计算机科学
COMPUTER SCIENCE

基于界面相似度的Android仿冒应用检测研究

付雄, 聂晓晗, 王俊昌

引用本文

付雄, 聂晓晗, 王俊昌.基于界面相似度的Android仿冒应用检测研究[J]. 计算机科学, 2023, 50(6A): 220300114-7.

FU Xiong, NIE Xiaohan, WANG Junchang. Study on Android Fake Application Detection Method Based on Interface Similarity [J]. Computer Science, 2023, 50(6A): 220300114-7.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220300114

基于界面相似度的 Android 仿冒应用检测研究

付　雄　　聂晓晗　　王俊昌
南京邮电大学计算机学院　南京 210023

摘　要　随着 Android 系统的发展,仿冒应用在 Android 平台出现并逐渐活跃。混淆等技术的普及使得仿冒应用难以被传统的检测方法检测。为了有效抵抗加固技术,提出了一种基于界面相似度的 Android 仿冒应用检测方法 InfSimiDetec。首先通过自动化测试工具提取运行界面的布局信息,接着基于布局信息获取界面结构特征,然后筛选出结构特征相似的界面进行界面相似度计算,最后基于相似界面的比率进行应用相似度计算。使用含有多种类型仿冒应用程序的数据集进行实验并将所提方法与传统的检测方法进行比较,结果表明该方法的准确率为 94.11%,召回率为 96.12%,与传统检测方法相比表现出更优越的性能。

关键词:Android;界面布局;仿冒检测;自动化遍历;特征提取
中图法分类号　TP393

Study on Android Fake Application Detection Method Based on Interface Similarity

FU Xiong, NIE Xiaohan and WANG Junchang
School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China

Abstract　With the development of the Android, fake applications appear and become active on the Android platform. The popularity of obfuscation and other technologies makes it difficult for fake applications to be detected by traditional detection methods. In order to effectively resist the reinforcement technology, an Android fake application detection method (InfSimiDetec) based on interface similarity is proposed. Firstly, the layout information of the running interface is extracted by the automatic test tool. Next, the interface structural features are extracted based on the layout information. Then the interfaces with similar structural features are selected for interface similarity calculation. Finally, the application similarity calculation is carried out based on the ratio of similar interfaces. Experiments are carried out using a dataset containing multiple types of fake applications and compared with traditional detection methods. The results show that the precision rate of this method is 94.11% and the recall rate is 96.12%. Compared with traditional detection methods, this method shows better performance.

Keywords　Android, Interface layout, Detection of fake application, Automation traversal of application, Feature extraction

1　引言

随着 Android 和 IOS 等移动终端操作系统的发展,移动应用市场迸发出前所未有的活力,与此同时仿冒应用也逐渐活跃。由于缺少有效的监督和管理,层出不穷的仿冒应用不仅对应用市场和用户的财产安全产生了巨大威胁,也为监管部门带来了严峻的挑战[1]。

自互联网时代开始以来,仿冒欺诈行为就广泛分布于各大平台。作为一个存在已久的问题,仿冒欺诈受到国内外学者的广泛研究。目前主流的 Android 仿冒应用检测技术主要分为基于代码特征的检测方案和基于界面特征的检测方案[2]。

基于代码特征的检测方案需要提取应用的静态代码片段,根据片段的相似性来进行仿冒鉴别,其思路主要来自代码克隆检测技术[3-4]。Crussel 等[5]通过比较 Android 应用的程序依赖图来衡量是否存在仿冒行为,程序依赖图通过提取应用的函数调用关系来构建生成。Niu 等[6]通过计算应用的方法调用依赖图的相似性来进行仿冒检测,方法调用依赖图通过对 Android 应用反编译获得的 smali 代码来分析构造生成。Zhou 等[7]提出了 Android 重打包检测工具 DroidMoss,通过对 APK 文件进行逆向分析来提取指令序列,对指令序列执行模糊哈希计算并将其结果作为代码特征进行重打包检测。基于代码特征的仿冒应用检测方案虽然简洁高效,但是恶意开发者可以通过在源代码中插入大量无意义的垃圾代码或者使用代码混淆技术来使检测工具提取到的代码特征与原版应用截然不同,从而达到逃避检测的目的。

基于界面特征的检测方法能够有效抵抗代码混淆。Malisa 等[8]截取应用运行界面的图片,对其进行灰度处理生成应用的界面特征,通过图像识别技术进行仿冒检测。Sun 等[9]根据界面视图树分别提出基于编辑距离和基于布局哈希

基金项目:国家自然科学基金(51977113);江苏省重点研发计划(社会发展)项目(BE2017743)
This work was supported by the National Natural Science Foundation of China (51977113) and Primary Research & Development Plan (Social Development) of Jiangsu Province (BE2017743).
通信作者:付雄(fux@njupt.edu.cn)

值的仿冒应用检测方法。Zhuaniarovich 等[10]通过计算应用间资源文件的相似性来检测应用的重打包行为，提出了 Android 重打包应用检测工具 FSquaDRA。Zhu 等[11]根据应用的界面控件特征进行仿冒检测。

本文提出了一种基于界面相似度的 Android 仿冒应用检测方法 InfSimiDetec。该方法使用自动化测试工具 Appium 遍历启动应用中的 Activity，对每个 Activity 的运行界面进行截图，提取界面布局信息并生成界面结构特征向量，最后根据界面截图和特征向量计算应用间的相似度进行仿冒鉴别。

2 基于界面相似度的 Android 仿冒应用检测方法

2.1 方法概述

InfSimiDetec 方法的总体架构如图 1 所示，该方法由预处理、布局信息提取、结构特征提取和相似度计算 4 个部分组成。

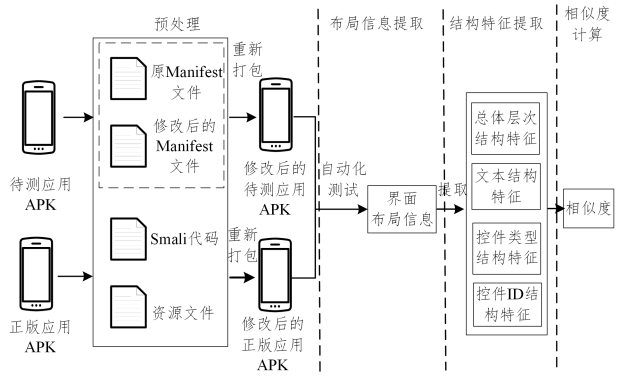

图 1 InfSimiDetec 方法架构图

Fig. 1 Framework of InfSimiDetec method

首先，需要对待测应用和正版应用的 APK 进行预处理。将 APK 文件反编译，对反编译生成的 AndroidManifest. xml 文件中的内容进行修改，为其中声明的 Activity 组件添加子节点，重新打包签名生成新的待测应用 APK 并在 Android 模拟器上安装。

然后，通过自动化测试工具 Appium 遍历启动 AndroidManifest. xml 中声明的非第三方库 Activity 组件，使用 Appium提供的 API 提取界面布局信息，以 XML 文件的方式存储并对应用界面进行截图。

接着，从界面布局信息中提取每个控件的坐标属性信息，根据控件坐标上下界对界面布局进行层次切割生成界面的总体层次结构特征。依次抽取各层中控件的文本、类型和 ID 属性，生成该界面的文本结构特征、控件类型结构特征和控件 ID 结构特征。

最后，通过总体层次结构特征和界面截图的空间包络特征(GIST)筛选出待测应用和正版应用中相似的界面并计算它们之间的相似度。根据相似界面的比率计算正版应用和待测应用间的相似度，根据其结果判断待测应用是否为仿冒应用。

2.2 预处理

Activity 主要负责显示应用的运行界面并提供交互功能，需要在 AndroidManifest. xml 中声明才能成功启动。当声明的 Activity 组件具有“android. intent. category. LAUNCHER”属性时，Appium 可以根据名称直接启动该 Activity。因此，需要对应用的 APK 文件进行预处理。通过逆向工具 Apktool 反编译 APK 文件，Apktool 会在指定路径下生成一个以 APK 命名的文件夹，其中保存了本次反编译的结果。在生成的文件中找到 AndroidManifest. xml 并进行修改，对于其中声明的所有 Activity 组件，添加 intent-filter 节点及 action 和 category 子节点。

对 AndroidManifest. xml 文件预处理的示例如图 2 所示。intent-filter 为意图过滤器，主要用于过滤隐式意图，Android 系统会根据配置的 intent-filter 进行匹配测试来寻找可以响应用户特定操作的组件或服务。它实行“白名单”管理，其中只列出能够接受意图的组件。action 表示动作测试，android:name 属性用来指定响应的动作名，动作名必须是独一无二的字符串。category 表示类别测试，android:name 属性用来指定组件响应的环境，其值必须为“android. intent. category. LAUNCHER”，表示该 Activity 可以作为应用的入口 Activity 启动。

```
〈activity android:name="cn. caocaokeji. menu. module. charge. ChargeActivity"〉
  〈intent-filter〉
    〈action android:name="android. intent. action. 1"/〉
    〈category android:name="android. intent. category. LAUNCHER"/〉
    〈/intent-filter〉
〈/activity〉
〈activity android:name="cn. caocaokeji. menu. module. trip. MyTripActivity"〉
  〈intent-filter〉
    〈action android:name="android. intent. action. 2"/〉
    〈category android:name="android. intent. category. LAUNCHER"/〉
  〈/intent-filter〉
〈/activity〉
```

图 2 AndroidManifest. xml 文件预处理示例

Fig. 2 AndroidManifest. xml file preprocessing example

对 AndroidManifest. xml 文件修改完成后需要用 Apktool 重新打包。Apktool 重打包生成的文件是未签名的 APK，无法在模拟器或手机上安装。因此需要通过签名工具 Signapk 重新签名生成新的可运行的应用程序。

2.3 界面布局信息提取

InfSimiDetec 使用自动化测试技术遍历启动应用的运行界面，提取界面布局信息。传统的自动化测试方法通常会遍历应用所有的运行界面，遍历过程中会耗费大量时间用于搜索下一个界面的入口点。Liu 等[12]的研究表明，传统的自动化测试方法平均需要几个小时才能遍历完一个应用所有的界面，效率较低。因此，我们基于应用在 AndroidManifest. xml 中声明的 Activity 组件实现自动化测试，读取应用在 AndroidManifest. xml 文件中声明的 Activity 组件的名称并通过 Appium 直接启动。这种方法虽然不能启动所有的 Activity，无法完整提取应用所有的界面布局信息，但其节省了传统自动化方法中在界面入口点搜索的分析时间，复杂度低，提取速度快，在 UI 覆盖范围和自动化测试性能之间实现平衡。

Appium 启动应用指定的 Activity 需要传入以下参数：platformName 表示操作系统名称；appPackage 表示包名；appActivity 表示待启动的 Activity 名；deviceName 表示设备名；udid 表示物理设备或模拟器的唯一设备标识符。将以上参数填写完毕后 Appium 将会根据参数启动指定设备中指定

应用的 Activity，并将 Activity 的运行界面和界面布局信息展示出来。

为防止第三方库的 Activity 组件对检测结果造成影响，根据一份第三方库白名单[13]顺序遍历启动该应用所有的非第三方 Activity。将每个 Activity 的布局信息以 XML 的格式存储在本地并对运行界面进行截图。对于每一个布局信息，都是由一个布局根节点及其下面的多个控件或布局子节点构成。在图 3 所示的布局信息示例中，层布局 FrameLayout 节点下包含一个 TextView 控件节点。

```
〈?xml version='1.0' encoding='UTF-8' standalone='yes'?〉
〈hierarchy index="0" class="hierarchy" rotation="0" width="720"
height="1 280"〉
  〈android.widget.FrameLayout index="0" package="cn.caocaokeji.user"
  class="android.widget.FrameLayout" text="" checkable="false"
  checked="false"
  clickable="false" enabled="true" focusable="false" focused="false"
  long-clickable="false" password="false" scrollable="false" selected=
  "false"
  bounds="[0,0][720,1 280]" displayed="true"〉
    〈android.widget.TextView index="1" package="cn.caocaokeji.user"
     class="android.widget.TextView" text="行程" checkable="false"
     checked="false" clickable="false" enabled="true" focusable=
     "false"
     focused="false" long-clickable="false" password="false" scrollable
     ="false"
     selected="false" bounds="[324,67][396,116]" displayed="true"/〉
  〈/android.widget.FrameLayout〉
〈/hierarchy〉
```

图 3　布局信息示例

Fig. 3　Example of layout information

对于每一个控件节点，我们主要关注以下属性：text 属性表示该控件在界面上显示的文本内容；class 属性表示控件的类型；bounds 属性表示控件在屏幕中的位置，Android 会根据 bounds 属性将控件绘制在界面中的一块矩形区域内，$[x_1, y_1]$ 表示矩形区域左上角的坐标，$[x_2, y_2]$ 表示右下角坐标，由于屏幕的横纵坐标分别按照从左到右、从上到下的顺序递增，因此，$[x_1, y_1]$ 表示横纵坐标上界，$[x_2, y_2]$ 表示横纵坐标下界；resource-id 属性表示当前控件的资源 id，若当前控件资源 id 非空，则其值唯一。

2.4　结构特征提取

在提取界面结构特征之前，需要对界面布局信息进行预处理。通过使用一种平行切割方法根据界面布局信息将界面划分为若干个矩形区域，这些矩形区域互相平行且不相交。在划分矩形区域时需要遵守以下准则：1）每个控件都必须处于矩形区域中且只能处于 1 个矩形区域内；2）每个矩形区域都无法再被划分成更小的矩形区域。图 4 给出了界面切割的正确示例，界面从上到下被划分为 6 个平行不相交的矩形区域。

将预处理生成的矩形区域定义为层。经过实践发现，大量应用以列表的方式将内容展示给用户，列表中各层控件的组成结构是一致的，在图 4 所示的历史订单界面中，订单列表被划分为层 3－层 6，其中的每一层都是由 4 个 ImageView 和 6 个 TextView 控件组成的。恶意开发者经常会修改这种列表层次结构来达到仿冒欺诈的目的，如减少层的高度使列表能一次显示更多的层。因此，对于相邻的含有相同数量和类型的控件的层，将其合并，并将合并生成的层定义为重叠层。如图 4 所示，层 3－层 6 可以合并为一个重叠层。最后，定义独立层表示所有的重叠层和普通层。在图 4 中，历史订单界面的所有层经过合并最终生成 3 个独立层。

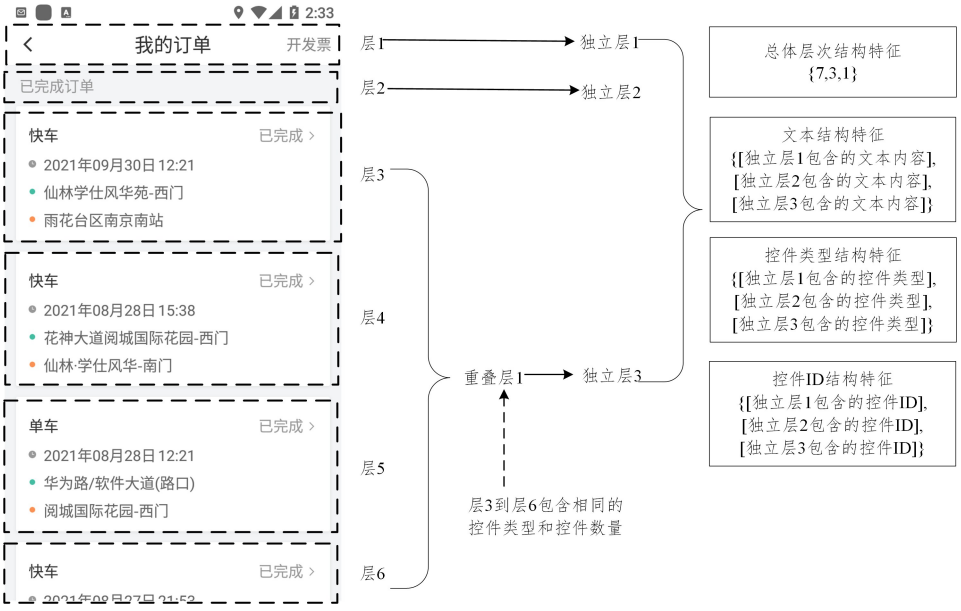

图 4　结构特征提取示例

Fig. 4　Structural feature extraction example

（1）层次结构特征

由于 Appium 提取的 XML 格式的布局信息中的控件是按照从外到内嵌套排序的，直接使用该布局信息进行层的划分合并和结构特征的提取操作会产生大量重复的坐标比较，从而降低运行效率，因此需要根据坐标信息对控件从上到下重新排序。需要注意的是，考虑到 Android 应用在竖屏和横屏时的界面布局有所差异，对于竖屏和横屏情况下的界面布局信息，我们分别基于控件的纵坐标和横坐标进行排序。

竖屏界面层次结构特征提取算法伪代码算法 1 所示。

算法 1 层次结构特征提取算法

输入：界面的控件信息 widgetList

输出：独立层信息 layer 和重叠层信息 overlapLayer

```
1. m,n=0
2. upperBound=widgetList[0][y2]
3. lowerBound=widgetList[0][y1]
4. For i=0,1,2,…,widigetList.length-1 do
5.     If(widigetList[i][y2]<=upperBound and widgetlist[i]
       [y1]>=layerLowerBound )
6.     layer[m].add(widgetList[i])
7. Else
8.     upperBound=widgetList[i][y2]
9.     lowerBound=widgetList[i][y1]
10.    layer[++m].add(widgetList[i])
11. End if
12. End for
13. For j=0,1,2,…,m do
14.    If(layer[j+1].equals(layer[j]))
15.      overlapLayer[n].add(layer[j])
16.      overlapLayer[n].add(layer[j+1])
17.      For k=j+2,j+3,…,m do
18.        If(layer[k].equals(layer[j]))
19.          overlapLayer[n].add(layer[k])
20.        Else
21.          layer.remove(overlapLayer[n++])
22.          Break
23.        End if
24.      End for
25.   End if
26. End for
27. Return overlapLayer,layer
```

该算法遍历所有控件划分为层的时间复杂度为 $O(n)$，将相邻层合并生成重叠层的时间复杂度为 $O(n^2)$，因此算法的整体时间复杂度为 $O(n^2)$。该算法创建了两个集合，分别用于保存未合并的普遍层和重叠层的信息，因此算法的整体空间复杂度为 $O(n)$。

定义向量 $\boldsymbol{f}_1=\{L_1,L_2,L_3\}$ 为层次结构特征。其中，L_1 表示层的数量，L_2 表示独立层的数量，L_3 表示重叠层的数量。如图 4 所示的历史订单界面的层次结构特征为 $\{7,3,1\}$。

(2)文本结构特征

文本结构特征由每个独立层中控件的 text 和 content-desc 属性中的内容获得，定义向量 $\boldsymbol{f}_2=\{T_1,T_2,\cdots,T_n\}$ 为文本结构特征，其中，T_i 表示第 i 个独立层中所有控件非空的 text 和 content-desc 属性值的集合。若第 m 个独立层为重叠层，则 T_m 为重叠层中所有层的控件的非空的 text 和 content-desc 属性值的并集。

(3)控件类型结构特征

控件类型结构特征由每个独立层中控件的 class 属性中的内容获得，定义向量 $\boldsymbol{f}_3=\{C_1,C_2,\cdots,C_n\}$ 为控件类型结构特征，其中，C_i 表示第 i 个独立层中所有控件的 class 属性值的集合。若第个独立层为重叠层，则 C_m 为重叠层中所有层的控件的 class 属性值的并集。

(4)控件 ID 结构特征

控件 ID 结构特征由每个独立层中控件的 resource-id 属性中的内容获得，定义向量 $\boldsymbol{f}_4=\{R_1,R_2,\cdots,R_n\}$ 为控件 ID 结构特征，其中，R_i 表示第 i 个独立层中所有控件的非空的 resource-id 属性值的集合。若第 m 个独立层为重叠层，则 C_m 为重叠层中所有层的控件的非空的 resource-id 属性值的并集。

2.5 相似度计算

首先通过结构特征向量计算应用间 Activity 界面的相似度，接着通过相似 Activity 的比率计算应用的相似度，最后根据应用相似度的阈值判断待测应用是否为仿冒应用。

(1)Activity 相似度对比

实验发现，大多数应用都包含至少 10 个 Activity，每个 Activity 至少含有 5 层，且大部分 Activity 间视觉效果相差较大。若直接对所有的 Activity 进行两两比较，会产生大量无意义的操作，其过程是十分耗时的。

针对上述问题，我们使用基于 Activity 层次结构特征和界面图像特征的优先比较方法，首先取出两个 Activity 的界面截图，使用 LMgist 算法分别计算两个界面截图的全局特征信息，即自然度(Degree of Naturalness)、开放度(Degree of Openness)、粗糙度(Degree of Roughness)、膨胀度(Degree of Expansion)和险峻度(Degree of Ruggedness)，由此生成 GIST 向量。接着对两个 Activity 的层次结构特征和 GIST 特征进行比较，如果二者的层次结构特征和 GIST 特征相差极大，则认定其不相似，否则需要进一步比较文本结构特征、控件类型结构特征和控件 ID 结构特征。

实验发现，两个在视觉方面差异较大的 Activity 的主要区别体现在以下几个方面：1)两者在层的数量上差距较大，即 Activity A 和 Activity B 的层次结构特征中的层的数量 L_{1A} 和 L_{1B} 差距较大；2)两者在重叠层的数量方面差距较大，即 Activity A 和 Activity B 的层次结构特征中的重叠层的数量 L_{3A} 和 L_{3B} 差距较大；3)两者的 GIST 特征余弦相似距离 cos AB 的值较小。

因此，由式(1)可以求得 Activity A 和 Activity B 的界面结构相似值 $Similarity_{A,B}$：

$$Similarity_{A,B}=(|L_{1A}-L_{1B}|>V_{\text{thres }1}\ \text{or}\ |L_{3A}-L_{3B}|>V_{\text{thres }2})\ \text{and}\ \cos AB<V_{\text{thres }3} \tag{1}$$

其中，阈值 $V_{\text{thres }1}$ 设定为 3，$V_{\text{thres }2}$ 设定为 1，$V_{\text{thres }3}$ 设定为 0.8，若 $Similarity_{A,B}$ 的值为 false，表明 Activity A 和 Activity B 相似，否则二者不相似，需要从待测应用和正版应用中挑选下一组 Activity 进行比较。

对于由公式(1)筛选出的两个相似的 Activity，进一步根据它们的文本结构特征、控件类型结构特征和控件 ID 结构特征来计算二者间的相似度，通过特征中相似向量的比率来衡量两个结构特征的相似度。取 Activity A 和 Activity B 的文本特征向量 $\boldsymbol{f}_{A,2}$，$\boldsymbol{f}_{B,2}$，控件类型特征向量 $\boldsymbol{f}_{A,3}$，$\boldsymbol{f}_{B,3}$ 和控件 ID 特征向量 $\boldsymbol{f}_{A,4}$，$\boldsymbol{f}_{B,4}$，根据公式(2)分别计算 $\boldsymbol{f}_{A,2}$ 与 $\boldsymbol{f}_{B,2}$，$\boldsymbol{f}_{A,3}$ 与 $\boldsymbol{f}_{B,3}$，$\boldsymbol{f}_{A,4}$ 与 $\boldsymbol{f}_{B,4}$ 的特征向量相似度 $SIM(\boldsymbol{f}_{A,2},\boldsymbol{f}_{B,2})$，$SIM(\boldsymbol{f}_{A,3},\boldsymbol{f}_{B,3})$，$SIM(\boldsymbol{f}_{A,4},\boldsymbol{f}_{B,4})$：

$$SIM(\boldsymbol{f}_{A,x},\boldsymbol{f}_{B,x})=\frac{\sum_{1}^{m,n}2SIM(C_{A,i},C_{B,j})}{m+n},x=1,2,3 \tag{2}$$

其中，m 为 Activity A 中独立层的数量，n 为 Activity B 中独立层的数量，$SIM(C_{A,i},C_{B,j})$ 为特征向量中向量元素间

的对比函数，即：

$$SIM(C_{A,i},C_{B,j})=\begin{cases}1, & \text{if } C_{A,i}\subset C_{B,j} \text{ or } C_{A,i}\supset C_{B,j}\\ 0, & \text{if not}\end{cases} \tag{3}$$

其中，$C_{A,i}$和$C_{B,j}$分别为 Activity A 和 Activity B 特征向量中的第 i 和第 j 个元素。考虑到恶意开发者存在通过调整控件属性值中的内容来规避检测的可能，因此判定当$C_{A,i}$和$C_{B,j}$之间存在包含关系时为相似，否则为不相似。

基于文本结构特征相似度 $SIM(\boldsymbol{f}_{A,2},\boldsymbol{f}_{B,2})$、控件类型结构特征相似度 $SIM(\boldsymbol{f}_{A,3},\boldsymbol{f}_{B,3})$和控件 ID 结构特征相似度 $SIM(f_{A,4},f_{B,4})$的值，可以进一步求得两个 Activity 的相似度 $SC(A,B)$，即：

$$SC(A,B)=\begin{cases}1, & \text{if } SIM(\boldsymbol{f}_{A,2},\boldsymbol{f}_{B,2})\geqslant 0.66 \text{ or } SIM(\boldsymbol{f}_{A,3},\boldsymbol{f}_{B,3})\geqslant 0.7 \text{ or } SIM(\boldsymbol{f}_{A,4},\boldsymbol{f}_{B,4})\geqslant 0.7\\ 0, & \text{if not}\end{cases} \tag{4}$$

若 $SC(A,B)$的值为 1，则 Activity A 和 Activity B 相似，否则二者不相似。

Activity A 和 Activity B 特征向量相似度计算的算法伪代码如算法 2 所示。

算法 2　特征向量相似度计算算法

输入：Activity A 和 Activity B 的结构特征向量 **VectorA** 和 **VectorB**

输出：特征向量相似度

```
1. count=0
2. For i=0,1,2,…,VectorA.length-1 do
3.    For j=0,1,2,…,VectorB.length-1 do
4.       flag=true
5.       m,n=0
6.       Set1=VectorA[i].length > VectorB[j].length ? VectorA[i]:
         VectorB[j]
7.       Set2=VectorA[i].length > VectorB[j].length ? VectorB[j]:
         VectorA[i]
8.       For m=0,1,2,…,Set2.length-1 do
9.          For n=0,1,2,…,Set1.length-1 do
10.            If(Set2[m]==Set1[n]) break
11.            End if
12.         End for
13.         If(n==Set1.length)
14.            flag=false
15.         End if
16.      End for
17.      If(flag==true) count++
18.      End if
19.   End for
20. End for
21. Return(2 * count)/(VectorA.length+VectorB.length)
```

该算法需要依次取出两个向量中的元素进行两两比对，时间复杂度为 $O(n^2)$，判断两向量是否具有包含关系的时间复杂度为 $O(n^2)$，因此该算法的整体时间复杂度为 $O(n^4)$。算法在执行过程中需要创建常数个变量用于保存中间计算的结果，因此该算法的空间复杂度为 $O(1)$。

(2)应用相似度计算

令 Activity 序列 $V=\{P_{A1},P_{A2},P_{A3},\cdots,P_{An}\}$表示含有 n 个 Activity 的应用 P，$U=\{Q_{B1},Q_{B2},Q_{B3},\cdots,Q_{Bm}\}$表示含有 m 个 Activity 的应用 Q，由式(5)计算应用 P 和应用 Q 间的相似度 $SIMAPP(APP_P,APP_Q)$，即：

$$SIMAPP(APP_P,APP_Q)=\frac{\sum_{1}^{m,n}SC(U,V)+SC(V,U)}{m+n}\in[0,1] \tag{5}$$

正反计算两个应用中相似 Activity 的数目，将求得的权重值作为应用 P 和应用 Q 的相似度。基于 3.2 节中的实验内容，将阈值设置为 0.63，即当 $SIMAPP(P,Q)>0.63$ 时，应用 P 与应用 Q 为仿冒应用对。

3　实验结果与分析

本节主要从准确率和运行效率两方面对 InfSimiDetec 方法的性能进行评估，并将其与传统的 Android 仿冒检测工具进行比较。实验表明本节提出的方法具有较高的效率和准确率。

3.1　数据集

实验选用的数据集如表 1 所列，共有 215 个应用，4 720 个仿冒应用对，由多个渠道收集而来的不同种类应用组成。该数据集由以下子数据集组成。1) AndroZoo：该子数据集中的应用取自 Luxembourg 大学公开的恶意应用和重打包应用集[14]，这些应用来自 Google Play 等各大应用市场；2) Malicious Apps：该子数据集中的应用取自 Arp 等[15]研究公开的恶意样本库；3) Application Market：该子数据集中的应用取自应用市场标记的各种视觉效果相似的疑似仿冒应用。

表 1　实验数据集

Table 1　Experimental datasets

数据集	应用数量	仿冒应用对
AndroZoo	97	1892
Malicious Apps	43	53
Application Market	75	2775
Total	215	4720

3.2　性能评估

(1)仿冒欺诈行为检测准确率

我们根据数据集中的 215 个应用构造了 28 760 个应用对，使用精确率和召回率来对 InfSimiDetec 检测的准确率进行评估。图 5 给出了通过 InfSimiDetec 方法计算的所有应用对的相似度分布情况。图 6 给出了 InfSimiDetec 在不同相似度阈值下检测的误报和漏报的应用对分布情况。在阈值为 0.63 的情况下，InfSimiDetec 能检测出 4 537 个仿冒应用对，另有 284 个应用对被误报。因此 InfSimiDetec 的精确率为 94.11%，召回率为 96.12%。我们对误报和漏报的应用对进行分析发现，误报的应用对只能启动少量的 Activity 且提取出的文本信息相同；漏报的应用对中有 37 对应用的 Activity 无法正常启动，其余应用对为游戏类应用，Activity 界面在启动后黑屏，无法提取到有效的布局信息。因此，InfSimiDetec 方法能够有效检测非游戏类的仿冒应用，具有较强的通用性和普适性。

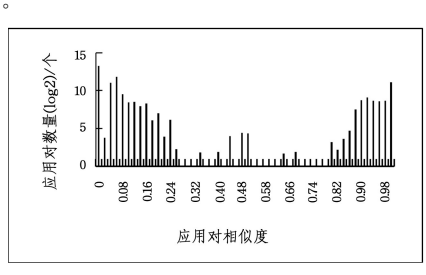

图 5　应用相似度分布

Fig. 5　Application similarity distribution

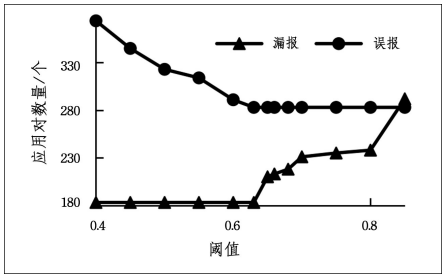

图 6 不同阈值下误报和漏报应用分布

Fig. 6 Application distribution of false positives and false negatives with different thresholds

(2)运行效率

由于 InfSimiDetec 直接通过 Appium 遍历启动所有 Activity 进行布局信息提取,因此应用的 Activity 数量决定了布局信息提取所消耗的时间。图 7 给出了通过 Appium 启动的数据集中所有应用的 Activity 数量分布,绝大多数应用成功启动的 Activity 数量在 0～40 之间。Appium 启动并提取一个 Activity 的布局信息耗时约为 10 s,提取一个应用所有 Activity 的布局信息平均耗时约 8 min。经实验估算,对 28 760 个应用对进行特征提取与相似度计算耗时约 6.2 h,每个应用平均耗时约 0.79 s,检测效率较高。

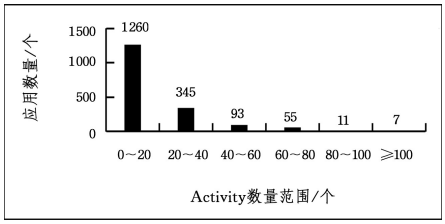

图 7 应用启动的 Activity 数量分布

Fig. 7 Distribution of the number of Activity started by application

(3)与传统仿冒欺诈检测方法的比较

本节使用 FSquaDRA[10]、SimiDroid[16]和根据 Zhu 等[11]的研究复现的检测工具 ZhuDetec 对数据集进行仿冒检测,它们分别是传统检测方法中基于资源文件、基于代码特征和基于视图布局的代表性检测方案。FSquaDRA 提取应用的文件类型生成静态特征向量进行仿冒检测;SimiDroid 在函数层面基于应用的静态代码特征使用 4 种度量方式进行仿冒检测;ZhuDetec 从应用的 UI 界面文件和图像中提取特征进行仿冒检测。根据先前的研究,我们将 FSquaDRA,SimiDroid 和 ZhuDetec 的相似度阈值分别设置为 0.8,0.9 和 0.8。

图 8－图 10 给出了 InfSimiDetec,FSquaDRA,SimiDroid 和 ZhuDetec 分别对 AndroZoo,Malicious Apps 和 Application Market 数据集中的应用进行仿冒检测的结果。对于 AndroZoo 和 Malicious App 数据集,FSquaDRA,SimiDroid 和 ZhuDetec 的检测效果较好,能够检测出其中的绝大多数仿冒应用。三者漏报的应用均为游戏应用,检测工具无法提取到有效的特征用于进行仿冒检测。对于 Application Market 数据集,FSquaDRA,SimiDroid 和 ZhuDetec 的检测效果较差。这是由于恶意开发者使用了混淆技术,对资源文件 hash 值和文件目录结构的混淆导致 FSquaDRA 无法有效检测仿冒应用;对 UI 的混淆引入了新的 Activity 和组件,使应用代码和 UI 界面产生较大变化,导致 SimiDroid 和 ZhuDetec 无法有效检测仿冒应用。InfSimiDetec 在 3 个数据集中均表现出较好的检测效果。

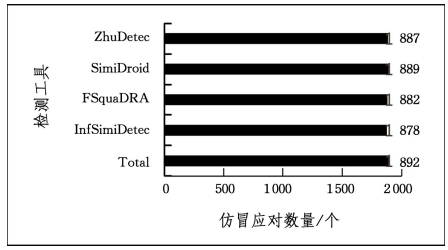

图 8 AndroZoo 数据集检测结果

Fig. 8 AndroZoo dataset detection results

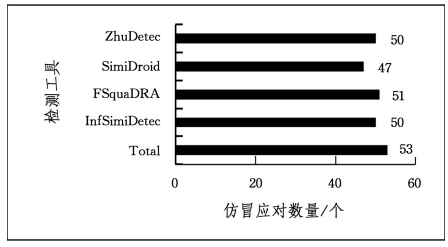

图 9 Malicious Apps 数据集检测结果

Fig. 9 Malicious Apps dataset detection results

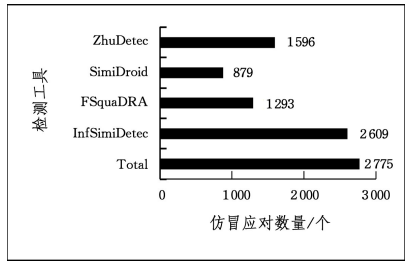

图 10 Application Market 数据集检测结果

Fig. 10 Application Market dataset detection results

综上所述,基于界面相似度的 Android 仿冒应用检测方法 InfSimiDetec 具有较高的准确率和运行效率,能够应对多种类型的仿冒欺诈行为。

结束语 本文研究了一种基于界面相似度的 Android 仿冒应用检测方法。首先介绍了方法的整体架构,将方法划分为预处理、布局信息提取、结构特征提取和相似度计算 4 部分;接着详细描述了每一部分的执行流程;最后将该方法应用于含有多种类型仿冒应用的数据集中,结合与传统检测工具的对比,证明了该方法的有效性和可行性。

后续研究应重点关注对游戏类应用的界面特征提取与仿冒检测,还应考虑应用版本差异对仿冒检测的影响。

参 考 文 献

[1] LI J L, WANG Y Z, LUO L G, et al. A Survey of Adversarial Attack Techniques for Android Malware Detection[J]. Journal of Cyber Security, 2021, 6(4): 28-43.

[2] WANG S Y, ZHANG Y S, CENG J R, et al. Overview of Android Malware Detection Methods[J]. Computer Applications and Software, 2021, 38(9): 1-9.

[3] UTKARSH S, KULDEEP K, DEEPAKUMAR G. A Study of Code Clone Detection Techniques in Software Systems[C]// Proceedings of the International Conference on Paradigms of Computing, Communication and Data Sciences. Springer, 2021: 347-359.

[4] HAMID A B,STAN J. A Data Mining Approach for Detecting Higher-Level Clones in Software[J]. IEEE Transactions on Software Engineering,2009,35(4):497-514.

[5] CRUSSEL J,GIBLER C,HAO C. Attack of the Clones:Detecting Cloned Applications on Android Markets[C]// European Symposium on Research in Computer Secu-rity. Computer Security-ESORICS,2012:37-54.

[6] NIU H,YANG T,NIU S. Clone Analysis and Detection in Android Applications[C]// International Conference on Systems and Informatics(ICSAI). IEEE,2016:520-525.

[7] ZHOU W,ZHOU Y J,JIANG X X,et al. Detecting Repackaged Smartphone Applications in Third-party Android Marketplaces [C]// ACM Conference on Data and Application Security and Privacy. ACM,2012:317-326.

[8] MALISA L,KOSTIAINEN K,CAPKUN S. Detecting Mobile Application Spoofing Attacks by Leveraging User Visual Similarity Perception[C]// ACM Conference on Data and Application Secu-rity and Privacy. ACM,2017:289-300.

[9] SUN M,LI M,LUI J C S. DroidEagle:Seamless Detection of Visually Similar Android Apps[C]// ACM Conference on Security and Privacy in Wireless and Mobile Networks. ACM,2015:1-12.

[10] ZHAUNIAROVICH Y,LEZZA A L,GADYATSKAYA O. Evaluation of Resource-Based App Repackaging Detection in Android[C]// Nordic Conference on Secure IT Systems. 2016:135-151.

[11] ZHU J,WU Z,ZHI G,et al. Appearance Similarity Evaluation for Android Applications[C]// International Conference on Advanced Computational Intelligence. IEEE,2015:323-328.

[12] LIU B,NATH S,GOVINDAN R,et al. DECAF:Detecting and Characterizing Ad Fraud in Mobile Apps[C]// USENIX Conference on Networked Systems Design and Implementation(NSDI'14). USENIX Association,2014:57-70.

[13] KAICHEN. A List of Shared Libraries and AdLibraries Used in Android Apps[EB/OL]. (2014-02-20)[2022-03-03]. http://sites. psu. edu/kaichen/2014/02/20/a-list-of-shared-libraries-and-adlibraries-used-in-android-apps/.

[14] LUXEMBOURG UNIVERSITY. AndroZoo [EB/OL]. (2019-03-21)[2022-03-03]. https:// androzoo. uni. lu/repackaging.

[15] ARP D,SPREITZENBARTH M,HUBNER M,et al. DREBIN:Effective and Explainable Detection of Android Malware in Your Pocket[C]// Network and Distributed System Security Symposium. 2014:23-38.

[16] LI L,BISSYANDE T F,KLEIN J. SimiDroid:Identifying and Explaining Similarities in Android Apps[C]// IEEE Trustcom/BigDataSE/ICESS. IEEE,2017:136-143.

FU Xiong, born in 1979, Ph. D, professor. His main research interests include cloud computing and distributed computing.