DOI:10.14188/j.1671-8836.2018.05.005

DroidFAR:一种基于程序语义的 Android 重打包应用抗混淆检测方法

汪　润[1,2],唐奔宵[1,2],王丽娜[1,2†]

(1. 武汉大学 空天信息安全与可信计算教育部重点实验室,湖北 武汉 430072;
2. 武汉大学 国家网络安全学院,湖北 武汉 430072)

摘　要:为了使 Android 平台重打包应用检测的方法在面向大规模移动应用中既能实现快速、准确地检测重打包应用又能对抗代码混淆攻击,本文提出了一种基于程序语义的重打包应用抗混淆检测方法.该方法首先进行粗粒度的检测,即先将应用的程序依赖图抽象成程序语义特征,通过计算程序语义特征之间的相似性,实现快速的可疑重打包应用检测;然后使用程序依赖图作为应用的特征,完成可疑重打包应用细粒度的准确检测.基于文中的方法设计并实现了原型系统 DroidFAR(Fast, Accurate and Robust).实验结果表明,本文方法检测的准确率达到 95.1%,误报率低于 1.2%,且能够有效地抵御代码混淆攻击.

关　键　词:重打包应用;代码克隆;程序依赖图;安全与隐私

中图分类号:TP 309.1　　**文献标识码**:A　　**文章编号**:1671-8836(2018)05-0407-08

DroidFAR: An Anti-Obfuscation Method for Detecting Android Repackaged Application Based on Program Semantics

WANG Run[1,2], TANG Benxiao[1,2], WANG Lina[1,2†]

(1. Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, Wuhan University, Wuhan 430072, Hubei, China;
2. School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, Hubei, China)

Abstract: A semantic-based application repackaging detection approach is proposed for addressing the requirement of fast, accurate and obfuscated code evade attacks in large scale Android repackaged application detection. Our approach first performs a coarse-grained detection, which abstracts the program dependency graph of application as a kind of program semantic feature and calculates the similarity of semantic features in order to achieve a fast potential repackaged applications detection. Then the program dependency graph is utilized as application feature to achieve an accurate and fine-grained detection of potential repackaged applications. A prototype system, called DroidFAR, is designed and implemented based on our proposed approach. Experimental results show that our approach can achieve an accuracy of 95.1% and obtain false positive rate lower than 1.2% in application repackaging detection and it can effectively evade the code obfuscation attack.

Key words: repackaged application; code clone; program dependency graph; security and privacy

0　引　言

近些年来,智能设备如智能手机、智能穿戴设备等变得越来越普遍.与此同时,提供给用户下载安装使用的移动应用数量也在飞速增长.据统计,Android 平台的官方商店 Google Play 和 iOS 平台的官方商店 Apple Store 的应用数量已经超过了 300 万之多,国内一些第三方应用商店如安智、豌豆荚、

收稿日期:2017-10-12　†通信联系人　E-mail:lnwang@whu.edu.cn
基金项目:国家自然科学基金(U1536204);国家高技术发展计划(863)(2015AA016004)资助项目
作者简介:汪　润,男,博士生,主要研究方向为移动安全与隐私. E-mail: wangrun@whu.edu.cn

小米应用商店等都有大量的应用供用户下载、安装和使用.但由于一些不法分子制造大量的恶意应用,使得第三方应用给用户带来了令人惊叹的功能体验的同时,也存在威胁用户安全和隐私的隐患[1~5].

iOS 和 Android 是目前两大主流移动平台,iOS 平台存在的安全问题多是由于应用漏洞导致的,而 Android 平台的安全问题主要是由于恶意应用的传播导致的.相关研究显示,在移动平台上传播的恶意应用,Android 平台占将近 98%.在 Android 平台的恶意应用中,有超过 86%的恶意应用是重打包应用[6].重打包应用是指恶意开发者通过修改已经发布在应用市场的合法应用,诱导用户下载达到谋取非法收益、传播恶意应用的目的[7].

针对 Android 平台的重打包应用检测,学术界展开了一系列的相关研究工作,主要分为基于代码克隆的重打包检测[8~16]和基于 Android 应用特征的重打包检测[17~21]两类方法.其中,第一类方法是先抽取应用的代码特征如程序依赖图、控制流图表示应用的特征,然后通过度量应用特征之间的相似性来判断是否为重打包应用.Juxtapp[8]利用 k-gram 算法对应用反编译得到的操作码进行编码处理,然后使用 Jaccard 距离计算其相似性.但如果攻击者在代码中插入多余的代码,会严重地干扰检测的准确率.DNADroid[9]生成应用的程序依赖图,利用图匹配算法识别可疑的重打包应用,该方法易遭受代码混淆攻击的影响且由于其检测速度慢导致无法应用于大规模的市场检测.Chen 等[13,14]利用应用程序流程图的质心表示应用的特征,但是该方法在面临严重的代码混淆时,检测效果并不理想.

上述基于代码克隆的检测方法容易受到代码混淆的干扰,当应用程序代码严重混淆时,检测准确率低.为了解决第一类方法难以应对代码混淆的问题,有学者提出第二类基于 Android 应用特征的重打包应用检测方法,这类方法使用应用的资源文件表示应用的特征,通过计算相似性来判断他们是否为重打包应用.其中 Android 应用的资源文件包括图片、用户界面和视频文件等.ViewDroid[17]使用有向图表示应用的特征,其中有向图的顶点是应用的视图,当视图之间存在事件可以进行视图切换时,顶点之间建立有向边连接.DroidEagle[18]也使用应用的 UI (user interface)表示特征,为了解决大规模的重打包应用的快速检测问题,DroidEagle 由部署在移动端的 HostEagle 和部署在服务端的 RepoEagle 构成.其中,RepoEagle 将 UI 表示成树形结构,HostEagle 将树形结构的布局进行哈希处理.这类方法的不足是由于部分应用的模板相似,导致该类方法误报率较高.

我们通过大量的研究发现,目前针对 Android 重打包应用的检测方法存在以下不足:可以实现面向大规模应用市场的快速重打包应用检测,但是检测的准确率低;或可以实现高准确率的检测,但是检测的速度慢,不能用于大规模应用市场中重打包应用的检测;一些基于代码克隆的高效、准确检测方法,易遭受代码混淆的影响,导致检测的误报和漏报率较高.

针对现有重打包应用检测方法中存在的不足,本文提出了一种基于程序语义的应用表示方法,实现面向大规模移动应用市场重打包应用的快速、准确检测,并且可以有效地对抗常见的代码混淆给应用检测带来的影响.

1 相关背景知识

在本节中,主要介绍本文涉及的相关背景知识.其中,1.1 节介绍程序依赖图,1.2 节介绍重打包应用的 4 种常见类型.

1.1 程序依赖图

程序依赖图(program dependency graph,PDG)用于表示应用程序中各个语句之间的依赖关系,包括数据依赖和控制依赖[22].数据依赖是指如果语句 S_1 中存在变量的值是由语句 S_2 决定的,则语句 S_1 和语句 S_2 之间存在数据依赖.如果语句 S_1 控制语句 S_2 执行与否,则语句 S_1 和语句 S_2 之间存在控制依赖.

用基于数据依赖的程序依赖图表示应用特征可以有效地防御代码插入、修改等攻击行为,相比于程序控制流图更适合表示应用相似性检测的特征.基于语义的程序依赖图(semantic-based program dependency graph,SPDG)是一种简化版的 PDG,与 PDG 不同的是,SPDG 的顶点包含有平台 API(application program interface)语句,顶点之间通过数据依赖关系建立连接.由于 SPDG 的顶点包含有 API 的语句,因此可以有效地抵御代码混淆攻击.

1.2 重打包应用类型

根据恶意开发者修改构造合法应用的精细程度以及程序代码拷贝的比例等,Android 平台重打包应用主要包括以下几种类型:

① 类型 1:完全拷贝,不修改程序代码.在合法应用的基础上,保留所有的程序代码内容不变,仅仅添加代码注释等内容;

② 类型 2:完全拷贝,修改程序代码变量名等.

恶意开发者仅仅修改部分变量名或函数名等;

③ 类型 3:部分或完全拷贝,修改程序代码行数.恶意开发者在程序代码中修改、删除或增加部分程序代码等;

④ 类型 4:程序语义功能拷贝.开发者在理解应用功能的基础上,重新编写代码实现相似或相同的功能.

2　总体设计

2.1　基本思想

通过大量的研究发现,恶意开发者在修改生成重打包应用时,通常会增加、修改包含有平台 API 的语句.但是包含有平台 API 的语句不易被代码混淆.换句话说,含有平台 API 的语句是应用程序的关键语句,包含有丰富的程序语义信息,可以用于表示程序的行为特征.基于以上发现,首先反编译应用的安装文件(apk, Android package files)生成基于语义的程序依赖图,顶点是含有 Android 平台 API 的语句,边表示顶点之间存在数据依赖.然后将生成的 SPDG 分割成子图,每个子图中仅包含两个顶点,顶点之间存在数据依赖关系.将子图构成的集合作为应用的语义特征表示,通过语义特征之间的相似性度量来快速地检测出可疑的重打包应用.最后,利用程序依赖图表示应用的特征,完成可疑重打包应用的细粒度检测.本文提出的基于 SPDG 的应用特征表示,其顶点包含有平台 API 的语句,它们不易被修改、混淆处理,可以有效地抵御常见的代码混淆对重打包应用检测带来的影响.而且,本文中 SPDG 的顶点之间通过数据依赖建立联系,可以防御程序代码的插入、修改等操作.

2.2　框　架

基于文中提出的程序语义特征表示方法设计的 Android 平台重打包应用检测系统 DroidFAR 的框架如图 1 所示.DroidFAR 主要由三个部分组成,分别是 Android 应用收集、粗粒度检测和细粒度检测.

本文的方法能够有效地检测 1.2 节中提到的类型 1、2、3 重打包应用,并不适用类型 4 基于语义功能的重打包应用检测.本文中分析和检测修改 DEX 代码生成的重打包应用,也就是 Java 语言编写的代码部分,而修改本地代码(native code)生成的重打包应用并不在本文的分析范围内.研究表明仅有 4.2%的应用包含有本地代码[23],而且鲜有研究报告显示有修改本地代码生成重打包应用的案例.

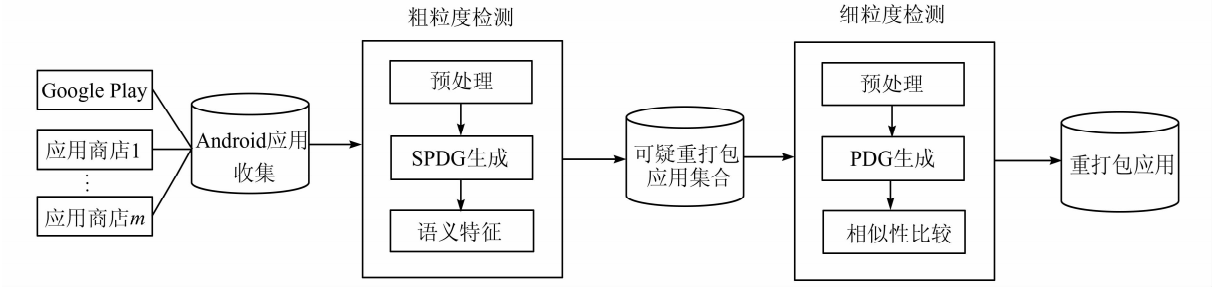

图 1　Android 重打包应用检测系统 DroidFAR 框架图

Fig. 1　The framework of application repackaging detection system, DroidFAR, in Android

3　DroidFAR 方法

本节介绍 DroidFAR 方法细节.其中 3.1 节介绍基本定义,3.2 节介绍 apk 文件预处理,3.3 节介绍基于 SPDG 的粗粒度重打包应用检测,3.4 节介绍基于 PDG 的细粒度重打包应用检测.

3.1　基本定义

本文中的重打包应用检测方法主要由粗粒度检测和细粒度检测两部分构成,其中,基于 SPDG 的应用特征表示用于实现快速的粗粒度检测,基于 PDG 的应用特征表示用于细粒度的准确重打包检测.在详细描述本文的检测方法之前,这里先给出一些基本的符号解释和定义.

定义 1　Android 应用语义语句 S. Android 应用语义语句表示为 $S=\{I_i\}, i=1,2,\cdots,M$,其中 I 表示语句中调用的函数名,M 表示应用中语句的个数,$I\in R$,R 表示 Android 平台 API 的集合.

定义 2　Android 应用 SPDG. Android 应用 SPDG 表示为 $G_s=\langle S, E\rangle$,其中 S 表示函数中含有系统 API 的语句集,见定义 1 所示,E 表示边集,当语句间有数据依赖时,语句间存在有向边连接.

定义 3　Android 应用 PDG. Android 应用的程序依赖图表示为 $G=\langle S_p, E_p\rangle$,其中 S_p 表示函数的语句集,E_p 表示边集,当语句之间有数据依赖时,语句之间存在有向边连接.

3.2 预处理

在 Android 应用市场,Android 应用以 apk 文件的形式发布,供用户下载、安装和使用.从各个应用市场包括 Android 官方应用商店 Google Play 中收集移动应用,作为应用检测分析的对象,其中包括合法应用和部分重打包应用.apk 文件包括 Dalvik 字节码、资源文件和 XML 格式的 Manifest 文件等.安装包中的 class.dex 文件包含提供给 Dalvik 虚拟机执行的 Dalvik 字节码,应用的签名也包含在安装包中作为应用的惟一标识.

在本文的 SPDG 和 PDG 生成中,均需要通过反编译解析安装包中 DEX 文件,生成相应的应用特征.应用的签名用于提取开发者的标识信息,不同开发者的签名均不同,用于识别两个应用是否为相同的开发者.

3.3 基于 SPDG 的粗粒度重打包应用检测

基于 SPDG 的快速相似性检测流程如图 2 所示.首先反编译获得 Android 安装包的 DEX 文件,针对应用程序代码中的每个方法生成相应的 SPDG,然后将其特征向量化,最后根据特征向量的相似性距离判断是否为可疑的重打包应用.在相似性检测中主要包括以下三个组成部分:SPDG 生成、特征向量化表示以及相似性检测等,下面详细介绍这三个组成部分的内容.

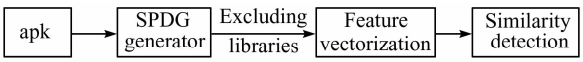

图 2 基于 SPDG 的应用快速相似性检测流程图

Fig. 2 The flow graph of application similarity detection based on SPDG

1) SPDG 生成

SPDG 的顶点为包含有 Android 平台 API 的语句.这些语句可以用于表示程序的行为,而且难以被代码混淆处理.当顶点之间存在数据依赖时,建立有向边连接.因此,SPDG 不仅可以对抗诸如在程序代码中插入语句、修改语句等行为,而且不易受控制流修改等攻击行为的影响[9,16].由于顶点是含有 Android 平台 API 的语句,不易被代码混淆处理,因此可以有效地抵御常见的代码混淆攻击.

本文对应用中每个函数都生成相应的 SPDG,表示为 G_s(见定义 2),其生成过程如算法 1 所示.由于应用中存在大量的第三方通用库,需建立白名单过滤应用的库文件,降低库文件给应用相似性检测带来的影响.另外,一些代码行数少的函数存在功能通用和模板相似的现象,这些函数的 SPDG 会出现相似或相同的问题.因此,本文中过滤掉 SPDG 节点个数小于特定值(本文中特定值设为 5)的函数.

算法 1 SPDG 生成算法

输入:应用 apk 文件 F;

Android 平台 API 集合 S_t.

输出:应用的 SPDG 集合表示 G_s.

```
methods=getMethod(F); /* 获取应用的方法 */
for m in methods
nodes=countNode(m, S_t); /* 计算方法中含有系统API的语句个数 */
if nodes < 5 or is ThirdLibrary(m): /* 去除节点数少于5和第三方库文件的函数 */
continue;
else:
G_s←genSPDG(m, TYPE=DataDep); /* 将每个方法的SPDG保存在集合G_s中,节点间存在数据依赖 */
end for
```

2) 特征向量化表示

SPDG 以一种图形化结构表示应用,但是由于子图同构是 NP 难问题,难以在多项式时间内完成.因此,基于 SPDG 的应用特征表示方法难以拓展到大规模的应用检测环境中.本文设计了一种基于 SPDG 的子图特征表示方法,将 SPDG 分割成节点对,如图 3 所示.

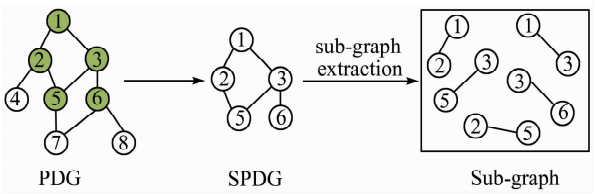

图 3 应用的 SPDG 的特征子图表示

Fig. 3 The featured sub-graph representation of application's SPDG

在图 3 的子图特征表示中,标记有颜色的节点为含有 Android 平台 API 的语句,将生成的 SPDG 分割成节点对,子图表示成 $g=(n_x, n_y)$,其中节点 n_x 和 n_y 之间存在数据依赖关系,n_x 和 n_y 表示语句中调用 Android 平台的 API.抽取数据集中所有应用的 SPDG 形成的子图集合表示成 $S_g=\{g_1,\cdots,g_k,\cdots,g_m\}$.每个函数的 SPDG 特征向量化表示为 $\boldsymbol{H}_i^{G_s}=[(h_i^{g_1})G_s,\cdots,(h_i^{g_k})G_s,\cdots,(h_i^{g_m})G_s]^{\mathrm{T}}$,当 g_k 在应用的子图中时,$(h_i^{g_k})G_s=1$,否则,$(h_i^{g_k})G_s=0$.

3) 相似性检测

在应用特征向量化之后,本文中使用 Jaccard 距离计算两个应用的特征相似性距离,即

$$J(H_i^{G_s},H_j^{G_s})=\frac{|H_i^{G_s}\cap H_j^{G_s}|}{|H_i^{G_s}\cup H_j^{G_s}|} \tag{1}$$

应用 A_1 和 A_2 相似性计算如下

$$\mathrm{sim}(A_1,A_2)=\sum_{1\leqslant i,\ j\leqslant n}\sigma_{i,j}\cdot J_{\mathrm{m}}(H^{G_s}_{A_1,i},H^{G_s}_{A_2,j}) \quad (2)$$

(2)式中 n 表示应用 A_1 和 A_2 的函数个数最小值,系数 σ 表示根据函数节点个数统计得出的权重,J_{m} 表示最大的 Jaccard 距离. 如果 $\mathrm{sim}(A_1, A_2)$ 大于阈值 0.75,判定应用 A_1 和 A_2 是重打包应用程序对. 关于阈值的确定见第 4 节.

3.4　基于 PDG 的细粒度重打包应用分析

与 SPDG 的检测类似,对应用的每个函数都生成 PDG,表示为 G(见定义 3). 在生成应用的 PDG 时删除节点数少于 8 的函数,并使用白名单过滤掉第三方库.

在利用 PDG 作为应用的特征相似性比较时,将其转化成子图同构,找出存在部分或全部克隆的程序代码片段. 基于 PDG 的应用相似性计算如下

$$\mathrm{dis}(P_1,P_2)=\frac{\sum\limits_{G\in P_1}|C(G)|}{\sum\limits_{G\in P_1}|G|} \quad (3)$$

(3)式中,P_1 和 P_2 表示一对可疑的重打包应用,$|G|$ 表示应用程序 P_1 的函数个数,$C(G)$ 表示 P_1 在 P_2 中找到最佳匹配子图的节点个数.

4　实验与结果分析

4.1　方法实现

DroidFAR 重打包检测主要有以下几个步骤:

① Android 安装文件的反编译. Keytool 用于提取应用的签名信息,作为应用的惟一标识. 使用 apktool 反编译 DEX 文件生成应用的 SPDG 和 PDG.

② 过滤第三方库. 文献[24]中公开了多达 1 000个第三方的功能库和 240 个广告库,本文利用这些公开的信息库作为白名单过滤应用中的库文件.

③ SPDG 和 PDG 的生成. 在本文中使用 soot[25] 生成应用中每个函数的 SPDG 和 PDG,其中 SPDG 和 PDG 的节点之间存在数据依赖关系.

④ PDG 同构子图发现. 使用 VF2 算法[26] 找出在细粒度分析中 PDG 的同构子图,识别部分或全部的代码克隆片段.

4.2　实验环境与数据来源

实验环境:CPU 为 Intel(R) Core(TM) i7-6700K 4 GHz,内存 32 GB,操作系统为 Ubuntu 14.04.

本文中的实验数据分为两部分:一部分是来源于第三方市场收集的应用,如表 1 所示;另一部分来自 AndroZoo 中提供的重打包应用程序对[24],包括 2 776 个合法应用和 15 297 个修改生成的重打包应用. 实验中收集的应用安装文件大小分布如图 4 所示,有超过 85% 的 apk 文件大小在 8 MB 以内,apk 文件的平均大小在 4 MB 左右.

表 1　实验数据采集

Table 1　Data collection of experimental data

应用市场	应用数量	比例/%
Google Play	52 789	42.4
Baidu	31 245	25.1
Anzhi	40 478	32.5
Total	124 512	100

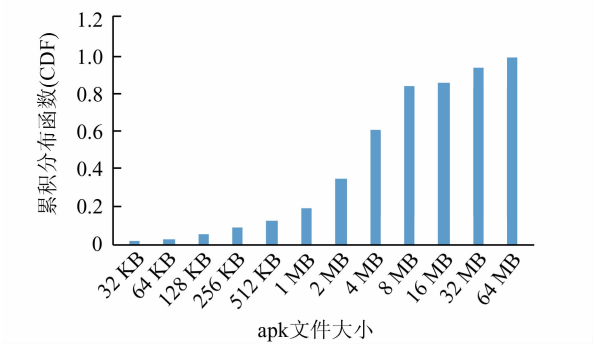

图 4　实验数据中安装文件大小分布

Fig. 4　The distribution of apk files in experimental data

4.3　实验结果分析

本文提出的基于 SPDG 和 PDG 的应用特征表示方法,旨在解决大规模的重打包应用检测问题. 在实验结果评价中,从应用检测的速度和检测的准确率两方面展开有关分析.

基于 SPDG 的粗粒度重打包检测,是为了实现快速检测并且对抗常见的代码混淆攻击对检测准确率的影响,同时降低在细粒度分析中的规模. 图 5 表示在细粒度检测中,代码比较的次数与应用规模变化的关系,实验数据为表 1 中提供的应用数据. 相比于 DNADroid 只通过应用代码的两两比较的方式实现检测,当达到 10×10^4 个应用时,比较的次数近 25×10^{14} 次,在本文中使用 SPDG 作为应用特征实现检测之后,在基于 PDG 的细粒度比较中 10×10^4 个应用只需要 4×10^{14} 次比较.

实验结果表明,基于 SPDG 的粗粒度检测可以实现分析规模的约减,随着应用的增长其代码对的比较次数可以保持在线性增长空间内,因此本文的方法可以有效地应用于大规模的第三方应用市场

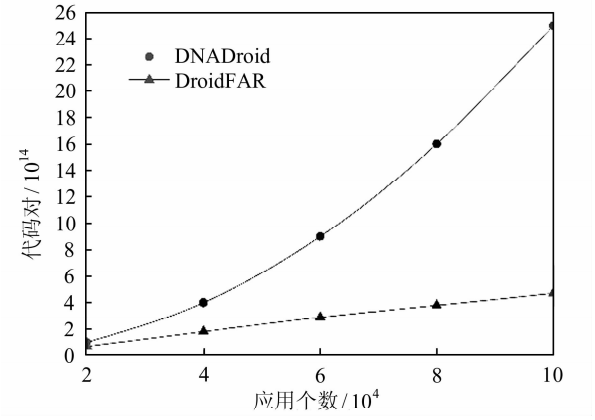

图 5 程序代码对与应用数量的关系

Fig. 5 The relation of program code pair and application number

检测.

图 6 表示在基于 SPDG 的粗粒度检测和基于 PDG 的细粒度检测中,阈值与准确率的关系. 实验中,我们在 AndroZoo 数据集上通过多次调整阈值,发现阈值设为 0.75 时,基于 SPDG 的粗粒度快速相似性检测准确率最高达到 87%;阈值设为 0.85 时,基于 PDG 的细粒度应用相似性检测准确率最高达到 0.951(图 6).

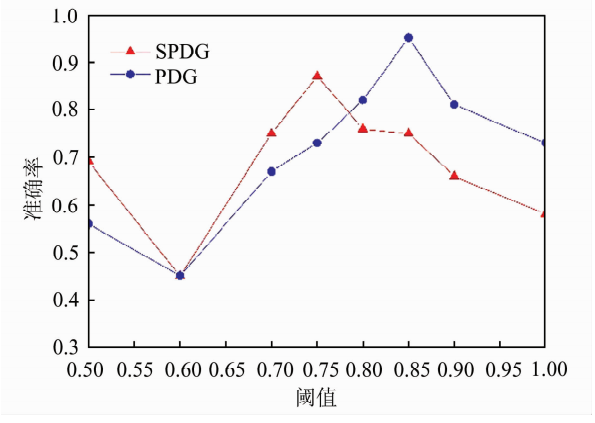

图 6 阈值与检测准确率的关系图

Fig. 6 The relation of threshold and detection accuracy

实验结果显示,本文方法的漏报率(FNR)为 4.2%,误报率(FPR)为 1.2%,召回率(recall)为 95.8%,准确率(accuracy)为 95.1%,精准率(precision)为 98.8%,准确率和召回率的调和平均值(F-score)为 97.3%. 实验结果的 ROC(receiver operating characteristic)曲线如图 7 所示.

本文的方法与现有的基于代码克隆[6,9,16]和基于应用资源文件[17,18]等重打包应用检测的方法在检测速度、检测准确率和抗代码混淆三方面比较结果如表 2 所示.

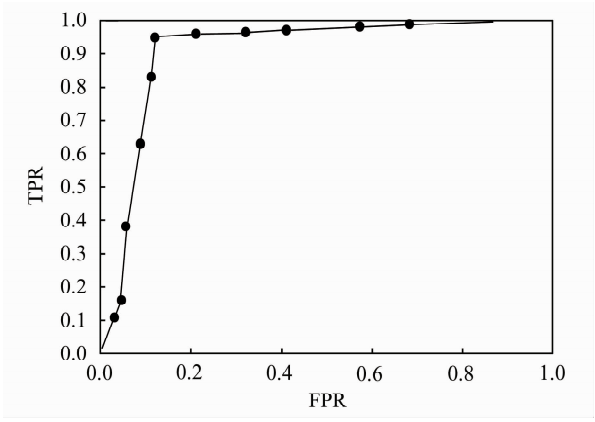

图 7 ROC 曲线

Fig. 7 ROC curve

TPR (true positive rate), FPR(ralse positive rate)

表 2 重打包应用检测方法对比

Table 2 Comparison of repackaging detection approaches

重打包应用检测方法	检测速度	检测准确率	抗代码混淆
基于代码克隆检测方法[6,9,16]	慢	高	弱
基于资源文件检测[17,18]	中	低	强
本文方法	快	高	强

5 相关讨论

在 Android 应用中存在大量的第三方库如广告库、功能库等,使用 Androzoo 提供的白名单过滤第三方库,由于提供的白名单并不能完全包括所有的第三方库列表,所以在库过滤中存在一定的误报和漏报的现象. 本文的研究对象是修改 DEX 文件生成的重打包应用,修改本地代码生成的重打包应用不在本文的研究范围内. 在未来工作中我们可以在检测系统中补充关于二进制程序克隆检测的功能,解决 DroidFAR 存在的不足. 由于目前鲜有报告指出存在大量修改本地代码生成的重打包应用,因此本文中针对 Java 代码的研究范围是合理的.

使用 SPDG 表示应用的特征,相比于现有的重打包应用检测方法可以有效地抵御常见的代码混淆攻击如布局混淆、数据混淆和控制混淆. ProGuard 是 Android 平台使用最广泛的代码混淆工具,实验表明本文方法可以有效地分析并检测使用 ProGuard 生成的 apk 文件. 但是,本文方法针对加壳保护的 apk 文件无效. 在未来的工作中,我们尝试使用动态分析方法研究加壳保护的重打包应用检测.

在进行粗粒度的快速重打包检测后,输出可疑的重打包应用 $|P| \ll M$(数据集中应用的数量),有

效地降低了基于 PDG 的细粒度检测阶段中的分析规模,因此本文方法可以满足大规模应用市场的检测要求.

6　结　论

本文提出了一种基于程序依赖图的语义特征表示方法,利用 Android 平台 API 包含丰富的语义可以用于表示程序行为的特点,设计了一种基于语义特征的快速相似性检测方法,识别可疑的重打包应用.使用程序依赖图表示应用的特征,实现细粒度的准确检测.本文的方法能够有效地检测 1.2 节提到的类型 1、2、3 的重打包应用,但不适用类型 4 的重打包应用.

本文的检测方法仍然是用两两比较的方式来计算某一应用与合法应用之间的相似性距离,从而判断该应用是否为重打包应用.如果合法应用数据缺失,则会导致漏报.在未来工作中,我们将尝试从应用程序的代码风格出发,提取能表示重打包应用潜在特点的特征,降低检测的漏报率.

参考文献:

[1] ZHOU Y J, JIANG X X. Dissecting Android malware: Characterization and evolution [C]//2012 *IEEE Symposium on Security and Privacy* (*SP*). Washington D C: IEEE, 2012: 95-109. DOI: 10.1109/sp.2012.16.

[2] 卿斯汉. Android 安全研究进展[J]. 软件学报, 2016, **27**(1): 45-71. DOI: 10.13328/j.cnki.jos.004914.
QING S H. Research progress on Android security [J]. *Journal of Software*, 2016, **27**(1): 45-71. DOI: 10.13328/j.cnki.jos.004914(Ch).

[3] 张玉清, 王凯, 杨欢, 等. Android 安全综述[J]. 计算机研究与发展, 2014, **51**(7): 1385-1396. DOI: 10.7544/issn1000-1239.2014.20140098.
ZHANG Y Q, WANG K, YANG H, *et al*. Survey of Android OS security [J]. *Journal of Computer Research and Development*, 2014, **51**(7): 1385-1396. DOI: 10.7544/issn1000-1239.2014.20140098(Ch).

[4] ACAR Y, BACKES M, BUGIEL S, *et al*. Sok: Lessons learned from Android security research for appified software platforms [C]//2016 *IEEE Symposium on Security and Privacy* (*SP*). Washington D C: IEEE, 2016: 433-451. DOI: 10.1109/SP.2016.33.

[5] 刘剑,苏璞睿,杨珉, 等. 软件与网络安全研究综述[J]. 软件学报, 2018, **29**(1): 42-68. DOI: 10.13328/j.cnki.jos.005320.
LIU J, SU P R, YANG M, *et al*. Software and cyber security—A survey [J]. *Journal of Software*, 2018, **29**(1): 42-68. DOI: 10.13328/j.cnki.jos.005320(Ch).

[6] ZHOU W, ZHOU Y, JIANG X, *et al*. Detecting repackaged smartphone applications in third-party Android marketplaces [C]//*Proceedings of the Second ACM Conference on Data and Application Security and Privacy*. New York: ACM, 2012: 317-326. DOI: 10.1145/2133601.2133640.

[7] LI L, LI D, BISSYANDE T F, *et al*. Understanding Android app piggybacking [C]//*Proceedings of the 39th International Conference on Software Engineering Companion*. Washington D C: IEEE Press, 2017: 359-361. DOI: 10.1109/ICSE-C.2017.109.

[8] HANNA S, HUANG L, WU E, *et al*. Juxtapp: A scalable system for detecting code reuse among Android applications [C]//*International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Heidelberg: Springer-Verlag, 2012: 62-81. DOI: 10.1007/978-3-642-37300-8_4.

[9] CRUSSELL J, GIBLER C, CHEN H. Attack of the clones: Detecting cloned applications on Android markets [C]//*European Symposium on Research in Computer Security* (*LNCS* 7459). Heidelberg: Springer, 2012: 37-54. DOI: 10.1007/978-3-642-33167-1_3.

[10] 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*. New York: ACM, 2015: 71-82. DOI: 10.1145/2771783.2771795.

[11] 王浩宇, 王仲禹, 郭耀, 等. 基于代码克隆检测技术的 Android 应用重打包检测[J]. 中国科学:信息科学, 2014, **44**(1): 142-157. DOI: 10.1360/N112013-00130.
WANG H Y, WANG Z Y, GUO Y, *et al*. Detecting repackaged Android applications based on code clone detection technique [J]. *Scientia Sinica* (*Informationis*), 2014, **44**(1): 142-157. DOI: 10.1360/N112013-00130(Ch).

[12] KIM D, GOKHALE A, GANAPATHY V, *et al*. Detecting plagiarized mobile apps using API birthmarks [J]. *Automated Software Engineering*, 2016, **23**(4): 591-618. DOI: 10.1007/s10515-015-0182-6.

[13] 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 Enginee-*

ring. New York: ACM, 2014: 175-186. DOI: 10.1145/2568225.2568286.

[14] CHEN K, WANG P, LEE Y, *et al*. Finding Unknown Malice in 10 Seconds: Mass Vetting for New Threats at the Google-Play Scale [C/OL]. [2018-05-12]. *https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-chen-kai.pdf*.

[15] ZHOU W, ZHOU Y, GRACE M, *et al*. Fast, scalable detection of "piggybacked" mobile applications [C]//*Proceedings of the Third ACM Conference on Data and Application Security and Privacy*. New York: ACM, 2013: 185-196. DOI: 10.1145/2435349.2435377.

[16] CRUSSELL J, GIBLER C, CHEN H. AnDarwin: Scalable detection of semantically similar android applications [C]//*European Symposium on Research in Computer Security*. Heidelberg: Springer-Verlag, 2013: 182-199. DOI: 10.1007/978-3-642-40203-6_11.

[17] ZHANG F, HUANG H, ZHU S, *et al*. ViewDroid: Towards obfuscation-resilient mobile application repackaging detection [C]//*Proceedings of the* 2014 *ACM Conference on Security and Privacy in Wireless & Mobile Networks*. New York: ACM, 2014: 25-36. DOI: 10.1145/2627393.2627395.

[18] SUN M S, LI M M, LUI J C S. DroidEagle: Seamless detection of visually similar android apps [C]//*Proceedings of the* 8*th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. New York: ACM, 2015: 1-12. DOI: 10.1145/2766498.2766508.

[19] 焦四辈,应凌云,杨轶,等. 一种抗混淆的大规模 Android 应用相似性检测方法[J]. 计算机研究与发展, 2014, **51**(7): 1446-1457. DOI: 10.7544/issn1000-1239.2014.20121775.

JIAO S B, YING L Y, YANG Y, *et al*. An anti-obfuscation method for detecting similarity among Android applications in large scale [J]. *Journal of Computer Research and Development*, 2014, **51**(7): 1446-1457. DOI: 10.7544/issn1000-1239.2014.20121775 (Ch).

[20] SHAO Y, LUO X, QIAN C, *et al*. Towards a scalable resource-driven approach for detecting repackaged android applications [C]//*Proceedings of the* 30*th Annual Computer Security Applications Conference*. New York: ACM, 2014: 56-65. DOI: 10.1145/2664243.2664275.

[21] GADYATSKAYA O, LEZZA A L, ZHAUNIAROVICH Y. Evaluation of resource-based app repackaging detection in Android [C]//*Nordic Conference on Secure IT Systems*(*LNCS* 10014). Heidelberg: Springer-Verlag, 2016: 135-151. DOI: 10.1007/978-3-319-47560-8_9.

[22] LIU C, CHEN C, HAN J, *et al*. GPLAG: Detection of software plagiarism by program dependence graph analysis [C]//*Proceedings of The* 12*th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York: ACM, 2006: 872-881. DOI: 10.1145/1150402.1150522.

[23] ZHOU Y, WANG Z, ZHOU W, *et al*. Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets [C/OL]. [2018-05-12]. *http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/07_5.pdf*.

[24] ALLIX K, BISSYANDE T F, KLEIN J, *et al*. AndroZoo: Collecting millions of Android apps for the research community [C]//*ACM* 13*th Working Conference on Mining Software Repositories*. New York: ACM, 2016: 468-471. DOI: 10.1145/2901739.2903508.

[25] LAM P, BODDEN E, LHOTAK O, *et al*. The soot framework for java program analysis: A retrospective [DB/OL]. [2017-10-02]. *https://sable.github.io/soot/resources/lblh11soot.pdf*.

[26] CORDELLA L P, FOGGIA P, SANSONE, *et al*. A (sub) graph isomorphism algorithm for matching large graphs [J]. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004, **26**(10): 1367-1372. DOI: 10.1109/TPAMI.2004.75.

□