基于资源签名的 Android 应用相似性快速检测方法

张 鹏[1,2],牛少彰[1],黄如强[1]
(1. 北京邮电大学智能通信软件与多媒体北京市重点实验室,北京 100876;2. 宁夏大学信息工程学院,宁夏银川 750021)

摘 要: 由于盗版 Android 应用(Android Application,简称 APP)通常保持着与正版 APP 相似的用户体验,因此本文提出一种基于资源签名的 APP 相似性快速检测方法. 该方法将 APP 的资源签名视为字符串集合,利用计算任意一对 APP 资源签名集合的 Jaccard 系数判断两者的相似性. 为了避免遍历全部的 APP 对,该方法将 MinHash 和 LSH (Locality Sensitive Hashing)算法的思路引入其中,通过从 APP 集合中挑选候选对并对候选对进行检验的方式获得最终的检测结果. 由于挑选候选对的方式将大量相似性较低的 APP 对排除在外,因此该方法可以明显地提高 APP 相似性的检测速度. 实验结果表明,该方法的检测速度比现有方法 FSquaDRA 提高了大约 30 倍,而检测结果与 FSquaDRA 几乎完全相同.

关键词: APP 相似性; 资源签名; MinHash; LSH; Jaccard 系数
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2019)09-1913-06
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2019.09.014

A Fast and Resource-Based Detection Approach of Similar Android Application

ZHANG Peng[1,2], NIU Shao-zhang[1], HUANG Ru-qiang[1]
(1. *Beijing Key Lab of Intelligent Telecommunication Software and Multimedia, Beijing University of Posts and Telecommunications, Beijing* 100876, *China*; 2. *College of Information Engineering, Ningxia University, Yinchuan, Ningxia* 750021, *China*)

Abstract: Since pirated Android applications (APPs for short) usually maintain a similar user experience to original APPs, a fast APP similarity detection approach based on resource signature has been proposed. In order to determine the similarity of a pair of APP, the approach calculates the Jaccard coefficient of resource signature sets of them because a set of resource signatures can be treated as a set of strings. With the help of the MinHash and LSH (Locality Sensitive Hashing) algorithm, it can avoid the traversal of all APP pairs by selecting candidate pairs from the APP set and verifying them at last. Because the procedure of selecting candidate pairs excludes a large number of APP pairs with lower similarity, this approach can significantly improve the detection speed of APP similarity. The experimental results show that the detection speed of this approach is about 30 times higher than the existing approach FSquaDRA while the detection result is almost identical.

Key words: APP similarity; resource signature; MinHash; LSH; Jaccard coefficient

1 引言

由于 APP 很容易被反编译出源代码,因此互联网上出现了大量的盗版. 为了扼制盗版 APP 数量的快速增长,除了可以采用代码混淆、软件防篡改等技术对 APP 进行"加固"之外,也可以对主要的 APP 市场(特别是第三方市场)进行监视分析,根据 APP 的签名、用户界面等信息,及时发现和清理 APP 市场中的盗版软件. 但如何能准确地计算任意两个 APP 的相似性,以及如何在大量的 APP 中快速发现相似的 APP,都是目前亟待解决的问题. 由于盗版 APP 为了吸引更多的用户,通常保持着与正版 APP 相似的用户体验(即"look and feel",如相似的界面布局、图像、声音等),因此本文将 APP 资源签名集合的 Jaccard 系数作为判断 APP 相似性

收稿日期:2018-05-28;修回日期:2018-08-04;责任编辑:梅志强
基金项目:国家自然科学基金资助项目(No. U1536121, No. 61370195)

的依据,提出一种快速的 APP 相似性检测方法,简称 FRSA(a Fast and Resource-based detection approach of Similar Android application). 该方法首先获取 APP 所有资源文件的签名信息,然后提取签名信息中的部分内容组成对应于 APP 集合的特征矩阵,继而利用 MinHash 算法[1]和 LSH 算法[2]的思路,以一定的概率从 APP 集合中挑选中 Jaccard 系数大于指定阈值的 APP 对(即 APP 候选对),最后对 APP 候选对进行检验,计算其 Jaccard 系数,从而判断 APP 之间的相似性. 由于挑选 APP 候选对的过程可以排除大量相似性较低的 APP 对,因此 FRSA 可以明显地提高 APP 相似性的检测速度.

2 相关工作

目前对 APP 进行相似性检测的方法主要有两个方向:特征提取法和水印法. 其中,特征提取法也称为零水印法,指的是不在 APP 中主动添加任何水印,而是通过从 APP 的各种属性中提取特征值,组成对应于 APP 的特征向量,然后利用某种向量相似性算法计算 APP 之间的相似性,例如从 Dalvik 字节码、用户界面、多媒体文件、APP 名称和图标等[3,4] APP 属性中进行特征提取,然后利用余弦距离、Jaccard 系数、欧氏距离等进行相似性计算. 特征提取法的优点是对 APP 性能的影响很小,但缺点是检测结果完全取决于对 APP 属性的特征提取,如果攻击者对 APP 的属性进行了修改(如利用代码混淆工具修改 Dalvik 字节码),将对特征提取法的正确性产生一定的影响.

水印法指的是将特定的数据(即水印,如图片、字符串密钥等)添加到 APP 中的特殊位置(如程序代码、程序调用图等),在检测时利用相应算法从 APP 中提取水印,然后根据提取的结果对 APP 的相似性进行判断. 水印的载体和水印数据多种多样,例如将 Dalvik 字节码的指令执行顺序、APP 的源代码作为水印载体;将字符化的图片、XML 和 DEX 文件中的字符和空格作为水印数据等[5,6]. 水印法的优点是检测结果相对稳定,不容易受到攻击者修改的影响,但缺点是 APP 的性能会有所下降.

本文提到的“资源”指的是 APP 包含的全部文件,如代码、布局、音频、图片文件等. 如前所述,盗版 APP 往往具有与正版 APP 相似的用户体验,因此盗版 APP 中必然存在与正版 APP 相同的资源. Zhauniarovich[7]等提出一种基于资源的 Android 相似应用检测方法 FSquaDRA,将 APP 中资源文件的 SHA1 签名信息作为特征,通过计算 APP 签名信息集合之间的 Jaccard 系数得出 APP 之间的相似性. 由于签名信息已经保存在 APP 的安装包内,因此这种方法可以达到较高的检测速度. 但缺点是在检测时需要遍历全部的 APP 对,当 APP 个数较多时仍然需要大量的检测时间.

3 相似性快速检测方法

为了提高 APP 相似性检测的速度,本文根据 FSquaDRA 方法的思路,提出一种基于资源签名的 APP 相似性快速检测方法 FRSA. 该方法可以有效地缩小 APP 相似性检测的范围,明显提高 APP 相似性的检测速度. FRSA 主要分为三个阶段:APP 特征矩阵生成阶段、APP 候选对集合生成阶段和 APP 候选对集合检验阶段,流程如图 1 所示.

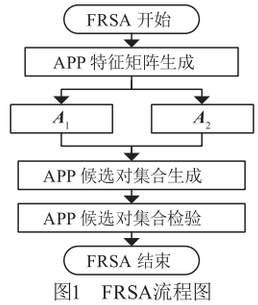

图1 FRSA流程图

如图 1 所示,FRSA 在第一阶段利用 APP 特征矩阵生成算法,从 APP 集合中的所有 APP 的资源签名中提取部分内容,生成两个对应于 APP 集合的特征矩阵 $\boldsymbol{A}_1$ 和 $\boldsymbol{A}_2$(生成两个而不是 N 个特征矩阵的原因将在第 4 节实验部分进行阐述). 在第二阶段,FSRA 利用 APP 候选对集合生成算法,根据两个特征矩阵生成两个 APP 候选对集合,然后计算两个集合的交集并生成最终的 APP 候选对集合. 在最后一个阶段,FRSA 对 APP 候选对集合进行检验,读取集合中每一对 APP 的所有资源签名信息,然后计算两个 APP 资源签名集合的 Jaccard 系数. 如果计算结果大于指定阈值,则将该 APP 对加入最终的检测结果集合,否则继续计算直到候选对集合中的 APP 对全部处理完毕.

3.1 APP 特征矩阵生成阶段

FRSA 在此阶段的任务是从 APP 的资源签名信息中提取特征信息,生成对应于 APP 集合的特征矩阵. 由于 APP 在打包之前需要对所有的资源文件进行 SHA1 签名并将这些签名信息以 BASE64 编码格式保存在 APP 安装包的“META-INF/MANIFEST. MF”文件,如图 2 所示,因此可以通过读取 MANIFEST. MF 文件来提取 APP 的资源签名信息.

由于每个 APP 具有的资源个数不同,因此 APP 特征矩阵生成算法(算法 1)采用了一种从签名信息中提取部分信息,组成对应于 APP 集合特征矩阵的方法,伪代码如算法 1.

```
Name: assets/drawable-hdpi.zip
SHA1-Digest: F678/SuFMPnExFCFwDmZlcURMtI=

Name: assets/bin/Data/sharedassets4.assets.split5
SHA1-Digest: ce9cianiGbfXFMMhAwNdkIfM1wg=

Name: assets/bin/Data/Managed/Assembly-Csharp.dll
SHA1-Digest: Z/Ts8XHkGJjaVFbif/dZkbJ6sMc=
```

图2　APP签名信息

算法 1　函数 1 genFeatureMatrix(F,t)

```
//Generatea feature matrix from part of digest information
//Input:a collection of "MENIFEST. MF" files F,an integer number t
//Output:a feature matrix A with 64^n rows and g columns
//combTable(64^n) ← combination list for n chars from 0~9,A~Z,a~z,/ and +
g ← F. count( )
initialize A (64^n,g) with all zero
for each file j in F do
    for each digest d in j do
        extracts a substring e from d that begins at t and ends at t+n-1
        v ← gets index position of e based on combTable(64^n)
        A(v,j) ← 1
```

算法 1 中的 F 表示一个"MENIFEST. MF"文件集合,g 表示文件个数. 算法 1 首先通过 F. count( )方法获取集合中文件的个数 g. 然后,算法创建一个大小为 $64^n \times g$ 的特征矩阵 $\boldsymbol{A}$,并将其初始化为全零. 对于 F 中任意第 j 个文件,算法从该文件的每个签名信息 d 中提取 n 个字符串到 e,提取的位置从 t 开始,$0 < t+n-1 \leqslant 27$. 如前所述,签名信息 d 以 BASE64 编码保存,n 个 BASE64 编码按照从"0"到"\"的次序可以组成 64^n 个字符串组合,记为 combTable(64^n). 根据提取到的字符串 e,算法按照以上次序可以计算出 e 在 combTable(64^n)中的位置 v,并将 $\boldsymbol{A}(v,j)$ 置 1. 最后,算法重复整个过程直到 F 中的全部文件处理完毕. 通过 APP 特征矩阵生成算法,可以将个数不等的 APP 签名信息统一到固定大小的特征矩阵,便于后续使用 MinHash 和 LSH 算法计算 APP 候选对. 从该算法流程中不难看出,随着 n 的增大,APP 特征矩阵的大小将会成指数级增加. 考虑到后续的计算量和大部分 APP 资源文件的实际个数,FRSA 选择 $n=2$,即生成的特征矩阵大小为 $4096 \times g$. 为了避免生成同样的特征矩阵,FRSA 在此阶段通过改变 t 的值(例如 $t=0$ 和 $t=12$),生成两个特征矩阵 $\boldsymbol{A}_1$ 和 $\boldsymbol{A}_2$.

3.2　APP 候选对集合生成阶段

FRSA 在此阶段的任务是利用 MinHash 算法和 LSH 算法的思路生成 APP 候选对集合. 首先,FRSA 采用 MinHash 算法将两个 APP 特征矩阵转化为两个 APP 签名矩阵. 根据 MinHash 算法,产生签名矩阵需要对特征矩阵的所有行进行均匀置乱. 但如果采取随机交换的方式进行置乱,需要较大的时间开销,因此 FRSA 在此阶段采用 Jure[8] 等提出的方法,产生 $r \times b$ 个 Hash 函数(r 和 b 为任意的正整数),然后利用这些 Hash 函数对两个特征矩阵的所有行进行置乱. 在置乱之后,按照 MinHash 算法选择特征矩阵每列中首个不为 0 数据的行号,分别生成两个大小为 $(r \times b) \times g$ 的签名矩阵 $\boldsymbol{Q}_1$ 和 $\boldsymbol{Q}_2$. 最后,FRSA 分别从 $\boldsymbol{Q}_1$ 和 $\boldsymbol{Q}_2$ 中挑选出相似度较高的 APP 对,生成两个 APP 候选对集合并计算两者的交集. APP 候选对集合生成算法的伪代码如算法 2 所示.

算法 2　函数 2 calculateCandidates ($\boldsymbol{Q}[\mathbf{0}..r \times b-\mathbf{1},\mathbf{0}..g-\mathbf{1}]$)

```
//Generate candidate pairs of similar apps
//Input:a signature matrix Q with r×b rows and g columns
//Output:a set of candidate pairs H
H ← Φ
for each band i in b bands do
    for each column c in g do
        m ← 0
        c_i ← read r rows in i of c from Q
        hash(c_i) ← hashFun(c_i)
        if hash(c_i) ∈ K_m
            add c into a candidate column set L_m
        else
            create a new hash bucket K_m and add hash(c_i) into it
            create a new candidate column set L_m and add c into it
            m ← m+1
        for k ← 0 to m-1 do
            for each pairwise column ⟨c_x,c_y⟩ in L_m do
                H ← H∪⟨c_x,c_y⟩
```

从算法 2 中可以看出,候选对集合生成算法实际是 LSH 算法的一种实现,$\boldsymbol{Q}$ 表示 APP 签名矩阵,由 $r \times b$ 行和 g 列构成. 其中,b 表示 LSH 算法中行条(band)的个数,r 表示每个行条中包含的行数,H 代表 APP 候选对集合. 候选对集合生成算法首先需要对每个行条上的列数据进行 Hash 计算,从而判断在一个行条上的任意两列 c_x 和 c_y 是否能映射到同一个哈希盒(Hash bucket). 根据 LSH 算法,任意两列 c_x 和 c_y 在至少一个行条上映射到同一个哈希盒的概率如式(1)所示:

$$P(\mathrm{LSH}(c_x)=\mathrm{LSH}(c_y))=1-(1-s^r)^b \qquad (1)$$

其中,s 表示任意两个 APP 的相似度,即 $s=\mathrm{sim}(\mathrm{app}_x,\mathrm{app}_y)$,$s \in [0,1]$. 从式(1)不难看出,通过设置不同的 r,s 和 b,可以对映射的概率进行控制,从而以一定的概率找到 APP 候选对. 为简单起见,FRSA 将 r 设置为 4,将 s 设置为 0.5. 算法 2 中 c_i 表示 c 列在第 i 个行条中的 r 行数据,通过 Hash 函数 hashFun 可以计算出 c_i 的 Hash 值 hash(c_i). hashFun 函数的计算方法是将 c_i 上的 r 行数据以字节形式连在一起. 如果 c_i 已经映射到第 m 个哈希盒 K_m,则将 c 列加入到候选列集合 L_m,否则创建一个新的

哈希盒 K_m 和新的候选列集合 L_m，将 hash(c_i) 和 c 分别加入其中，然后对 m 加 1. 最后，算法遍历 m 个候选列集合，将集合中的每一对列组合$\langle c_x, c_y\rangle$加入 H 集合，重复执行直到 b 个行条全部处理完毕. 由于 FRSA 已经对 r 和 s 设置了固定值，只需要改变 b 即可实现映射概率的控制. 而 b 的设置关系到 FRSA 的计算量和准确性，因此将在第 4 节中通过实验对 b 的取值进行讨论.

3.3 APP 候选对检验阶段

FRSA 在此阶段的任务是对前一阶段生成的 APP 候选对集合进行检验，即计算候选对集合中每一对 APP 的 Jaccard 系数，从而得出最终的相似性检测结果. APP 候选对集合检验算法的伪代码如算法 3 所示.

算法 3 函数 3 checkAppCandidates(H)

```
//Calculate similarity of each candidate pairs and generate a final set of similar pairs
//Input:a set of candidate pairs H
//Output:a set of similar pairs J
for each file f in H do
        res(f) ← read all digests from the "MENIFEST. MF" file of f
J ← Φ
for each pair ⟨f_i,f_j⟩ in H do
        sin(f_i,f_j) ←|res(f_i) ∩ res(f_i)|/|res(f_i) ∪ res(f_j)|
        if sim(f_i,f_j) > s
          J ← J∪⟨f_i,f_j⟩
```

算法 3 中的 H 代表 APP 候选对集合，J 表示相似性检测结果集合. 为了加快检验的速度，算法首先将 H 集合中的全部 APP 资源签名信息加载到内存中，即对于任意一个 f，算法将 f 的全部资源签名信息读取到集合 res(f). 然后，算法计算 H 中任意一个 APP 候选对$\langle f_i, f_j\rangle$的 Jaccard 系数 sim(f_i,f_j)，如果结果大于阈值 s，则将该 APP 候选对加入 J. 对于 J 中的每一对 APP，可以进一步根据其证书信息判断 APP 是否为“克隆”版本. 由于 FRSA 着重讨论的是 APP 相似性的检测，因此没有设计 APP 证书信息的判断过程.

4 实验结果

为了对 FRSA 的效率和准确性进行评估，作者使用 Visual Basic. NET 语言对 FRSA 进行了实现并且设计了与 FSquaDRA 的对比性实验，分别从检测时间和检测结果两个方面对两者进行对比分析. 由于 FSquaDRA 只提供了 JAVA 源代码，为了避免编程语言差异对实验结果产生影响，作者在参考 FSquaDRA 源代码的基础上，使用 Visual Basic. NET 语言重新实现了 FSquaDRA 并删除了其中比较 APP 证书信息的功能. 实验的硬件为普通 PC 机(Intel CORE I7-6700，16G 内存，操作系统为 Windows 10 专业版，开发工具为 Visual Studio 2015 专业版). 实验所用的 APP 集合包含 38572 个 APP，分别来自于三个第三方应用市场：anzhi(4114)，apkhome(5576 个)，gfan(10855 个)和一个 Android 恶意应用库 androzoo(18073 个). 为了测试 FRSA 和 FSquaDRA 针对不同数量 APP 集合的检测时间和结果，从 38572 个 APP 中任意取 5000，10000，17800，29000 个 APP，产生 4 个 APP 子集，加上原有的 APP 集合，共形成 5 个 APP 实验集合.

实验步骤如下：

(1)针对行条个数 b 的不同取值(b = 35，40，45，50，55，60，65)，使用 FRSA 对 5 个 APP 集合进行检测，获取运行时间和检测结果.

(2)使用 FSquaDRA 对 5 个 APP 集合进行检测，获取运行时间和检测结果.

实验结果如图 3 和表 1 所示.

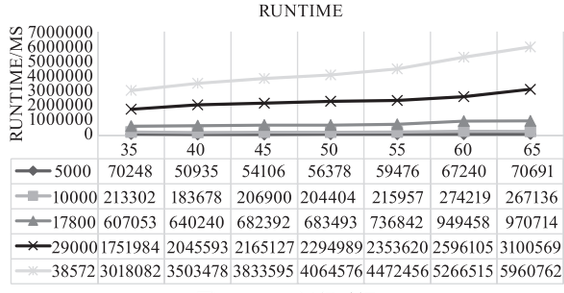

图3 FRSA运行时间

表 1 FRSA 检测结果

	35	40	45	50	55	60	65
5000	67508	67508	67508	67508	67508	67508	67508
10000	292354	292361	292368	292370	292369	292370	292369
17800	863082	863124	863141	863144	863151	863153	863152
29000	2320996	2321034	2321119	2321128	2321174	2321182	2321182
38572	4259748	4259754	4259953	4259982	4260050	4260067	4260068

图 3 显示的是 FRSA 在不同 b 的取值下的运行时间，横坐标表示 b 的取值，纵坐标表示运行时间(毫秒). 从图 3 中可以看出，b 的不同设置对 FRSA 的运行时间产生了明显的影响. 随着 b 的增大，FRSA 的运行时间逐渐增加. 这是因为 FRSA 在 APP 候选对集合生成算法中需要遍历所有的行条，行条个数越多，需要的时间越长.

表 1 显示的是 FRSA 在不同 b 的取值下的检测结果，第一行表示 b 的不同取值，第一列表示 APP 集合的大小，其余数据表示 FRSA 的检测结果，即 Jaccard 系数大于 0.5 的 APP 对的个数. 从表 1 中可以看出，随着 b 的增大，FRSA 的检测结果逐渐增加. 这是因为根据式(1)，b 的取值越高，任意两列在至少一个行条上映射到同一个哈希盒的概率就越高. 映射概率的增大，降低了 FRSA 出现检测误差的可能性，因此使得 APP 对的个数

逐渐增加. 但从图 3 中可以看出,随着 b 值的增加,FRSA 的检测时间也不断增加,因此用户可以根据自身的实际需要对 b 进行调整,从而在检测结果和运行时间之间达到平衡.

对于 FRSA 选择生成两个候选对集合而不是生成 N 个集合(N 为正整数,$N>2$),作者利用一个小实验对原因进行说明. 实验选择 APP 个数为 5000 的集合,取 $b=65$,使用 FRSA 分别生成 $N=1$、2、3 和 4 个候选对集合,然后对 $N=1$ 时的候选对集合,以及 $N=2$、3、4 时的候选对集合的交集进行检验,实验的运行时间如表 2 所示.

表 2　候选对集合的个数对运行时间的影响

	$N=1$	$N=2$	$N=3$	$N=4$
集合大小	572429	234743	173861	143961
生成集合时间	26907	32944	53786	66413
运行时间	119234	70691	81828	89633

表 2 中的第一行表示 N 的取值,第二行表示 FRSA 生成的候选对集合的大小,即 APP 候选对的个数,其中 $N=2$、3、4 表示使用 FRSA 分别生成 2、3、4 个候选对集合并计算交集的结果. 第三行表示 FRSA 生成候选对集合并计算交集的时间($N=1$ 时不计算交集),单位为毫秒. 最后一行表示 FRSA 的运行时间,即生成候选对集合的时间与检验候选对集合的时间之和. 从表 2 中可以看出,当 $N=1$ 时,候选对集合中的 APP 对个数最多. 随着 N 的增加,APP 对的数量逐渐下降. 但 FRSA 的运行时间并没有持续降低,在 $N>2$ 后反而出现了升高. 这是因为虽然候选对个数的减小缩短了检验候选对集合的时间,但仍然无法抵销 FRSA 在生成 N 个候选对集合方面的时间开销,从而导致了 FRSA 运行时间的增加. 因此,FRSA 选择了 $N=2$,即只生成两个候选对集合.

为了便于分析 FRSA 的运行效率,定义“检测速度=集合中 APP 对的个数÷检测时间”. 对于实验中所用的 5 个 APP 集合,不难计算出 APP 对的个数分别为 12497500、49995000、158411100、420485500 和 743880306 对. 取 $b=65$ 时 FRSA 的检测速度与 FSquaDRA 进行对比,结果如图 4 所示.

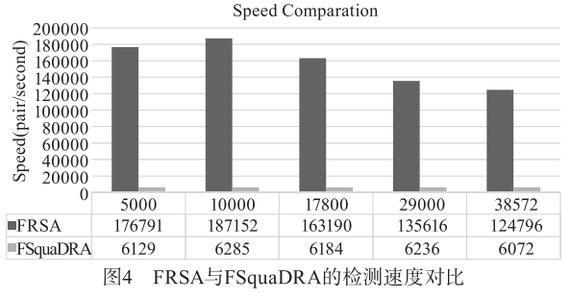

图4　FRSA与FSquaDRA的检测速度对比

图 4 的横坐标表示 APP 集合的大小,纵坐标表示检测速度,单位为“对/秒”. 从图 4 中可以看出,FSquaDRA 的检测速度大约为每秒 6200 对,比 Zhauniarovich[7] 等提及的每秒 6700 对略低. 这是因为 Zhauniarovich 等在计算 FSquaDRA 的检测速度时只考虑了 Jaccard 系数的计算时间,没有考虑资源签名信息的加载时间. 而 FRSA 的检测过程包括资源签名的加载,因此为了公平起见,作者将资源签名信息的加载时间计入 FSquaDRA 的运行时间,因此 FSquaDRA 的检测速度降低到了大约每秒 6200 对.

从图 4 中可以看出,FRSA 的检测速度大约是每秒 180000 对,约为 FSquaDRA 检测速度的 30 倍. 从两种检测方法的流程上看,FSquaDRA 需要遍历 APP 集合中所有的 APP 对,时间复杂度为 $O(n^2)$,n 代表 APP 的个数. 而 FRSA 首先从大量的 APP 对中挑选出可能相似的 APP 候选对,继而再遍历候选对集合,时间复杂度为 $O(m^2)$,m 代表候选对集合中的 APP 个数. 由于 $m^2 \ll n^2$,因此 FRSA 可以节省大量的检测时间. 表 3 是针对不同 APP 集合,FRSA($b=65$)和 FSquaDRA 需要处理的 APP 对的个数对比(即 m^2 和 n^2 的对比).

表 3　APP 对的个数对比

	FRSA($b=65$)	FSquaDRA	FRSA/FSquaDRA
5000	234743	12497500	0.018783197
10000	979379	49995000	0.019589539
17800	3030306	158411100	0.019129379
29000	7969508	420485500	0.01895311
38572	14334039	743880306	0.019269281

表 3 的第一列表示 APP 集合的大小,第二列是 FRSA 生成的候选对集合大小,第三列是 FSquaDRA 需要检测的 APP 对个数,第四列表示前两列数据的比值. 不难看出,FRSA 需要检验的 APP 对只有 FSquaDRA 的百分之二. 由于排除了大量的 APP 对,因此 FRSA 的检测速度远高于 FSquaDRA.

对于检测正确性的对比,选择 $b=65$ 时的 FRSA 检测结果与 FSquaDRA 的检测结果进行对比,如表 4 所示.

表 4　FRSA 与 FSquaDRA 的检测结果

	FRSA($b=65$)	FSquaDRA
5000	67508	67508
10000	292369	292370
17800	863152	863153
29000	2321182	2321183
38572	4260068	4260068

表 4 的第一列是 APP 集合的大小,第二、三列分别是 FRSA 和 FSquaDRA 的检测结果,即 Jaccard 系数大于 0.5 的 APP 对的个数. 从表 4 中可以看出,FRSA 的检测结果与 FSquaDRA 几乎完全一致,误差小于 0.001%. 这是由于通过对 b 控制,FRSA 可以用较高的概率挑选出

Jaccard 系数大于 0.5 的 APP 对,因此在后续的 APP 检验阶段不会出现明显的误差.

5 总结

由于盗版 APP 通常保持着与正版 APP 相似的用户体验,本文将 APP 的资源文件作为相似性检测的基础,提出一种快速的 Android 应用相似性检测方法 FRSA. FRSA 将 APP 的资源签名视为字符串集合,利用计算任意一对 APP 资源签名集合的 Jaccard 系数判断两者的相似性. 为了避免遍历全部的 APP 对,FRSA 首先利用 APP 特征矩阵生成算法,从 APP 集合中提取每个 APP 的部分资源签名信息,组成对应于 APP 集合的特征矩阵. 继而,FRSA 利用 APP 候选对集合生成算法,以一定的概率从 APP 集合中挑选出 Jaccard 系数大于指定阈值的 APP 对,生成 APP 候选对集合. 最后,FRSA 对 APP 候选对集合进行检验,计算集合中每一对 APP 的 Jaccard 系数,将大于指定阈值的 APP 对加入最终的检测结果集合. 由于 FRSA 在生成候选对集合时排除了大量相似性较低的 APP 对,因此可以有效地提高 APP 相似性检测的速度. 实验结果表明,FRSA 的检测速度大约是现有方法 FSquaDRA 的 30 倍,而检测结果与 FSquaDRA 几乎完全相同.

参考文献

[1] Broder, Andrei Z. On the resemblance and containment of documents[A]. Proceedings of Compression and Complexity of Sequences[C]. Positano, Salerno, Italy: IEEE, 1997. 21 – 29.

[2] Indyk Piotr, Motwani Rajeev. Approximate nearest neighbors: towards removing the curse of dimensionality[A]. Proceedings of Thirtieth Symposium on Theory of Computing[C]. Dallas, Texas, USA: ACM, 1998. 604 – 613.

[3] Zhou W, Zhou Y, Jiang X, et al. Detecting repackaged smartphone applications in third-party android marketplaces [A]. Proceedings of the second ACM conference on Data and Application Security and Privacy[C]. San Antonio, Texas, USA: ACM, 2012. 317 – 326.

[4] Li L, Bissyande T F D A, Klein J. Simidroid: Identifying and explaining similarities in android apps[A]. The 16th IEEE International Conference on Trust, Security and Privacy in Computing and Communications[C]. Sydney, Australia: IEEE, 2017. 136 – 143.

[5] Zhang Y, Chen K. AppMark: a picture-based watermark for android apps[A]. IEEE Eighth International Conference on Software Security and Reliability[C]. San Francisco, California, USA: IEEE, 2014. 58 – 67.

[6] Zeng L, Ren W, Lei M, et al. DroidMark: A lightweight android text and space watermark scheme based on semantics of XML and DEX[A]. The 5th International Conference on Emerging Internetworking, Data & Web Technologies[C]. Wuhan, China: Springer, 2017. 756 – 766.

[7] Zhauniarovich, Yury, Gadyatskaya, et al. FSquaDRA: Fast detection of repackaged applications[A]. Proceedings of the 28th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy[C]. Vienna, Austria: Springer, 2014. 130 – 145.

[8] Jure Leskovec, Anand Rajaraman, Jeff Ullman. Minging of Massive Datasets[M]. Cambridge: Cambridge University Press. 2014

作者简介

张 鹏 男,1980 年生,博士生,副教授,研究方向为移动互联网、软件保护等.
E-mail: longbow27@163.com

牛少彰 男,1963 年生,博士,教授,博士生导师,研究方向为信息隐藏、移动互联网等.
E-mail: szniu@bupt.edu.cn

黄如强 男,1995 年生,硕士生,研究方向为智能终端安全等.