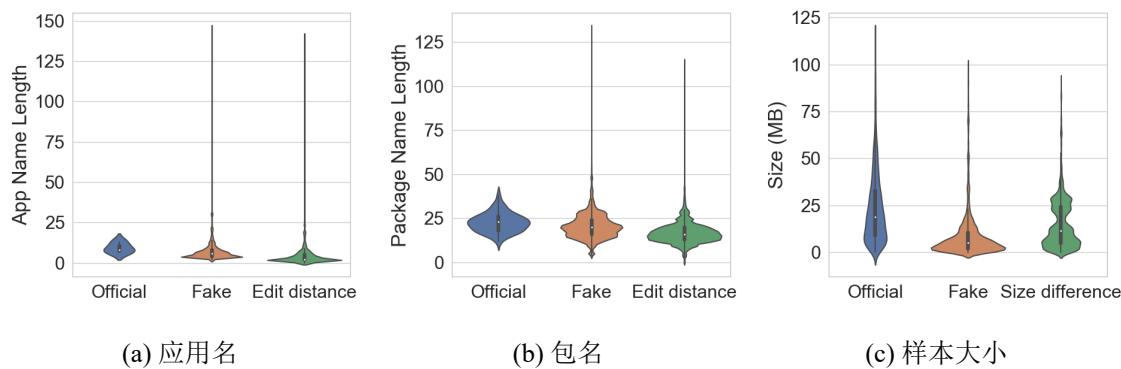


图 4.1: 对 App 各项属性的统计结果

方样本和仿冒样本的应用名的平均长度十分相近。这两个数据组之间的数据分布的主体略微相近，也表面了他们的相似程度。更重要的是，图中“编辑距离”图例的中位数值十分低（在 y 轴上为“2”的位置）。这意味着过半数仿冒应用通过从官方 App 的应用名中修改少于 3 个字符来获得其应用名。这表示大部分仿冒应用正在使用与官方 App 非常相似的应用名。与此同时，我们也留意到了一些仿冒应用有着异乎寻常的长名称（其中最长的仿冒样本的应用名中甚至有 146 个字符）。我们认为这些异常样本有可能是出于测试第三方市场审查机制的目的而被上传到市场上的。另一种可能是，一些长应用名拼合了多个热门 App 的名称（比如“潮流女装-美丽说蘑菇街淘宝天猫京东美团精选”），这可能是为了应用更容易地被用户搜索到而采取的策略。

4.1b 显示了针对包名的结果。和 4.1a 类似，官方 App 的平均包名长度和仿冒样本的平均包名长度依然很类似（双方的值分别为“23”和“20”）。然而，他们之间编辑距离的中位数则比应用名编辑距离的中位数明显更高（在 y 轴上“16”的位置），这意味着将一个仿冒应用的包名转换为一个官方 App 的包名平均需要 16 次修改，反之亦然。也就是说，官方 App 的包名和仿冒应用的包名会相当不一样。我们可以据此推出仿冒应用更倾向于使用自定义的包名。

4.1c 显示了 APK 包大小的信息。为了能更好地显示结果，我们作图前从数据集中剔除了一些极端样本：他们是大于 150MB 的 APK 包，在所有 69,614 个官方 App 的样本中占 851 个（约为 1%），在 52,638 个仿冒样本中占 447 个（少于 1%）。这些极端样本大部分来源于“游戏”类别下。图表显示，仿冒样本大小的中位数约

为 5MB，而约半数的正版 App 有着大于 18MB 的大小。因此，仿冒应用更有可能：(1) 由仿冒开发者自行开发，而不是使用重打包技术制作，因为重打包之后的应用通常不会在大小上与原版有太大差距；(2) 是恶意应用，因为恶意应用除了恶意代码之外，通常没有太多其他内容。

简而言之，图 4.1 说明了仿冒应用：(1) 更倾向于用一个与正版 App 相似（甚至是相同）的应用名，但会使用自定义的包名；(2) 通常大小更小。

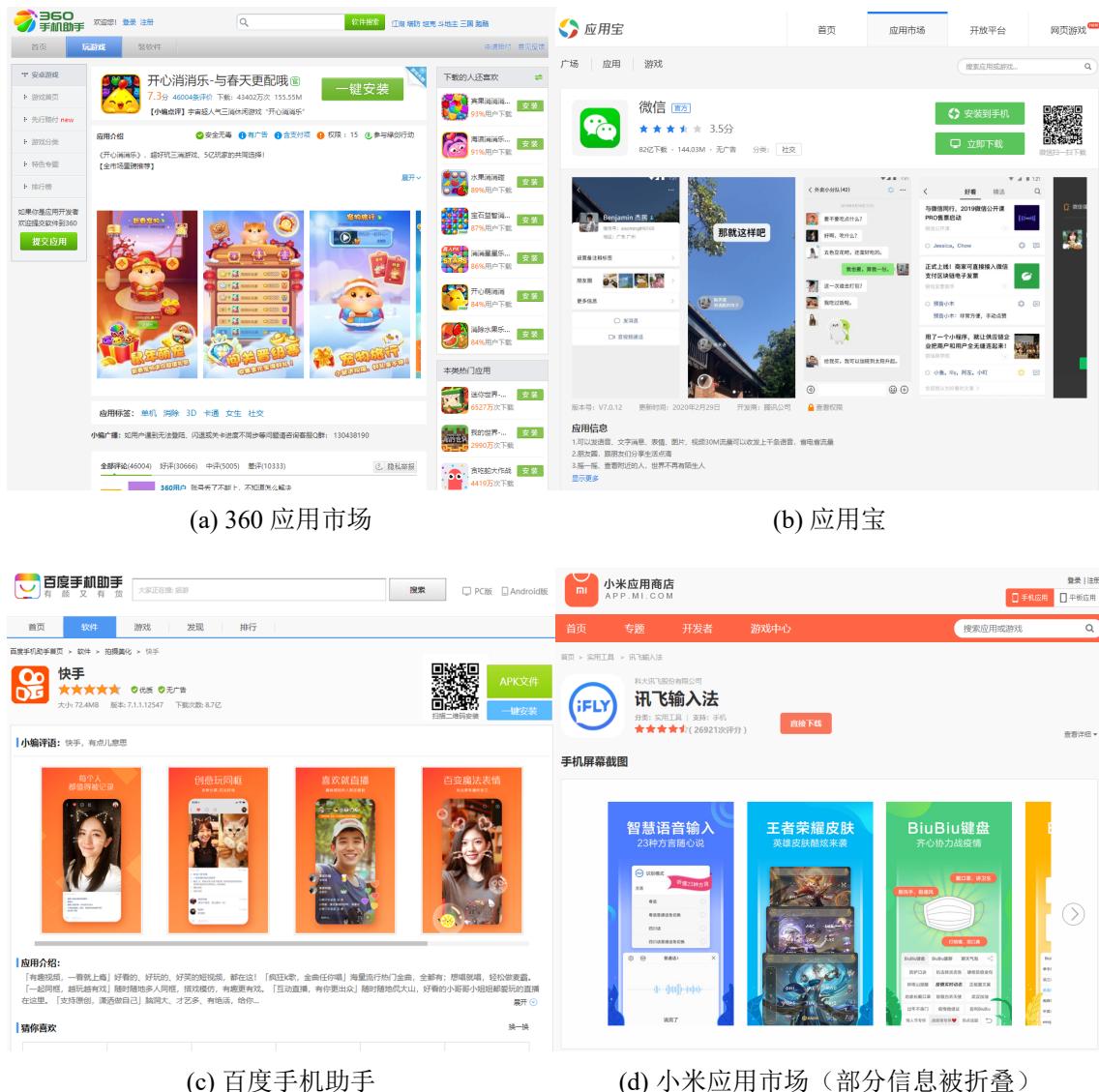


图 4.2: 各大应用市场应用详情页（从桌面端浏览）

在很大程度上，我们认为产生第一点结论的原因是第三方应用市场上提供的 App 信息的缺失和用户对 Android App 了解的缺乏。如图 4.2 所示，在第三方应用

市场上，当用户浏览 App 的详情页时，能看到应用名、下载量、应用描述、其他用户对应用的评论和评分等信息。但是，大部分应用详情页上并不会出现技术详情，比如应用包名。而普通用户也不会了解 Android App 中包名和应用名的区别和关联。个别第三方应用市场（如 4.2d 所示的小米应用市场）上，应用的某些信息甚至被折叠了起来，用户要点开折叠页才能发现一个 App 会有多大。因此，不会被普通用户关注的包名自然可能会出现各种五花八门的情况，而一定会显示的应用名则是越接近正版 App 越容易诱导用户下载。

本节小结：绝大部分安全证书只与少数仿冒样本有所关联。这很可能是仿冒开发者规避第三方市场监管机制采用的策略。同时，我们也观测到仿冒应用倾向于与官方 App 相同或者是十分相似的应用名。但是，仿冒样本和官方 App 在包名和 APK 包大小方面都不相似，这表明重打包应用在仿冒应用中并不普遍存在。

4.2 针对仿冒样本的量化分析

天下熙熙，皆为利来；天下攘攘，皆为利往。我们不妨假设获利是驱动仿冒应用开发者的最终目标，进而假设仿冒应用的数目与其来源市场、受欢迎程度还有应用分类密切相关。因为一个应用越受欢迎，就越可能获利，对仿冒应用也是如此。此外，App 的更新频率也可以被视作一个潜在因素，频繁更新的 App 或许会阻碍仿冒应用开发者对其进行仿冒。对应地，我们假设以下这些因素可能会影响某款 App 对应的仿冒样本数量：

假设 2.1：仿冒样本的比率与第三方应用市场架上的 App 数量有关联。

假设 2.2：仿冒应用的数量与其对应的正版 App 的受欢迎程度有密切联系。

假设 2.3：应用的更新频率影响着其对应仿冒应用的数量。

假设 2.4：App 类别是影响仿冒应用数量的因素之一。

与之相对，我们定义了下面几个研究问题：

RQ 2.1：这些仿冒应用市场都集中来源于哪里？哪个第三方应用市场有最多的仿

冒应用？

RQ 2.2: 一个 App 受欢迎的程度会影响对其仿冒的应用数量吗？

RQ 2.3: 一个 App 的更新频率会影响对其仿冒的应用数量吗？

RQ 2.4: 一个 App 所在类别会影响对其仿冒的应用数量吗？

RQ 2.1. 结果 图 4.3 展示了我们收集到的所有样本的来源。在左边的图中，我们可以看到，在我们收集数据的所有 31 个第三方应用商店中，源于百度手机助手的样本量是最大的。同时，从百度手机助手中搜集到的仿冒样本也是最多的。各个应用市场的仿冒率在右图呈现。

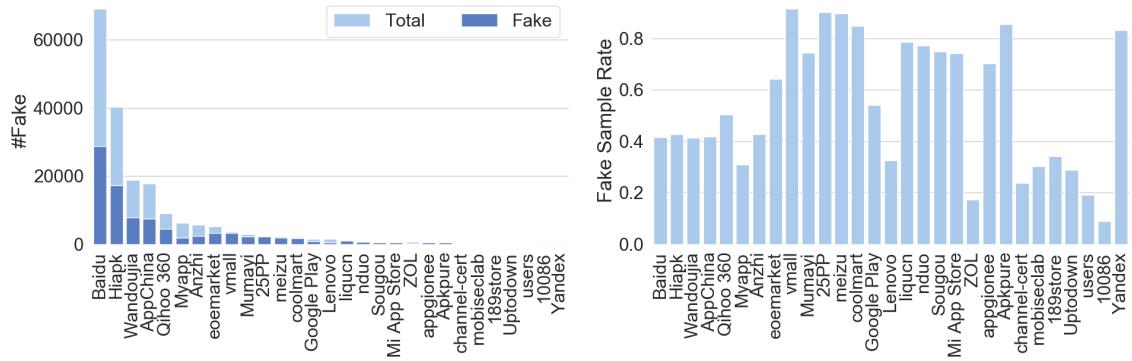


图 4.3: 从不同第三方应用市场中收集到的应用数量以及各市场仿冒率

定义 2 仿冒率

某款 App_a 的仿冒率 $fake\ sample\ rate_a$ 指与其关联的仿冒样本的数量 $fake_a$ 与该 App 正版样本的数量 $total_a$ 的商（式 4.1）。

某第三方应用市场 AS 的仿冒率 $fake\ sample\ rate_{AS}$ 则是其中包含的所有目标 App 的均值（式 4.2）。

$$fake\ sample\ rate_a = \frac{fake_a}{total_a} \quad (4.1)$$

$$fake\ sample\ rate_{AS} = Avg(fake\ sample\ rate_a, \forall a \in \{\text{目标 App}\}) \quad (4.2)$$

尽管百度手机助手^[25]和安卓市场^[29]的仿冒率均为40%左右，在所有的31个渠道中处于中等水平，但由于源于这两个渠道的样本基数最大，所以从这两个第三方应用市场搜集到的仿冒样本数也是最多的。

图中数据显示，应用市场的样本数量和仿冒率并没有直接联系，但是我们仍然有一个有趣的发现，那就是App本身和市场的关系有可能是影响App仿冒率的一个因素。腾讯旗下的应用市场应用宝^[24]中较低的仿冒率可以很好地为这个发现提供数据支持，因为我们的50款目标App中，有12款都是腾讯公司开发的应用。

RQ 2.2. 结果 从直觉上看，某款App越受欢迎，仿冒应用开发者就越有动机对其仿冒，然后诱导用户下载仿冒版本以获取利润。

要注意的是，每款目标App都有不同的样本数（无论是官方的或是仿冒的），所以我们不能拿仿冒样本的数量直接作比较。为了消除偏差，我们应该将样本数量归一化，使用式4.1定义的仿冒率对每个目标App进行比较。接下来，我们使用了皮尔逊积矩相关系数（Pearson product-moment correlation coefficient，简称PPMCC）来计算一款App被仿冒的严重程度与其热度是否具有相关性。相关计算会使用上述的仿冒率和从易观千帆^[30]获取的App月度热度指数计算。

定义3 皮尔逊积矩相关系数

两个变量之间的皮尔逊相关系数定义为两个变量之间的协方差和标准差的商（式4.3）。

$$p_{x,y} = \text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_x \sigma_y} = \frac{E[(X - u_x)(Y - u_y)]}{\sigma_x \sigma_y} \quad (4.3)$$

式4.3的值域为[-1, 1]，该值越接近0，表示两个变量之间的相关关系越弱。出乎我们意料的是，根据我们的数据，这两个因子之间的相关系数只有0.246。这表明仿冒应用的数量和App的热度在相关性上只处于较弱水平，和我们的预期并不符合。

RQ 2.3. 结果 我们猜想更新频率有可能会与App被仿冒的次数相关联，因为升级通常会有漏洞修复等举措，可以帮助App免受攻击。所以我们一款App的更新频率越高，其安全性能应该就会越好。

为了评估一款目标 App 的平均更新频率，我们标记了每个官方应用样本被发行时的时间，精确到日。然后我们找到最新发布的那个样本和最早发布的那个样本，求出他们发行时间的差值的绝对值与版本数的商，即为平均更新频率（单位：天/版本）

相关系数的计算结果表示，更新频率和仿冒数量之间的关联度只有 0.084，意味着两者之间几乎没有关联。我们认为这个结果可能有两个原因导致：(1) 在我们的数据集中，每款 App 的平均更新间隔为 10 天/版本。这个较高的更新频率表面开发者可能不会每次都在更新中修正安全性问题，从而削弱了更新频率作为安全性指标的功能。(2) 结合前文的结果，我们数据集中的大部分仿冒样本都不是重打包应用，而是仿冒应用开发者自行开发的。因此，无论官方版本受到的保护程度如何，仿冒应用开发者都可以制造出对应的仿冒应用。

RQ 2.4. 结果 某些 App 类别比其他类别更有可能带来收益。根据一份来自 App 营销机构 LIFTOFF^[31] 的报告预测，在未来，游戏类有望成为带来最高收入的应用分类。

表 4.2: 目标 App 与其相关统计

应用名	类别	月度热度指数	更新频率 (天/版本)	样本总数	仿冒 样本数	仿冒率	平均仿 冒延迟
微信*	社交网络	91.2K	6.4	9248	6447	69.7%	12.1
QQ*	社交网络	54.6K	10.7	11167	3780	33.8%	9.2
爱奇艺	视频	53.5K	6.4	7586	3481	45.9%	9.3
支付宝	生活	48.1K	10.2	983	231	23.5%	10.1
淘宝*	移动购物	47.5K	7.0	6003	3010	50.1%	8.1
腾讯视频	视频	47.3K	6.3	1429	68	4.8%	10.7
优酷	视频	40.9K	7.3	2058	262	12.7%	6.7
新浪微博*	社交网络	39.2K	5.3	5947	2715	45.7%	5.7
WiFi 万能钥匙	系统工具	36.4K	3.1	4808	2999	62.4%	3.0
搜狗输入法	系统工具	33.3K	11.0	898	40	4.5%	21.8
百度	资讯	32.4K	11.1	15651	3514	22.5%	12.8
腾讯新闻	资讯	28.7K	8.5	1051	11	1.0%	8.9
QQ 浏览器	资讯	27.8K	5.6	1369	43	3.1%	11.6
今日头条	资讯	27.4K	4.4	3538	179	5.1%	5.6
应用宝	应用市场	27K	11.4	2419	266	11.0%	11.6
快手	视频	24.4K	3.2	8273	4270	51.6%	3.5

腾讯手机管家	系统工具	24.2K	8.7	2463	1340	54.4%	8.7
高德地图	生活	24K	6.5	1225	51	4.2%	13.1
酷狗音乐	音乐	23K	8.6	1313	122	9.3%	12.2
QQ 音乐	音乐	21.7K	9.4	1132	65	5.7%	14.6
百度地图	生活	21.3K	8.8	2609	1438	55.1%	15.3
抖音	视频	19.4K	11.1	317	12	3.8%	8.3
京东*	移动购物	18.5K	10.9	5000	2377	47.5%	12.3
UC 浏览器	资讯	16.7K	7.4	4232	1624	38.4%	7.0
360 手机卫士	系统工具	15.4K	12.4	3670	1423	38.8%	19.1
全民 K 歌	音乐	14.7K	21.1	618	215	34.8%	17.3
美团	生活	13K	8.0	4752	1415	29.8%	6.9
拼多多*	移动购物	12.9K	6.6	2327	551	23.7%	7.8
王者荣耀*	游戏	12.5K	15.5	2350	1319	56.1%	12.3
美图秀秀	摄影录像	12.4K	5.4	1705	784	46.0%	5.8
火山小视频	视频	12.2K	11.9	410	16	3.9%	9.6
墨迹天气	生活	12K	4.2	10081	7093	70.4%	4.7
滴滴出行	生活	11.8K	8.6	943	117	12.4%	7.0
华为应用市场	应用市场	11.8K	N/A	0	0	0.0%	N/A
开心消消乐*	游戏	11.2K	19.7	2406	1738	72.2%	20.6
酷我音乐盒	音乐	11K	2.9	3778	69	1.8%	4.2
西瓜视频	视频	11K	11.5	866	100	11.5%	8.8
OPPO 应用商店	应用市场	10.8K	N/A	0	0	0.0%	N/A
猎豹清理大师	系统工具	9.9K	10.3	1803	388	21.5%	13.5
360 清理大师	系统工具	9.6K	17.3	327	8	2.4%	8.5
360 手机助手	应用市场	9.2K	7.6	1616	137	8.5%	8.4
WiFi 管家	系统工具	8.8K	19.5	1636	658	40.2%	15.7
讯飞输入法	系统工具	8.6K	6.0	1451	8	0.6%	10.1
百度手机助手	应用市场	8.2K	11.4	3849	437	11.4%	14.5
小米应用市场	应用市场	7.8K	N/A	0	0	0.0%	N/A
WPS Office*	商务办公	7.4K	6.0	1152	69	6.0%	7.8
美颜相机	摄影录像	7.1K	5.3	1600	691	43.2%	6.3
网易云音乐	音乐	7K	10.5	616	6	1.0%	12.2
网易新闻	资讯	6.7K	7.0	1441	93	6.5%	5.0
QQ 邮箱*	商务办公	6.6K	16.4	520	11	2.1%	10.4

* 详情会在 **RQ 2.4.** 结果中给出

根据应用功能划分，我们的 50 款目标 App 可以被分为 11 个类别。表 4.2 按每款 App 的热度排序，展示了每款 App 的类别和他们的仿冒率，同时还有他们的更新频率、关联样本总数等数据。在相同的类别下，多数应用之间的仿冒率差值都位

置在一个合适的水平。毫无疑问地，娱乐向类别（比如游戏和社交网络）吸引了更多仿冒应用开发者对其仿冒。而另一个类别移动购物也特别受到了仿冒应用开发者的“关照”，因为移动购物在中国也正在快速地发展。相对来说，商务方向的商务办公分类的应用就不是那么地吸引仿冒应用开发者了，这个领域下的 App 平均仿冒率只有 4.05%。上述四个类别的应用都在表中被加粗标识，读者可以自行查阅。

这个结果与我们在日常生活中观察到的结果相符，比起商务类用途，普罗大众更倾向于使用移动设备用作娱乐用途，消遣时间。仿冒应用的数量从某种角度上反映了人们在日常生活中如何使用移动设备，这是个十分有趣的发现。

本节小结：正如由我们的统计和计算揭示的那样，从一个应用市场中能找到的应用样本数量并不能与应用商店的仿冒率相对应。此外，App 本身与应用市场的关系也会影响到市场中对应 App 仿冒样本的数量。出乎我们意料的是，App 的更新频率并没有与仿冒率相关联。我们认为这是由于应用更新频率太高、而且重打包应用在我们的数据集中占少数而导致的结果。我们进一步观测到，与更新频率和应用热度相比，应用分类这一因素对 App 的仿冒率有更大的影响。

4.3 仿冒应用的发展轨迹

在这个视角下，我们希望结合时间维度，从我们的数据中挖掘信息。随着时间推移，仿冒应用的特征和行为模式是否有发生改变？这些年来，仿冒应用这个灰色产业是否有过变迁？为了解答这些问题，我们提出了以下几个研究问题：

RQ 3.1：在一个官方应用的新版本推出之后，仿冒应用开发者需要花多少时间去推出对应版本的仿冒版？换句话说，这些“山寨”版本会过多久出现？

RQ 3.2：一个仿冒应用开发者的安全证书可以存活多久？

RQ 3.3：随着时间的推移，仿冒应用是否存在一个变化模式？

RQ 3.1. 结果 我们计算出了每版 App 被仿冒的延迟时间和延迟的分布情况，结果显示在图 4.4 中。

出于多种原因，我们很难为研究中的所有 50 款目标 App 回溯出它们每个版本

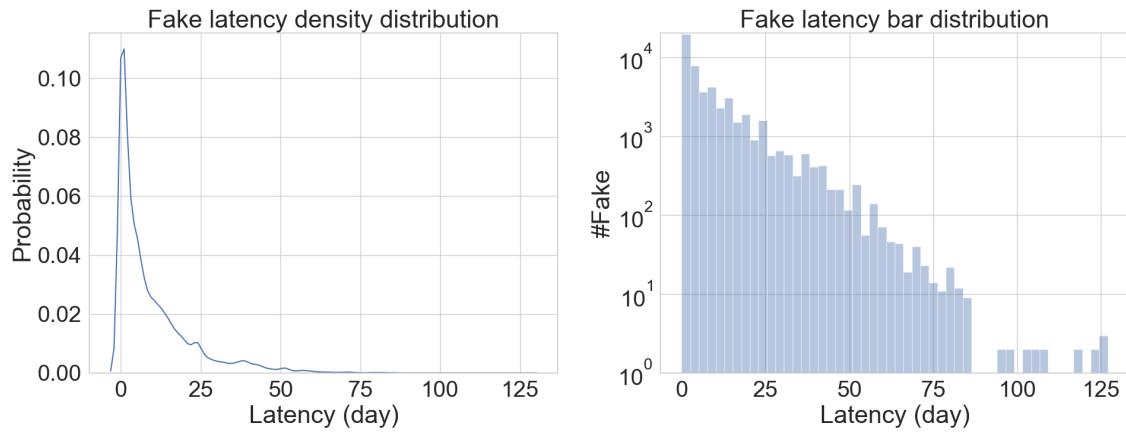


图 4.4: 仿冒延迟总体分布

更新的时间点，然而，我们凭借数据库中的数据大致地重现了他们的更新时间线。首先，我们对所有收集到的官方样本按照他们的 App 源进行分类，然后再按照版本号对 App 源类下的所有样本分类。举个例子，一开始是使用“爱奇艺”作为搜索源的所有官方样本都会被归回“爱奇艺”类下，然后再按版本号分类。这么做的原因是，即使是官方渠道发布的版本，有些 App 在不同应用市场上架的时间和内容也会略微有差别，可能会导致一个版本的正版 App 有多个样本的结果。这个做法就是为了消除上述情况带来的影响。之后，对于每款 App 和每个版本，我们都按照被爬取到的日期时间戳对对应的样本进行排序。这样的话，通过提取每个版本中第一个版本被爬取的时间，我们就可以知道这个版本的官方 App 最早的发布日期了。最后，将每款 App 中每个版本最早的发布日期串联起来，我们就可以大致重现每款 App 的更新时间线。

由于仿冒样本大多不是重打包应用，我们没办法为每个仿冒样本精确匹配一个对应的正版版本，为了找到某个仿冒样本的仿冒延迟，我们会提取出它被爬取的日期时间戳，然后将这个时间戳与正版更新时间线上的所有版本进行对比，找到不晚于这个仿冒样本发布的最晚发布官方版本的发行时间，然后取他们的时间差作为仿冒延迟。按照图 4.4 的结果，绝大多数仿冒样本在官方应用推出后的 20 天内就被发布了。根据我们的统计，有 60% 仿冒应用在正版 App 被推出的 6 天内就被发布出来了。这表明仿冒开发者的行动十分迅速。

RQ 3.2. 结果图 4.5 展示了不同仿冒应用安全证书在应用市场里存活时间的总体分

布。在左边的密度分布函数图中， x 轴表示其存活的时长， y 轴则表示了与 x 轴上的值对应的概率密度。曲线下的总面积为 1，任意两个 y 值 y_1 、 y_2 之间的曲线下面积是其对应的 x 轴上的值 x_1 、 x_2 在数据中占有的概率。比如说，图 4.5 中 x 轴从 0 到 200 之间的值对应的 y 轴曲线下方的区域面积接近 0.8，意味着约 80% 的仿冒应用安全证书不会存活多于 200 天。

断定一个仿冒应用安全证书存活市场的方法和我们计算应用更新频率的方法稍有类似，我们会把某个安全证书关联的所有样本都找出来，提取出其中最早和最晚被爬取的样本的发布日期时间戳，然后将他们的差值的绝对值当作是这个安全证书的存活时长。从时长上爬取到样本的日期与样本在应用市场上实际能存活的时间稍有不同，但由于我们的爬虫工具每日都从应用市场中爬取样本，而我们又没有其他方法可以知晓某款 App 具体在应用市场中上架了多久，我们只能近似地将上述提到的时间差当作是某个安全证书能在应用市场上存活的时间了。

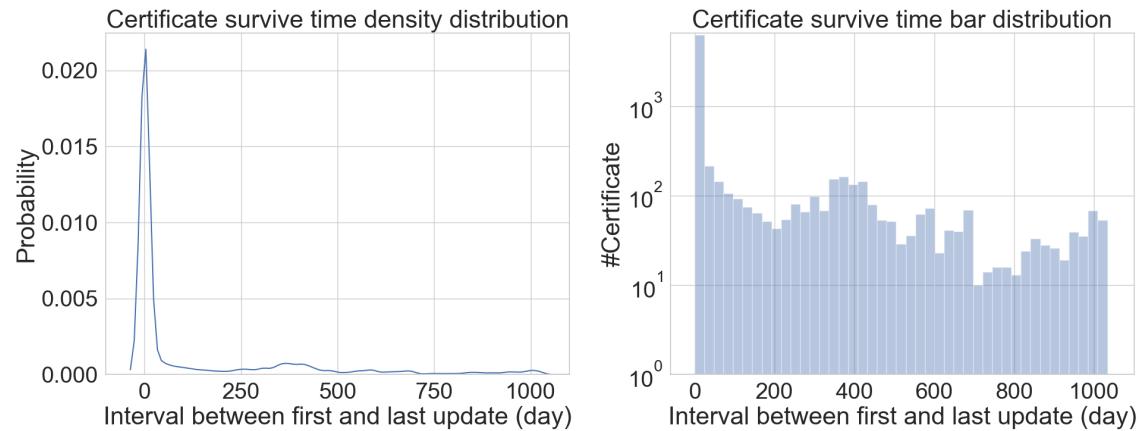


图 4.5: 仿冒应用安全证书存活时间分布

正如图 4.5 所示，仿冒应用安全证书存活时间的分布表明几乎所有仿冒应用安全证书都只能存活相当短的时间，这表明大多仿冒应用只会在一个很小的时间窗口里出现，然后迅速消失。这可以由大部分市场都有的一个安全机制解释。只要一款 App 被发现具有恶意行为或者违法行为，应用市场就会禁止开发者再使用该证书上传应用，也就是我们常见的封号处理。但是，我们也能发现有一部分的仿冒应用安全证书存活了相当久的一段时间。根据图表信息，我们可以看到有的仿冒应用安全证书的生命周期甚至贯穿了我们整个研究截取的时间周期。对于这个异常

样本，我们会在第??章 中有更详尽的案例分析。

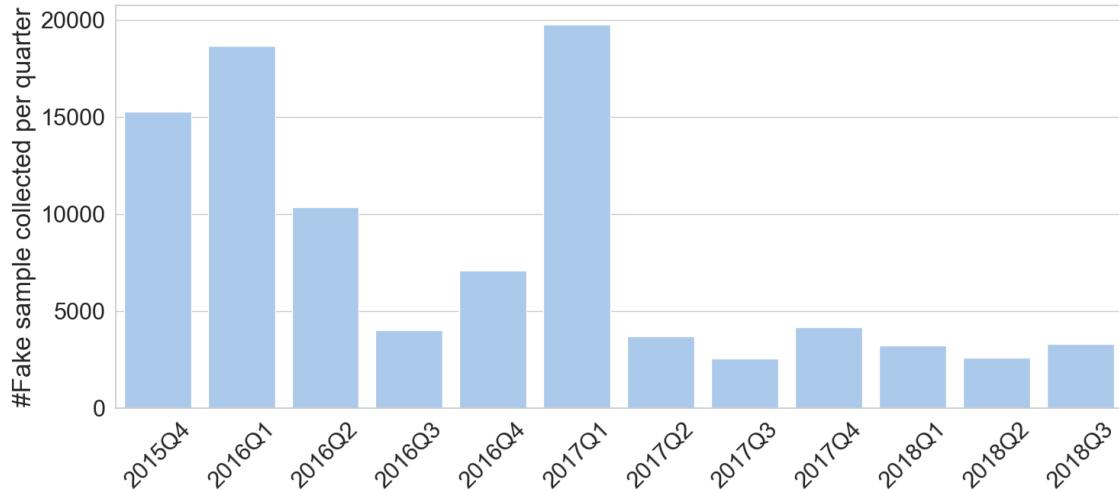


图 4.6: 每季度爬取到的仿冒样本数量（2015 年第四季度到 2018 年第三季度）

RQ 3.3. 结果 图 4.6呈现了我们的爬取工具自 2015 年第四季度起在每个季度爬取到的仿冒样本的总量。尽管我们每个季度都能爬取到新上架的大量仿冒样本，图表信息表明，市场上仿冒应用的总数正在呈现逐年减少的趋势。要注意的是，我们的研究仅仅针对仿冒应用。因此这个现象并不表明移动黑产的发展具有萧条的趋势。相反地，我们认为这个现象更有可能是由黑产内部改革而导致的。

从一方面看，随着信息技术发展，应用市场逐渐配备了更智能、严格而又强大的监管机制和安全系统，本研究中的仿冒应用开发者无疑更难以将仿冒的应用投放到市场上了。从另一方面看，新一代的恶意软件，比如 WannaCry 等勒索软件^[32] 也正在影响着整个黑产工业。与仿冒应用相比，这些新一代的恶意软件不仅更难以防范（传统的反病毒软件思路是提取已有恶意代码的特征，从而识别恶意行为，但这无法遏制新型的恶意行为），而且也似乎更容易能获取暴利。Wannacry 作为一种新型恶意软件，自 2017 年第二季度被首次发现开始，就能在数周内攻克数以千万计的设备，并让比特币的价格像搭火箭一样直线飙升^[33]。之后不久，在 2018 年的第一季度，移动设备上也爆发了一系列的加密采矿恶意软件^[34]。所以我们有理由猜测 2017 年第二季度和 2018 年第一季度的仿冒应用上架数量下跌是受到了新形态黑产的冲击。当然，要去证明这个猜想还需要采集更多数据、进行更深

入的研究，并不在本文的讨论范围之内，我们可将这个疑问放入第五章 展望与未来工作中。

本节小结：仿冒应用可以在极短时间内被研发并上架，而仿冒样本逐年下降的新增量，也许表明了仿冒应用产业正在陷入衰退期。此外，只有很小一部分的仿冒应用安全证书可以存活很长的一段时间，这表明应用市场的保护监管机制在一定程度上的确能发挥作用。

4.4 本章小结

本章分别从仿冒应用的基本特征、针对仿冒样本的量化分析和仿冒应用的发展轨迹三个不同视角对采集到的数据进行了分析，并在每个视角后的本节小结中概括了每个视角的结论。在下一章，我们将会从数据中挑选出一些较有代表性的
真实数据，印证我们在本章中的发现，同时为上述发现提供有力的事实依据。

第五章 总结与展望

5.1 总结

本文提出并实现了一个可用于生成 Android 应用程序动态函数调用图的技术方案——RunDroid。RunDroid 生产的函数调用图，可以准确的反映应用程序的执行过程，取得了有意义的研究成果。本文的主要工作如下：

1)：本文在传统的函数调用概念的基础上，提出了函数触发关系，用于描述两个方法之间的因果关系；并结合方法触发关系、方法对象等概念提出了拓展函数调用图的概念。

2) RunDroid 的设计与实现：RunDroid 利用源程序代码插桩和运行时方法拦截相结合的方式，获取应用方法执行信息，构建函数调用图；并在此基础上，利用方法和对象的关系补全到调用图中的方法间触发关系，展现运行过程中的 Android 特性行为。

3) 静动态工具的实验结果对比：本文将 RunDroid 产生的动态函数调用图和 FlowDroid 产生的静态函数调用图进行对比。相比 FlowDroid，RunDroid 产生的函数调用图能够体现应用程序的执行过程，表现函数间的调用关系和触发关系，准确地还原 Android 组件的生命周期。

4) 开源应用的统计实验：利用 RunDroid 构建开源 Android 应用的动态函数图，统计数据佐证了事件回调、Handler 等函数间触发关系在 Android 应用中的普遍性。

5) RunDroid 在错误定位领域的应用：相关实验结果实现，从 RunDroid 提供的函数关系信息可以反映出更多程序依赖信息，相比之前技术方案，方法间的因果关系模型更健全，实验的可靠性有所提升。

5.2 展望

虽然通过 RunDroid 还原得到的 Android 应用程序动态函数调用图，反映程序的运行时状态，但在实验过程中我们发现以下问题：

1) RunDroid 在捕获应用用户层方法时，采用的方案是源代码插桩方案。调用图构建的前置条件需要提供 Android 应用的源代码，因此，RunDroid 的运行对源代码高度依赖。

2) 在实验阶段，我们发现，当应用程序长时间运行时，应用程序会产生较多的日志。通常的，移动设备上的存储是有限的。因此，对于一些调用关系较为复杂的应用，RunDroid 的日志方案比较容易遇到日志存储的瓶颈。

3) RunDroid 中的运行时拦截器是基于 Xposed 框架实现的。Xposed 框架并不是适用于所有的 Android 手机，在一定程度给 RunDroid 的实验环境提出了额外的要求。

整体上，本文提出的 RunDroid 较为准确地还原出 Android 应用程序在运行过程的函数调用图。针对 RunDroid 实验中发现的不足，RunDroid 的后续工作可以从以下几个方面进行改进：利用字节码修改技术代替源代码修改方案以减少 RunDroid 运行过程中对源代码的依赖；引入基于 JVMTI 的调试环境，借助调试技术实现系统方法执行的拦截，摆脱对 Xposed 环境的依赖；通过静态分析技术确定运行过程的确定性路径，缩减待插桩的用户方法数量，进而减少运行时日志的产出量。

参考文献

- [1] STATCOUNTER. Mobile Operating System Market Share Worldwide, 2009 - 2020[EB/OL]. 2019 [February 23, 2020].
<https://gs.statcounter.com/os-market-share/mobile/worldwide/#yearly-2009-2020>.
- [2] CLEMENT J. Number of available applications in the Google Play Store from December 2009 to December 2019[EB/OL]. 2019 [January 4, 2020].
<https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>.
- [3] ANDOW B, NADKARNI A, BASSETT B, et al. A Study of Grayware on Google Play[J]. 2016 IEEE Security and Privacy Workshops (SPW), 2016: 224–233.
- [4] LUO L, FU Y, WU D, et al. Repackage-proofing android apps[C] // 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). 2016: 550–561.
- [5] WASSERMAN A I. Software engineering issues for mobile application development[C] // Proceedings of the FSE/SDP workshop on Future of software engineering research. 2010: 397–400.
- [6] CHEN S, FAN L, CHEN C, et al. StoryDroid: Automated Generation of Storyboard for Android Apps[C] // Proceedings of the 41th ACM/IEEE International Conference on Software Engineering, ICSE 2019. 2019.
- [7] CHEN S, XUE M, TANG Z, et al. Stormdroid: A streaminglized machine learning-based system for detecting android malware[C] // Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. 2016: 377–388.
- [8] CHEN S, XUE M, FAN L, et al. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach[J]. computers & security, 2018, 73 : 326–344.

- [9] CHEN S, XUE M, XU L. Towards adversarial detection of mobile malware: poster[C] // Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking. 2016 : 415–416.
- [10] FAN L, XUE M, CHEN S, et al. POSTER: Accuracy vs. Time Cost: Detecting Android Malware through Pareto Ensemble Pruning[C] // Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016 : 1748–1750.
- [11] MCAFEE. McAfee Mobile Threat Report Q1, 2018[R/OL]. 2018 [September 26, 2018].
<https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2018.pdf>.
- [12] YIN H, SONG D, EGELE M, et al. Panorama: capturing system-wide information flow for malware detection and analysis[C] // Proceedings of the 14th ACM conference on Computer and communications security. 2007 : 116–127.
- [13] FELT A P, FINIFTER M, CHIN E, et al. A survey of mobile malware in the wild[C] // SPSM@CCS. 2011.
- [14] ZHOU W, ZHOU Y, JIANG X, et al. Detecting repackaged smartphone applications in third-party android marketplaces[C] // CODASPY. 2012.
- [15] ZHENG M, SUN M, LUI J C S. DroidAnalytics : A Signature Based Analytic System to Collect , Extract , Analyze and Associate Android[C] // . 2013.
- [16] LINARES-VÁSQUEZ M, HOLTZHAUER A, POSHYVANYK D. On automatically detecting similar Android apps[C] // Program Comprehension (ICPC), 2016 IEEE 24th International Conference on. 2016 : 1 – 10.
- [17] GLANZ L, AMANN S, EICHBERG M, et al. CodeMatch: obfuscation won't conceal your repackaged app[C] // Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 2017 : 638 – 648.
- [18] WANG H, GUO Y, MA Z, et al. WuKong: a scalable and accurate two-phase ap-

- proach to Android app clone detection[C] // Proceedings of the 2015 International Symposium on Software Testing and Analysis. 2015 : 71 – 82.
- [19] ZHANG F, HUANG H, ZHU S, et al. ViewDroid: towards obfuscation-resilient mobile application repackaging detection[C] // WISEC. 2014.
- [20] CRUSSELL J, GIBLER C, CHEN H. Attack of the clones: Detecting cloned applications on Android markets[C] // European Symposium on Research in Computer Security. 2012 : 37 – 54.
- [21] CRUSSELL J, GIBLER C, CHEN H. Scalable semantics-based detection of similar Android applications[C] // Proc. of ESORICS : Vol 13. 2013.
- [22] CHEN K, LIU P, ZHANG Y. Achieving accuracy and scalability simultaneously in detecting application clones on Android markets[C] // Proceedings of the 36th International Conference on Software Engineering. 2014 : 175 – 186.
- [23] Google Play[EB/OL]. [February, 23, 2020].
<https://play.google.com/store/apps>.
- [24] Myapp[EB/OL]. [September 26, 2018].
<http://sj.qq.com/myapp/>.
- [25] Baidu App Store[EB/OL]. [September 26, 2018].
<https://shouji.baidu.com/>.
- [26] Xiaomi App Store[EB/OL]. [February 23, 2020].
<http://app.mi.com/>.
- [27] iiMedia GROUP. 2018-2019 中国中国移动应用商店市场监测报告 [R]. 2018 [February 23, 2020].
- [28] LEVENSHTEIN V I. Binary codes capable of correcting deletions, insertions, and reversals[C] // Soviet physics doklady : Vol 10. 1966 : 707 – 710.
- [29] Hiapk[EB/OL]. [September 26, 2018].
<http://apk.hiapk.com/>.
- [30] Analysys Figure[EB/OL]. [September 26, 2018].
<http://zhishu.analysys.cn/>.

- [31] LIFTOFF. Mobile Gaming Apps Report[EB/OL]. 2018 [September 26, 2018].
https://cdn2.hubspot.net/hubfs/434414/Reports/2018%20Gaming%20Apps/Liftoff_2018_Mobile_Gaming_Apps_Report_Aug.pdf.
- [32] WIKIPEDIA. Ransomware[EB/OL]. 2018 [September 26, 2018].
<https://en.wikipedia.org/wiki/Ransomware>.
- [33] JONAS B. Global Malware Campaign WannaCry is Affecting Bitcoin's Price[EB/OL]. 2017 [September 26, 2018].
<https://hacked.com/global-malware-campaign-wannacry-affecting-bitcoins-price/>.
- [34] COMODO. Comodo Cybersecurity Q1 2018 Global Malware Report: cybercriminals follow the money, cryptominers leap ahead of ransomware[EB/OL]. 2018 [September 26, 2018].
<https://blog.comodo.com/comodo-news/comodo-cybersecurity-q1-2018-global-malware-report/>.

致 谢

XXX

二〇二〇年五月

攻读学位期间发表的学术论文

1. Chongbin Tang, Sen Chen, Lingling Fan, Lihua Xu, Yang Liu, Zhushou Tang, and Liang Dou, “A large-scale empirical study on industrial fake apps,” in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, IEEE Press, 2019: 183-192. (发表于 CCF A 类推荐学术会议, 软件工程方向顶级会议 ICSE2019)