计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2014.20121775
51(7):1446-1457, 2014

一种抗混淆的大规模 Android 应用相似性检测方法

焦四辈　应凌云　杨　轶　程　瑶　苏璞睿　冯登国
(中国科学院软件研究所可信计算与信息保障实验室　北京　100190)
(jiaosibei@tca.iscas.ac.cn)

An Anti-Obfuscation Method for Detecting Similarity Among Android Applications in Large Scale

Jiao Sibei, Ying Lingyun, Yang Yi, Cheng Yao, Su Purui, and Feng Dengguo
(*Trusted Computing and Information Assurance Laboratory*, *Institute of Software*, *Chinese Academy of Sciences*, *Beijing* 100190)

Abstract　Code obfuscation exerts a huge impact on similarity detection among Android applications based on behavior characteristics. In order to deal with the situation, we propose a novel way of similarity detection among Android applications based on file content characteristics, which computes the similarity of file content features and can be applied to large-scale scenario in real world. Our method is not subject to code obfuscation or file obfuscation. We choose to utilize the characteristics of image, audio and layout files which are shown in our statistics as the most representative features in Android applications. Meanwhile, different weights are given to these features through machine learning, which further enhances the accuracy of our method. In addition, we implement a prototype system and particularly optimize each step to speed up the calculation, making our system suitable for large-scale scenario and give a good calculation performance. The experiments dataset contains 59 000 applications. And for both legitimate application and malware applications, our system successfully detects those repackaged pirate applications and those with the similar malicious component, which prove the effectiveness of our method. The experiment results demonstrate that similarity detection based on file content characteristics could resist the file obfuscation and give better performance in both accuracy and efficiency.

Key words　file content characteristics; fuzzy Hash; perceptual features; Android; application similarity; anti-obfuscation

摘　要　随着代码混淆、加壳技术的应用,基于行为特征的 Android 应用相似性检测受到的影响愈加明显.提出了一种抗混淆的大规模 Android 应用相似性检测方法,通过提取应用内特定文件的内容特征计算应用相似性,该方法不受代码混淆的影响,且能有效抵抗文件混淆带来的干扰.对 5.9 万个应用内的文件类型进行统计,选取具有普遍性、代表性和可度量性的图片文件、音频文件和布局文件作为特征文件.针对 3 种特征文件的特点,提出了不同内容特征提取方法和相似度计算方法,并通过学习对其相似度赋予权重,进一步提高应用相似性检测的准确性.使用正版应用和已知恶意应用作为标准,对 5.9 万

收稿日期:2013-12-10;**修回日期**:2014-03-06
基金项目:国家"九七三"重点基础研究发展计划基金项目(2012CB315804);国家自然科学基金重大研究计划项目(91118006);国家自然科学基金项目(61073179);北京市自然科学基金项目(4122086)
通信作者:应凌云(yly@tca.iscas.ac.cn)

个应用进行相似性检测实验,结果显示基于文件内容的相似性检测可以准确识别重打包应用和含有已知恶意代码的应用,并且在效率和准确性上均优于现有方案.

关键词 文件内容特征;模糊散列;感知特征;安卓;应用相似性;抗混淆

中图法分类号 TP309; TP181

Android 系统是目前手机市场上最主要的智能手机系统,IDC 发布的"Worldwide Quarterly Mobile Phone Tracker"[1]显示,2013 年第 1 季度 Android 手机市场份额达到 75%,Android 应用的数量也随着系统的普及而快速增长,谷歌官方市场的 Android 应用数量已达到 100 万[2].然而,在 Android 智能手机与应用数量迅速增长的同时,恶意代码已经悄然渗透入 Android 领域并感染大量的用户设备,网秦"2012 上半年全球手机安全报告"[3]显示,2012 年发现了超过 65 227 个新型恶意软件,较 2011 年同比增长 263%.据估计,2012 年超过 3 200 万 Android 设备被感染,同比增长 200%,其中应用重打包是传播恶意软件的主要途径之一.

Android 应用是采用集中分发制,即开发者将应用提交给应用市场,用户通过应用市场下载应用.谷歌官方应用市场采用多种安全机制对提交的应用进行安全检测,并不断完善安全审核机制以提高检测准确性与全面性,但是官方市场内依然无法避免恶意软件的存在[3];而第三方应用市场对于应用审核不严格,安全保障机制不够完善,导致市场内恶意软件的比例居高不下[3].其中,嵌入已知恶意代码和重打包应用是主要威胁.

针对目前嵌入已知恶意代码和重打包应用带来的安全问题,国内外研究学者提出了一系列解决方法,其中通过检测应用相似性来识别嵌入已知恶意代码应用和重打包应用是目前研究的主流方法.

目前 Android 应用相似性检测方法主要有基于行为的检测和基于文件的检测.基于行为的检测方法采用反编译工具[4-8]或者动态行为分析工具[9-12]得到应用行为序列,对行为序列进行预处理得到行为序列特征,通过比较行为序列特征的距离得到应用相似性的量化数据.该方法可以检测应用代码的改变,解决包括代码重用、剽窃和重打包等问题,但是行为序列特征的提取容易受到代码混淆技术的影响[13],因而在针对实际问题进行分析时具有一定的局限性.基于文件的检测是根据应用文件目录结构或者文件内容来提取特征,其优势在于不受代码混淆的影响,但是该方法容易受垃圾文件的干扰,应用在重打包时被恶意插入大量垃圾文件后,文件目录结构和文件数量会发生变化,进而大幅度影响相似性检测结果.

如上所述,目前应用市场应用数量庞大,增长迅速,安全形势严峻,研究抗混淆的大规模 Android 应用相似性检测方法对保证应用市场的安全性和提高智能手机环境安全水平具有重要意义.根据目前的研究现状,本文提出了一种抗混淆的大规模 Android 应用相似性检测方法.通过分析 Android 应用的文件属性,选取具有普遍性、代表性和可度量性的文件类型提取内容特征,并根据文件类型采用不同的内容特征提取算法,通过机器学习对其相似度赋予权重,从而提高应用相似性检测的准确性和运算效率.本文的贡献主要包括如下 3 个部分:

1) 提出了一种抗混淆的 Android 应用相似性检测方法,通过提取应用内的 3 类文件内容特征进行应用相似性检测,该方法不受代码混淆技术的影响,且可以有效抵抗插入无用文件和文件目录结构改变带来的干扰.

2) 选取适应大规模运算的系统框架和算法,通过统计分析和人工测试选择合适的特征文件和内容特征提取算法,缩小文件内容特征规模,提高运算效率,适应大规模运算.

3) 完成了原型系统开发,实现了抗混淆的 Android 应用特征提取和相似性检测功能.并以 Kongfu、水果忍者等样本作为标准检测相似应用.实验结果表明,本文提出的方法可以准确识别重打包应用和嵌入已知恶意代码的应用,并能有效抵抗文件混淆带来的干扰.

1 相关工作

目前 Android 应用相似性检测方法包括行为相似性检测和文件相似性检测.在行为相似性检测方面,Hanna 等人在文献[14]中提出一种根据行为相似性检测 Android 应用代码重用的方法,该方法首先通过反编译工具得到 Android 可执行字节码,从中提取应用行为序列,采用特征散列算法对行为序

列进行处理得到应用行为特征;然后采用群落系数相似性度量方法计算行为特征相似性.该模型适用于检测恶意代码重用、应用重打包、代码抄袭.同时Zhou等人[15]提出了DroidMOSS———一个根据行为相似性检测第三方Android应用市场重打包应用的系统.该系统首先假设谷歌官方市场中的应用为正版应用,使用正版应用与第三方市场的应用进行比较,如果第三方市场中的应用和谷歌官方市场的应用相似度超过预设阈值,且相似应用的签名信息不相同,则判定第三方市场中的应用为重打包应用;DroidMOSS通过工具反编译得到Android可执行字节码,从中提取操作数信息,然后对操作数序列进行模糊散列得到应用行为特征,比较应用行为特征得到应用间的相似度.该文分析了6个Android第三方应用市场中的22 906个应用,实验结果发现第三方市场中有5%~13%的应用为重打包应用.

但是,基于应用行为的相似性检测,首先需要从可执行文件中逆向分析得到应用行为序列,而逆向分析的过程容易受到代码混淆的影响,Huang等人[13]主要关注Android应用代码混淆技术对基于行为特征的相似性检测带来的影响,该文提出了一套评估基于行为特征的应用相似性检测算法有效性的机制,通过举例指出Zhou等人[15]采用的模糊散列方法和Hanna等人[14]的特征散列方法均无法有效应对混淆代码,重打包人员只需要插入几行混淆语句就会导致其相似性检测方法失效.该文采用了多种代码混淆技术,用基于应用行为的相似性检测工具androguard[16]进行实验,发现每种代码混淆技术都会直接影响相似性检测结果,其中"Class Encrypter"等技术导致重打包前后的应用相似性低于5%.

由于代码混淆技术主要针对可执行代码,而Android应用程序内还有其他多种文件,这些资源文件的内容特征和文件属性难以进行混淆,因此,基于应用文件的相似性检测也成为目前的热点研究方法之一.Li在文献[17]中采用文件目录结构对应用相似性进行评估,用树结构表示应用的目录结构,计算树之间的编辑距离,由此得到应用之间的距离.该方法可以检测重打包应用、嵌入已知恶意代码应用.但是,该方法的成功得益于恶意开发者基本不改变文件目录结构,如果插入无用文件或者改变文件目录名则会导致该方法失效.

针对目前Android应用相似性检测方法的局限性,本文通过分析Android应用内的文件属性和文件内容特征,提出了一种抗混淆的大规模应用相似性检测方法,该方法不受代码混淆技术的影响,且可以有效地抵御插入无用文件等文件混淆方法.

2 基本思路

Android应用以压缩文件的形式存在,内部以文件夹的形式组织存放可执行字节码文件、证书文件和资源文件,其中可执行dalvik字节码存储在class.dex文件中;证书文件是应用的签名文件,位于META-INF目录下;资源文件包括数据库文件、函数库文件、XML文件、图片文件等.我们以水果忍者的Android应用为例进行描述,图1所示为解压水果忍者应用得到的部分文件,其中META-INF目录下的3个文件是签名文件,后缀名为ogg[18]的文件是音频文件,后缀名为png[19]是图片文件,后缀名为xml是布局文件.

```
Archive:  com.halfbrick.fruitninjafree_19199200_0.apk
  Length      Date    Time    Name
---------  ---------- -----   ----
   187151  08-20-2012 10:25   META-INF/MANIFEST.MF
   187272  08-20-2012 10:25   META-INF/IDREAMSK.SF
      949  08-20-2012 10:25   META-INF/IDREAMSK.RSA
     7975  04-19-2012 13:21   assets/sound/bonus-banana-x2.ogg
     4035  04-19-2012 13:21   assets/sound/bonus-count-up.ogg
    16062  04-19-2012 13:21   assets/sound/bonus-drum-roll.ogg
     5976  04-19-2012 13:21   assets/sound/bonus-explosion-1.ogg
     6032  04-19-2012 13:21   assets/sound/bonus-explosion-3.ogg
     6253  04-19-2012 13:21   assets/sound/bonus-explosion-5.ogg
   194807  08-15-2012 10:15   res/drawable/wood2.png
     3578  08-15-2012 10:15   res/drawable/yourall.png
     3422  08-15-2012 10:15   res/drawable/yourfriend.png
     2592  08-15-2012 10:15   res/layout/achivementdesc.xml
     1168  08-15-2012 10:15   res/layout/admob.xml
```

Fig. 1 The files in Fruit Ninja.

图1 Android应用水果忍者内的部分文件

根据对Android应用样例的分析,可以将Android应用描述为集合$app=\{dalvik;lib;layout;image;sound;etc\}$,其中$dalvik$表示应用中的可执行字节码,$lib$表示程序中的原生代码库,$layout$表示用于程序数据存储和布局描述的XML文档,$image$表示程序中的图片文件,$etc$表示程序中的其他文件.根据集合$app$的描述可知:本文的目标是根据$dalvik$,$lib$,$layout$,$image$等相关文件的内容特征,提出一种Android应用相似性判定方法.

为了准确、有效地通过文件内容分析应用相似性,并符合实际的检测需求,本文提出的方法着力达到以下3个目标:1)适应大数据运算,Android应用市场内的应用数量基数大、增长快,能快速处理大量数据的系统框架是适应大数据运算的基础;2)选取合适的特征文件,Android应用内有上千种文件类

型,提取哪些文件的内容直接影响相似性检测的效率和准确性;3)高效的特征提取和准确的相似性计算算法,提取文件内容特征的速度决定了系统效率,同时准确的相似性计算算法是保证系统能够正确给出判定结果的基本保证.下面本文将从系统基本框架、特征文件选取、特征提取和相似性算法设计 3 方面阐述其思路.

2.1 适应大数据运算的基本框架

为了适应大数据运算的性能要求,本文从特征文件选取、特征提取算法设计、相似度计算算法设计入手,在提取文件内容特征、计算应用相似度的过程中保证提高效率的同时不失运算结果的准确性.

要求系统适应大数据运算,首先要求算法针对的目标不能过于复杂,如果针对目标过于复杂,那么需要对这个目标进行缩减,选出其中关键的要素进行对比;其次算法效率高;最后,在构建算法过程的时候,要尽可能对算法的运行环境进行优化,减少算法的中间步骤,削减算法中可能引起大量时间和空间消耗的内容.

抗混淆的大规模相似性检测方法如图 2 所示,首先需要选取合适的特征文件,一个 Android 应用中的文件从几百个到几千个不等,如对全部文件的内容进行特征提取,容易造成目标过于复杂、分析效率低下的结果,且容易受到插入无用文件的干扰.因此本文根据普遍性、代表性和可度量性原则,选取部分合适的文件类型作为特征文件,在最大程度保证特征文件有效表示 Android 应用的情况下缩小特征规模,从而减小运算量,具体的选取原则在 3.2 节中详细阐述.接下来,从 Android 应用中提取已选定文件的特征,本文采用 Java 的 ZipFile[20] 接口获取应用内的文件接口,根据压缩文件位置偏移定位特征文件,省去对其他无关文件进行解压的步骤以提高运算效率.特征提取算法是保证特征能够正确代表应用的根本,本文首先对 Android 应用中的特征文件进行统计,根据统计规律对比不同的算法实现,对算法进行最合适的优化,在保证准确性的前提下采用效率最高的算法,并在提取过程中应用多线程方案,重写 Java 中不支持多线程的部分函数,保证所有运算的线程安全性,进一步提高运算效率.最后,基于文件内容特征进行相似性检测,在相似度度量算法设计时,根据 Android 应用的统计特征,采用散列表计数,用空间消耗换取时间优化,使得相似性计算算法速度提高 10 倍左右,本文将在 2.3 节阐述文件内容特征提取和相似度计算算法.

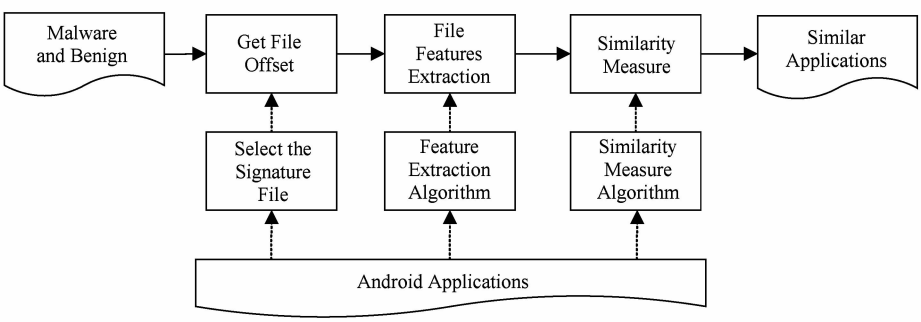


Fig. 2 System overview.

图 2 系统框架图

2.2 选取合适的特征文件

Android 应用内的文件数量多达几千个,通过文件内容特征计算应用相似度,首先要从复杂的 Android 文件类型中选取合适的特征文件.本文认为合适的特征文件需要具有普遍性、代表性和可度量性 3 个特点.普遍性表示大多数 Android 应用内包含该类型的文件,如果某个文件类型仅在少数应用内存在,则无法通过该类文件内容特征进行相似度比较;代表性表示文件内容特征具有“签名”特性,可以代表该 Android 应用,不同 Android 应用中提取出的文件内容特征具有差异性;可度量性表示文件内容具有距离特性,相似应用中的文件内容距离近,反之不同应用中的文件内容距离远.经过统计实验(见 4.3.1 节),选取布局文件、图片文件、音频文件作为特征文件,可描述为 $app\_files=\{image;sound;layout\}$,主要思路是计算文件内容特征相似度,以此分析应用相似度,可用以下公式表示:

$$sim(app_1,app_2)=sim(app\_files_1,app\_files_2).$$

2.3 文件内容特征提取和相似度计算算法

本文选取了 3 类文件作为特征文件,用这 3 类文件的内容特征表示应用特征,具体的算法实现将在第 4 节阐述.每类文件内容特征集合包含了此类所有文件的特征,用如下公式表示:

$$image\_fea = \sum_{i=1}^{n} image\_fea[i];$$

$$sound\_fea = \sum_{i=1}^{n} sound\_fea[i];$$

$$layout\_fea = \sum_{i=1}^{n} layout\_fea[i].$$

计算图片、音频、布局文件的内容相似度,对 2 个应用的每种特征进行对比,可以推导出文件特征相似度计算公式如下,表示应用内文件相似性等价于 2 个应用内所有该类型的相似性:

$$sim\_image(app_1, app_2) = \sum_{i=1}^{n}\sum_{j=1}^{m} sim(image\_fea_1[i], image\_fea_2[j]);$$

$$sim\_sound(app_1, app_2) = \sum_{i=1}^{n}\sum_{j=1}^{m} sim(sound\_fea_1[i], sound\_fea_2[j]);$$

$$sim\_layout(app_1, app_2) = \sum_{i=1}^{n}\sum_{j=1}^{m} sim(layout\_fea_1[i], layout\_fea_2[j]).$$

在实验过程中发现,单独使用图片、音频或者布局文件内容特征相似度代表应用相似度,结果不够理想,如果阈值设置较高会导致漏报;如果阈值设置过低则会导致误报. 因此,本文通过机器学习对图片、音频和布局文件内容相似度赋予权重,通过 3 种文件内容特征的加权相似度表示应用相似度,加权相似度公式表示如下:

$$sim(app_1, app_2) = sim(app\_files_1, app\_files_2) = sim\_image \times \alpha + sim\_sound \times \beta + sim\_layout \times \gamma.$$

该公式表示应用 app_1 和 app_2 的相似性等价于 app_1 和 app_2 内部文件的相似性,等价于 2 个应用内图片、声音、布局文件相似性的加权值. 此处 α, β, γ 的值根据 sim_image, sim_sound, sim_layout 的不同而不同. 从图 6~8 可知,图片、音频和布局文件在 Android 应用中的数量不一,而且部分应用不包含音频文件,所以固定的 α, β, γ 无法有效地计算应用相似度. 经过实验,本文提出一种动态权值的解决方法:即根据 sim_image, sim_sound, sim_layout 三个值的大小赋予权重,通过学习确定 3 个最合适的权重,分别为 0.6, 0.3, 0.1, sim_image, sim_sound, sim_layout 中值最大的权重为 0.6,其次权重为 0.3,最小的权重为 0.1.

经过以上运算可以获得 2 个应用的相似度,通过比较相似度和阈值 T 的大小可以判断 2 个应用是否属于相似应用.

3 系统实现

本文采用文件内容特征代表应用特征,根据第 3 节提出的系统框架和选取的 3 种特征文件,针对不同文件的特点提出具体的特征提取方法以及相似度计算算法.

3.1 图片内容特征提取

目前,图片内容相似度比较方向已经有了大量的研究,但是大部分图片处理算法需要较大的空间和时间开销[21],无法应用在大规模计算环境中. 在研究过程中,本文作者注意到 Android 系统的重打包应用通常采用 2 种方式影响图片:1)在原有图片基础上进行修改;2)改变原图片清晰度. 基于这样的考虑,在图片内容特征提取过程中,需要选取一种算法,要求这种算法能够降低修改图片和消除清晰度降低带来的干扰. 感知散列算法是一种高效的图片指纹提取算法,该算法根据图片的明暗和构图为每张图片生成一个字符串作为图片的"指纹",2 张图片的指纹越相似,则 2 张图片越相似. 但是文献[21]指出,感知散列算法的缺点是误报多,且运算复杂度随着图片规模增大成指数级增长. 因此,本文根据实验集合中的图片内容特征对感知散列算法进行改进,提高了准确性的同时降低了运算复杂度. 图 3 为传统的感知散列算法流程. 其中缩小图片尺寸和对比图片指纹计算消耗最大,在这 2 个环节进行改进.

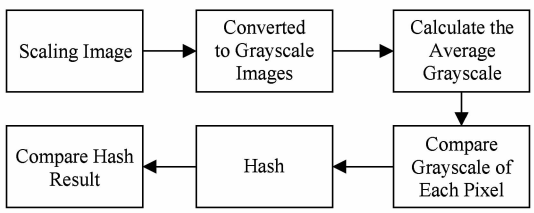


Fig. 3 Image Hash algorithm.

图 3 感知散列算法流程

缩小图片尺寸是将图片缩小到 $K \times K$ 像素,该缩小过程主要用于消除图片清晰度对相似度比较的干扰、去除图片尺寸和图片比例的差异,该过程只保留结构、明暗等基本信息,这里的 K 值一般设为 128. 根据统计(见 4.3.1 节),40×40 分辨率的图片在实验集合的 Android 应用中出现比例最高,所以文献设定 $K=40$,有效减小图片缩放计算消耗.

图片内容相似度比较需要计算指纹的汉明距离,即 2 个指纹字符串对应位置不一样的字符个数,$K=40$,则字符串长度为 $K \times K/8=200$,则每次对比

计算复杂度为 $O(n)=200$. 本文对该步骤进行简化,采用字符串是否相等代替汉明距离,代价是相似度结果只能显示 2 个图片指纹是否一致,无法通过汉明距离检测图片指纹是否相似.

3.2 布局文件内容特征提取

Android 应用内的布局文件以 XML 文件格式存储,因此,布局文件内容特征提取等同于 XML 文件内容特征提取. XML 文件相似度比较包括结构相似度和内容相似度 2 方面,文献[22]将 XML 文件转换为树结构,通过比较树的差异得到 XML 结构差异,通过比较树的节点差异得到 XML 内容差异. 但是基于树的相似度比较计算开销较大,在 Android 应用中布局文件是按照一定规则存储的,在已知规则的情况下,本文采用了一种简单的结构和内容特征提取方法:首先,根据谷歌官方的 Android 布局文件说明[23],得到结构名列表;然后,根据结构名列表提取结构特征,过滤掉布局文件内的结构特征和符号信息,得到内容信息;最终对结构和内容信息计算散列值,得到结构特征值和内容特征. 如图 4 所示,布局文件经过处理后得到一个散列数组,从而将布局文件的内容相似度转化为比较散列数组的相似度.

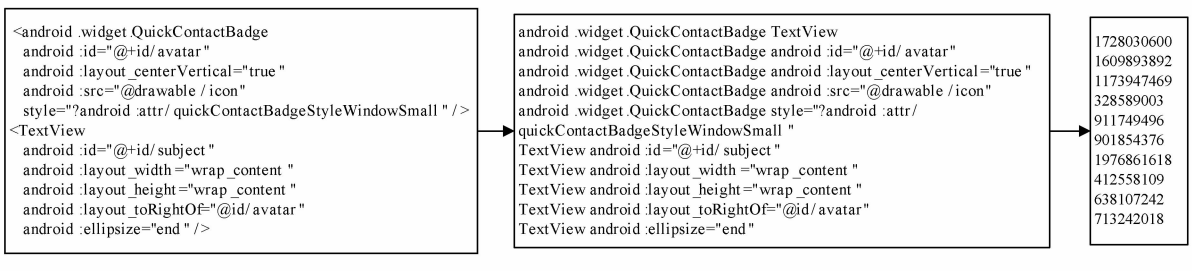


Fig. 4 Layout file content features generation.

图 4 布局文件内容特征提取

3.3 音频文件内容特征提取

音频文件内容相似度已有许多成熟算法,但是已有算法的计算开销大,不适合大规模运算. 本文对 Android 应用内的音频文件进行分析发现,重打包应用不对音频文件进行大的修改,因此本文采用文件散列值作为音频文件特征. 下面对 2 种常见的散列算法进行性能对比,表 1 是 2 种散列算法对字符串进行 10 万次散列计算的时间 t 对比,$MD5$(Java)是 Java 提供的 API 实现散列运算,由于该 API 不是多线程安全的,所以这里仅用作对比;$MD5$(Ours)是本文实现的散列算法,支持多线程运算;$FNV1$[24]是一种高度分散的散列算法. 可以看出,即使是经过 Java 优化的 $MD5$ 算法,运算速度也比 $FNV1$ 慢 40 倍左右. 所以,本文采用 $FNV1$ 计算音频文件散列. 结果为 unsigned int 型值,对于大规模运算来说其散列空间较小,散列结果易发生碰撞. 因此,本文提出多次散列方法,在保证运算速度的情况下大大减小散列碰撞. 算法如下:

算法 1. 多次 $FNV1$ 散列算法.

输入:S;

输出:$hashResult$.

① $M=$"ABCDE";

② $hash1 \leftarrow FNV1(S)$;

③ $S' \leftarrow S+M$;

④ $hash2 \leftarrow FNV1(S')$;

⑤ $hashResult \leftarrow hash1+hash2$;

⑥ return $hashResult$.

Table 1 Time-Consuming of Hash Algorithm

表 1 散列算法耗时对比

Algorithm	t/ms
$MD5$(Java)	2 744
$MD5$(Ours)	5 217
$FNV1$	70

通过以上算法得到音频文件的多次散列值作为音频文件的内容特征.

3.4 相似度计算

Android 应用文件内容特征包含图片内容特征、布局文件内容特征和音频文件内容特征. 图片内容特征为图片"指纹"集合;一个布局文件内容特征为一个散列集合,应用内的所有布局文件内容特征由多个散列集合组成;音频文件内容特征为散列集合. 3 种文件内容特征集合均可视为字符串集合. 而常用的字符串相似度计算算法有欧氏距离、杰卡德距离等. 本文对一组相似应用中包含的 3 种特征文件进行统计(见 4.3.1 节),发现计算文件内容相似度不适合使用欧氏距离和杰卡德距离. 经过对比,本文选择内容相似度作为标准,其计算方法是:集合 A

和 B 的交集元素在 A 和 B 中较小的集合所占的比例. 这种方法可以有效地衡量不同长度的集合之间的相似度. 内容相似度用 $L(A,B)$ 表示如下:

$$sim(A,B)=L(A,B)=\frac{|A\cap B|}{\min(|A|,|B|)}.$$

由此,特征文件集合相似度计算公式由文件特征相似度公式和内容相似度公式推导得到式(1),表示文件集合相似度等价于文件集合的内容相似度;应用相似度计算由式(1)和加权相似度公式推导得到式(2),表示应用相似度等价于文件集合相似度加权值,即 3 种特征文件内容相似度的加权值.

通过计算特征文件内容相似度得到应用相似度,不受文件目录结构改变的干扰;而选取的相似度计算方法采用 2 个集合中较小的集合长度作为标准,因此可以有效抵抗插入垃圾文件的干扰.

4 实验与分析

本节主要介绍实验的数据来源和实验环境,并根据本文提出的相似性检测方法进行重打包应用和嵌入已知恶意代码检测实验.

$$sim(app\_files_1,app\_files_2)=L(app\_files_1,app\_files_2)=\frac{|app\_files_1\cap app\_files_2|}{\min(|app\_files_1|,|app\_files_2|)},\quad(1)$$

$$sim(app_1,app_2)=sim(app\_files_1,app\_files_2)=\frac{|image\_fea_1\cap image\_fea|}{\min(|image\_fea_1|,|image\_fea|)}\times\alpha+\frac{|sound\_fea_1\cap sound\_fea|}{\min(|sound\_fea_1|,|sound\_fea|)}\times\beta+\frac{|layout\_fea_1\cap layout\_fea|}{\min(|layout\_fea_1|,|layout\_fea|)}\times\gamma.\quad(2)$$

4.1 实验数据

本文中使用的恶意代码样本来自文献[25]提供的恶意代码样本库. 大规模实验 Android 应用数据来自第三方 Android 应用市场——安智市场,应用收集时间为 2012 年 11 月至 12 月之间,共收集 5.9 万个应用,全部数据集合大小为 268 GB,提取的特征大小为 1.98 GB.

图 5 是实验数据中应用按照包含的文件数量分布,可以看到大部分应用的文件数量在 10～200 之间,但是仍有 2 000 多个应用的文件数量超过了 1 000;图 6～8 分别是实验数据中应用包含的图片、音频、布局文件数量分布,如图 6 所示,其中大部分应用图片文件数量在 10～100 之间,图片分布规律和应用包含的总文件个数趋势基本一致;从图 7 可以看出,包含音频文件的应用中,有 20% 的应用仅含有 1 个音频文件,18% 的应用包含 9 个音频文件;另外,如图 8 所示,90% 的应用包含的布局文件数量在 30 个之内.

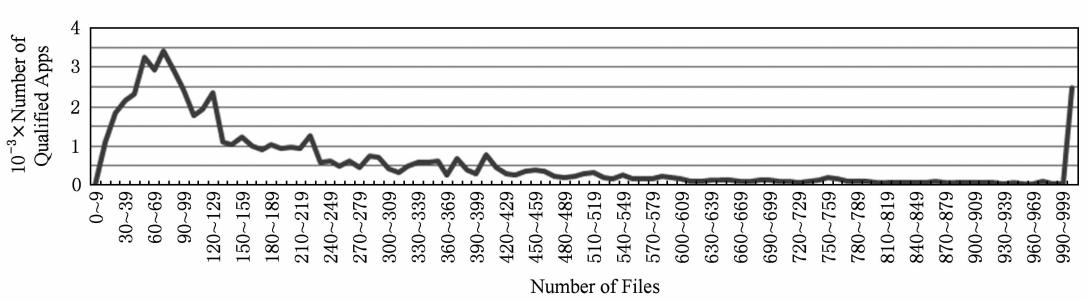


Fig. 5 The number of files statistics in experimental collection.

图 5 实验数据中的文件数量分布

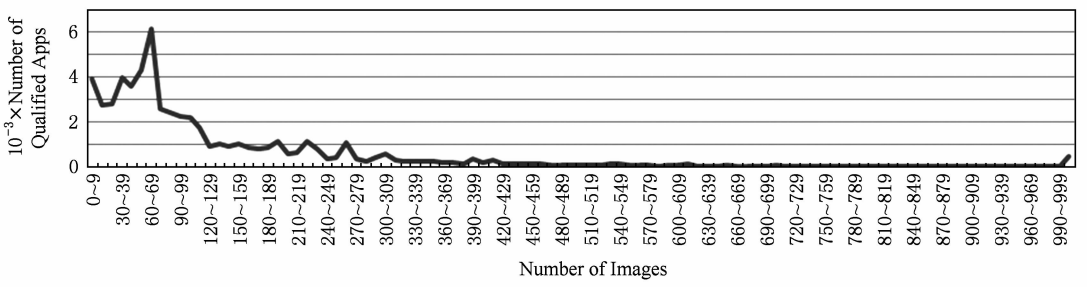


Fig. 6 The number of images statistics in experimental collection.

图 6 实验数据中的图片文件数量分布

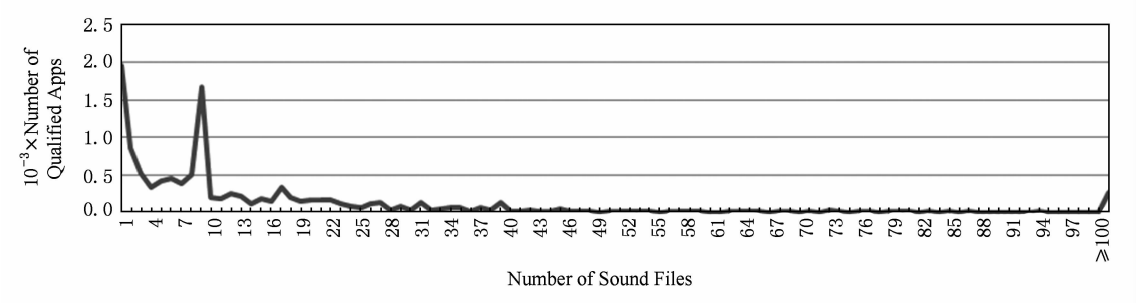


Fig. 7 The number of sound files statistics in experimental collection.

图 7 实验数据中的音频文件数量分布

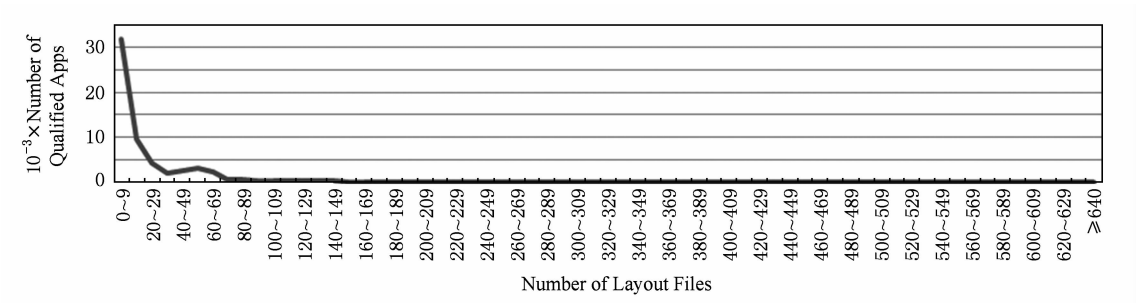


Fig. 8 The number of layout files statistics in experimental collection.

图 8 实验数据中的布局文件数量分布

通过分析实验集合的特征,可以发现本文采用的特征文件仅占应用全部文件的一小部分,所以 268 GB 的原始数据提取后的特征数据为 1.98 GB,大大缩减了特征文件的数量,保证本文提出的方法可以有效处理海量数据.

4.2 实验环境

本文实现了基于文件内容特征的大规模 Android 应用相似性检测原型系统,实验环境为 DELL Optex380,CPU 为 Intel® Core™ 2 Duo E7500 2.93 GHz,4 GB 内存,操作系统版本为 Win7 32b 旗舰版.

4.3 实验结果

本节首先对实验集合中的 Android 应用内的文件属性进行统计,根据统计特征选择合适的特征文件、图片压缩率以及相似性计算算法.然后部分通过实验讲述如何使用相似性检测系统来检测重打包应用和已知恶意代码.

4.3.1 实验数据统计

为了选出合适的特征文件,本文首先对 Android 应用内的文件类型进行统计,发现 Android 应用内的文件类型有 2 000 多种,且一种类型有多种后缀名,如音频文件后缀包括 mp3,mp4,wav,ogg,因此,对同样类型的后缀名进行合并,其中图片文件用 *image* 表示,描述为 *image*={gif;png;jpg},音频文件用 *sound* 表示,描述为 *sound*={mp3;mp4;wav;ogg},然后根据合并后的文件类型进行统计.如图 9 所示(图 9 中取百分比排名前 25 的文件类型进行展示).

结合文件内容对图 9 进行分析可知,mf,sf,rsa 文件为 Android 应用的签名文件,arsc 文件为

Fig. 9 The rate of the specified type of file in experimental collection.

图 9 实验数据集合中包含指定类型文件的应用比例

Android 的资源索引文件,属于特定用途文件,不满足可度量性,无法作为特征文件;dex 文件为 Android 的可执行字节码文件,基于行为特征的相似性检测方法即从该文件中提取特征,由于行为特征容易受到代码混淆的影响,不选取该文件作为特征文件.除去以上文件类型外,*layout*,*image*,*sound* 在符合条件的文件类型中出现频率排名前 3,因此选取图片文件、音频文件和布局文件为特征文件.

为了优化图片内容特征提取算法,对实验数据集合中 Android 应用内的图片分辨率进行分析,如图 10 所示(取排名前 30).可以发现 40×40 分辨率的图片比例最高,因此将图片缩放的分辨率设为 40×40.

最后,为了选取合适的相似度计算算法,选取一组相似应用,对相似应用内的 3 类特征文件进行信息统计,如图 11 所示,相似的应用内包含的 3 类特征文件数量不一,每种特征文件在应用内的数量差别较大,提取到的文件内容特征字符串集合包含的元素数量不一致,因此不适合使用欧氏距离和杰卡德距离衡量其相似度.

Fig. 10 The rate of the image resolution in experimental collection.

图 10 Android 应用内的图片分辨率分布

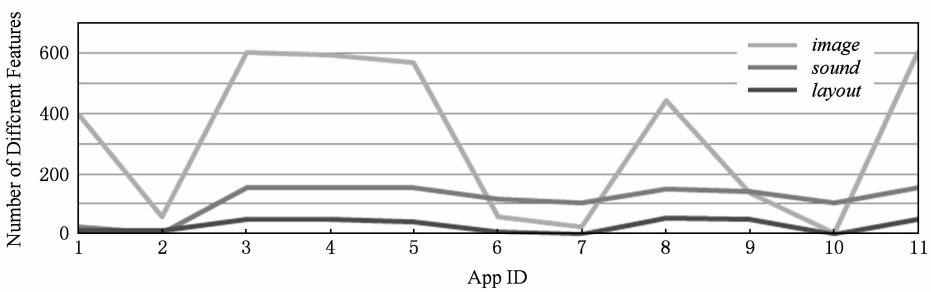


Fig. 11 The number of features in similar applications.

图 11 相似应用包含的特征文件数量

通过对实验集合内的 Android 应用文件属性进行统计分析,选取图片文件、音频文件和布局文件作为特征文件,并确定图片文件合适的缩放标准,以及对比得到了最适合的相似度计算算法.

4.3.2 重打包应用检测

Android 应用很容易被反编译,攻击者可以利用现有的工具[5]轻松对应用进行修改、嵌入广告库或者恶意代码,重新编译成可执行程序发布在应用市场.

选取水果忍者为标准检测重打包应用.水果忍者是一款著名的游戏应用,由澳大利亚布里斯班的 Halfbrick 工作室开发[26],从 Halfbrick 官方网站下载了水果忍者 Android 安装包,使用基于文件的相似性检测系统对样本进行检测.图 12 是 3 种文件内容特征检测到的相似度大于 90% 的和加权相似度

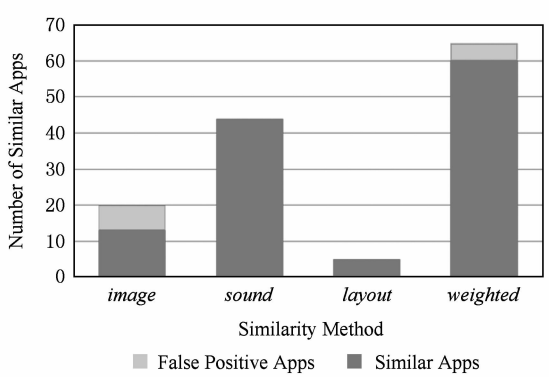


Fig. 12 The result of similar application of Fruit Nijna with different features.

图 12 不同特征检测到的水果忍者相似应用数量

大于 60%的应用数量. 图 12 中深色表示经过人工验证是水果忍者相似应用的应用数量,浅色表示验证后不是水果忍者相似应用的应用数量.

可以看出,基于音频特征的相似性检测误报率为 0,但是有一定的漏报率;基于布局特征的相似性检测误报率低,但是漏报率较高;基于图片特征的相似性检测,误报率和漏报率均较高,而加权相似性的检测效果最好,误报率仅有 7%. 实验表明,经过学习的加权相似性度量方法效果最好.

4.3.3 恶意应用检测

为了验证本文提出的系统检测已知恶意代码的能力,选取了 3 类恶意应用,对其进行特征提取,这 3 类恶意应用均属于常见的恶意代码族,分别是 GoldDream,Kongfu4 和 Kongfu1. 根据每类恶意应用的文件内容特征对实验集合中的应用进行相似性检测. 表 2 是系统检测到的相似应用结果:

Table 2 The Number of Malware in Similar Applications

表 2 检测到的已知恶意应用相似应用数量

Known Malware	Number of Similar Applications	Number of Malware in Similar Applications
KongFu4	51	48
KongFu1	12	7
GoldDream	20	14
Total	83	69

使用金山毒霸对检测到的相似应用进行扫描,发现这些应用中大部分是包含恶意代码的应用. 对包含恶意代码的相似应用进行进一步分析,发现其中大量应用是同一应用的重打包. 实验结果表明,系统可以有效地根据已知恶意代码检测到包含恶意代码的应用.

4.3.4 抗混淆测试

本文提出抗混淆的大规模 Android 应用相似性检测方法,是基于 3 类特征文件内容特征检测应用相似性. 而文件相似性检测容易受到插入无用文件和改变文件目录结构的干扰,针对这 2 种混淆方法设计实验,测试系统的抗混淆性.

本文选取一组相似应用,用其中一个应用作为标准,通过本文的系统计算其他应用的相似性. 相似应用总数为 21 个,除去作为标准的应用,余下 20 个为待测应用. 测试环境包括未混淆、插入无用垃圾文件、插入垃圾特征文件、改变文件目录结构 4 种. 实验结果如表 3 所示.

从表 3 可以看出,本文实现的系统完全不受插入无用垃圾文件和改变文件结构目录的影响,同时能较好地抵抗插入垃圾特征文件的干扰. 进一步分析发现,2 个漏报应用和标准应用的特征文件数目差别较大(标准应用内包含 500 个左右图片文件,漏报应用内仅含有 50 个左右,其他 2 种特征文件类似),通过插入垃圾特征文件,使得漏报应用内的特征文件数量和标准应用内的数量基本一样,进而导致基于内容的相似度计算结果出现较大偏差.

Table 3 Anti-Obfuscation Test

表 3 抗混淆测试

Items	Number of Similar Application
No Obfuscation	20
Insert Useless Files	20
Insert Repressive Files	18
Change Directory Structure	20

4.3.5 确定阈值 T

采用本文提出的相似性计算系统,计算出实验集合和指定应用的相似性,然后通过对比阈值 T 来识别重打包应用. 前文的实验是人工确定阈值,基本思路是对比不同阈值的效果,选择最优的阈值.

图 13 是根据加权相似性计算水果忍者应用的相似性后,设定不同阈值得到的重打包应用结果. 图中浅色表示人工确认误报的应用数量.

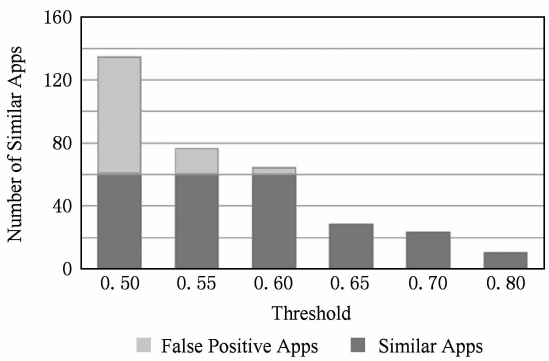


Fig. 13 The result of different threshold.

图 13 不同阈值 T 的重打包应用结果

如图 13 所示,阈值 T 在不同值时,检测重打包应用的有效性和准确性随之发生变化. 在阈值 $T=0.6$ 时检测效果最好,所以,第 4 节的实验加权相似性阈值设置为 0.6.

4.3.6 系统运行效率

本文的相似性比较是基于应用文件特征的相似性比较. 同类的研究有 Li[17] 提出的 DStruct,表 4 是本文的系统和 DStruct 的运行时间对比. 值得注意的是本文实验的系统为普通的台式机,配置较低,

如果提高 CPU 和内存等配置,运算时间会进一步缩短.

Table 4 System Efficiency

表 4 运算效率对比

Algorithm	Data Scale	t/min
DStruct	59 000	≈150
Ours	61 200	≈45

从表 4 可以看出,本文的系统运算速度远远优于同样基于 Android 应用文件特征的 Dstruct.

4.4 漏报和误报

从图 12 可以看出,3 种文件内容特征的误报率和漏报率差别较大,除了选取的标准样本差异以外,还有多个因素影响文件内容特征相似性检测的漏报和误报. 误报受以下 2 个因素影响:

1) 同一个公司发布的不同产品,可能会采用相同的图片、音频和类似的布局,容易导致相似性检测误报,如水果忍者应用的相似性检测结果中,一部分误报是 Halfbrick 开发的另一款游戏 Jetpack;

2) 应用抄袭是导致误报的另一个重要原因,开发者倾向于模仿体验优秀的应用,采用类似的布局、图片和音效,以此提高应用质量. 误报主要由开发者正常的开发行为导致,而漏报主要受到恶意开发者的误导影响. 恶意开发者为了躲避相似应用检测,先是采用代码混淆,使得基于行为特征的相似性检测困难重重,然后还会对应用的文件进行修改,包括文件结构、文件数量等,这些修改会导致基于文件内容特征的相似性检测失效.

4.5 局限性分析

根据本文提出的方案,开发者可以通过 2 种手段来混淆:1)开发者可以有意识地修改图片、音频和布局文件,以达到改变应用文件内容特征,进而避免相似性检测;2)开发者可以通过代码实现布局,而非通过 XML 文件来布局,进而避免基于布局文件的检测. 但是,对于以上手段,可以采用不同的技术来进行反混淆:1)目前图片、音频和布局文件的相似性检测有更完善的技术,图片相似性检测算法 pHash,可以检测修改内容不超过 25%相似图片;基于感知特征的音频相似性检测,可以检测一个音频片段是否包含另一个片段;而复杂的 XML 相似性检测,则可以有效地识别部分相似性的 XML 文件. 2)可以通过分析 Android 代码,提取其中关于布局的部分,对其进行分析. 这些内容是以后的研究方向.

5 总 结

本文提出了一种基于文件内容特征的抗混淆大规模 Android 应用相似性检测方法,通过对 5.9 万个 Android 应用进行统计,选择图片、音频和布局文件作为 Android 应用的特征文件,同时针对 Android 市场应用数量基数大、增速快的特点,提出了适用于大规模环境下的多种文件内容特征提取算法与相似性计算算法. 通过原型系统的实验验证,本方法可以在大量应用中有效识别重打包应用和嵌入已知恶意代码的应用,能有效地抵抗已知的文件混淆技术,并且在运算效率和准确性上优于现有解决方案.

参 考 文 献

[1] IDC. Worldwide quarterly mobile phone tracker [EB/OL]. [2013-01-20]. http://www.idc.com/getdoc.jsp?containerId=prUS24108913

[2] Engadget. Google play hits 25 billion app downloads [EB/OL]. (2012-09-16) [2013-01-20]. http://www.engadget.com/2012/09/26/google-play-hits-25-billion-app-downloads/

[3] NetQin. Global mobile phone report of first half of 2012 [EB/OL]. [2013-01-20]. http://cn.nq.com/neirong/2012shang.pdf (in Chinese)
(网秦. 2012 上半年全球手机安全报告[EB/OL]. [2013-01-20]. http://cn.nq.com/neirong/2012shang.pdf)

[4] Wisniewski R. Brut.alll @ gmail.com. android-apktool [CP/OL]. [2013-01-20]. https://code.google.com/p/android-apktool/

[5] Gruver B. jesusfreke @ jesusfreke.com. smali [CP/OL]. [2013-01-20]. http://code.google.com/p/smali/

[6] Google. DDMS [CP/OL]. [2013-01-20]. http://developer.android.com/guide/developing/debugging/ddms.html

[7] Dupuy E. JD-GUI [CP/OL]. [2013-01-20]. http://java.decompiler.free.fr/

[8] Panxiaobo. pxb1988 @ gmail.com, yyjdelete @ gmail.com. dex2jar [CP/OL]. [2013-01-20]. http://code.google.com/p/dex2jar/

[9] Shabtai A, Kanonov U, Elovici Y, et al. "Andromaly": A behavioral malware detection framework for android devices [J]. Journal of Intelligent Information System, 2012, 38(1): 161-190

[10] Xie L, Zhang X, Seifert J P, et al. pBMDS: A behavior-based malware detection system for cellphone devices [C] // Proc of the 3rd ACM Conf on Wireless Network Security. New York: ACM, 2010: 37-48

[11] Cheng J, Wong S H Y, Yang H, et al. SmartSiren: Virus detection and alert for smartphones [C] //Proc of the 5th Int Conf on Mobile Systems, Applications and Services. New York: ACM, 2007: 258-271

[12] Enck W, Gilbert P, Chun B-G, et al. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones [C] //Proc of the 9th USENIX Conf on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2010: 1-6

[13] Huang H, Zhu S, Liu P, et al. A framework for evaluating mobile app repackaging detection algorithms [G] //Trust and Trustworthy Computing. Berlin: Springer, 2013: 169-186

[14] Hanna S, Huang L, Wu E, et al. Juxtapp: A scalable system for detecting code reuse among android applications [G] //Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin: Springer, 2013: 62-81

[15] Zhou W, Zhou Y, Jiang X, et al. Detecting repackaged smartphone applications in third-party android marketplaces [C] //Proc of the 2nd ACM Conf on Data and Application Security and Privacy. New York: ACM, 2012: 317-326

[16] Desnos A, Gueguen G. anthony. desnos @ gmail. com. androiguard [CP/OL]. [2013-01-20]. https://code. google. com/p/androguard/

[17] Li S. Juxtapp and DStruct: Detection of similarity among android applications [EB/OL]. [2013-01-20]. http://www. eecs. berkeley. edu/Pubs/TechRpts/2012/EECS-2012-111. html

[18] Montgomery C. ogg [EB/OL]. [2013-01-20]. http://www. xiph. org/ogg/

[19] Boutell T. PNG [EB/OL]. [2013-01-20]. http://www. libpng. org/pub/png/png

[20] Oracle. ZipFile [EB/OL]. [2013-01-20]. http://docs. oracle. com/javase/7/docs/api/java/util/zip/ZipFile. html

[21] Kudla R. findimagedupes [CP/OL]. [2013-01-20]. http://www. ostertag. name/HowTo/findimagedupes. pl

[22] Zhang Chong, Tang Jiuyang, Xiao Weidong, et al. XML structural clustering based on cluster-core [J]. Journal of Computer Research and Development, 2011, 48(11): 2161-2176 (in Chinese)
(张翀, 唐九阳, 肖卫东, 等, 基于簇核心的 XML 结构聚类方法[J]. 计算机研究与发展, 2011, 48(11): 2161-2176)

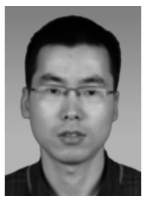
[23] Google. Layouts [EB/OL]. [2013-01-20]. http://developer. android. com/guide/topics/ui/declaring-layout. html

[24] Noll L C. FNV hash [CP/OL]. [2013-01-20]. http://www. isthe. com/chongo/tech/comp/fnv/

[25] Yajin Z, Xuxian J. Dissecting android malware: Characterization and evolution [C] //Proc of the 33rd of Security and Privacy. Piscataway, NJ: IEEE, 2012: 95-109

[26] Studios H. Fruit Ninja [CP/OL]. [2013-01-20]. http://halfbrick. com/our-games/fruit-ninja/

Jiao Sibei, born in 1986. PhD candidate. His main research interests include Android security and malicious code analysis.



Ying Lingyun, born in 1982. Assistant professor. His research interests include malware analysis and mobile security.

Yang Yi, born in 1982. Assistant professor. His research interests include malware analysis and defense.

Cheng Yao, born in 1987. PhD candidate. Her research interests include mobile security and privacy, including security and privacy iss-ues in systems and software.

Su Purui, born in 1976. Professor. His research interests are malware analysis and prevention.

Feng Dengguo, born in 1965. Professor and PhD supervisor. His research interests are crypto graphy and information security.