移动应用安全解析学: 成果与挑战

杨 威[1], 肖旭生[2], 李邓锋[1], 李豁然[3], 刘譞哲[3], 王浩宇[3],
郭 耀[3], 谢 涛[1]

[1]伊利诺伊大学香槟分校 计算机系 厄巴纳 美国 61801
[2]NEC 美国研究所 普林斯顿 美国 08540
[3]北京大学 信息科学技术学院 北京 中国 100871

摘要 随着移动应用(App)的广泛使用, 移动应用的安全事件也频频发生。从数以亿计的移动应用中准确地识别出潜在的安全隐患成为了信息安全领域重要的难题之一。移动应用数量级增长的同时, 也产生了海量的应用安全数据。这些数据使得移动应用的安全解析成为了可能。本文分别从用户界面解析、重打包应用检测、应用功能与安全行为一致性检测、基于上下文的恶意行为检测、终端用户应用管理和使用行为分析这五个方面介绍了移动应用安全解析学目前的成果。同时, 基于以上的研究成果, 对未来的研究方向进行了展望, 并讨论了这些研究方向面临的挑战。

关键词 移动安全, 数据解析, 机器学习, 程序分析
中图法分类号 TP309.1 DOI 号 10.19363/j.cnki.cn10-1380/tn.2016.02.001

Security Analytics for Mobile Apps: Achievements and Challenges

YANG Wei[1], XIAO Xusheng[2], LI Dengfeng[1], LI Huoran[3], LIU Xuanzhe[3], WANG Haoyu[3], GUO Yao[3], XIE Tao[1]

[1]Department of Computer Science, University of Illinois at Urbana–Champaign, Urbana, 61801, USA
[2]NEC Laboratories America, Princeton, 08540, USA
[3]School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China

Abstract With the increasing popularity of mobile apps, security incidents of mobile apps frequently occur. Accurately identifying security threats among billions of mobile apps has become an important and difficult topic in information security. In the meantime, the increasing number of mobile apps leads to a massive amount of mobile security data, which enables security analytics for mobile apps. In this article, we illustrate recent research achievements on mobile security analytics from five different perspectives: User Interface (UI) analytics, identification of repackage apps, consistency checking between apps functionality and security behaviors, context-sensitive detection of malicious behaviors, and client app management/usage. We also discuss future directions on security analytics for mobile apps, along with challenges in these directions.

Key words mobile security, data analytics, machine learning, program analysis

1 引言

1.1 移动应用安全解析学概述

随着移动终端(如智能手机, 平板电脑等)的普及, 基于应用商店的发布模式成为软件应用的重要模式, 移动应用产业得到了快速的增长。截至 2015 年, 在 Apple Store、Google Play 等主流应用商店和 360、豌豆荚等第三方应用商店上已经积累了数百万移动应用。由于移动应用可从移动终端获得大量的敏感信息, 且其本身能通过移动市场及广告商产生高利润, 移动应用频频遭到了黑客的攻击。 移动应用安全事件频发。究其原因, 主要来源于如下几个方面: 简单的应用市场安全审查、粗粒度的权限系统、 有限的系统级安全监测, 以及鼓励快速传播的应用分派模

通讯作者 1: 刘譞哲, 博士, 北京大学信息科学技术学院副教授, Email: liuxuanzhe@pku.edu.cn。**通讯作者 2:** 郭耀, 博士，北京大学信息科学技术学院副教授，Email: yaoguo@pku.edu.cn。
本文工作得到国家科技部 863 计划项目(2015AA01A202)、国家自然科学基金委创新群体项目(61421091)、海外合作基金项目(61529201)和国家自然科学基金青年-面上连续资助项目(61370020)、美国自然科学基金(CNS-1513939)、谷歌教授研究奖的资助。
收稿日期: 2016-3-8; 修改日期: 2016-4-26; 定稿日期: 2016-5-1

式, 等等。海量的移动应用缺乏集中有效的安全管理等, 都导致大量的恶意移动应用, 或是存在安全漏洞的低质量移动应用, 被发布在移动应用市场。如何从海量的移动应用中精确地识别出可能会给移动终端带来安全隐患的应用, 成为移动应用安全研究的重要问题之一。

从量大面广、不断推陈出新的移动应用中准确识别恶意应用是很有挑战性的问题之一, 得到了学术界和工业界的广泛关注[19] [13] [20]。但同时, 这些海量应用为移动应用的安全分析提供了充足的历史数据进行参考。利用机器学习的方法来进行数据解析(Data Analytics)已成为应用安全分析的重要手段。图 1 描述了移动应用安全解析的几个关键步骤。

首先, 安全解析会通过程序分析或文本分析在移动应用中提取相关特征(feature), 同时, 如有必要, 将提取相关的领域知识(domain knowledge)。此后提取的特征将会被进一步地处理。处理之后, 整个特征数据集会被分为训练样本集(training data set)及测试样本集(testing data set)。机器学习算法利用训练样本数据集, 并参考提取的领域知识, 生成机器学习模型。而测试样本数据则用于样本测试以评估生成的机器学习模型, 并产生最后的评估结果。

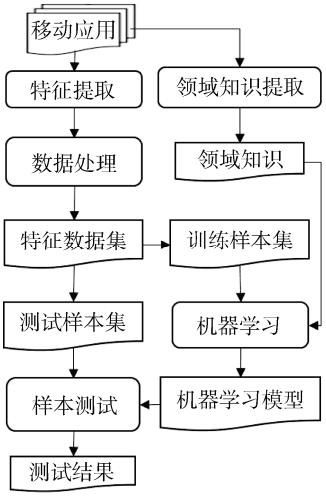


图 1 移动应用安全解析的关键步骤

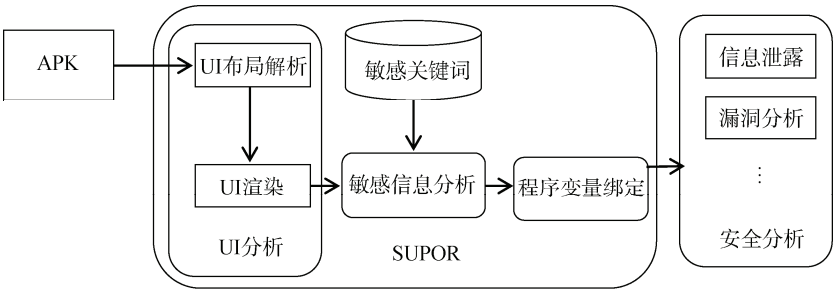


图 2 SUPOR 系统的概述

本文从五个方面, 概述了目前学术界如何利用海量的移动应用数据对应用的安全性进行解析, 并选取一些典型工作加以介绍。本文第二节介绍了基于用户界面解析的敏感信息安全管理机制。其中着重介绍了两个用于发现并保护用户输入的敏感信息的安全技术:SUPOR [13]和 UIPicker [25]。本文第三节介绍了重打包应用的检测技术。其中着重介绍了 WuKong [9], 去除第三方库干扰且可扩展的重打包应用检测系统。本文第四节介绍了软件功能与安全行为的一致性(consistency)研究。这类研究致力于检测应用中非用户所期望的敏感行为。此类研究一般基于一个重要假设, 即应用市场中的软件功能描述反映了用户对应用行为的期望。在这个假设的基础上, 此类研究利用文本分析等技术来检测真实的应用安全行为是否与用户所能觉察到的功能描述保持一致。本文第五节介绍基于上下文(Context)的恶意行为检测技术。这类研究表明一个行为发生的上下文可以有效反映应用执行这个行为的目的。而执行行为的目的可以作为甄别行为的恶意性的主要依据。这类研究通过程序分析来提取安全行为的上下文, 并通过这些提取的上下文来辅助机器学习算法检测恶意行为。本文第六节介绍了移动终端应用的管理和使用。这类研究一般基于 Android 用户数据分析, 然后从数据中发现可能存在恶意行为的应用。此小节探究了三种可供安全分析参考的用户数据, 即非主动安装, 用户卸载行为及异常流量检测。当数据维度增多、规模增大、更高效准确的算法被提出之后, 此类方法帮助我们发现更多的安全隐患。

2 用户界面解析(UI Analytics)

智能手机用户通过手机应用的图形用户界面(GUI, Graphical User Interface)与手机应用进行交互, 例如选择图标或者输入用户名和密码。通过这些交互过程, 用户与手机应用之间进行了信息的输入与

输出。在这些用户输入的信息中, 包括很多用户的敏感信息, 例如用户名、身份证号码、银行卡号码等。如果这些输入的敏感信息被手机应用无意的以明文形式发送出去, 或者恶意的发送到远端服务器, 将会对用户的隐私安全造成严重的危害。

当前, 移动安全研究主要集中在能够被智能手机操作系统和应用开发 API 所管理的用户敏感信息, 例如联系人信息、GPS 位置, 等等。智能手机操作系统通过权限对这些敏感信息进行管理。为了访问这些敏感信息, 移动应用开发者必须声明相应的权限, 并经过用户同意后, 该手机应用才会被授予请求的权限。然而, 这些被权限管理的数据通常并没有包括所有的用户敏感信息。作为一种主要的用户敏感信息, 用户输入的信息就不能被手机操作系统的权限进行管理。因此, 我们需要新的安全管理机制来发现并且保护用户输入的敏感信息。

SUPOR 提出了一种结合静态分析和自然语言处理的方法, 如图 2 所示。该方法对用户界面进行静态分析, 自动找出接受用户输入敏感信息的文本框, 并把这些文本框跟手机应用程序里面存储这些文本框输入的程序变量关联起来。利用这种关联, 我们就可以使用现有的信息泄露分析技术来检测手机应用是否泄露了用户输入的敏感信息。

此外, UIPicker 提出了一种结合机器学习、自然语言处理和程序分析的方法, 如图 4 所示。 该方法利用大量的用户界面显示的信息生成分类器, 用于判断描述用户界面控件的提示文字是否含有敏感信息的提示。为了提高发现潜在安全隐患的精准度, UIPicker 进一步利用程序分析对用户界面的控件进行分析, 从而保证对用户界面显示的信息的分析结果更加精确和全面。

2.1　问题描述

智能手机的界面包含着各式各样的 UI 控件, 例如按钮、文本框和图文列表。每一种控件都接受某一类型的用户输入。比如按钮接受点击、文本框接受用户输入的文字、图文列表接受用户的选择等。这些控件一般都有相应的文字提示, 来告诉用户他们希望得到的用户输入。图 3 是某 Android 应用的用户登录界面。该用户界面包括两个文本框, 第一个文本框接受用户输入的帐号, 第二个文本框接受用户输入的密码。显而易见, 这些输入都属于敏感信息, 以明文传输这些信息到远程服务器会造成对用户隐私的危害。

图 3　用户登录界面

自动检测 UI 控件是否接受包含有敏感信息的用户输入主要存在三点挑战: (1) 由于用户输入不由操作系统的权限进行管理, 现有的针对系统 API 的方法都没法使用。 判定用户输入是否包含有敏感信息, 需要分析相关 UI 控件的文字提示信息。 (2)如何精确的把用户界面显示的文字提示与 UI 控件对应起来? 每个用户界面都有很多个 UI 控件, 而每个 UI 控件的附近又可能有多处文字提示。例如, 在图 3, 第一个文本框上面的文字提示是帐号, 而下面的文字提示是密码。因此, 准确的匹配 UI 控件及其文字提示是判断用户输入是否包含敏感信息的前提。(3) 如何判定文字提示的内容是要求用户输入敏感的信息? 文字提示的内容都是用自然语言写的, 而这些内容的语义没法被程序直接分析。

a) 用户界面的分析

针对检测用户输入敏感信息的第一点挑战, 即分析相关 UI 控件的文字提示信息, SUPOR 和 UIPicker 都提出了对用户界面分析的技术。智能手机的用户界面一般采用声明式语言进行定义。比如, Android 应用通常有多个用户界面。每个用户界面统一使用 XML 定义用户界面, 包括用户界面的布局, UI 控件的定义, 还有 UI 界面显示的资源(图标和字

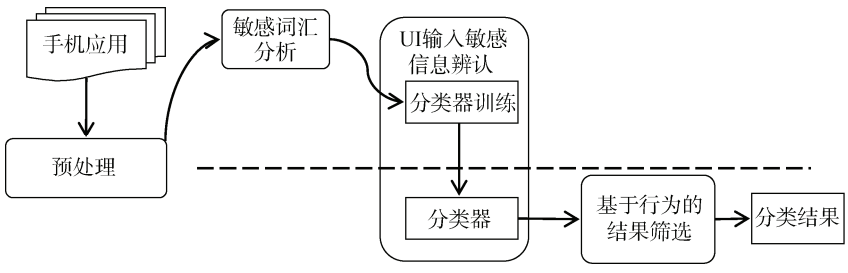


图 4　UIPicker 系统的概述

符串)。通过分析这些定义文件, 我们可以在不运行应用的情况下就知道一个手机应用有多少个界面和每个界面的控件。 同时, 通过利用 ADT(Android 开发工具)的类库, 我们也可以静态地渲染每个 UI 界面。这些分析使得 SUPOR 和 UIPicker 都能把 UI 控件的文字信息抽取出来, 并通过分析这些文字信息来判定 UI 控件是否会接受用户输入的敏感信息。

b) 敏感关键词的数据收集

由于智能手机的屏幕尺寸的限制, UI 控件的文字提示一般都是短语或者单词, 很少有成段的文本提示用于描述一个 UI 控件所要接受的用户输入。因此, 使用基于关键词的自然语言分析技术可以有效地分析这些文字提示。为了收集表示敏感信息的关键词, SUPOR 和 UIPicker 提出了两个方法。

方法一: 自然语言处理与分析

SUPOR 首先收集了 54,371 个 Android 应用的 UI 文字信息, 然后利用自然语言处理的技术, 对频繁出现的短语和句子进行结构分析, 抽取出名词和名词短语。在抽取出的这些名词和名词短语里面, 再通过人工分析并划分出 10 类关于敏感信息的关键词。

方法二: 基于 Chi-Square 的聚类

UIPicker 首先根据初始给定的敏感词语对用户界面的定义文件进行分类, 然后利用 Chi-Square 测试的方法抽取在这些界面定义文件里面具有代表性的词语。这些抽取出来的词语再通过人为筛选, 然后扩充敏感关键词。

c) 用户输入分类

针对检测用户输入敏感信息的第二和第三点, SUPOR 和 UIPicker 提出了不同的解决方法。

方法一: 用户界面分析

SUPOR 通过静态的 UI 渲染, 计算出每个 UI 控件和文字提示的绝对坐标。这样, SUPOR 就进一步地能计算出每个文字提示到达任何一个文本框的距离。对于一个 UI 控件, 用户一般都是看离该控件最近的文件提示来确定该 UI 控件接受哪种内容的用户输入。基于这个观察, SUPOR 挑选最近的文字提示, 并且利用收集的敏感关键词来判定该 UI 控件是否接受用户输入的敏感信息。

方法二: 建模解析

不同于 SUPOR, UIPicker 收集 UI 控件的文字信息以及其布局定义文件里面的资源定义, 并利用这些信息来对 UI 控件进行归类。 UIPicker 利用收集的敏感关键词做为特征, 采用支持向量机(Support Vector Machine, SVM)训练出一个分类器, 然后利用该分类器可以对任意一个 UI 控件进行分类, 判定其是否接受敏感输入。

2.2 实验结果

实验结果表明, SUPOR 和 UIPicker 提出的方法都达到了大于 90%的精确度。 SUPOR 由于采用了用户界面分析的方法来匹配文字提示和文本框, 因而在精确度上可以达到 97.3%, 高于 UIPicker 的 93.6%。UIPicker 的机器学习方法除了可以应用在文本框上, 在其他 UI 控件上也达到了很高的精确度。

2.3 展望和挑战

用户通过用户界面与手机应用进行交互。SUPOR 和 UIPicker 专注于分析用户界面中的文字信息。除了文字信息, 用户界面还包含图标等其他信息。全面分析这些信息有助于检测更多的恶意手机应用。例如, 恶意手机应用可以通过假的用户界面以骗取用户输入的信息。如何对用户界面进行更深度的分析以保护用户的安全和隐私将会成为一个重要的研究方向。

3 重打包检测分析

3.1 研究背景

相对于 iOS 应用, Android 应用更加容易被破解, 且由于 Android 系统的开源特性, Android 应用造成的安全隐患也更大。目前有很多的反编译工具可以使用。恶意的开发者可以将原应用反编译之后, 进行修改, 例如替换掉原应用中的广告库或者加入一些恶意代码, 然后重新打包并且发布出去。应用重打包的行为不仅侵犯了开发者的利益, 也严重威胁到了用户的安全和隐私。应用重打包已经成为移动平台上恶意软件传播的主要方式。

目前关于 Android 应用的重打包检测有很多相关工作, 可以分为如下几类:

(1) 基于哈希的检测技术

DroidMOSS [22] 使用模糊哈希技术来对应用中指令序列生成指纹来检测重打包应用。类似的, Juxtapp [18] 使用特征哈希来对应用产生指纹。

(2) 基于应用静态特征的技术

FSquaDRA [26]和 PlayDrone [15] 使用应用中的资源文件作为特征来进行重打包检测。PiggyDroid [22] 使用 API 调用等静态特征来做检测。ViewDroid [6] 使用用户界面的视图作为特征。

(3) 基于代码克隆检测的技术

DNADroid [12] 使用了程序依赖图(PDG)作为特征来进行检测。Centroid [14] 通过找到程序依赖图

的集合中心来进行比较。

但是现有的应用重打包检测工作主要存在两点挑战: (1) 如何在达到准确性的同时保持可伸缩性? 应用市场存在上百万的应用, 因此应用重打包检测系统需要在上百万个应用中快速准确的找出重打包应用, 并保证在有新应用添加时能够增量式的快速检测新应用是否是重打包应用。(2) 如何准确的去除第三方库的影响? Android 应用的特点之一是大量的使用了第三方库, 而第三方库会对检测重打包应用带来很大的干扰。

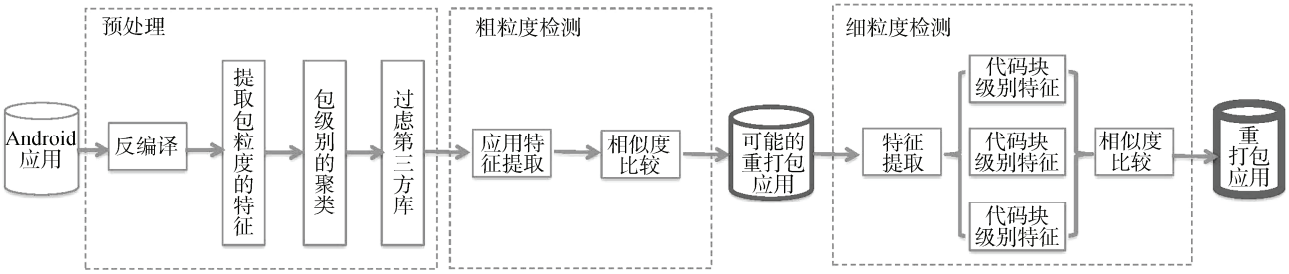


图 5　WuKong 的工作流程图

针对这两点挑战, WuKong [9] 提出了两个关键技术: (1)提出一个两阶段的应用重打包检测方法。第一阶段先使用简单的静态特征快速找出可能的重打包应用对, 然后在第二阶段使用更准确的代码克隆检测技术对找出的应用对进行细粒度比较。(2)提出一种基于聚类的方法来准确的识别并过滤第三方库, 这种方法可以找出不同版本的第三方库并且不受混淆的影响。

3.2　系统概述

如图 5 所示, WuKong 的工作流程包括预处理阶段、粗粒度检测阶段和细粒度检测阶段。预处理阶段先对应用程序进行反编译, 提取包粒度的特征, 进行聚类, 然后根据聚类结果过滤第三方库。在粗粒度阶段, 根据应用的静态特征快速的找出可能的重打包应用对。然后在细粒度检测阶段, 对于可能的重打包应用对, 提取代码块级别特征, 然后进行相似度比较, 从而找出重打包应用。

3.3　第三方库过滤

3.3.1　挑战

很多研究工作都是通过白名单的方式来过滤第三方库, 即通过比较包名是否与白名单中的第三方库包名相同。然而, 这种方法存在一些问题。首先, 现有的工作都不可能建立一个完整的第三方库白名单。大部分工作都使用了大约 100 个左右的第三方库白名单, 远远少于现有第三方库的数量。其次, 在 Android 应用中, 有超过 50%的第三方库都存在不同程度的混淆 [5]。被混淆过的代码中, 包名有可能发生改变, 例如包名被更改为“com/a/b”。因此, 被混淆后的第三方库不能通过白名单的方式来过滤。

3.3.2　基于聚类的第三方库过滤

Android 应用中的第三方库使用有两大特点: (1) 第三方库一般会被大量的应用所使用; (2) 开发者在使用第三方库的时候一般不会对其进行修改, 因此可以通过聚类的方法将第三方库检测出来。根据这两个特点, 如果我们对大量应用在包粒度提取特征并且进行聚类, 那么属于第三方库的包就会聚到一个很大的类里面, 通过这种方法我们可以将第三方库检测并过滤。

对于每个应用, 我们在包粒度提取所有的 Android API 调用特征, 其中包括不同 API 的调用的次数。然后, 根据 API 调用特征来进行聚类。在聚类的时候进行严格匹配, 即只有 API 调用特征完全一样的包我们才会聚在一起。因此, 我们的聚类算法是首先对每个包根据 API 调用次数进行排序, 只有调用次数完全一样的时候, 才比较它们的特征并且进行聚类。

基于聚类的第三方库过滤有以下优点:

(1) 能够在没有先验知识的情况下检测出第三方库。我们检测到的第三方库数量远远超过白名单方式标记的库的数量。

(2) 能够检测第三方库的不同版本, 只要有足够多的应用使用这个第三方库。

(3) 代码混淆对于我们的检测没有影响, 因为我们使用了 Android API 特征, 在 Android 应用中常用的命名混淆对我们的特征没有影响。

3.4　两阶段的重打包检测

3.4.1　粗粒度检测

为了进行快速比较, 在粗粒度检测阶段, WuKong 使用比较简单的 Android API 特征, 主要是不同 API 的调用次数。每个应用被表示成一个 API 向量。原理是如果两个应用核心代码相似的话, 那么它们的 API 调用也比较相似。尽管这样会造成一些误

报(两个不同的应用的 API 调用向量相似), 但是漏报率会非常低。WuKong 使用曼哈顿距离来表示两个 API 向量的相似度, 如果两个应用的距离小于给定阈值, 就会被选出来进行细粒度检测。

3.4.2 细粒度检测

对于粗粒度检测阶段过滤出来的可能的重打包应用对, WuKong 提取更详细的特征来比较它们的相似度。细粒度检测是基于已有的基于计数的代码克隆检测技术 Boreas [23] [24]。

WuKong 将每个变量在不同计数环境下出现的次数作为特征, 对每个变量得到一个特征计数向量。计数环境用来描述代码片段中变量的行为特征。WuKong 使用了如下的计数环境来描述变量的特征: (1)出现且使用次数;(2)被定义的次数;(3)在条件判断语句中出现次数;(4)在第一层循环中出现次数;(5)在第二层循环中出现次数;(6)在第三层及更深循环中出现次数;(7)在加/减运算中出现的次数;(8)在乘/除运算中出现的次数;(9)作为数组下标次数;(10)被常量表达式定义次数。

对每个变量得到一个特征计数向量之后, 可以对于每个代码片段得到一个特征计数矩阵。因此, 每个应用的特征由一系列的特征计数矩阵表示, 从而两个应用之间的相似度就是比较它们的特征计数矩阵有多少是相似的。

特征向量的相似度可以通过比较两个向量之间的余弦相似度得到。为了计算特征矩阵 A 和 B 的相似度, 对于矩阵 A 中的每个向量, 首先找到与其最大相似匹配的矩阵 B 中的另一向量, 然后两个矩阵的相似度是由所有向量相似度的乘积得到。应用之间的相似度, 是比较两个应用中有多少相似的代码块。

3.5 实验结果

WuKong 选取了国内五个第三方应用市场中超过 10 万个应用进行实验, 如表 1 所示。

对应用进行预处理之后, 我们发现有超过 60%的代码来自第三方库, 因此这些代码可以被过滤掉。在粗粒度检测之后, 共检测到 93,122 个可能的重打包应用对, 其中包含 14,702 个应用。虽然这些应用占了总应用的 14%左右, 但是可能的重打包应用对比总应用对减少了五个数量级, 极大的缩短了细粒度阶段所需要的检测时间。

表 1 实验数据来源

应用市场	应用的数量	比例
anzhi	14,047	13.3%
eoe	40,134	38.1%
gfan	13,672	13.0%
baidu	16,613	15.8%
myapp	20,833	19.8%
total	105,299	100%

在细粒度检测之后, 总共检测到 80,439 个重打包应用对, 其中包含 12,922 个不同的应用, 占了总共数据集的 12%左右。市场内部与市场间的重打包关系如图 6 所示。除了市场间的重打包应用, WuKong 还发现了在同一个应用市场内部有相当数量的重打包应用。例如, WuKong 发现 eoe 市场中有 4,368 个应用是属于重打包应用, 占了 WuKong 分析的该市场应用的 10%以上。这个结果也说明了在应用市场内进行重打包检测的重要性。

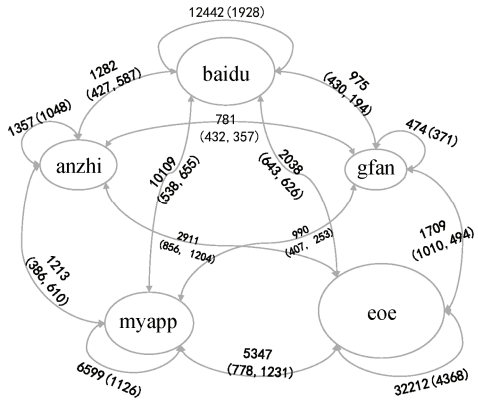


图 6 应用市场间及市场内的重打包应用分布

3.6 展望和挑战

应用重打包是移动平台上恶意软件传播的主要方式。在应用市场中进行重打包应用分析有助于找出可疑的应用, 根据添加和修改的代码可以来分析应用的恶意行为。同时, 应用重打包也促进了应用加固技术的研究, 如何对应用进行保护以及防止恶意的开发者进行破解也是主要的一个研究方向。

4 软件功能与安全行为间一致性研究

4.1 功能和行为

应用商店, 例如 Apple Store 和 Google Play Store, 为用户提供了丰富的应用软件, 在智能手机的普及上起到了关键的作用, 但是这些应用商店也给恶意软件的传播提供了捷径。目前, 主流的 iOS 和 Android 操作系统都有其独有的权限控制机制来遏制恶意软件。

苹果 iOS 系统采用实时权限请求机制, 即当软件在每次触发到需要权限的行为时都会告知用户。而 Android 系统在软件安装时就会向用户确认该程序正常使用的权限列表, 等用户同意安装后才该软件将会被授权所有被请求的权限。这些操作系统都

要求手机用户通过阅读软件描述、用户评价和软件评分, 以及通过判断该软件所请求的权限的合理性来区分正常软件和恶意软件。但是, 现阶段大部分的研究工作主要侧重于单方面分析应用软件的权限、代码或实时监测来检测软件行为是否恶意, 却忽略了一个重要的问题: 用户所期望的软件行为是什么?

WHYPER [17] 基于用户所期望的软件行为这样一种思路, 首次提出了一种基于自然语言处理的软件风险分析方法, 如图 7 所示。该方法在软件描述和软件请求的权限之间建立了一种映射关系, 并用这种映射关系量化软件功能和软件真实行为之间的差异性。实验结果表明, 自然语言处理技术能够有效的识别出这种差异性, 并能准确对软件提供风险评估。

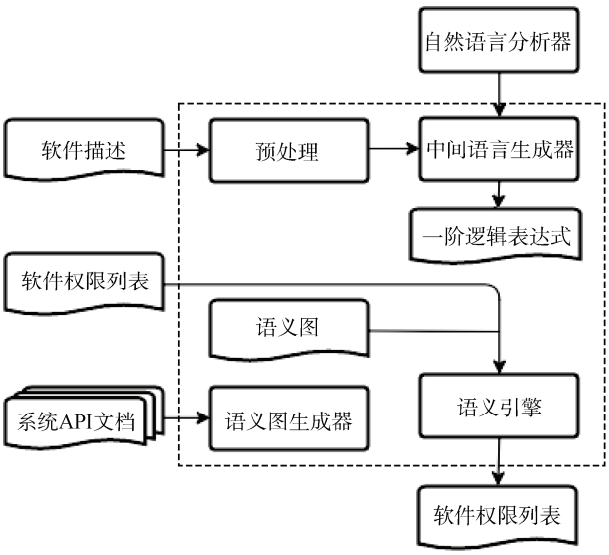


图 7　WHYPER 系统概述

此外, AutoCog [28] 提出了一种结合机器学习和自然语言处理的方法, 如图 8 所示。该方法利用大量的数据生成软件描述和软件请求权限之间的关系模型, 从而使分析结果更精准和全面。

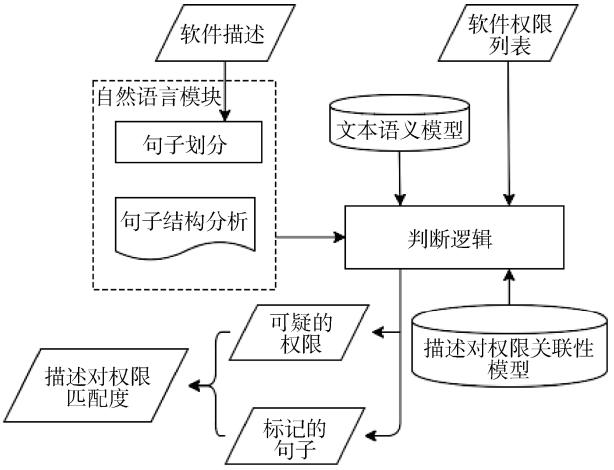


图 8　AutoCog 系统概述

4.2　问题描述

通常情况下, 应用软件不会在软件描述中阐述请求特定权限的原因。然而大部分恶意软件经常会请求一些和软件功能无关的权限 [27] [1]。例如, 一个恶意时钟软件会使用 GPS 权限发送用户的地理位置信息。但是对于一个 GPS 追踪软件, 把用户的地理位置信息通过网络端发送出去是正常行为, 而并非隐私泄漏。同理, 一个用来 Root 手机的软件, 通过利用权限提升漏洞获取 Root 权限的行为也应被看做是正常行为。因此, 用户所期望的软件功能和软件真实行为之间的差异性应当是检测恶意软件重要指标之一。我们所面临的问题是如何自动化识别出用户所期望的软件功能和软件真实行为之间的差异性, 并用此差异性来对软件进行风险评估。

具体来说, 我们需要寻求一种方法, 通过软件描述估测用户所期望的软件功能, 同时从软件请求的权限估测软件的真实行为。前者需要我们对软件描述进行语义上的分析, 后者需要我们对软件权限进行语义上的分析。

尽管关键词搜索法是最直接、最易实现的, 但是它在语义推理和处理语义混合效应上有诸多局限性。因此研究人员采用自然语言处理技术来取代关键词搜索法。

4.3　语义处理

4.3.1　句子划分

首先需要将软件描述中的句子区分出来, 同时尽可能地降低词法记号(lexical token)的数量。这样能够有效的帮助我们提高分析的准确度。具体来说需要处理如下几种情况:

1. 句号: 英文中句号('.')除了可以当做句子结尾符号, 也可以在小数点, 省略号和简化符号(例如"Mr.")中出现。

2. 句子划分: 除了句号外, 标号, 点句, 分隔符等其他特殊符号都能划分句子。

3. 命名: 代表名字的一连串单词没有必要进行再拆分。例如"Google Map"。

4. 简写: 有时候缩写和原词常常放在一起, 例如: "Instant Message (IM)"。对简写进行分析是没有必要的。

4.3.2　句子结构分析

WHYPER 和 AutoCog 都采用 Stanford Parser 对句子里的命名, 从属关系(Stanford-Typed Dependency)和词性进行标注, 之后还对单词进行了主次关系标注, 如图 9(对句子"Also you can share the yoga exercise to your friends via Email and SMS"的标注)

所示。

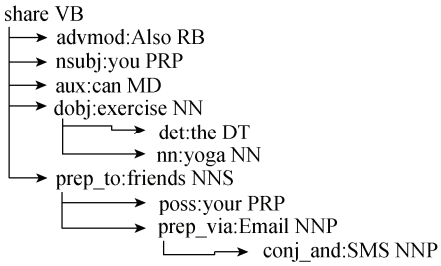


图 9　句子的主次关系图

有研究表明, 一阶逻辑(First Order Logic)能够有效的表达文本中的相互关系[2] [3] [16], 因此 WHYPER 进一步采用一阶逻辑树形图, 当作语义解析的中间语言, 如图 10 所示。在这种表达形式中, 叶节点都是实体节点, 叶节点外所有节点都是谓语(predicate)节点; 在每个谓语节点下的第一个子节点是主实体(节点 7), 第二个子节点是次实体(节点 10), 即这两个子节点为主次关系。

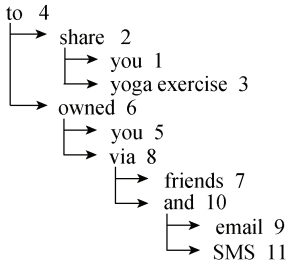


图 10　一阶逻辑图

WHYPER 还研究实现了一个结合 shallow parsing [4] 技术的级联有限状态机(cascading finite state machine),将标注了从属关系后的文本转换为一阶逻辑树形图的表达形式。在该表达形式之后将会与分析出的软件真实行为进行差异性比较。

4.4　差异匹配

方法一: 单一分析

为了获取应用的真实行为, 我们分析软件需要调用的系统 API 函数。这是因为, 区别于传统 PC 软件, 手机端的软件所能用到的系统资源有限。分析手机软件对系统资源的访问能够帮助我们估测出软件的真实行为。WHYPER 首先列出所有 API 所需的权限; 其次, 对每个权限, 找出其调用的资源与行为并整合成语义图。权限 CONTACT 能够允许软件从联系簿获取手机号、邮件、地址等信息, 并且可以对联系簿进行读取、显示、发送等操作, 如图 11 所示。

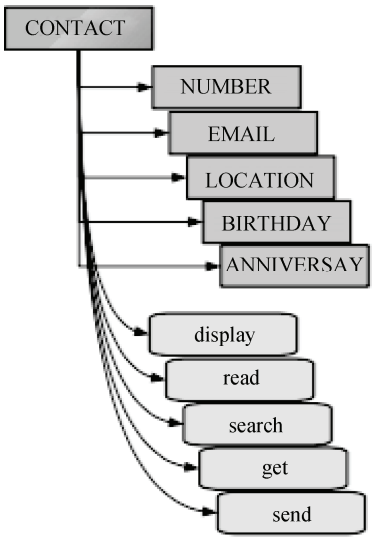


图 11　CONTACT 权限的语义图

最后, WHYPER 对通过一阶逻辑树形图里面的实体节点分和谓语节点与语义图里资源的访问类型和方法进行匹配, 并找出差异。

方法二: 建模与大数据解析

AutoCog 分别提出了“文本语义模型”和“描述对权限关联性模型(description-to-permission relatedness model, 简称 DPR, 如图 12 所示)”。

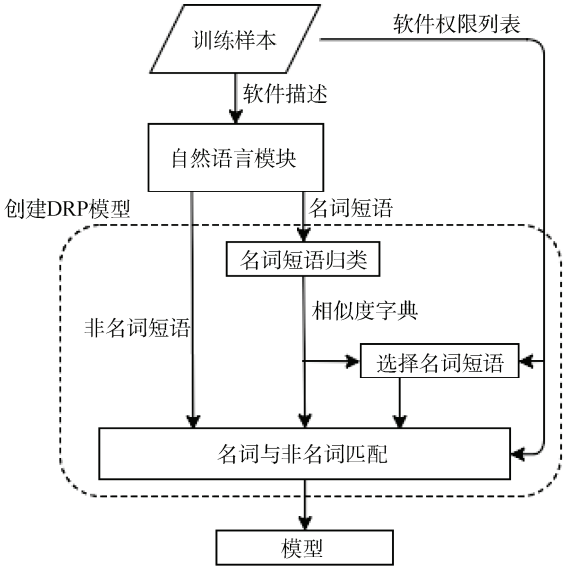


图 12　描述对权限关联性模型创建图

文本语义模型用来描述词汇之间语义上的关联性。AutoCog 从维基百科采集大量文本数据, 并用显示语义分析(Explicit Semantic Analysis)技术构建文本语义模型。

描述对权限关联性模型通过词汇来表征每个权限之间安全特性。AutoCog 用基于学习技术的算法实现了权限关联性模型。该算法对海量软件描述进行

分析, 测量出每一对主次关系名词与各个权限的关联性。最后, 通过添加 np-counterparts 和保留那些与权限有统计关联的词汇来优化模型。

最后, 给定一个应用软件, 用语义模型测量该软件的描述与其权限的关联性。

4.5 展望和挑战

近几年, 当基于自然语言处理技术的软件风险评估框架与其它技术结合(例如机器学习)后, 会越来越全面和高效地发现移动应用中潜在的风险。今后的工作可以继续尝试与其它技术融合, 如在程序分析方面, 可进一步分析程序的权限列表, 静态代码或者动态程序分析, 都能更好的描述程序的真实行为。

5 基于上下文的恶意行为解析

5.1 研究背景

随着移动应用的普及, 恶意移动应用从数量上和类型上都有显著地增长。如此快速的增长使得传统的基于签名的监测方法无法跟上恶意应用的进化速度。针对这种情况, 部分研究开始利用机器学习的方法来对恶意应用以及恶意行为进行解析。这些研究利用机器学习的方法来总结恶意行为的模式来对恶意行为进行自动检测。恶意行为的模式

表 2 AppContext 分类所用特征集

特征集类型	特征集元素		
行为信息特征集	请求的权限	安全方法调用	
触发事件特征集	硬件事件	系统事件	界面事件
上下文因素特征集	环境因素列表		

可以是: 方法调用的模式, 申请权限的模式, 信息流的模式等。

与此同时, 恶意应用也在不断进化以避开防病毒程序的检测。恶意应用软件的作者会模仿良性软件的行为模式, 使得安全分析无法有效检验恶意应用的行为。针对这种情况, 一些研究提出了利用行为发生上下文的方法以进一步分辨恶意行为和良性行为。这些研究表明, 一个行为发生的上下文可以有效反映应用执行这个行为的目的。而执行行为的目的可以作为甄别行为的恶意性的主要依据。

在这些研究中, AppContext [20] 是一个利用行为发生上下文以检测恶意行为的典型工作。AppContext 通过程序分析来提取安全行为的上下文, 并通过这些提取的上下文来辅助机器学习算法检测恶意行为。 AppContext 主要基于对恶意应用的一个重要观察, 即恶意应用的设计要满足两点需求: (1)尽量避开防病毒程序的检测以延长恶意应用在市场中的存活时间, (2)尽量可能多的触发恶意行为以实现恶意应用的效率最大化。

5.2 系统概述

AppContext 共分五个部分:首先通过分析程序的字节码来创建程序的调用图(Call Graph),并根据程序中所调用的安全方法来定位程序中的安全行为。然后, 通过调用图来提取触发事件(Activation Event), 并将调用图转化成适合安全分析的扩展调用图(Extended Call Graph)。接着, AppContext 通过扩展调用图来创建约减跨过程控制流图(Reduced Inter-procedure Control Flow Graph), 并通过跨过程控制流图来找到控制安全行为的条件语句。通过条件语句定位到上下文因素(Context Factor)。最后, 利用机器学习算法来通过触发事件和上下文因素将安全行为分类为良性行为以及恶意行为。

5.3 安全行为解析

如表格 1 所示, AppContext 结合了三种特征集来定义及识别程序中的安全行为。其中, 行为信息特征集包括程序中请求的权限和程序中所调用的安全方法。这两类信息通过分析程序的配置文件(AndroidManifest.xml)以及字节码获得。

AppContext 所用的第二种特征集是触发事件特征集。AppContext 之所以采用此类特征集是因为恶意应用频繁地利用手机系统中的各种事件来触发恶意行为。其中, 用于触发恶意行为的事件往往与用于触发良性行为的事件不同。因为恶意软件为了避开防病毒程序检测以及实现恶意行为效率的最大化, 往往使用触发频率频繁且用户不经常注意到的事件。比如说, 手机接收信号强度的变化。基于这个发现, 我们将触发事件特征集分为了三种不同的事件: 硬件事件、系统事件以及界面事件。其中, 硬件事件是由通过与手机硬件界面的交互触发, 如 HOME 或者 BACK 按键。系统事件通过系统状态的变化触发, 如之前所讲信号强度的变化, 或者系统收到了新的短消息。而界面事件通过用户与程序本身的界面而触发, 例如点击程序中的按钮。根据我们的实验显示, 系统事件往往会被作为触发恶意行为的事件在恶意

应用中使用。

AppContext 所用的第三种特征集是上下文因素特征集。上下文因素是指那些会影响安全行为是否会触发的环境因素, 比如说当前的时间日期。 AppContext 之所以将这些因素作为特征集加以考虑, 是因为恶意软件为了避开用户或防病毒程序的检测, 它们会控制恶意行为在不太会被检测的环境(如特定的时间地点)下触发。比如说恶意应用 DroidDream 只会允许其恶意行为在晚上 11 点至第二天早上 5 点间被触发。由于用户在这个时间段一般在睡觉, 这样恶意程序能有效避开用户的注意力以延长自己的生存时间。

当 AppContext 通过静态分析提取出如上所述的特征集后, 通过人工将所有提取的安全行为标记成良性行为与恶意行为, 并利用十倍交叉验证的方法(ten-fold cross-validation)来检测这些特征集是否能有效检测恶意应用。具体的实验效果可参照 AppContext 论文[20]。

5.4 总结与展望

为了应对恶意应用智能性的增加, 手机应用需要对这种智能性进行检测。利用安全行为的上下文是检测这种智能性的方法之一。而本章中介绍的 AppContext 仅为其中有代表性的工作之一。在未来, 随着恶意应用的隐蔽性增加、触发频率更高, 利用上下文解析的方法会变得更加关键, 以辅助研究者提出相应的更智能的检测方法。

6 移动应用管理和使用行为分析

6.1 研究背景

随着移动应用的普及, 恶意移动应用从数量上和类型上都有显著增长。如此快速的增长使得传统的基于签名的监测方法无法跟上恶意应用的进化速度。针对这种情况, 部分研究开始利用统计和机器学习的方法, 基于对终端用户使用和管理应用的行为数据进行分析, 来推断恶意应用以及恶意行为并进行解析。

通过观察、分析真实用户的使用数据, 可以从中发现一些潜在的问题, 继而可以此为根据有针对性地分析问题产生的原因, 进一步寻求解决问题的方法。基于这一思路, 本节从一个基于 Android 用户数据的数据分析工作出发, 从三个角度探讨如何从数据中发现可能存在恶意行为的应用。

6.2 非主动安装

非主动安装一直以来都是困扰智能手机用户的一大问题。非主动安装即指一个应用在未经用户允许、不是通过用户的主动选择而被安装在了用户的手机上, 或者是通过一些带有迷惑性的界面、语言, 导致用户进行了安装应用的“主动”操作, 但这一“主动”行为并不是出于用户本身的意图。在传统的 PC+Windows 平台, 这一问题较为普遍。近年来随着智能手机的发展, 非主动安装也随之进入了这一领域。

非主动安装对于用户的信息安全来说是具有潜在威胁的。究其根本原因, 在于一个原本应当经过用户同意的行为绕开了用户的确认, 导致用户对于自己的手机上存在这样的应用并不知情。一个直接而明显的影响是这些应用占用了手机的存储空间。如果这些应用同时具有自启动特性(一个不经用户同意自行安装的应用, 亦十分可能不经用户同意而主动运行), 则也会对手机的整体性能造成拖慢。更为严重的问题是, 这些应用可能包含恶意行为, 其自行安装、运行的目的即为窃取用户数据。对于这种行为, 用户须提高警惕加以防范。

非主动安装的最典型形式是绑定安装, 即用户在试图安装一个应用时, 被隐式的、不明显的、或者强制性的安装了其他若干个“绑定”在一起的其他应用。这一情况可以一定程度上从用户使用数据中得以体现。通过分析用户使用数据, 我们发现存在着一些应用对, 它们被共同安装在同一个用户的设备上的比例相当之高。并且, 这些应用对中有相当大的比重来源于同一个开发者。以某公司开发的安全软件 A1 和手机管理软件 A2 为例, 在所有安装过这两个应用的用户当中, 同时安装了它们的用户所占的比例相当之高, 超过了 70%。通过本地试安装 A1, 我们发现其确实绑定安装的问题, 在安装 A1 的过程中, A2 被同时安装在了手机上, 即发生了绑定安装行为。

根据这一发现, 我们可以得到如下启发。当一些应用, 尤其是来自于同一开发者的应用被共同安装的比例较高时, 我们需要提高对其的警惕性, 警惕该行为是否是由于非主动安装导致。

在传统 PC 情境下, 已有相关研究工作利用应用的安装行为数据进行分析, 并试图从中发现恶意行为。Kwon 等人 [29]提出过一种利用分析下载图的方式对下载应用是否具有恶意进行分析。比如,应用 A 下载应用 B 和 D, 且未知 B、D 是否具有恶意。但是 B 后序进行了下载 C、进而 C 下载 F 的行为, 而 C 与 F 被判断为恶意应用。由此构造下载图并通过对该图的分析, 可以很大程度上怀疑 B 具有恶意性。在移动平台环境中, 在拥有行为数据的前提下, 我们也可以考虑利用类似技术进行应用的恶意性分析。这对智能手机上的恶意应用发现相关工作提供了一

个新思路。本节前段提到的非主动安装行为可以考虑结合下载图进行分析。

6.3　用户卸载行为

当用户在不知情的情况下被安装了应用, 或者是发现了某一应用为恶意应用之后, 很有可能会主动卸载该应用。因此, 通过分析用户的应用卸载行为, 也可以从一定程度上帮助我们发现恶意应用。这里, 单纯分析一个应用被用户卸载的次数是不够准确的, 因为不同的应用有着不同的用户基数, 一个用户量很大的即时通讯软件(比如微信)可能下载量很高, 而卸载量也很高。如果直接比较该即时通讯软件和另一个小众社交软件的卸载次数, 所得到的结果并没有什么意义。Li 等人通过对国内某著名 Android 应用市场上 1700 万用户连续 5 个月下载、卸载 App 的行为进行分析后, 提出了一个直观的指标来度量一个应用被用户接受的程度[8]: 安装/卸载比(I/U Ratio), 用来衡量一个应用被卸载的比例, 定义如下: 令一个应用 app 下载的集合为 I(app), 卸载的集合为 U(app), 则其安装卸载比为 $R = \frac{I(app)}{U(app)}$。直观而言, 一个应用的安装卸载比越高, 表示用户更少的倾向于卸载这个应用, 反之则说明用户更多的倾向于卸载这个应用。

图 13 通过线性回归, 反映了 105 万个 Android App 下载次数与其安装卸载比之间的关联关系。可以看出, 应用的下载卸载比与其总下载次数之间的关联并不紧密。从这一角度而言, 应用的下载次数并不能完全反映出用户对其的喜好。很多下载量很高的应用卸载量也很高, 因此卸载所占比例也非常高。我们进一步分析了一个应用的安装卸载比与其在用户手机上的平均存在时间之间的关系。所谓存在时间, 即一个应用从被安装到用户的手机上直至用户卸载它所经过的时间。这一关系展现在图 14 中。可以看

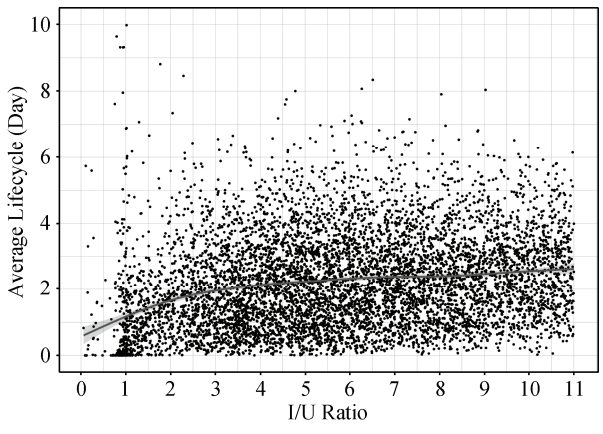


图 13　安装卸载比与下载数的关系图

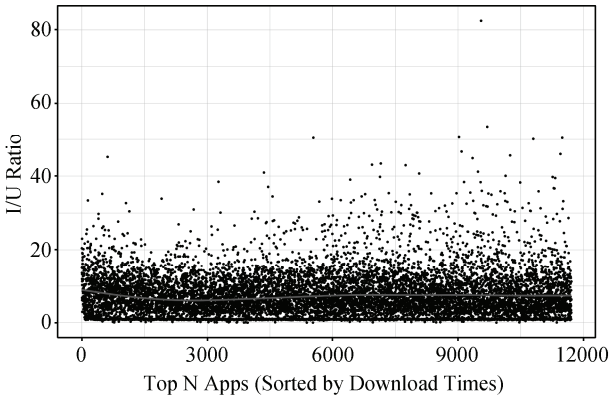


图 14　安装卸载比与存在时间的关系

到, 随着安装卸载比的提高, 应用的平均存在时间有增长的趋势。

因此, 可以认为应用的安装卸载比是一个有意义的指标。对于那些安装量较高、但是安装卸载比却不高的应用, 应当提出额外的关注。如上节中提到的 A1, 其下载量较高, 但是安装卸载比缺较低, 可以从一定程度上说明其存在问题。

6.4　异常流量

隐私信息的泄露最终需要通过网络来进行传输, 因此关注应用的网络连接情况, 对于帮助我们发现恶意软件也有一定借鉴意义。Li 等人 [7] 分析了百万级用户在 26 万个 Android 应用上网络使用数据(连接时长、流量消耗), 通过对相似功能的应用进行聚类和异常检测之后, 可以观察到有一些应用产生的网络行为数据和其自身功能存在较为明显的差异。比如, 有一款手电筒类型的应用, 其功能仅为简单的通过手机闪光灯进行照明(无论是该应用自己的描述亦或是用户对其的与其都仅此而已)。但是, 这个手电筒应用却保有很长的后台联网时间, 同时在后台产生一定的网络数据传输, 平均每天产生 7MB 的蜂窝数据流量和约 5MB 的 Wi-Fi 流量。对于联网时间, 也许可归结为操作系统在后台进程管理方面存在不足, 但是对于后台流量消耗, 则可能是应用本身的设计问题, 比如申请了不必要的联网权限、使用了可能存在问题的第三方库、或者是应用本身的逻辑中存在 Bug。这一现象很可能是由于恶意行为导致, 至少会导致用户的隐私信息直接面临威胁。类似应用应当提醒我们的注意。

6.5　总结与展望

在传统的 PC 情境下, 已有大量工作通过数据比较、数据分析、行为分析等方式对恶意应用进行鉴别。随着智能手机用户的使用数据的增多以及大数据技术的兴起, 相关技术可以得到新的扩展, 新的技术也有待继续开发, 以期从用户的使用数据中发

现可能的安全问题。

随着智能手机用户的使用数据的增多以及大数据技术的兴起, 从用户的使用数据中可以发现一些可能的安全问题。本章中介绍的方法仅为冰山一隅。在未来, 当数据维度增多、规模增大、更高效准确的算法被提出之后, 数据可以帮助我们发现更多的安全隐患。

7 小结

本文分别从用户界面解析、重打包应用检测、应用功能与安全行为一致性检测、基于上下文的恶意行为检测, 以及终端应用管理和使用这五个部分介绍了移动应用安全解析学目前的成果。同时, 基于以上的研究成果, 对未来的研究方向进行了展望, 并分析了这些研究方向面临的挑战。

参考文献

[1] A. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. "A survey of mobile malware in the wild," in *Proc. ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2011, pp. 3-14.

[2] A. Sinha, A. M. Paradkar, P. Kumanan, and B. Boguraev, "A linguistic analysis engine for natural language use case description and its application to dependability analysis in industrial use cases," in *Proc. IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, 2009, pp. 327 – 336.

[3] A. Sinha, S. M. SuttonJr., and A. Paradkar, "Text2Test: Automated inspection of natural language use cases," in *Proc. International Conference on Software Testing, Verification and Validation (ICST)*, 2010, pp. 155-164.

[4] B. K. Boguraev, "Towards finite-state analysis of lexical cohesion." in *Proc. International Conference on Finite-State Methods and Natural Language Processing (FSMNLP)*, 2000.

[5] B. Liu, B. Liu, H. Jin, and R. View, "Efficient privilege de-escalation for ad libraries in mobile apps," in *Proc. International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2015, pp. 89-103.

[6] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu, "ViewDroid: towards obfuscation-resilient mobile application repackaging detection," in *Proc. ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2014, pp. 25-36.

[7] H.Li, X. Lu, X. Liu. "Characterizing Smartphone Usage Patterns from Millions of Android Users," in *Proc. ACM Conference on Internet Measurement Conference (IMC),* 2015, pp. 459-472.

[8] H.Li, W. Ai, X. Liu. "Voting with Their Feet: Inferring User Preferences from App Management Activities" in *Proc. International Conference on World Wide Web (WWW) Companion,* 2016, pp. 1351-1361.

[9] H. Wang, Y. Guo, Z. Ma and X. Chen, "WuKong: A Scalable and Accurate Two-Phase Approach to Android App Clone Detection," in *Proc. International Symposium on Software Testing and Analysis (ISSTA),* 2015, pp. 71-82.

[10] J. Crussell, R. Stevens, H. Chen, "MAdFraud: Investigating ad fraud in Android applications," in *Proc. annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2014, pp. 123-134.

[11] J. Crussell, C. Gibler, and H. Chen, "Attack of the clones: detecting cloned applications on Android markets," in *Proc. European Symposium on Research in Computer Security (ESORICS)*, 2012, pp. 37-54.

[12] J. Crussell, C. Gibler, and H. Chen, "Scalable semantics-based detection of similar Android applications," in *Proc. European Symposium on Research in Computer Security (ESORICS)*, 2013, pp. 182-199.

[13] J. Huang, Z. Li, X. Xiao, Z. Wu, K. Lu, X. Zhang, and G. Jiang, "SUPOR: Precise and scalable sensitive user input detection for Android apps," in *Proc. USENIX Conference on Security Symposium (USENIX Security),* 2015, pp. 977-992.

[14] K. Chen, P. Liu, and Y. Zhang, "Achieving accuracy and scalability simultaneously in detecting application clones on Android markets," in *Proc. International Conference on Software Engineering (ICSE)*, 2014, pp. 175-186.

[15] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of Google Play," in *Proc. ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2014, pp. 221-233.

[16] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar, "Inferring method specifications from natural language API descriptions," in *Proc. International Conference on Software Engineering (ICSE)*, 2012, pp. 815-825.

[17] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: Towards automating risk assessment of mobile applications," in *Proc. USENIX Conference on Security Symposium (USENIX Security),* 2013, pp. 527-542.

[18] S. Hanna, L. Huang, E. Wu, S. Li, C. Chen, and D. Song, "Juxtapp: A scalable system for detecting code reuse among Android applications," in *Proc. International Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, 2012, pp. 62-81.

[19] W. Yang, M. R. Prasad, and T. Xie, "A grey-box approach for automated GUI-model generation of mobile applications," In *Proc. International Conference on Fundamental Approaches to Software Engineering (FASE)*, 2013, pp. 250-265.

[20] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, & W. Enck, “AppContext: Differentiating malicious and benign mobile app behaviors using context,” In *Proc. International Conference on Software Engineering (ICSE),* 2015, pp. 303-313.

[21] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, “Detecting repackaged smartphone applications in third-party Android marketplaces,” in *Proc. ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2012, pp. 317-326.

[22] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou, “Fast, scalable detection of piggybacked mobile applications,” in *Proc. ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2013, pp. 185-196.

[23] Y. Yuan and Y. Guo, “CMCD: count matrix based code clone detection,” in *Proc. Asia Pacific Software Engineering Conference (APSEC)*, 2011, pp. 250-257.

[24] Y. Yuan and Y. Guo, “Boreas: an accurate and scalable token-based approach to code clone detection,” in *Proc. IEEE/ACM International Conference on Automated Software Engineering (ASE),* 2012, pp. 286-289.

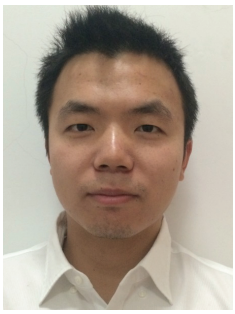
[25] Y. Nan, M. Yang, Z. Yang, S. Zhou, G. Gu, and X. Wang, “UIPicker: User-input privacy identification in mobile applications,” in *Proc. USENIX Conference on Security Symposium (USENIX Security),* 2015, pp. 993-1008.

[26] Y. Zhauniarovich, O. Gadyatskaya, B. Crispo, F. La Spina, and E. Moser, “FSquaDRA: fast detection of repackaged applications,” *Data and Applications Security and Privacy XXVIII, volume 8566 of Lecture Notes in Computer Science,* 2014, pp. 130-145.

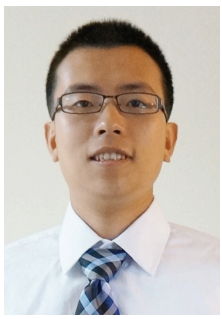
[27] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," in *Proc. Network and Distributed System Security Symposium (NDSS)*, 2012.

[28] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen. "AutoCog: Measuring the description-to-permission fidelity in Android applications," *In Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS),* 2014, pp. 1354-1365.

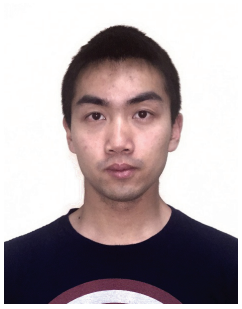
[29] B.J. Kwon, J. Mondal, J. Jang, L. Bilge, and T. Dumitras. "The dropper effect: Insights into malware distribution with downloader graph analytics." In *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2015, pp. 1118-1129.



杨威　于 2013 年在美国北卡罗莱纳州立大学计算机科学专业获得硕士学位。现在伊利诺伊大学香槟分校计算机专业攻读计算机科学学位博士。研究领域为软件工程与安全。研究兴趣包括: 软件安全, 程序分析,自动化测。Email: weiyang3@illinois.edu



肖旭生　于2014年在美国北卡罗莱纳州立大学计算机科学专业获得博士学位。现任 NEC 美国研究所研究员。研究领域为企业安全智能系统和移动安全管理。研究兴趣包括: 软件测试, 程序分析, 移动安全, 系统和企业安全。Email: xsxiao@nec-labs.com



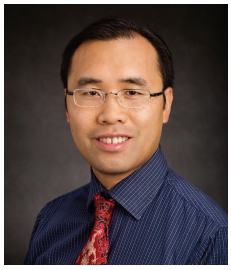
李邓锋　于 2015 年在美国宾夕法尼亚州立大学获得电子工程和计算机科学双学位。现在伊利诺伊大学香槟分校计算机专业攻读计算机科学硕士学位。研究领域为软件工程与安全。Email: dli46@illinois.edu

李豁然　现在是北京大学信息科学技术学院的博士研究生。他的研究兴趣包括移动计算、软件工程、数据分析及人机交互。Email: lihuoran@pku.edu.cn

刘譞哲　北京大学信息科学技术学院副教。研究兴趣包括移动计算, 网页系统, 和程序分析。Email: liuxuanzhe@pku.edu.cn

王浩宇　于 2011 年进入北京大学信息科学技术学院攻读博士学位, 现在是博士五年级。研究领域为移动计算, 系统软件。研究兴趣包括: 移动系统中的安全与隐私, 移动应用分析技术。Email: howiepku@pku.edu.cn

郭耀 北京大学信息科学技术学院副教授, 美国马萨诸塞大学计算机工程博士, 中国计算机学会高级会员, ACM 和 IEEE 会员。主要研究方向为操作系统、低功耗设计、软件工程等。Email: yaoguo@pku.edu.cn



谢涛 美国伊利诺伊大学香槟分校(UIUC)计算机科学系副教授和 Willett Faculty Scholar, 美国西雅图华盛顿大学博士, ACM 杰出科学家和 IEEE 高级会员。主要研究方向为软件工程、软件测试和分析、软件解析学等。Email: taoxie@illinois.edu