云环境下基于函数编码的移动应用克隆检测

杨佳[1]，付才[1]，韩兰胜[1]，鲁宏伟[1]，刘京亮[2]
（1. 华中科技大学计算机学院，湖北 武汉 430074；2. 北京航空精密机械研究所，北京 100876)

摘 要：提出了一种云环境下基于汇编函数层编码的 App 克隆检测方法，实现了 Android 应用克隆检测原型系统 Pentagon。首先，针对 App 执行文件代码设计了二进制函数基本代码块特征提取方法；其次，提出了一种对二进制函数进行单调编码的算法，基于图形嵌入原理，融合执行流程图与函数代码基本块特征，形成每个函数代码的投影特征；最后，提出了一种高效的第三方库函数过滤方法，进一步提升了克隆检测效果。实验证明，所提方法具有 97.6%的准确率，在实验条件下判定一个 App 克隆对只需要 79 ms，能够支撑云环境下应用市场级别的大规模克隆检测。

关键词：克隆检测；函数编码；控制流程图；知识产权保护
中图分类号：TP391.4
文献标识码：A
doi: 10.11959/j.issn.1000−436x.2019106

Function encoding based approach for App clone detection in cloud environment

YANG Jia[1]，FU Cai[1]，HAN Lansheng[1]，LU Hongwei[1]，LIU Jingliang[2]
1. School of Computer Science & Technology, Huazhong University of Science and Technology, Wuhan 430074, China
2. Avic Beijing Precision Engineering Institue Aircraft Industry, Beijing 100876, China

Abstract: An efficient function-based encoding scheme in the cloud environment for detecting the cloned Apps was designed, called Pentagon. Firstly, a basic block feature extraction method was proposed. Secondly, a monotonic encoding algorithm for the App function was designed, which encoded the function based on the control flow graph structure and basic block attributes. Finally, a three-party libraries filtering method was proposed by using an efficient clustering algorithm based on the function feature. Experiments verified the effectiveness of the proposed scheme. The average search time is close to 79 ms, and the clone detection accuracy achieves 97.6%.

Key words: clone detection, function encoding, CFG, intellectual property right protection

1 引言

近年来，随着移动智能终端设备的迅速发展，Android 应用占据了移动应用市场的主要份额。据统计，2017 年已有超过 15.5 亿部智能手机被使用，其中超过 80%的手机搭载了 Android 系统。海量规模的各类移动应用（App）投放在多个第三方应用市场供用户下载，例如谷歌应用市场中 Android 应用数量已经接近 140 万。这些移动应用已在办公、娱乐等各个方面深入人们的生活，成为不可或缺的一部分。

然而，由于 Android 系统的开放性和 Android 应用的易被破解性，许多非法开发者利用克隆技术或者反编译工具仿造出大量的克隆 App。目前，应用重打包已经成为恶意软件传播的主要途径之一。目前，有超过 85%的安卓应用存在重打包行为，这不仅侵害了

收稿日期：2018-09-06；修回日期：2019-04-06
通信作者：付才，fucai@hust.edu.cn
基金项目：国家自然科学基金资助项目（No.61572222，No.61772194，No.61272405）
Foundation Item: The National Natural Science Foundation of China (No.61572222, No.61772194, No.61272405)

原开发者的知识版权和利益，也对移动用户的安全和隐私造成危害。此外，一些攻击者会修改原始 App 中的函数逻辑，如去除 App 中的支付函数、修改原 App 中的广告库，甚至插入各种各样的恶意代码，再用一个新的签名重新打包这些恶意 App。克隆 App 在第三方应用市场中大量地被下载使用，严重干扰了应用市场的正常环境。为保护合法用户和原创开发者的正当权益，如何实现高效准确的 App 克隆检测是当前移动应用市场面临的技术挑战。

由于 App 市场的规模庞大，App 的数量不断快速增长，仅基于本地的计算资源进行 App 克隆检测会因为硬件配置较低而出现计算能力有限、电源功耗过高等问题，本地检测技术已不能满足百万数量级的 App 分析需求。移动互联网中的云计算、大数据及虚拟化技术为解决大规模 App 克隆检测问题提供了技术支撑。本文基于云环境提出了一种高效 App 克隆检测方法，构建了检测原型系统，提高了 App 克隆检测的效率，能有效应对大规模 App 克隆检测挑战。

许多研究者提出了多种 App 克隆检测技术，主要有以下三类：基于简单散列特征提取的技术[1]、基于静态语义特征提取的技术[2-6]、更加复杂有效的基于 Android 应用 UI 结构特征的提取技术[7-10]。第三类检测技术中基于 UI 结构特征的检测方法主要分析用户接口之间的使用相似度进而判断 App 是否克隆。目前，Androguard[11] 和 FSquaDRA[12]是用于 App 克隆检测的主要公用工具。Androguard 能够在 Dalvik 字节码层进行比较，FSquaDRA 能够基于源文件进行相似度分析。然而这些研究还存在以下问题：基于散列值和某一特定静态语义分析方案的适应性还需增强，例如，在克隆中使用代码混淆可避免被基于静态语义特征的方法检测出来；而 UI 结构特征提取方案需要提取整个 App 的 UI 调用结构，同时需要复杂的图匹配计算来比较 App 相似度，因此对于小结构 UI 图的 App 难以达到较好的检测效果。

本文主要从以下 3 个方面来讨论目前的工作。

1) 二进制层函数克隆检测方案

在二进制层的克隆检测方案中，一种通用的方法是使用基于函数执行序列的路径进行代码相似度检测。这种方法能够检测控制流程图（CFG, control flow graph）的变化，但是不能准确地检测在不同平台或者代码混淆下的函数克隆。另一种方法是利用代码的静态特征进行 App 克隆检测。但当恶意开发者对这些静态特征进行修改的时候，例如修改字段变量的常数或 API Call 的调用序列等，这种方法的正确率就无法得到保障。Rendezvous[13]在二进制代码上使用代码搜索的办法进行克隆检测，这种方法有 2 个限制：首先，需要代码的 ngram 特征来提高准确率；其次，需要将整个 CFG 分为几个子图来处理。Pewny 等[14]提出了一种基于已知的漏洞签名的搜索方案，该方案计算二进制代码块的语义散列来进行函数的克隆比较，然而这种方案并不高效。Feng 等[15]提出了基于控制流程图的恶意代码检测方案，但是需要复杂的图匹配过程。

2) App 克隆检测

传统方法检测恶意 App 主要通过权重很大的静态和动态特征来进行黑白名单检测，但是不能检测未知的恶意代码，如果之前并没有收录这种恶意代码，则无法进行检测。例如，Zhou[16]利用软件权限、API 调用顺序等静态特征进行检测，但是仅利用这些静态特征进行检测会导致较低的准确率。David[17]能够自动划分一个函数调用图的社区结构，这些划分的社区结构能够用于新的恶意代码检测。Arp 等[18]提出一种新的轻量级的检测方案，这种方案从安卓应用中提取八类特征，然后对这些特征用 SVM 进行分类实现检测[3]。

3) 动态检测

该检测方式需要 App 在一个虚拟的环境下运行，从动态特征中提取信息。Andrubis[19]即是动态地进行检测，但是耗时过长。Enck[20]提出了一个系统 Taintdroid，通过追踪敏感信息的流动来进行克隆行为的检测，这种方法同样非常耗时，在效率上有所欠缺，无法应对大规模的检测。

实现云环境下高效的函数层 App 克隆检测方案存在以下一些困难和挑战。

1) 相比面向源代码的克隆分析，移动应用 App 中基于可执行字节码的克隆分析更为复杂。移动应用的源码并不能轻易获取，同时，Android 应用中大量使用的代码混淆技术也给重打包检测增加了难度。

2) 海量的应用规模为克隆检测效率提出了更高的要求。随着各类 Android 应用的普及，App 应用数量呈现出飞速增长的趋势，通过 App 的 UI 调用图或者动态分析等方法都无法应对大规模级别的克隆分析。

3) 移动应用中第三方库函数的大量使用干

扰了克隆检测的效率与准确率。只有App自身核心函数代码来源于其他应用才能认定是克隆App，之前的工作仅通过黑白名单过滤第三方库函数，在正确性和效率方面存在很大的问题。

4) 系统需要具备良好的可扩展性。在有新应用添加到云端时，既要快速分析判断，又要将新应用特征快速融合到整体克隆特征库中，实现增量特征更新。如果需要和之前的App克隆检测数据库同时学习与提炼，其应用效率会受到影响。

针对上述问题，本文面向云环境提出了一种基于函数编码的App克隆检测方法，并实现了该方法的原型系统，称为Pentagon。该方法在App字节码层面对函数的控制流程图与基本代码块进行特征编码，从函数层面比较2个App之间的相似度。所有的特征提取过程和检测算法均在云环境下构建和实施。本文主要贡献如下。

1) 设计了根据App字节码提取函数基本代码块特征的方法。每个执行函数由多个基本代码块通过跳转连接组成，首先针对每个执行函数提取原始CFG，然后提取每个CFG图形节点的特征，即基本代码块特征，这些特征值表示了该代码块的静态统计特征以及该代码在CFG中调用跳转结构特征。

2) 提出了融合代码块特征和函数CFG特征的编码算法。该编码算法将App中任一函数编码成几个关键值的特征向量，提取的函数CFG有向图特征空间被压缩成低维的数字特征空间；然后，基于该特征空间使用高效的聚类方法过滤第三方库函数；同时，证明了算法中函数特征编码的单调性，一个特征编码仅代表一个函数。

3) 解决了规模化App增量更新的问题。Pentagon对App中每个函数进行独立编码，对于新增加的应用程序，Pentagon可单独编码该App中所有函数特征，然后在数据库中对App进行搜索比较，避免了同时整体提炼所有App特征值的复杂过程。

4) 从第三方应用市场收集了超过152 789个应用程序，对原型系统进行了大规模的实验验证。实验结果表明本文方法的预处理效率和克隆检测效率是目前典型App克隆方案的8~100倍。在1 000个函数中一个克隆函数的平均搜索时间为7.9×10^{-9} s。同时，App克隆检测的正确率达到97.6%。

2 问题描述和方案总览

本节首先说明设计移动应用App克隆检测系统需要解决的问题，然后介绍本文提出的面向云环境下的App克隆检测方案。

2.1 问题描述

本文需要解决的关键问题是在云环境下如何快速判断一个未知App是否为克隆App，并判断克隆App中具体哪些函数是相似的及其相似度大小。判断App是否克隆，需要满足以下条件。1) 不同的开发源有相同的App功能函数。通过查看Android应用的签名密钥和App中所有函数的特征来判断该App是否为克隆。如果这些不同签名的App具有相似或相同的核心函数特征，则说明是克隆App。2) 相似的函数不能包括第三方库函数。App的代码中一般包含了许多相同的第三方库函数，这些库函数不能作为判断克隆的依据，克隆检测方案需要过滤这些第三方库函数。

检测App是否克隆的基本步骤如下。从非结构化的App字节码中提取CFG特征，然后将每个含有基本代码块特征信息的CFG编码为一个低维数字特征，该特征向量表示App中的函数特征。通过不同App中所有的函数特征匹配来实现App函数层的克隆检测。App克隆检测问题数学描述如下。

问题1 如何设计一个函数f计算每个App函数的编码特征$\vec{c}_j$。通过App中所有函数的编码特征获得App的矩阵表示，即$\boldsymbol{A}=[\vec{c}_1,\vec{c}_2,\cdots,\vec{c}_j,\cdots,\vec{c}_m]$，其中，$\vec{c}_j$表示整个函数的编码特征向量，$j=1,2,\cdots,m$，也是矩阵$\boldsymbol{A}$中第$j$列列向量；$m$表示App中函数的个数。在函数$f$中，$\vec{v}_i$表示CFG中基本代码块$i$的特征，$|v|$表示一个CFG中代码块的个数。

问题2 如何设计一个有效的过滤函数F以去除App中第三方库函数。这是提高App克隆检测正确率的重点问题，由于App克隆检测需要比较所有的代码块特征，如果不同App中有大量相同的第三方库函数，一方面会增加克隆检测时间，另一方面会干扰克隆检测的准确率，相同的第三方库函数不能作为克隆的检测依据。

2.2 方案总览

本节主要介绍函数层的特征编码方案，并说明本文方案如何解决第2.1节中提出的问题。本文的App克隆检测方案的基本思想是：首先，提取App反编译smali文件中每个函数的CFG；然后，针对每个CFG中的每个基本代码块提取关键特征值，并根据编码算法将所有的基本代码块特征和CFG跳转结

构编码为低维的向量表示，将该向量存储到 App 特征数据库中；最后，用余弦相似度计算 2 个函数特征之间的向量距离，通过该距离判断 App 是否克隆。

如图 1 所示，Pentagon 有以下 3 个检测步骤。1) CFG 提取云环境，包括安卓市场下载 App、反编译和 CFG 提取过程。2）函数特征生成及过滤云环境，主要为 Pentagon 编码过程，Pentagon 编码的函数特征用来进行第三方库函数过滤。3）云环境下 App 克隆检测，主要为特征向量相似度比较。每一个步骤都在云环境下实现，具体实现过程为：首先编写自动下载 App 的爬虫引擎，将下载的 App 存储到对应的服务区中；然后通过并发方式对每个下载的 App 进行 CFG 提取以及函数特征编码；结合云环境下内存流计算方式并行地处理这些 App，获得 App 中每个函数的低维数据表示，并将编码特征存储在云端的非结构化数据库中。

上述检测步骤中，第一步，主要提取函数 CFG 中的基本代码块属性。在 Pentagon 中，CFG 的每一个节点提取 5 个属性值，这 5 个属性值分别表示字节码特征和该基本代码块在 CFG 中的拓扑结构。第二步，根据 CFG 的拓扑结构，设计了一种基于图形嵌入的编码算法，将 CFG 中的所有节点特征编码为一个单调的低维数字特征，并通过反证法证明其单调性；使用函数编码特征进一步过滤第三方库函数，在过滤之前，先删除每个 App 中重复的函数，然后通过聚类算法删除第三方库函数。第三步，给定一个待检测的 App，使用局部敏感散列（LSH, local sensitive hash）搜索算法实现高效的函数特征匹配。由于函数特征是独立编码，不需要更新之前的 App 函数特征库，只需在数据库中增加新的 App 特征。

3 Pentagon 函数编码

本节重点描述 Pentagon 函数特征编码算法，该算法将 App 中每个函数编码为一个一维向量，该一维向量能够单调表示 App 中的函数特征。

3.1 原始 CFG 提取

Android 应用一般是由开发者将所有的源码和资源打包成 APK (Android package)文件，然后发布供用户下载。APK 文件是一个压缩包，解压缩之后的文件夹内主要包含该应用的 Dalvik 字节码、UI 资源、UI 布局、配置文件、签名信息等。其中，Dalvik 字节码是 App 的可执行函数文件。Android 应用程序通常用 Java 语言实现，然后被编译成 Dalvik 字节码。

APK 文件本质上是一个压缩文件。反编译的 classes.dex 文件是 App 主要的代码文件，源代码编译成 class 后，转成 jar，再压缩成 dex 文件。dex 文件是可以直接在 Android 虚拟机上运行的文件。smali 语言是 davlik 的寄存器语言，语法上和汇编语言相似，一个 smali 文件中包含了很多函数，每个函数中包含了一系列的操作码及该操作码对应的寄存器和处理数据。本方案首先对 smali 文件提取 CFG。CFG 中的每个节点代表函数中的一个代码块。CFG 中跳转结构开始于一个代码块，结束于另一个代码块，有向边表示控制流程图的跳转结构。

3.2 函数特征编码

编码函数特征的主要思想是从 App 的 smali 文件的字节码中提取一个带有代码块特征的 CFG，并将该特征 CFG 投影编码为一个包含 5 个属性值的低维向量特征。与函数代码块的其他特征（如 I/O 特征和其他语义特征[16]）不同，CFG 能更加全面地匹配函数的执行流程特征。CFG 中的边表示基本代码块之间的调用关系，CFG 中的每个代码块包含了一系列的操作码和数据。本文基于图形嵌入的思想来编码带有代码块信息的 CFG 结构。Pentagon 融合了 CFG 拓扑结构和基本代码块的静

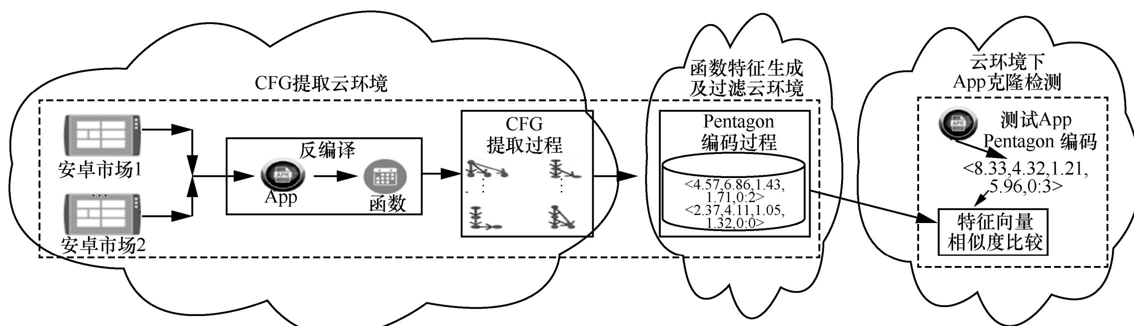

图 1 Pentagon 检测步骤

态特征综合完成函数特征编码。图 2 表示 CFG 的抽象图，一个圆圈表示一个基本代码块，也是 CFG 图中的一个节点，圆圈中的字母表示该基本代码块的序号，圆圈旁边每一行表示一行 dex 汇编代码，其中包含了操作码及使用的寄存器（v0~v4 表示寄存器），图中的箭头表示 2 个代码块跳转的一条有向边。

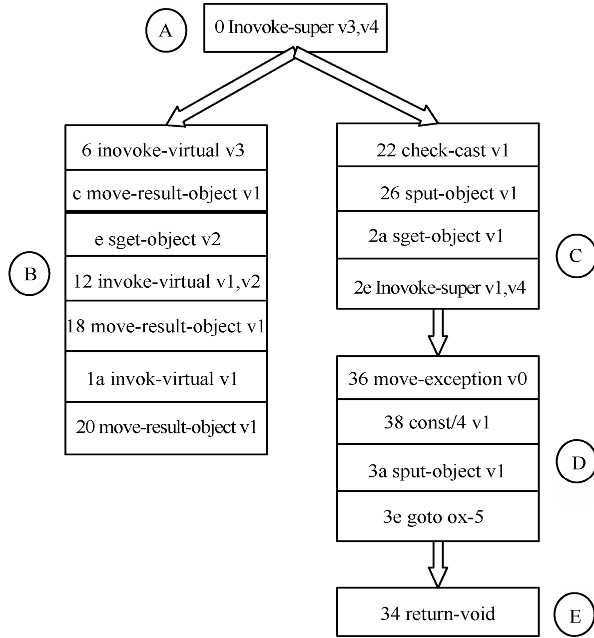

图 2 CFG 抽象图

定义 CFG 为一个有向图 (V,E)，其中，V 是一个函数的代码块集合，$E \subseteq V \times V$ 是一系列代码块之间的连接边。每个基本代码块的特征表示为一个一维向量 $\vec{v}_i = (s_i, p_i, a_i, o_i, l_i)$。其中，$s_i$ 为代码块 i 在 CFG$_j$ 中的使用序列，p_i 为代码块 i 的操作码的数目，a_i 为调用不同的 API Call 的个数，o_i 为代码块 i 的出度，l_i 为代码块的循环结构的个数，并定义代码块 i 有唯一的权重 ω_i。根据提出的算法，使用权重 ω_i 编码所有的节点特征。具体的计算过程如式(1)所示。

$$c_{js} = \frac{\sum\limits_{e(i,k)\in G_j} (\omega_i s_i + \omega_k s_k)}{\omega}$$

$$c_{jp} = \frac{\sum\limits_{e(i,k)\in G_j} (\omega_i p_i + \omega_k p_k)}{\omega}$$

$$c_{ja} = \frac{\sum\limits_{e(i,k)\in G_j} (\omega_i a_i + \omega_k a_k)}{\omega}$$

$$c_{jo} = \frac{\sum\limits_{e(i,k)\in G_j} (\omega_i s_i + \omega_k s_k)}{\omega}$$

$$c_{jl} = \frac{\sum\limits_{e(i,k)\in G_j} (\omega_i l_i + \omega_k l_k)}{\omega}$$

$$\omega = \frac{\sum\limits_{i=1}^{|V|} \omega_i}{|V|} \tag{1}$$

其中，$e(i,k)$ 表示 G_j 中节点 i 到节点 j 之间的一条边 $|V|$表示节点数。计算结果 $\vec{c}_j = (c_{js}, c_{jp}, c_{ja}, c_{jo}, c_{jl})$ 是每个函数的特征，其中，c_{js}、c_{jp}、c_{ja} 表示代码块的统计特征，c_{jo}、c_{jl} 表示 CFG 拓扑结构的跳转结构。

3.3 权重参数生成

Pentagon 的关键是计算每个节点 i 的权重 ω_i，ω_i 用于函数特征的融合编码。为了得到每个节点的权重，首先定义带有代码块特征的 CFG 一级跳转结构和全局跳转结构[21-22]。

1) 一级跳转结构。一级跳转结构 L 描述 CFG 中任意 2 个节点之间的跳转情况。

$$L = \{e(1,2), e(1,3), \cdots, e(i,k)\}, \forall i,k \in \{1,2,\cdots,|V|\}$$

如果节点 i 和 k 之间至少有一条有向边，则 $e(i,k)$=1；否则 $e(i,k)$=0。一级跳转结构对 CFG 编码非常重要，表示在函数代码中 CFG 结构代码块的一级继承和调用情况。

2) 全局跳转结构。全局跳转结构 N 描述了节点与 CFG 中的其他节点之间的一级跳转情况。

$$N = \{N(1,2), N(1,3), \cdots, N(u,v), \forall u,v \in \{1,2,\cdots,|V|\}\}$$

$$N_u = \{e(u,1), e(u,2), \cdots, e(u,|V|)\}$$

其中，N_u 表示节点 u 和 CFG 图中其他节点的一级跳转，节点的总数为$|V|$。比较 N_u 和 N_v 之间的相似度，获得节点 u 与节点 v 之间的全局跳转 $N(u,v)$。直观来说，全局跳转结构表示如果 2 个节点连接了更多相同的节点，则它们之间的联系会更加紧密，这些设想在很多领域都被合理证明。

在 CFG 的有向边 $e(i,k)$中，定义从节点 i 到节点 k 之间的跳转概率为

$$p(i,k) = \frac{1}{1+\exp(-\omega_i \vec{v}_i \omega_k \vec{v}_k)} \tag{2}$$

其中，$\vec{v}_i$ 表示码块 i 特征，$\vec{v}_i = (s_i, p_i, a_i, o_i, l_i)$。式(2)中，使用 sigmoid 函数作为概率函数定义图 CFG 中的跳转概率分布，2 个节点之间连接的经验概

率定义为

$$p'(i,k)=\frac{e(i,k)}{E}$$

其中，

$$E=\frac{\sum_{h=1}^{|V|}e(i,h)}{\sum_{g=1}^{|V|}e(g,k)}$$

经验概率表示节点 i 和 k 在图 CFG 的实际跳转概率。为了计算权重参数，本文将跳转概率分布和经验概率分布之间的距离函数作为目标函数，通过计算 2 个概率之间 KL（Kullback-Leibler）距离，得到损失函数 O_1。

$$O_1=d(p',p)=-\sum_{e(i,j)\in E}p'(i,k)\ln p(i,k)$$

全局跳转结构定义了代码块和 CFG 中其他节点的连接情况，以及该代码块在 CFG 中的上下文。本文定义全局跳转结构的连接概率为

$$p_2(k\,|\,i)=\frac{\exp(\omega_i\overrightarrow{\boldsymbol{v}_i}^{\mathrm{T}}\omega_k\overrightarrow{\boldsymbol{v}_k})}{\sum_{h=1}^{|V|}\exp(\omega_h\overrightarrow{\boldsymbol{v}_h}^{\mathrm{T}}\omega_k\overrightarrow{\boldsymbol{v}_k})}$$

其中，$|V|$ 表示节点的个数。为了保留图 CFG 中的全局跳转结构，需要上下文条件概率分布接近于经验概率分布，上下文经验概率为

$$p'_2(k\,|\,i)=\frac{N(i,k)}{o_i}$$

其中，o_i 表示节点 i 的出度。同样根据 KL 距离定义以下的损失函数。

$$O_2=d(p',p)=-\sum_{e(i,j)\in E}p'_2(i\,|\,k)\ln p_2(i\,|\,k)$$

Pentagon 在编码函数之前，需要先计算每个节点的权重 ω_i。根据一级跳转结构和全局跳转结构获得的损失函数 O_1 和 O_2，对 CFG 中的所有边进行取样，然后对目标函数进行求导，如式(3)所示。

$$\frac{\partial(O_1+O_2)}{\partial\omega_i}=\frac{\partial\left(-\sum_{e(i,j)\in E}\frac{e(i,j)}{E}\ln\frac{1}{1+\exp(-\omega_i\overrightarrow{\boldsymbol{v}}^{\mathrm{T}}\omega_j\overrightarrow{\boldsymbol{v}_j})}\right)}{\partial\omega_i}+\frac{\partial\left(-\sum_{e(i,j)\in E}\frac{N(i,j)}{d_i}\times\frac{\exp(\omega_i\overrightarrow{\boldsymbol{v}_i}^{\mathrm{T}}\omega_j\overrightarrow{\boldsymbol{v}_j})}{\sum_{h=1}^{|V|}\exp(-\omega_i\overrightarrow{\boldsymbol{v}_i}^{\mathrm{T}}\omega_j\overrightarrow{\boldsymbol{v}_j})}\right)}{\partial\omega_i}\quad(3)$$

由于损失函数是凸函数[19]，当式(3)中导数为 0 时，可得到损失函数的最小值。令其一阶导数趋近于 0，求得所有的权重参数 ω_i。然后，使用权重序列 $\Omega=\{\omega_1,\omega_2,\cdots,\omega_n\}$ 编码整个 CFG 获得函数的特征。

3.4　Pentagon 的单调性

由 Ω 编码 CFG 结构计算的函数特征为 $\overrightarrow{\boldsymbol{c}_j}=(c_{js},c_{jp},c_{ja},c_{jo},c_{jl})$。编码过程需要确保每个特征向量能唯一表示一个函数特征。

函数特征由 Ω 组成的一系列的线性等式函数计算而得。由于不同的 CFG 有不同的跳转结构，所以当 Ω 不同时，即有不同的线性等式，可以使用反证法来证明单调性。假设不同的 CFG 有相同的特征，即对于 CGF$_i$ 和 CFG$_j$，$\vec{\boldsymbol{c}}_i=\vec{\boldsymbol{c}}_j$，考虑到节点个数不同其 Ω 也不同，本文使用反证法，假设 2 个 CFG 的节点个数相同，根据 $\vec{\boldsymbol{c}}_i=\vec{\boldsymbol{c}}_j$ 和式(1)，得到式(4)。

$$\begin{aligned}\sum_{k=1}^{n}\omega_{ik}o_{ik}s_{ik}&=\sum_{k=1}^{n}\omega_{jk}o_{jk}s_{jk}\\ \sum_{k=1}^{n}\omega_{ik}o_{ik}p_{ik}&=\sum_{k=1}^{n}\omega_{jk}o_{jk}p_{jk}\\ \sum_{k=1}^{n}\omega_{ik}o_{ik}a_{ik}&=\sum_{k=1}^{n}\omega_{jk}o_{jk}a_{jk}\\ \sum_{k=1}^{n}\omega_{ik}o_{ik}o_{ik}&=\sum_{k=1}^{n}\omega_{jk}o_{jk}o_{jk}\\ \sum_{k=1}^{n}\omega_{ik}o_{ik}l_{ik}&=\sum_{k=1}^{n}\omega_{jk}o_{jk}l_{jk}\end{aligned}\quad(4)$$

其中，n 表示 CFG$_i$ 和 CFG$_j$ 的节点的个数，ω_{ik} 表示 CFG$_i$ 中的节点 k 的权重。由于式(2)中，sigmoid 是单调函数，并且当 2 个 CFG 的结构不同时，这 2 个 CFG 的实际边分布概率也不相同，即 $p_i\neq p_j$，p 是单调函数，则对任意 k，CFG$_i$ 和 CFG$_j$ 中每个节点的权重 $\omega_{ik}\neq\omega_{jk}$。当节点的数目相同时，节点的序列集是相同的，但是每一对有相同的调用序列的函数对都有不同的权重参数序列，并且由于 CFG 的结构不同，即 $o_{ik}\neq o_{jk}$ 且 $l_{ik}\neq l_{jk}$，因此，式(4)不成立。

通过上述反证法，证明了不同的 CFG 能编码为不同的特征，Pentagon 编码的函数是单调的。证毕。

3.5　CFG 编码举例

根据图 2 提取的原始 CFG，以下用 i，j 表示该 CFG 中基本代码块的编号，首先根据一级跳转结构和全局跳转结构计算权重参数，具体的计算过程如下。

$$\frac{\partial(p+p_2)}{\partial\omega_i}=0,\forall i\in\{1,2,3,4,5\}$$

考虑到 $k\omega_i\omega_j \gg 1$，令

$$\ln(1+\exp(k\omega_i\omega_j))=k\omega_i\omega_j$$

并且 CFG 结构中最后一个出口节点的权重 $\omega_5=0$，然后对导数两边同时取对数，有以下的等式。

$$\begin{cases}9\omega_2+10\omega_4+6\omega_5=0\\35\omega_3+37\omega_4+18\omega_5-18\omega_1=0\\20\omega_5+35\omega_2-18\omega_1=0\\25\omega_5+37\omega_2+10\omega_1-60\omega_3=0\\20\omega_3+6\omega_1+18\omega_2-50\omega_4=0\\\Omega=\{\omega_1,\omega_2,\omega_3,\omega_4,\omega_5\}\bmod 4\end{cases}$$

最后，得到权重参数为 $\Omega=[1,2,3,1,0]$，计算函数的特征如下

$$\begin{cases}c_{js}=\dfrac{1\times2\times1+2\times1\times2+3\times2\times3+4\times2\times1+5\times1\times1}{1+2+3+1+0}=5.29\\c_{jp}=\dfrac{1\times2\times1+7\times1\times2+4\times2\times3+4\times2\times1+1\times1\times0}{1+2+3+1+0}=6.86\\c_{ja}=\dfrac{0\times2\times1+2\times1\times2+1\times2\times3+1\times2\times1+1\times1\times0}{1+2+3+1+0}=1.71\\c_{jo}=\dfrac{1\times2\times1+2\times1\times2+1\times2\times3+1\times2\times1+0\times1\times0}{1+2+3+1+0}=2\\c_{jl}=\dfrac{0\times2\times1+0\times1\times0+0\times2\times3+0\times2\times1+0\times1\times0}{1+2+3+1+0}=0\end{cases}$$

3.6　基于函数编码特征的 App 表示

一个完整 App 的 smali 文件中包含了很多函数，每个函数被编码成一维向量特征 $\vec{c}_j=(c_{js},c_{jp},c_{ja},c_{jo},c_{jl})$。根据 App 中所有函数的编码特征，获取每个 App 的矩阵表示 $\boldsymbol{A}$，$\boldsymbol{A}=[\overrightarrow{c_1},\overrightarrow{c_2},\cdots,\overrightarrow{c_m}]$。对于不同大小的 App，所包含函数的个数 m 不同，由于 App 中会有重复函数，重复函数的字符串特征和 CFG 拓扑结构相同，重复的函数编码表现为矩阵 $\boldsymbol{A}$ 中有相同的列。

大量的重复函数需要重复比对，降低了效率和正确率。根据统计调研，App 所包含的函数个数集中在 2 000~20 000，实验的预处理过程首先要删除重复的函数。使用 python 库的 numpy 包中的"unique()"函数来删除每个 App 矩阵中的重复函数。最后 App 的矩阵表示为

$$\boldsymbol{A}=[\vec{c}_1,\vec{c}_2,\cdots,\vec{c}_i,\cdots,\vec{c}_j,\cdots,\vec{c}_m]$$

其中，$\forall\vec{c}_i\neq\vec{c}_j$。

4　Pentagon 的第三方库函数过滤

App 的代码中一般包含了许多相同的第三方库函数，这些相同函数不能作为克隆依据，克隆检测方案需要过滤这些第三方库函数。本节描述如何根据聚类算法删除第三方库函数。

假设 App 数据库足够大，并且第三方库函数在许多 App 中都被使用。由于编码之后的函数特征能够单调表示 App 中的函数，这些函数特征可用于过滤第三方库函数。为了提高效率和准确度，本文主要通过两大步骤来过滤第三方库函数：第一步根据每个函数特征对应的平均权重参数进行粗粒度过滤；第二步在第一步的基础上，进行严格比对聚类。具体如下。

第一步，统计所有函数中相同 ω 出现的频率，使带有相同的 ω 的函数特征聚集在同一类中，第三方库函数会被聚集到一个远远大于其他类别的类中，计算所有不同 ω 的类中的函数个数，选取其中聚类函数个数在前 60%的聚类作为下一步需要比较的对象。第二步，实施严格比对，只有当 2 个函数向量特征完全对应相同时，才会重新将这些函数放置到同一类别中，最后将聚类中函数数目较多的一些函数作为第三方库函数，删除每个 App 表示矩阵中包含这些第三方库函数的特征，剩下的就是 App 核心功能的函数。

5　Pentagon 克隆决策

Pentagon 系统克隆决策由 2 个层面的相似度比较功能构成，首先是函数级别的特征相似度比较，其次是 App 之间的相似度比较。

5.1　函数特征相似度比较

本系统使用余弦相似度函数进行相似度计算，每个函数特征向量 $\vec{c}_j$ 有 5 个值。任意 2 个函数的特征向量 $\vec{c}$ 和 $\vec{c}\,'$ 的相似度定义如下

$$S_f(\vec{c},\vec{c}\,')=\begin{cases}1, 1-|\cos(\vec{c},\vec{c}\,')=\dfrac{<\vec{c},\vec{c}\,'>}{\|\vec{c}\|\cdot\|\vec{c}\,'\|}\leqslant\delta_f\\0, 1-|\cos(\vec{c},\vec{c}\,')=\dfrac{<\vec{c},\vec{c}\,'>}{\|\vec{c}\|\cdot\|\vec{c}\,'\|}>\delta_f\end{cases}$$

其中，δ_f 表示函数之间差异度阈值。如果 2 个函数相同，则 $S_f=1$。

根据 Pentagon 的函数特征编码过程，如果修改了原始执行代码块中操作码的一小部分，函数特征的编码结果不会更改过多；如果改变某一个代码块

中操作码的执行顺序，同样也不会过多地改变函数特征；但如果整个 CFG 的跳转结构发生了大量的变化，或者代码块的统计特征改变了很多，对函数特征就会产生相应的影响。

5.2 App 相似度比较

函数层相似度比较用于检测相似的 App，但是相似的 App 不一定是克隆的 App。有以下 2 种情况不属于 App 克隆。1) 2 个 App 中，一个 App 相比另一个 App 有超过某一固定数目的不同的核心函数。2) App 来自相同的作者，仅是不同的版本。

首先，比较 2 个 App 中函数的个数。App a 和 App a' 中包含的函数个数分别为$|c|$和$|c'|$。如果

$$\frac{\min(|c|,|c'|)}{\max(|c|,|c'|)}<\sigma$$

则判定这 2 个 App 不是克隆的。其中，σ 是任意 2 个 App 的函数数目比例常数，具体根据实验 App 中的函数数目决定,。

其次，需要通过使用函数层的相似度来比较 App 的相似度，具体计算式如下

$$S_a(a,a')=\frac{\sum_{i=1}^{|c|}\sum_{|c'|}^{j=1}S_f(c_i,c'_j)}{\max|c|,|c'|}$$

设定 δ_a 是 App 相似度阈值，如果 $S_a>\delta_a$，这 2 个 App 定义为克隆 App。

6 性能测试与分析

本文实验分为以下 3 个主要部分：App 中所有函数 CFG 特征提取、函数编码、App 克隆检测。在准备阶段，使用 Python 语言编写爬虫程序，该程序自动在 6 个 Android 市场中下载 App。然后，使用 Androgurad 工具，将 App 文件反编译为 smali 文件。根据 smali 文件提取每个函数的 CFG，使用 Pentagon 对每个函数的 CFG 进行特征编码。由于函数间的编码互不影响，Pentagon 能够并行编码每个 App 中的函数。在克隆检测部分，比较 2 个待检测的 App 中的所有函数，来确定这 2 个 App 是否为克隆 App 以及其中具体的克隆函数。

本文提出的原型系统在 Linux 平台实现，并且添加了对 OS X 平台的支持。在应用程序预处理模块，使用了 3 个开源工具 Androguard、Keytool、Dex2jar。Androgurad 工具用来反编译 App 产生原始 CFG，Keytool 用来提取应用程序的签名信息，Dex2jar 用来将得到的 Dalvik 字节码文件反编译成 JAR 包。通过编写脚本，将这些工具组合成一个自动化的工具链，可以批量对应用程序进行预处理。Pentagon 特征提取和相似度分析主要用 Python 来实现。App 样本库的特征提取后保存在云端文件系统中，所有应用的预处理和特征提取只需要进行一次。

6.1 实验环境与数据来源

本文共进行了 2 个阶段的实验，第一阶段是小规模实验，用来验证第三方库函数过滤的效率及本文方法的准确性；第二阶段是大规模实验，用来测试系统的可扩展性和性能。实验数据集包含以下几个数据库：游戏 App 数据库、社交 App 数据库、娱乐 App 数据库、金融数据库和其他数据库。在这 5 个数据库中共下载了 152 789 个应用程序，其中，包含的函数个数为 7 490 721 877 个。数据库 1 中函数个数为 1 596 096 888 个，数据库 2 中函数个数为 1 142 135 436 个，数据库 3 中函数个数为 2 151 246 841 个，数据库 4 中函数个数为 1 120 121 321 个，数据库 5 中函数个数为 1 481 121 391 个。

测试数据包含两部分。一部分是从网络中下载的克隆 App，从知名的 Android 第三方应用市场以及各种 Android 论坛下载有可能是重打包的应用（通过名字、描述来判断，如破解版、修改版等），并且从其他官方市场寻找可能对应的原版应用。另一部分是通过人工重打包应用获取的克隆 App。对下载的合法应用进行一些简单修改（更改变量和函数名、添加删除代码和第三方库等），最后将修改后的应用重新签名并且打包。本文选取大小在 50 KB 到 68 MB 之间的 App 做实验，这个区间的 App 大小基本能代表 Android 市场中一般应用程序的大小。在实验之后手动安装和检查应用代码来验证结果的准确性。实验运行在 12 核 1.6 GHz 的 Linux 服务器，内存为 16 GB，硬盘为 7 TB，同时开启 4 个线程并行处理。

6.2 实验比较

在函数层的克隆分析方面，本文在准确率和效率上对比目前一种较好的基于函数层的 App 克隆检测方法：基于 CFG 质心运算提取特征的 App 克隆检测方案 Centroid[23]。在 App 克隆检测层面，本文对比当前较好的工作：FSquaDRA[12]、Wukong[24]、Centroid[23]。

6.3 重复函数删除

图 3 表示删除重复函数前不同函数数目的 App 出现的次数，也就是对应于某一具体函数数目 App

的个数，横坐标表示每个App中函数的数目，纵坐标表示这一函数数目的App的出现频率。图4表示删除重复函数之后不同函数数目的 App 出现的次数。可以看出在整个样本中函数的个数下降了17.07%～67.27%。某些App中，重复函数的个数超过了一半，一些大的 App 中包含了更多的重复函数。可以看出，重复函数的个数和App的大小呈现下降的趋势。一些只含有20个函数的App中基本没有重复函数。删除一个含有17 863个函数的App中的重复函数的平均时间少于1 s。

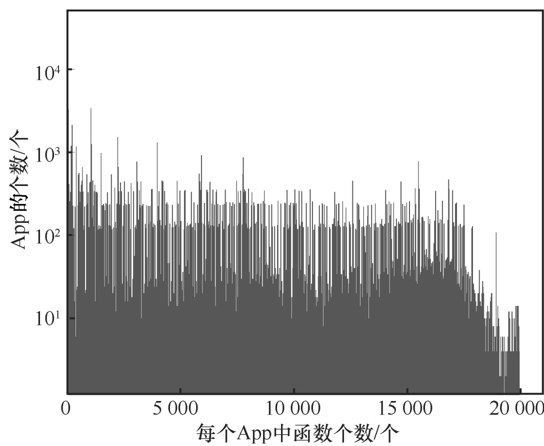

图3　删除前的函数分布

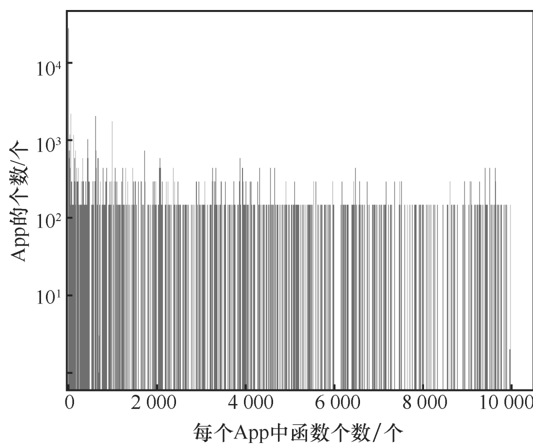

图4　删除后的函数分布

6.4　第三方库函数过滤

本节实验对数据集中所有 App 编码的函数特征进行聚类，用于第三方库函数的过滤，前文说明了不同的权重代表不同的函数，即如果$\omega_i \neq \omega_j$，$\vec{c}_i \neq \vec{c}_j$，则i和j表示2个不同的函数，首先，计算所有不同的权重的出现次数。图5中分别展示了所有ω值对应的所有函数特征的函数数目分布，以及所有ω值对应的不同的函数特征的函数数目的分布。比如，同一个ω可能出现多个函数特征，即有ω的函数特征可能相同也可能不同，因此要区分同一个权重ω对应的函数特征是否相同。图5横坐标表示出现的所有ω的值，纵坐标表示权重为该ω的函数数目，也就是函数的出现频率。从图中可以看出，出现频率较高的函数数目的权重ω为$1\sim10^3$。然后选择权重ω为$1\sim10^3$对应的函数特征作为第三方库函数的特征。其次，在相同ω的聚类中严格比较函数特征，选取聚类中函数各数较大的类作为第三方库函数，并查看这些函数特征对应的函数名和函数可执行代码。实验显示函数个数在前十的聚类都是安卓支持库，或者广告支持库，这些都不在白名单中。为了删除第三方库函数，在聚类之后需要求得聚类大小的阈值作为下一步聚类的条件。

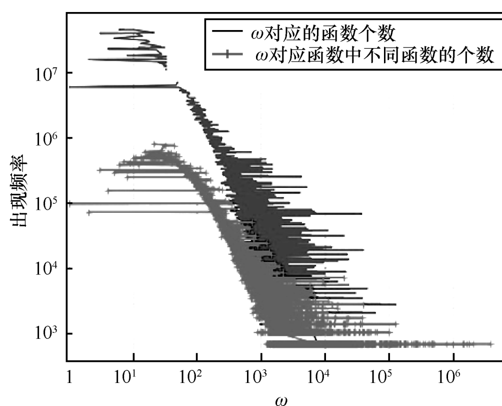

图5　不同权重中所有的函数和不同的函数的数目分布

本实验使用2种方法来决定第三方库函数的过滤阈值。首先，下载300个App，这些App中包含超过 14 个不同的已经确定的第三方库。手动反编译这些App，对相应的函数进行标记，在最后的检测结果中，将出现的频率数目的前29.02%的聚类作为第三方库函数进行过滤。其次，将所有数据库的App的函数特征进行聚类，选择大小在前29.02%的聚类，手动检测这些类别的函数特征对应的函数，标记为“第三方库函数”“关键代码”或者“不能确定”。根据这些标记判断聚类阈值选择的准确性。

结合上述的 2 种方法，最终选择聚类大小的数目为1 610，如果相同函数特征的聚类数目超过了 1 610，则判定为第三方库函数。过滤了第三方库函数后，整个 App 数据库中的函数数目下降了79.26%。过滤前，有超过 63.60%的 App 中包含的函数数目在504~5 973之间，平均每个App有2 259个函数；过滤后，大部分的App包含的函数数目为84~1 976，平均每个App只有627个函数，超过249、419、203个函数被过滤。

6.5　准确率

为了表示 App 函数检测的正确率，用以下 2 个参数来评估实验的好坏，即 TPR（true positive rate）和 FPR（false positive rate），TPR 是在搜索问题中正确找到克隆函数的比例，FPR 表示找到的克隆匹配是不正确的匹配函数的比例，由 TPR 和 FPR 的数据组成的曲线为 ROC 曲线。从测试数据库随机选取一些检测样本序列 q，一个样本实际上有 m 个克隆函数，在样本函数数目 L 中，如果认为前 F 个函数是检测出来的克隆函数，实际上，F 个函数中正确匹配的克隆函数有 c 个，剩下的 $L-c$，就是误报率。设 $\mathrm{TPR}=\dfrac{c}{m}$，$\mathrm{FPR}=\dfrac{F-c}{L-m}$，当 F 越大的时候，TPR 和 FPR 都会逐渐变高，这两者的相加值不为 1。

由于本文是基于函数层编码的 App 克隆检测方法，为了说明本文的准确率，本文将和目前的函数克隆检测的方案进行比较，根据本文提供的测试集，将这些测试集的 App 反编译的函数编码成一维特征后作为输入，进行 App 克隆搜索检测，同时与本文前面提到的方法进行比较，图 6 表示数据库中不同大小的 App 对应的数目。图 7 表示设置不同的函数向量差异阈值时，对应的失败率大小。从图中可以看出，如果函数差异阈值小于 0.005 7，则在函数层的检测失败率为 2.4%。如果函数差异阈值大于 0.02，则函数的错误率大于 20%。函数差异阈值越大，则 FPR 越大。阈值需要确保相似的函数有相近的特征，由于 Pentagon 编码的特征是单调的，不同的函数之间的特征差异比较大，如果阈值被设置得过大，那么相似的函数就被认为不同的函数，这样就会增加 FPR。

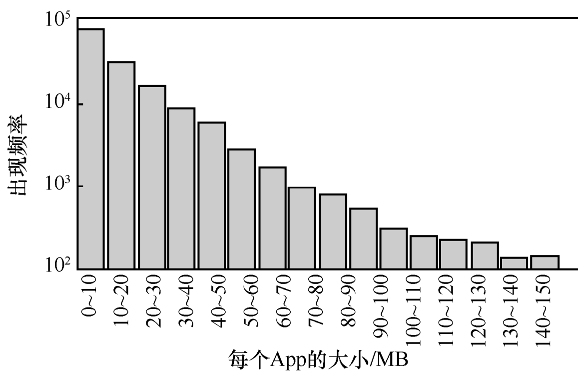

图 6　不同大小的 App 的总数

Pentagon 系统在当前已经投放市场正在使用的游戏 App 中检测发现了 2 个克隆样本（为了保护版权，本文不对这 2 个 App 样本进行具体说明）。这 2 个 App 在应用市场都显示为具有自主知识产权，但经过本文系统检测发现，它们之间存在克隆关系，并且通过动态调试，也确认了这 2 个 App 中存在克隆关系，表现出本文方案的实际应用效果。

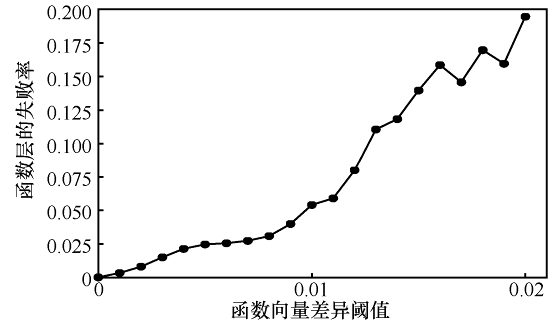

图 7　错误率和阈值之间的关系

图 8 展示了实验结果的 ROC 曲线，可以看到本文提出的 Pentagon 编码方式的正确率高于 Centroid 方案，取 App 差异阈值为图 7 所示的约为 0.01 时候，根据 ROC 曲线可以看到检测的准确率可以达到 97.6%。由于 Pentagon 是单调的，而且对函数的信息编码得更完整，不仅获取了每个代码块的主要统计特征信息，还获取了每个 CFG 的结构特征，所以 Pentagon 的准确率很高。Centroid 并不是严格单调地表示特征，相比较，本文提出的编码方案能更正确地表示函数的特征。

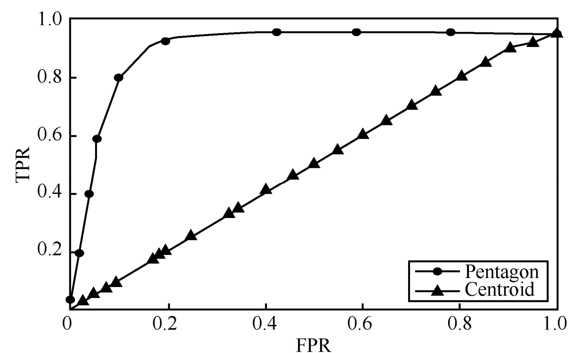

图 8　ROC 曲线

6.6　函数层效率

本文主要从以下 3 个方面对系统效率进行比较：CFG 特征的提取效率；Pentagon 编码时间；函数克隆检测时间。这三方面指标表示 App 克隆检测的预处理效率。本文有关效率的实验使用全部的 152 789 个 App，对所有的 App 提取特征进行存储，并生成用于 App 克隆检测的数据库。

1) CFG 特征提取时间。图 9 展示了 Pentagon 的 CFG 提取时间。该特征提取时间不包含 App 生成 CFG 时间，因为 Pentagon 的编码只需要给 CFG 的每个代码块提取 5 个值，Centroid 给 CFG 的每个代码块提取 3 个值，但 Centroid 在给 CFG 中的基本代码块进行编号的过程稍微复杂一些，所以需要更多的时间，但总体来说，2 种方案的 CFG 特征提取时间相差不大。

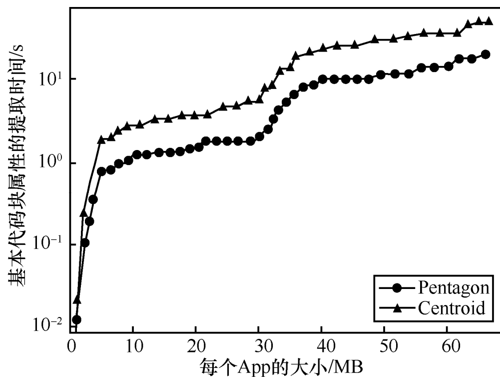

图 9　每个函数的 CFG 中代码块的特征提取时间

2) Pentagon 编码时间。图 10 展示了 Pentagon 生成时间随着函数的数目增加而增长。Pentagon 将 APP 中一个函数的 CFG 编码为一个特征，Pentagon 是一种线性计算的编码，当函数个数有 20 000 个时，即使串行编码，Pentagon 的编码时间仅仅需要 1 h。由于 Pentagon 的编码方式是解耦合的，因此利用并行的方式可以大大缩短编码时间。

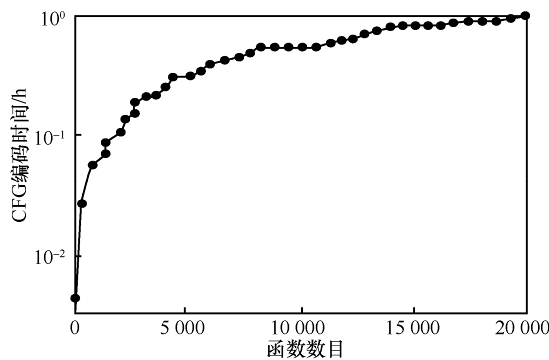

图 10　Pentagon 编码时间

3) 搜索时间。图 11 表示 Pentagon 的编码特征用于 App 克隆检测出的搜索时间，使用 6 个不同度量的样本大小来检测搜索效率，这 6 个文本中函数个数 c 为 $10^3 \sim 10^8$，随机选择 1 到 10 000 个函数序列作为提交的搜索序列，平均搜索时间随着函数数目的增加持续增长，在 1 000 个函数中平均的搜索时间为 7.9×10^{-9} s。

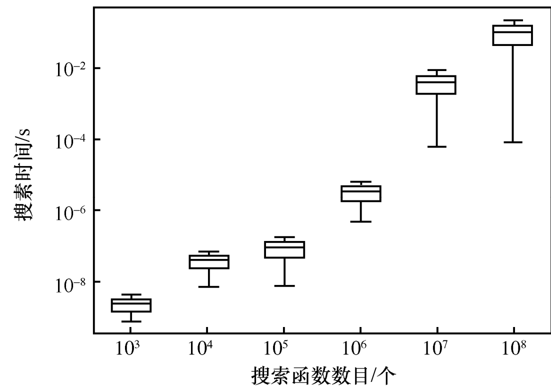

图 11　Pentagon 函数层的克隆搜索检测时间

6.7　App 克隆检测时间

当获取了所有的 App 函数特征之后，先按照本文方案删除第三方库函数，每个 App 大概需要 0.01 s 去除重复的函数，删除了第三方库函数之后，检测一个 App，需要比较检测的 App 和待检测的 App 之间所有的函数，实验表明，本文方案计算 2 个 App 之间的相似值需要 0.079 s。表 1 比较了当前几个比较好的 App 克隆检测方案 FSquaDRA[5]、Wukong[20]、Centroid[4]和本文方案之间的效率。

表 1　App 克隆检测效率

App	检测时间/s	预处理时间/s
Pentagon	0.079	0.74
FSquaDRA	0.4	4
Centroid	0.085	5.82
Wukong	0.29	0.99

7　结束语

与现有的 App 克隆检测方案相比较，本系统具有以下 2 个突出的优势。1）在对第三方库函数过滤的方案中，不局限于已有的白名单进行过滤，过滤更为精确有效。现有方案使用白名单通过 App 函数中的包名来过滤第三方库函数，在准确性上面存在问题，因为首先不可能完整全面地确定第三方库函数，其次，混淆可能改变包函数的名字，使过滤方案失效，本文发现部分第三方库函数没有具体的名称。 2）本文方案不仅考虑了基本代码块之间的静态特征，同时也考虑了 CFG 中代码块之间的跳转结构，现有的 App 检测方案主要考虑静态特征，比如提取不同的 API Call 调用顺序来检测克隆，这些特征同样会因为混淆而失效。本文的方法采用 CFG 函数结构编码，并融合代码块属性特征，相比

较其他方案在克隆分析上可靠性更高，实验结果也验证了上述结论。

Pentagon 能够快速应用于云环境下大规模 App 函数层克隆检测，实现了更准确高效地查找克隆 App 以及定位具体克隆函数的目标，对维护移动互联网应用知识产权的良好生态具有积极支撑意义。下一步将在特征自动化提取、第三方函数库优化、跨平台克隆检测等方面做进一步工作，以促进 Pentagon 系统应用推广。

参考文献：

[1] CHEN Y, GAN L, ZHANG S. Plagiarism detection in homework based on image hashing[C]// The Third International Conference of Pioneering Computer Scientists, Engineers and Educator. ICPCSEE, 2017: 424-432.

[2] ANDREWS S, VARUNBABU B S,SUBASH U. Finding the high probabilistic potential fishing zone by accelerated SVM classification[J]. International Journal of Information and Communication Technology. 2017, 11(4):576-585.

[3] PAPALEXAKIS E E,FALOUTSOS C,SIDIROPOULOS ND. Tensors for data mining and data fusion: models, applications and scalable algorithms [J]. ACM Transaction Intelligence, System, Technology, 2017, 8(16):1-4.

[4] FU D, XU Y. WASTK: a weighted abstract syntax tree kernel method for source code plagiarism detection [J]. Scientific Programming, 2017:1-8.

[5] FANG L,LIN Y,YANG J. An efficient and packing-resilient two-phase Android cloned application detection approach [J]. Mobile Information Systems, 2017:1-12.

[6] DU Y, WANG J LI Q. An Android malware detection approach using community structures of weighted function call graphs [J]. IEEE Access, 2017:1.

[7] LYU F, LIN Y, YANG J. Suidroid: an efficient hardening-resilient approach to Android App clone detection[C]// 2016 IEEE Trustcom BigdataSE ISPA. IEEE, 2016: 511-518.

[8] SIVIC J ZISSERMAN A. Video google: a text retrieval approach to object matching in videos[C]// The Ninth IEEE International Conference on Computer Vision. IEEE, 2003: 1470-1477.

[9] SOH C, TAN H B K, ARNAATOVICH Y L. Detecting clones in Android applications through analyzing user interfaces[C]//2015 IEEE 23rd International Conference on Program Comprehension. IEEE, 2015: 163-173.

[10] CHEN Q, WANG J, WANG Y. An online approach for detecting repackaged Android applications based on multi-user collaboration[C]// Ubiquitous Intelligence and Computing and 2015 IEEE International Conference on Autonomic and Trusted Computing. 2016: 312-315.

[11] DREBIN A. Reverse engineering, malware and goodware analysis of Android applications and more[R]. Technical Report, 2013.

[12] ZHAUNIAROVICH Y, GADYATSKAYA O , CRISPO B. Fsquadra: fast detection of repackaged applications[R]. Working Conference on Data and Applications Security and Privacy, 2014.

[13] KHOO W, MYCROFT M A, ANDERSON R. Rendezvous: a search engine for binary code[C]// The 10th Working Conference on Mining Software Repositories (MSR). IEEE, 2013: 329-338.

[14] PEWNY J,GARMANY B, GAWLIK R, et al. Cross-architecture bug search in binary executables[C]//2015 IEEE Symposium on Security and Privacy. IEEE, 2015: 709-724.

[15] FENG Q, ZHOU R. Scalable graph-based bug search for malware images[C]// The 2016 ACM SIGSA.2016: 480-491.

[16] ZHOU W, ZHOU Y, GRACEM. fast, scalable detection of piggy- backed mobile applications[C]//The Third ACM Conference on Data and Application Security and Privacy. 2013: 185-196.

[17] DAVID Y, YAHAV E. Tracelet-based code search in executables[C]// The 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2014: 349–360.

[18] ARP D,SPREITZENBARTH M,HUBNER M. Drebin: effective and explainable detection of Android malware in your pocket[C]//In Network and Distributed System Security Symposium. 2014.

[19] LUKAS W, MATTHIAS N, MARTINA L. Andrubis: Android malware under the magnifying glass[R]. Vienna Univeristy of Technology, 2014.

[20] ENCK W, GILBERT P, CHUN B. Taintdroid: an information tracking system for realtime privacy monitoring on smartphones[C]//The 9th USENIX Conference on Operating Systems Design and Implementation.2010: 393-407.

[21] TANG J QU M. LINE: large-scale information network embedding[C]//The 24th International Conference on World Wide Web. 2015: 1067-1077.

[22] LIU Z, CHEN C. POSTER: neural network-based graph embedding for malicious accounts detection[C]//The 2017 ACM CCS. 2017: 2543-2545.

[23] CHEN K, LIU P, ZHANG Y. Achieving accuracy and scalability simultaneously in detecting application clones on android markets[C]// 36th International Conference on Software Engineering. 2014: 175-186.

[24] WANG H, GUO Y, MA Z. Wukong: a scalable and accurate two-phase approach to Android App clone detection[C]//The 2015 Internatinal Symposium on Software, Testing and Analysis. ACM, 2015: 71-82.

[作者简介]

杨佳（1993- ），女，湖北武汉人，华中科技大学博士生，主要研究方向为网络空间安全中的恶意代码的克隆搜索、恶意代码在移动互联网中的传播以及大数据的隐私保护。

付才（1976- ），男，湖北武汉人，博士，华中科技大学教授，主要研究方向为移动网络安全、路由器算法以及分布式计算。

韩兰胜（1973- ），男，湖北武汉人，博士，华中科技大学副教授，主要研究方向为网络安全、恶意代码检测以及大数据安全。

鲁宏伟（1964- ），男，湖北武汉人，博士，华中科技大学教授，主要研究方向为网络空间安全、IoT、社交网络、安全协议分析。

刘京亮（1984- ），男，北京人，北京航空精密机械研究所高级工程师，主要研究方向为非接触测量方法、测量机软件工程、智能检测技术。