2020 届硕士专业学位论文

分类号：_____ 学校代码：10269

密　级：_____ 学　　号：**********

華東師範大學

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：*****

研 究 方 向：*********

指 导 教 师：*** **

学位申请人：***

2020 年 5 月

Thesis for master's degree in 2020 University Code: 10269

Student ID: **********

EAST CHINA NORMAL UNIVERSITY

A Large-Scale Empirical Study on Industrial Fake Apps

Department:	School of Computer Science and Technology
Major:	*****
Research Direction:	*****
Supervisor:	***
Candidate:	***

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）。

*** 硕士学位论文答辩委员会成员名单

姓名	职称	单位	备注
***	***	******	主席
***	***	******	
***	***	******	

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，我们按照排行榜确定了全网最受欢迎的前 50 个应用，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，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，从仿冒应用的基本信息特征开始，再将仿冒应用数据与其来源的应用市场结合，最后引入时间因素对数据进行挖掘分析。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，最后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 An-

droid underground industry.

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

目录

插　图

表　格

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]。

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工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，

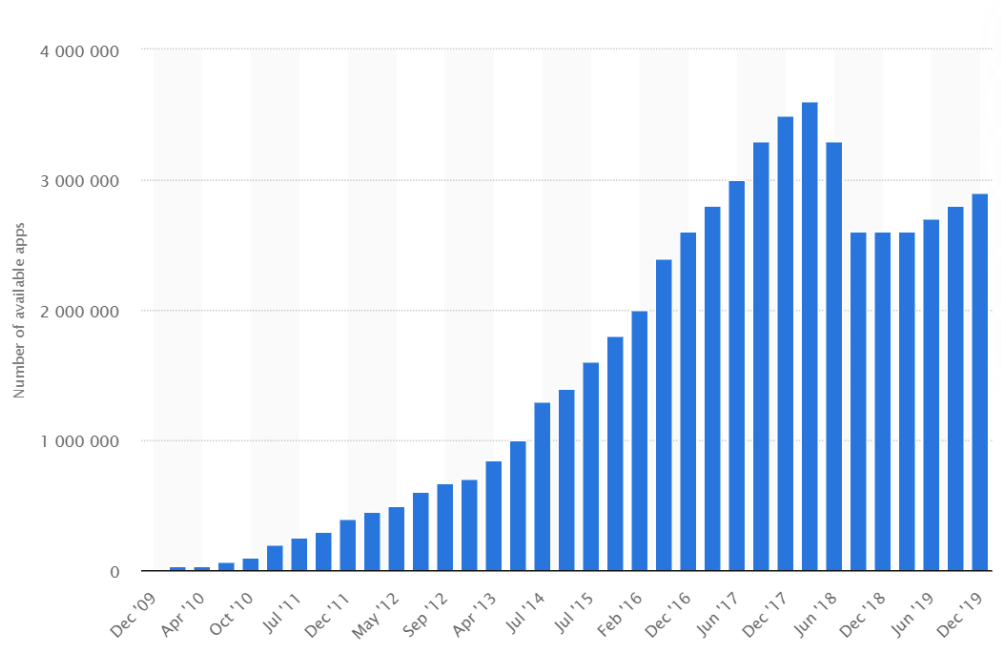

图 1.1: Google Play 应用商店架上应用总数变化趋势

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出重打包应用，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] 则用从每个方法从提取的 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，作者借助犇众信息的移动安全威胁数据平台 Janus[1]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：

[1]https://www.appscan.io/

- **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。
- **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　本文遇到的困难与挑战

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。前文提及犇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??章 提供了本篇研究的概览，并仔细分布解释了数据收集流程。

第??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??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??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。

第三章 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，同时也会介绍仿冒应用和 Android 安全证书机制的相关知识。

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。

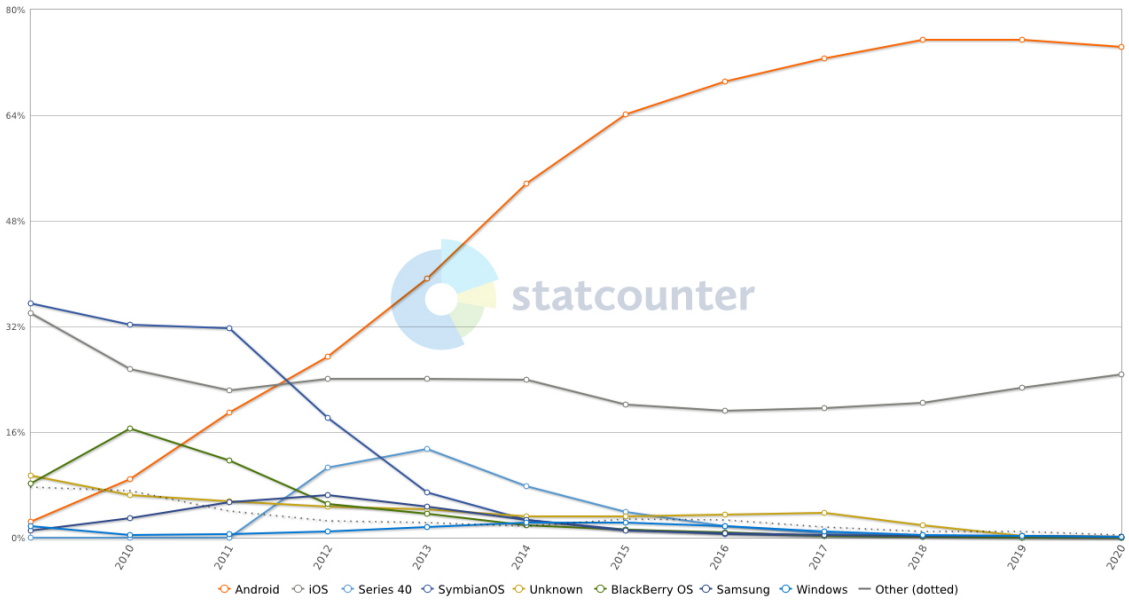

图 2.1: 2009 至 2020 年移动端操作系统市场份额变化图

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 App）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 Develop 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。

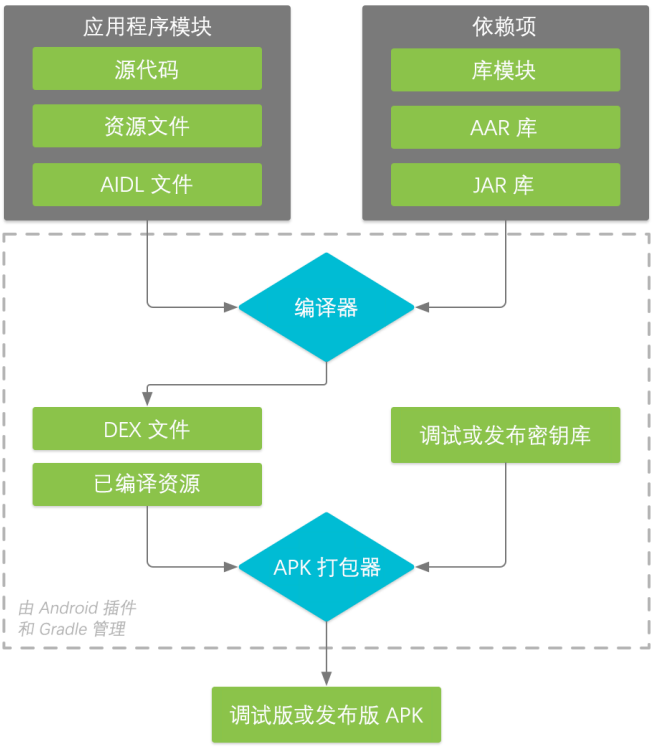

图 2.2: Android App 构建流程

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。

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，也先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）。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。

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。

与上述的四个问题对应，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

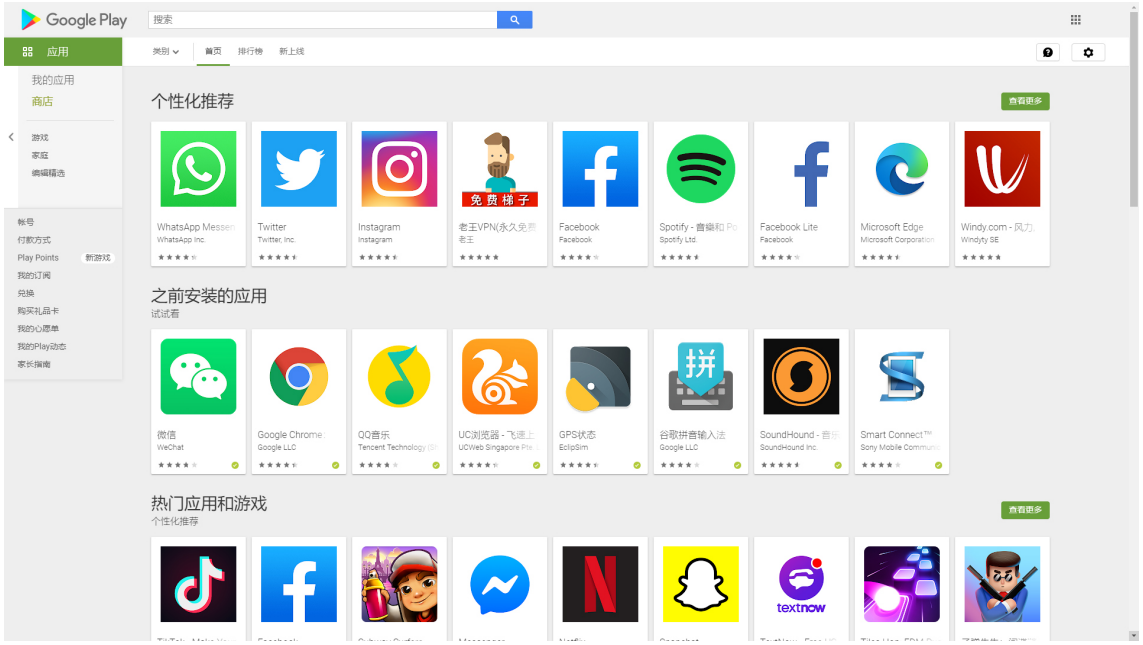

图 2.3: Google Play 应用商店首页（从桌面端浏览）

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 本身的开发，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

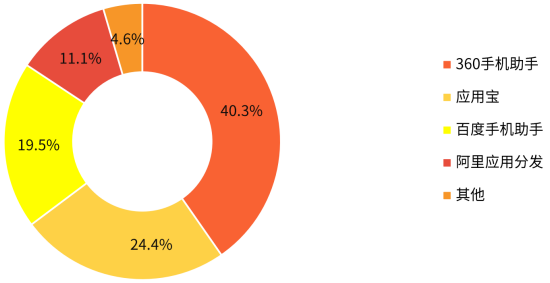

图 2.4: 2018 中国第三方移动应用商店用户首选使用品牌分布

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。

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

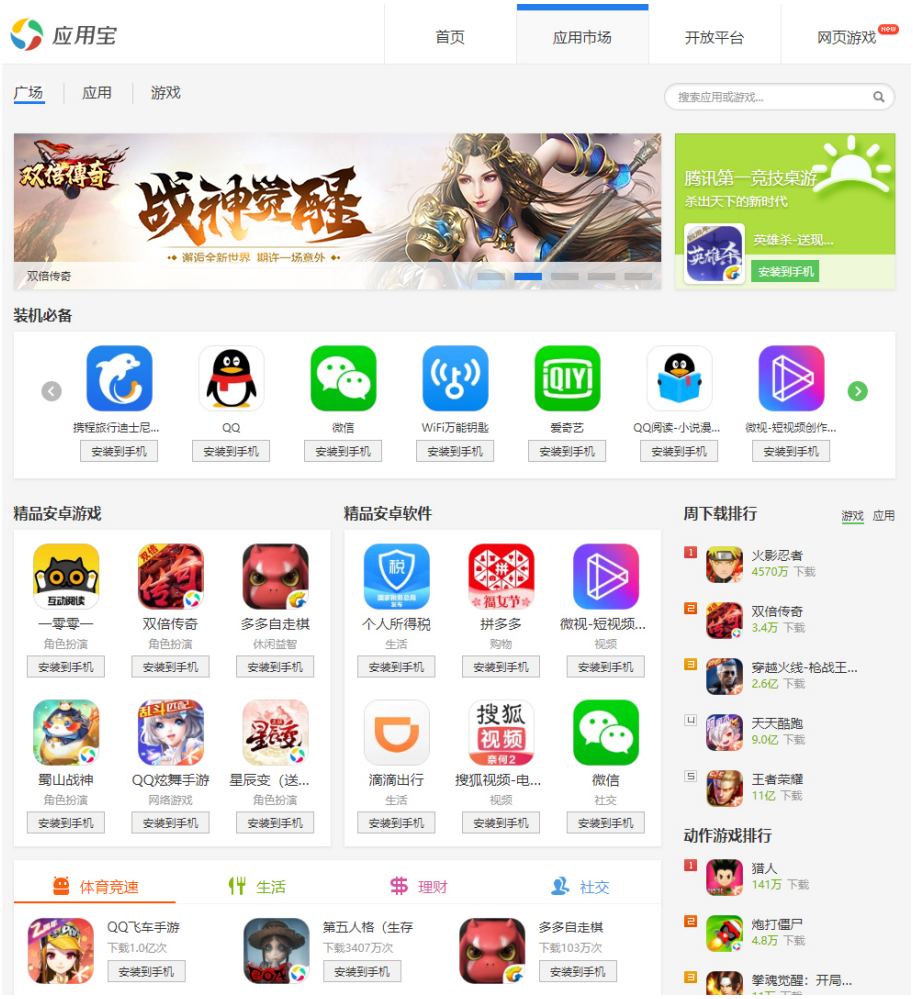

图 2.5: 腾讯应用宝应用市场首页（从桌面端浏览）

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。

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，越新的签名模式能越好地保障 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。

最后要提及的是，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。

2.6 本章总结

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　总结与展望

3.1　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）拓展函数调用图：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。

3.2 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 approach 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].

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)