基于动静结合的 Android 恶意代码行为相似性检测

作者	陈鹏，赵荣彩，韩金，孟曦
机构	数学工程与先进计算国家重点实验室
发表期刊	《计算机应用研究》
预排期卷	2018 年第 35 卷第 5 期
访问地址	http://www.arocmag.com/article/02-2018-05-053.html
发布日期	2017-05-25 09:22:43
引用格式	陈鹏，赵荣彩，韩金，孟曦. 基于动静结合的 Android 恶意代码行为相似性检测[J/OL]. [2017-05-25]. http://www.arocmag.com/article/02-2018-05-053.html.
摘要	针对同家族恶意软件行为具有相似性的特点进行研究，提出通过静态分析和动态运行程序相结合的方式度量软件行为的相似性。通过反编译和 soot 代码转换框架获取程序控制流图，利用行为子图匹配算法从静态方面对程序行为相似性进行度量；通过自动化测试框架运行程序，利用文本无关压缩算法将捕获到的 trace 文件压缩后进行相似性度量。该检测方法综合静态检测执行效率高和动态检测准确率高的优点，提高了软件行为相似性度量的效率和准确率。实验分析表明，该检测技术能够准确度量程序之间行为的相似性，在准确率上相较于 Androgua…
关键词	Android API 调用，控制流图，子图匹配，系统调用，trace 文件
中图分类号	TP309.2
基金项目	国家“863”计划资助项目（2009AA012201）；国家自然科学基金资助项目（61472447）

基于动静结合的 Android 恶意代码行为相似性检测 *

陈　鹏，赵荣彩，韩　金，孟　曦*

(数学工程与先进计算国家重点实验室，郑州 450000)

摘　要：针对同家族恶意软件行为具有相似性的特点进行研究，提出通过静态分析和动态运行程序相结合的方式度量软件行为的相似性。通过反编译和 soot 代码转换框架获取程序控制流图，利用行为子图匹配算法从静态方面对程序行为相似性进行度量；通过自动化测试框架运行程序，利用文本无关压缩算法将捕获到的 trace 文件压缩后进行相似性度量。该检测方法综合静态检测执行效率高和动态检测准确率高的优点，提高了软件行为相似性度量的效率和准确率。实验分析表明，该检测技术能够准确度量程序之间行为的相似性，在准确率上相较于 Androguard 有大幅提升。

关键词：Android API 调用；控制流图；子图匹配；系统调用；trace 文件

中图分类号：TP309.2

Android malware behavior similarity detection based on dynamic and static combination

Chen Peng, Zhao Rongcai, Han Jin, Meng Xi*

(State Key Laboratory of Mathematical Engineering & Advanced Computing, Zhengzhou 450000, China)

Abstract: According to the characteristics of similar behavior of racial malware，this paper presented a method to measure the similarity of software behaviors by means of static analysis and dynamic operation.The program control flow graph was obtained by decompiling and soot transcoding framework,the behavior similarity of the program was measured from the static aspect by using the behavior subgraph matching algorithm.Through an automated testing framework to run the program,using the text-independent compression algorithm to measure the similarity of the captured trace files.The method had the advantages of high static efficiency and high accuracy of dynamic detection, the efficiency and accuracy of software behavior similarity measurement had improved. Experimental analysis shows that the detection technique can accurately measure the similarity between programs and the accuracy rate compared to Androguard has greatly improved.

Key Words: android API calls; control flow graph; subgraph matching; system calls; trace files

0　引言

2016 年 IDC 发布最新数据显示，Android 总的市场份额超过 80%，并预计 2019 年 Android 市场份额将达到 82.6%占据绝对优势。随着移动终端时代的到来，移动安全远远落后于移动互联网络的发展速度成为越来越突出的矛盾[1]。例如 2016 年 9 月席卷全球的手机“霸屏”病毒在俄罗斯、墨西哥等一些国家造成极其严重的危害，该病毒会显示一个置顶窗口，让用户无法通过屏幕操作手机，全球每天的感染手机在 3 万台以上[2]；2016 年初到 2016 年 10 月“九头虫”病毒利用多家知名 root SDK 对上万种机型轻松提权，向系统分区植入多款恶意 app，并禁用掉国内多家知名杀软，致使设备安全防护功能全线瘫痪[3]；2016 年 12 月一款新的 Android 恶意软件 Gooligan 能够劫持谷歌账号，从各类谷歌 App 中获取敏感信息，包括 Gmail、Google Photos、Google Docs、Google Play 和 Google Drive，目前已经入侵超过 100 万谷歌账户，仍以每日 13000 台设备的速度蔓延[4]。可见 Android 恶意软件已经渗透到了人们生活的方方面面，安全问题层出不穷，无论是 OS 还是 APP 层面，都可能爆发新一轮更大规模的安全事件。

Android 应用的发布通过开发者提交给应用市场，用户进入各类应用市场下载 apk。除了谷歌推出的 Google play 外，国内还有众多第三方应用市场，而第三方应用市场面对海量应用无法进行全面的安全审核。目前各类恶意软件生成技术不断发展成熟，攻击者很容易批量对 Android 应用修改插入恶意代码。目前第三方市场上以家族变种存的形式在着大量恶意应用软件，它们在行为表现上具有相似性，如何通过代码检测手段快速的识别出行为相似的 Android 恶意程序成为亟待解决的问题。

基金项目：国家“863”计划资助项目（2009AA012201）；国家自然科学基金资助项目（61472447）

作者简介：陈鹏（1990-），男，河南南阳人，硕士，主要研究方向为信息安全（buder_cp@sina.com）；赵荣彩（1957-），男，教授，博士，主要研究方向为高性能计算、信息安全；韩金（1993-），男，安徽蚌埠人，硕士，主要研究方向为恶意代码检测；孟曦（1992-），女，山东济宁人，硕士，主要研究方向为恶意代码分析.

1 相关研究

同族恶意软件的行为表现具有相似性，因此可以采用基于行为相似性的检测手段，利用已知行为模式评估软件的恶意性和可能产生的危害。基于行为相似的恶意代码检测又划分为动态和静态两种方法。动态检测方法通过在真实或者模拟环境中运行软件，关注程序运行的异常行为，可以不受代码加密混淆的影响，但是动态检测效率低且不易全面触发恶意行为，很难进行自动化测试；静态检测方法不需要执行程序，主要通过反编译技术对 smali 或 java 代码进行控制流和数据流分析，代码执行效率高可以快速完成分析和检测，但是无法完全还原经过加密混淆过的代码，无法检测在运行过程中释放的恶意行为，因此静态检测准确率较低。

Zhou 等人[5]提出了 DroidMoss 基于行为相似性的静态重打包应用检测工具，利用逆向工具对 apk 进行反编译进而提取行为特征码，实验中作者选取了多个第三方市场的 22906 个 Android 应用，最终得到了的 5%到 13%的重打包检测率。文献[6]实现了一个程序相似性检测系统 BuaaSim，采用编译优化和反汇编技术将源程序转换为汇编指令集合，使用一个与指令顺序无关的决策函数计算程序相似度，还给出一个简单有效的聚类算法从程序集合中聚类出相似的程序子集。Lin 等人[7]在系统调用级别上提出了重打包恶意应用检测系统 SCSdroid，该系统首先提取出代表同一家族的恶意系统调用序列，利用已经获取的系统调用序列与同族其他软件序列对比，在同族恶意软件重打包检测中取得了良好的效果。Huan 等人[8]采用云端协作的软件架构，在云端对正常软件的行为模式进行分析，形成正常行为模式；在终端，从云端下载正常行为模式库，检测已安装应用，计算异常率。该领域中较为前沿的研究理论和方法还有 Hanna 等人[9]提出的基于特征值哈希算法，利用文件目录的结构对应用进行相似性评估；Hu 等人[10]利用图之间的编辑距离对恶意程序之间相似性进行度量。

结合上述研究，本文提出了基于动静结合的 Android 恶意软件行为相似性检测技术。同族样本的行为具有相似性，对同族代码分别进行应用框架层 API 和内核层系统调用序列的相似性度量，综合利用静态代码检测的高效性和动态行为检测的准确性设计完成了一套行为相似性度量系统。

2 基本思路

程序行为检测可以通过静态代码分析或是动态运行分析。利用反编译技术将恶意程序还原为高级语言，对 API 调用的解析可以高效全面的描述程序的运行轨迹，但是目前市场上无论是攻击者还是开发人员均会利用各种加密混淆等方式保护自己的代码不被逆向分析。基于这个事实，本文结合动态运行的方式触发恶意行为，记录触发恶意行为的系统调用序列以全面掌握程序运行状态，充分利用动态运行不受代码混淆和加密的影响，结合静态检测的高效，实现不同层次的 Android 恶意代码行为的相似性检测。系统框架层 API 调用与内核层系统调用的关系如图 1 所示。

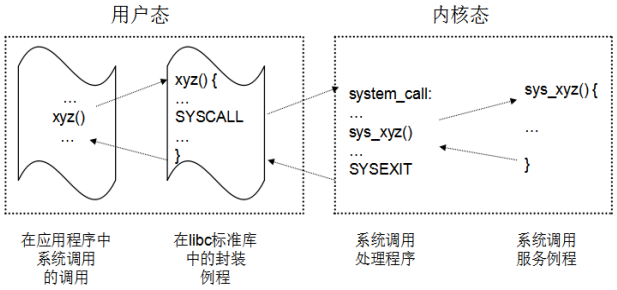


图 1 系统调用及用户函数关系

为度量程序行为的相似性，本文将整个过程划分静态检测和动态检测两个部分：

静态检测部分首先利用 apktool 进行反编译，对 AndroidManifest.xml 文件进行分析，提取应用包名、组件、类名、方法等信息作为动态检测的运行参数；其次利用 soot 开源框架对反编译的结果进行代码转换，绘制出基于 API 的控制流图；最后经过行为子图的过滤，利用子图匹配算法对应用程序的行为相似性进行度量。

动态检测部分主要采用沙盒技术，首先利用 appium 测试框架自动运行程序，应用程序在模拟器中自动运行触发恶意行为；然后提取系统调用序列，利用文本无关压缩算法对原始 trace 文本处理，对压缩后的 trace 结果集进行相似性度量；最终利用简单平均法结合动静两方面的度量结果得到待检测软件的相似性。整个系统流程如图 2 所示：

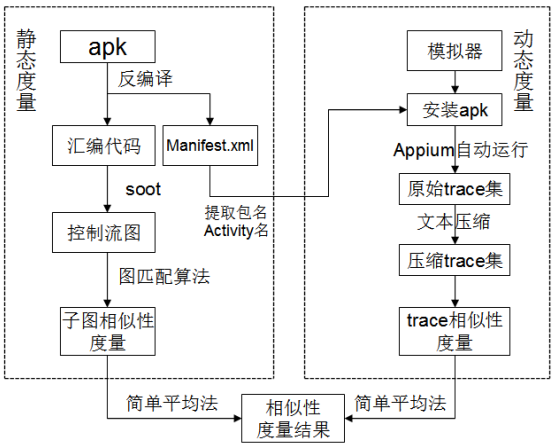


图 2 基于动静结合的恶意代码行为相似性检测模型

3 系统设计与实现

3.1 静态行为控制流图

首先利用 apktool 反编译应用程序以获取 AndroidManifest.xml 和 smali 文件。每个应用程序都包含一个 AndroidManifest.xml 文件，位于根目录下，它定义了应用程序的内容和行为；smali 就是 Dalvik VM 内部执行的核心代码，它有自己的一套语法，经过深度的优化，可以在 Dalvik 虚拟机中更快的运行且占用更少空间。

本文依据 Soot 开源框架提供的 API 对行为子图进行绘制。

Soot 是 McGill 大学的 Sable 研究小组开发的 Java 二进制代码分析和转换框架，它为分析和优化 Java 字节码提供了四种不同的中间语言表示，目前 soot 加入了扩展插件以支持对 Dalvik/Android 字节码的分析和转换，它包含一个入口点函数生成器，并且支持显式和隐式信息流的自动跟踪。通过 Soot 提供的过程内和过程间的分析与优化支持，本文可以很方便地获得每个方法的控制流图和各个折衷级别上的全局调用图[11]。

Soot 在程序包 soot.toolkits.graph 中提供了生成控制流图的接口：图的入口点和出口点，给定节点的后继者和前导者，迭代器遍历图形节点的数量。利用 Soot 创建的控制流图 CFG（Control Flow Graph）可以用 G=（N，E，Nentry,Nexit）表示。其中，N 为节点，程序中的语句块对应图中的一个节点；E 为边集，E={<n1,n2>|n1,n2∈N 且 n2 是 n1 的后继节点}；Nentry 是程序入口点；Nexit 是程序出口点。Soot 构建 CFG 的抽象工具类是 UnitGraph，它有三个不同的实现类：BriefUnitGraph、ExceptionalUnitGraph 和 TrapUnitGraph。为一个给定的方法体构建 CFG，需要将该方法体传递给 CFG 构造函数：UnitGraph g = new ExceptionalUnitGraph(body)；利用 Soot 包中的 CFGViewer 工具类为反编译出来的 Java 文件中的每个方法生成 DOT 语言描述的控制流图。DOT 是一种抽象的图形描述语言，内部存储的是基本块和有向边，利用可视化工具将其转换为图形格式的控制流图。编写简单的 Java 程序，利用 Soot 为如下 Java 程序生成控制流图。

```
public class hello {
    public static void main (String[] args) {
        for (int i = 0; i < 100; i++) {
            int a = 0;
            a = i % 3;
            switch (a) {
            case 0:
                System.out.println(“Zero”);
                break;
            case 1:
                System.out.println(“First”);
                break;
            case 2:
                System.out.println(“Second”);
                break;
            default :
                break;
            }
        }
    }
}
```

如图 3 所示即为上述 Java 程序生成的控制流图。

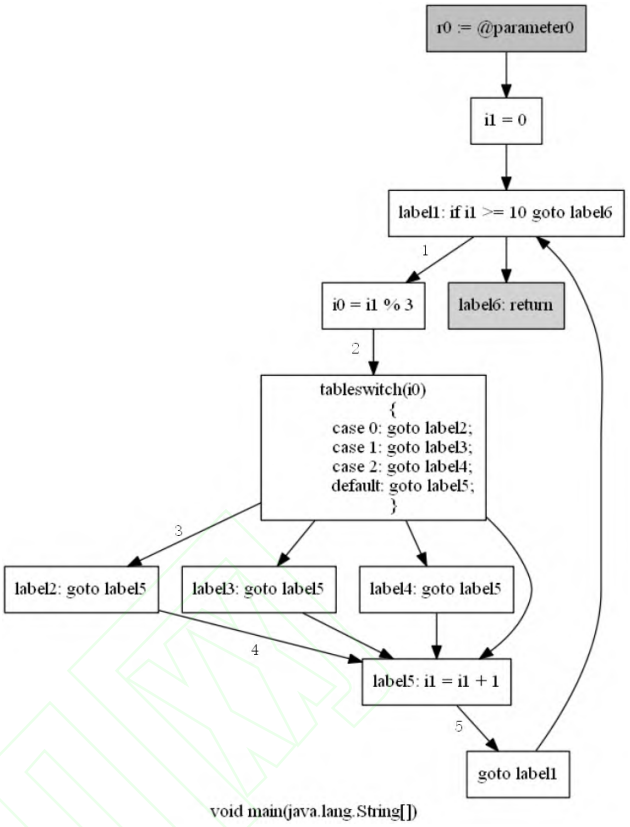


图 3 Java 代码控制流图

3.2 静态行为相似性计算

利用 soot 代码转换框架绘制出 apk 的所有行为子图，要比较两个程序行为的相似度就需要对比所有子图中相似子图所占的比例。下面本文根据两个子图之间的最大公共子图来计算图之间的相似度。

定义 1 公共子图。 非空图 G1 和 G2 中，如果 G1 中存在一个子图与 G 同构，且 G2 中也存在一个子图与 G 同构，则称 G 是 G1 和 G2 的公共子图

定义 2 最大公共子图。 如果 G 是 G1 和 G2 的公共子图，且不存在一个所含节点数比 G 还多的 G1 和 G2 的公共子图 G’，那么就称 G 是 G1 和 G2 的最大公共子图，记为 G=mcs(G1,G2)。

定义 3 图的相似度。 令|G|为图 G 中节点的数量，非空图 G1 和 G2 间的最大公共子图为 mcs(G1,G2)，那么它们的相似度 similarity(G1,G2)为：

$$Similarity(G_1,G_2)=|mcs(G_1,G_2)|/\max(|G_1|,|G_2|)$$

定义 4 图的包含度。 图 G1 在 G2 中的包含度 containment(G1,G2)定义为：

$$Containment(G_1,G_2)=|mcs(G_1,G_2)|/G_1$$

如果：

$$Containment(G_1,G_2)\geq\gamma,\gamma\in(0,1]$$

就说 G1 相对于 G2 是 γ -同构的。

控制流图刻画了程序的行为特征，同族程序相似性的比较可以转换为控制流图的相似性度量。如前所述，利用 Soot 代码转换框架将反编译的 Java 代码转换成为控制流图，也即该程序的所有行为子图，通过相似行为子图占所有行为子图的比例来度量程序静态行为的相似性。

由于不同程序的 CFG 图不可能完全相同，因此本文期望两个 CFG 是 γ -同构的，也就是根据定义 4 中描述的计算方法，两个图之间的相似度大于 γ 即为 γ -同构。

采用 ssLCHY 算法[12]找出两个程序中的所有同构子图：P 和 Q 为两个待检测相似性的程序，K 是 CFG 中最少应该拥有的节点数（设置为 6），只有当 CFG 的节点数大于这个值时才被认为是值得分析的。γ 是检查子图是否同构时的宽限参数。

DETECT（P,Q,K, γ ）

(1) 对每个函数 pi∈P（qj∈Q）构造他的控制流图 Gi(Hj)。

(2) 令 R 为所有可能的图的配对组合（Gi，Hj）组成的集合。

(3) 将 R 中所有不可能是相似行为子图的的部分删掉：

(a) 将所有 Gi 或者 Hj 中节点数小于 K 的(Gi，Hj)从 R 中删掉；

(b) 将所有|Hj| < γ |Gi|的(Gi，Hj)从 R 中删掉；

(4) 将 R 中剩下的配对逐个地进行比较：

对于每一个配对(Gi，Hj)∈R 进行下列操作：

if Hj 与 Gi 不是 γ -同构的 then

R ← R - {(Gi，Hj)}

(5) 把 R 作为所有可能的候选相似子图组成的集合返回。

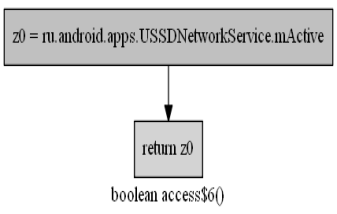
算法中的步骤（3）为子图过滤操作，由于图的相似性比较是 NP 问题会消耗较多处理器时间和内存资源，因此为了提升系统效率，需要对原始行为子图集合进行筛选，减少图之间的比较次数，将与程序行为无关或者无法代表程序恶意行为的子图删除。

由于在使用 soot 框架绘制行为子图时会出现较多只有很少节点的行为子图，这类子图并不构成危险行为所需的必要步骤，因此本文将子图中节点个数少于 6 的子图删除以减少比较次数。研究者 Kymie[17]在对软件行为模型的研究过程中发现，短序列模式的行为中至少需要 6 步以上的调用序列才能对软件的行为进行判别，因此本文将行为节点数少于 6 的行为子图进行删除，以减少需要匹配的子图数量。

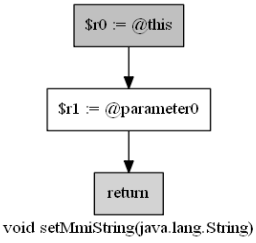
根据得到的行为子图，可以将不影响实际检测效果的行为子图分为如下六种类型：

a）节点数过少的子图，由于恶意行为的产生一般需要较长的行为过程，因此在行为子图中节点数过少的子图便可以删除掉，如下(a）(b)即为所要删除的子图；

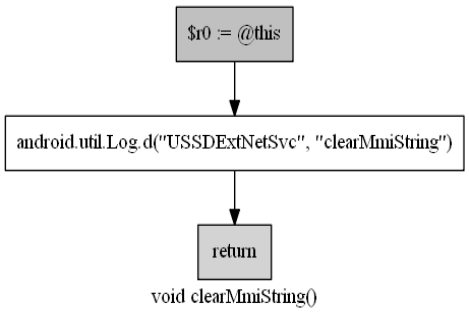
b）源代码作者调试程序时的行为子图，如下图（c）所示，这类行为子图是软件作者在开发过程中的调试代码信息，一般不产生恶意攻击行为。



(a)

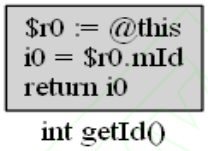


(b)

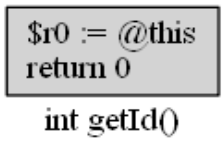


（c）

c）Android 程序获取资源 ID 的行为子图，Android 程序开发时需要获取程序的资源 ID，这类行为也不构成恶意行为路径，如图（d）、（e）所示：

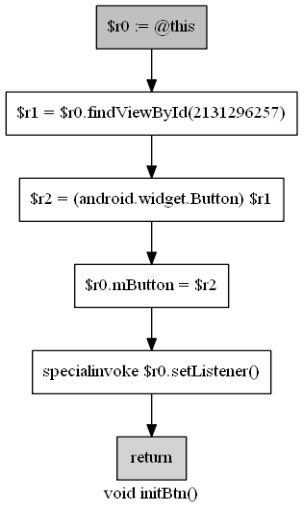


(d)



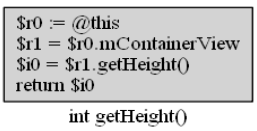
(e)

d）初始化组件操作如图（f），在开发过程中需要对布局组件进行初始化操作，主要是通过布局组件的 ID 查找对应的控件，进行页面布局和逻辑代码的链接，因此查找布局控件的操作行为一般不构成恶意代码的逻辑流程，需要在子图集合中删除此类子图。

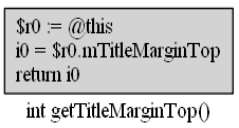


（f）

（e）获取控件位置的操作行为如图（g）（h），这类行为只是获取屏幕中的控件宽度和高度以及其他应用控件的图形位置信息，一般不构成恶意行为。



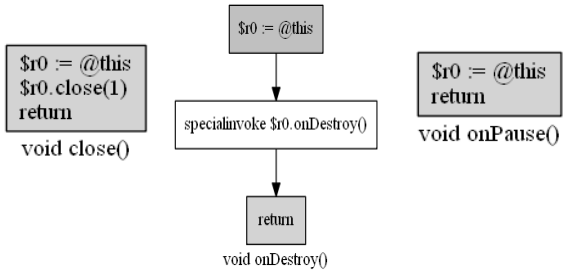
(g)



(h)

f）结束初始化操作行为子图，如图（i）、（j）、（k）所示，一般如网络连接释放、数据库资源请求释放、组件生命周期结束等情况，一般含有 destory()，finish()，close()等关键字，

由于此类型子图不会构成恶意行为，这类图的相似性匹配也没有意义，将此类型子图删除。



(i) (j) (k)

通过子图的过滤阶段后，将得到一个候选子图集合，在接下来的图同构检测中利用公式计算出两个应用程序的相似性：

$$\text{CFGSimilarity}(P,Q)=\frac{\sum_{k=1}^{r} mcs(Pk,Qk)}{\sum_{i=1}^{n} Pi+\sum_{j=1}^{n} Qj-\sum_{k=1}^{r} mcs(Pk,Qk)}$$

3.3 动态行为特征提取与度量

目前市场上的大部分应用都会使用代码混淆和加密手段以防止被逆向分析，不仅仅是开发人员，攻击者为了阻止安全人员对恶意代码的分析与研究也会加强自己的代码保护，例如使用 NDK 编程、Dex 文件保护、第三方加密平台等措施，致使在很多情况下 apktool 等反编译工具都会失效而无法获取程序的逻辑代码，也就无法使用静态手段对程序的行为进行相似性度量，因此仅使用静态手段将无法全面分析程序的恶意行为。

动态运行程序可以解决代码加密和混淆的难题。平行地运行两个程序，利用文献[13]提供的基于 appium 测试框架自动运行程序，记录系统调用序列比较它们的 trace 记录，以此度量它们的相似度。但是 trace 可能会产生极为庞大的数据集，冗长的 trace 记录给相似性分析带来极大阻碍。本文需要将庞大的 trace 结果压缩成更紧凑的易于相似性度量的表示形式。

算法 RELJ[14]使用内容无关语法压缩 trace 数据，它比未经压缩的原始数据要小，有利于进一步分析。整个语法的核心需要满足下列两个属性：

p1：没有哪两个相邻的符号对会出现两次，同时

p2：每个规则都能增加压缩率，即每个规则都会被使用至少两次。

在此基础上本文对此算法进行了改进，具体描述如下(T 为 trace 结果):

COMPRESS（T）：

（1）产生一个语法 G，在一开始，起始规则 S→ ε 。

（2）将 T 中的下一个元素赋给 c。

（3）将 c 附加到 S 的结尾处。

（4）如果在 G 中，某个二字母组合出现了两次，则添加一个规则 R→D，并把所有 D 替换成 R，以满足属性 p1。

（5）如果规则只使用了一次，那就将该规则删除。以满足属性 p2。

（6）调回步骤（4），直到属性 p1 和 p2 均被满足。

（7）调回步骤（2），直到 T 全部被添加到 G。

（8）将 G 返回。

X86 架构的系统调用命令有 360 个，将提取的 trace 序列作出字符映射表，每个字符表示一个系统调用名。假设所采样的一个合法调用序列为：

execve,read,write,read,write,open.close,open,close,futex,pselect6

所对应的映射数字为：

execve->1,read->2,write->3,open->4,close->5,futex->6,pselect6->7

这样原 trace 序列转换成了一串数字序列 12323454567，在序列中每前进一步，都会把 trace 中的下一个系统调用添加到起始规则的 S 尾部（符号^表示它之前的所有序列已经添加到 S 中了），序列压缩过程如表 1 所示。

表 1 trace 压缩过程

结果	语法
（1）^12323454567	{ S -> ε
（2）1^2323454567	{ S -> 1
（3）12^323454567	{ S -> 12
（4）123^23454567	{ S -> 123
（5）1232^3454567	{ S -> 1232
（6）12323^454567	{ S -> 12323 => $\begin{cases}S->1AA\\A->23\end{cases}$
（7）123234545^67	$\begin{cases}S->1AA4545\\A->23\end{cases}$ => $\begin{cases}S->1AABB\\A->23\\B->45\end{cases}$
（8）12323454567^	$\begin{cases}S->1AABB67\\A->23\\B->45\end{cases}$

从步骤 f）开始，起始规则中开始出现重复的二字符组合 23，为了满足性质 p1，需要在语法中增加一个规则 A->23，继续将 trace 结果的下一个元素添加到 S 尾部，当出现重复的二字符组合时，增加新的规则如（7）所示。算法的压缩率与 trace 结果中重复出现模式的个数和次数相关。在文献[15]中 Jim Larus 称，对于 SPEC 基准测试而言，该算法的压缩率在 7.3 至 392.8 之间。

利用上述文本压缩算法所返回的字符串结果集，Pi={Pi|程序 Pi 的 trace 压缩字符串}，Qj={Qj|程序 Qj 的 trace 压缩字符串}，计算两程序 trace 结果集的相似度：

$$\text{TraceSimilarity}(P_1,Q_1)=\frac{P_1\cap Q_1}{(|P_1|+|Q_1|)/2}$$

最终使用简单平均法将静态和动态分析结果的数值综合，得出程序的相似性：

$$\text{Similarity}(P,Q)=\frac{\text{CFGSimilarity}(P,Q)+\text{TraceSimilarity}(P,Q)}{2}$$

4 实验与测评

4.1 实验环境

实验主机配置为：CPU 主频为 2.30GHz 的双核 Intel(R)

Core(TM) i5-6500(HQ)，内存 16.00GB，Linux 操作系统为 Ubuntu 16.04 LTS，内核版本为 Linux version 4.4.0-21-generic，模拟器系统为 Android 7.0 N。

本文所分析的 3 个同族恶意代码样本均选自 FakeInstaller 恶意代码家族。FakeInstaller 是一个广泛传播的恶意软件系列，主要存在于俄罗斯。迈克菲所处理过的针对 Android 系统的恶意样本中，超过 60%均是来自 FakeInstaller 系列[16]。该系列恶意软件有大量变体，通过将自身伪装成主流的应用程序（如 Flash Player、Opera 浏览器、Skype 等），在数以百计的网站和假冒市场进行传播，并向付费号码发送短消息获利。为了横向对比，本文编写了 2 个无恶意属性并且功能类似的 Android 程序。表 2 为本实验选取样本的详细信息。

表 2 实验样本信息

样本名称	MD5	大小	样本描述
Trillian_1.2.0.5_rus_kol	64de83eba9a760fb35319a2a73984252	5643KB	伪造美国即时通信软件 Trillian 版本：1.0
Download	fd430a928d403fc657ba228d93f88e33	381KB	伪造“会说话的汤姆猫” 版本：1.0
QIP	d8f1e0a2b82e65f48242c53fd2fc8e3a	33KB	伪造俄罗斯即时通信软件 QIP 版本：2.0
My_app	cffac9795e87350e05dd4510c9d9afb0	582KB	编写的正常软件
Hello_app	f80bc8b6df2cff81bb4d27fa6cf8b82f	502KB	编写的正常软件

4.2 实验结果及分析

针对上述实验样本使用 apktool 进行反编译，利用 soot 框架进行代码转换绘制程序的控制流图，使用 ssLCHY 算法找出两个程序中的所有同构子图。静态分析结果如表 3 所示。

表 3 静态分析结果

对比程序名称	行为子图数	对比子图数	相似度
1）Download/QIP	28/20	25/16	65%
2）Download/Trillian	28/80	25/35	63.4%
3）My_app/Hello_app	30/27	20/18	87.6%
4）My_app/Download	30/28	20/25	2%

a）Download 和 QIP 为同族的恶意软件，两个应用均伪装成正版软件的登陆界面，用户开启后立即弹出用户许可窗口，显示用户是否接受“正版 APP”的使用协议并告诉用户将会发送一条或多条短信，用户点击后病毒立马生效。两款病毒在后台向特定号码发送付费短信的同时显示一个虚假的下载进度条，最终被重定向到一个假冒市场。由于程序逻辑类似，因此这两款软件的行为相似性较高。

b）Download 和 Trillian 为同族软件，不同的是 Trillian 在用户登录时有两个选项，但最终引导用户必须点击同意许可才能使用或是导向“正版软件”的下载地址，逻辑上和 Download 有细微差别，因此相似度没有第一组程序高。

c） My_app 和 Hello_app 是自己编写的功能类似的软件，没有经过加密和混淆处理，反编译较为全面的反映了源程序函数调用序列，利用 soot 完整的刻画了原程序的控制流图，因此该组程序的相似度很高。

d）My_app 和 Download 分别为自己编写的程序和 FakeInstaller 家族程序，实现完全不同，因此相似度很低。

为了比较同族样本的动态行为特征，本文在模拟器中安装待检测程序，利用自动化测试框架 appium 执行样本程序，采用 strace 命令对四组运行中的程序进行跟踪和系统调用序列的搜集，trace 结果按照频数统计如图 4 所示：

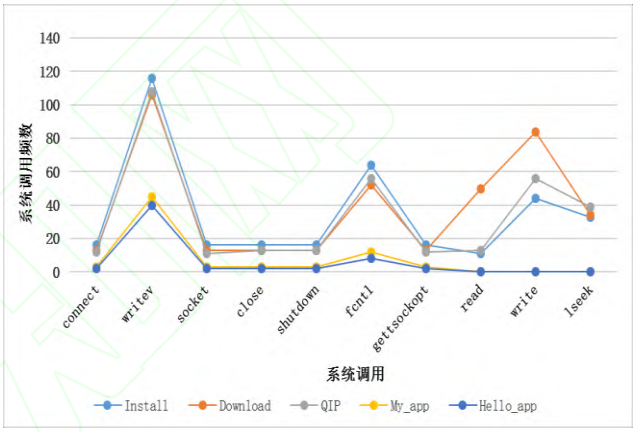


图 4 系统调用命令频数统计

从图中可以看出同族的恶意软件之间折线的重合度较大，系统调用频数大致相同，说明同族内软件的相似度较大；不同家族之间的系统调用折线相差较为明显，反映出不同类型的软件之间的关联性较低，系统调用的使用频数反映了软件之间的相似性。

为了对本文提出的行为相似性检测技术的有效性进行评测，使用著名的 Androguard 恶意软件分析工具对上述实验样本进行了对比测试。在操作系统 Santoku 中集成了大量的 Android 程序分析工具，其中包含了版本为 1.5 的 Androguard。系统中提供了 androsim.py 工具，它能够计算两个 apk 文件的相似度，利用该软件计算的相似度结果和本文的实验方案进行对比，两种检测方法对软件相似性检测结果如图 5 所示。

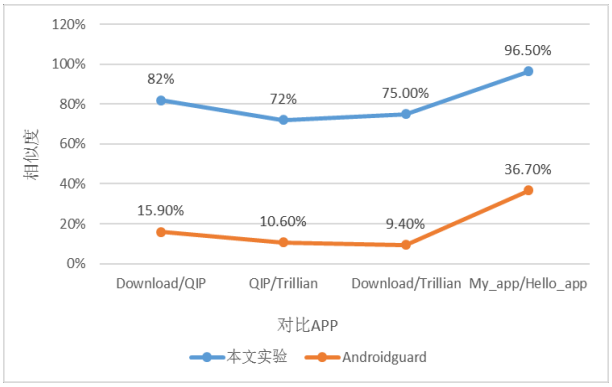


图 5 本文方法和 Androguard 检测的准确率对比

从实验结果可知，Androguard 对本实验所选取样本相似度检测的准确率明显低于本文所使用方法。从同族软件的相似性来看，实验结果与预期相同，较为准确的判断同族软件之间行为的相似性。

5 结束语

本文针对同族软件行为具有相似性的特征，提出了基于动静结合的 Android 恶意软件行为相似性检测技术。在静态检测方面利用 soot 代码转换框架完成程序行为子图的绘制，通过子图匹配算法度量程序行为的相似性；动态运行时捕获系统调用序列，通过对压缩后 trace 文本的相似性度量完成动态行为相似性检测。实验选取相同家族和不同家族的样本对比分析，最终结果表明，在软件行为相似性检测的准确率上，本文的检测方法明显优于现有软件相似性度量工具 Androguard。

下一步的研究工作将从以下方面展开：a) 子图相似性过滤算法有些简单，有待进一步提高，以减小匹配范围，进一步提高系统的执行效率；b)面对逻辑和界面更复杂的程序，还需设计覆盖率更高的测试框架，全面自动化运行程序，进一步提高检测准确率。

参考文献

[1] 周圣韬. Android 安全技术揭秘与防范[M]. 北京: 人民邮电出版社.

[2] 猎豹移动安全实验室. 席卷全球的“霸屏”病毒: 技术分析与处理[EB/OL]. (2016-08-30)[2017-02-17]. http: //www. freebuf. com/articles/terminal/118773. html.

[3] 阿里聚安全. “九头虫”病毒技术分析报告 [EB/OL]. (2016-11-02)[2017-02-17]. http: //www. freebuf. com/articles/terminal/117028. html.

[4] Check Point Research Team. More Than 1 Million Google Accounts Breached by Gooligan[EB/OL]. (2016-11-30)[2017-02-17]. https: //blog. checkpoint. com/2016/11/30/1-million-google-accounts-breached.

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

[6] 赵长海, 晏海华, 金茂忠. 基于编译优化和反汇编的程序相似性检测方法[J]. 北京航空航天大学学报, 2008, 34(6): 711-715.

[7] Lin Y D, Lai Y C, Chen C H, *et al*. Identifying Android malicious repackaged applications by thread-grained system call sequences[J]. Computers & Security, 2013, 39: 340-350.

[8] 桓自强, 倪宏, 胡琳琳, 等. 一种基于行为的 Android 重打包应用检测方案[J]. 计算机应用与软件, 2016, 33(8): 298-301.

[9] Hanna S, Huang L, Wu E, *et al*. Juxtapp: A scalable system for detecting code reuse among android applications[C]//Proc of International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin: Springer, 2012: 62-81.

[10] Hu X, Chiueh T, Shin K G. Large-scale malware indexing using function-call graphs[C]//Proc of the 16th ACM Conference on Computer and Communications Security. 2009: 611-620.

[11] 李远玲, 陈华, 刘丽. Soot 的 Java 程序控制流分析及图形化输出[J]. 计算机系统应用, 2009, 18(10): 88-92.

[12] Liu C, Chen C, Han J, *et al*. GPLAG: detection of software plagiarism by program dependence graph analysis[C]//Proc of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2006: 872-881.

[13] 张胜桥, 尹青, 常瑞, 等. 基于 APK 的 Android 应用程序 GUI 遍历自动化方法[J]. 计算机应用, 2016, 36(11): 3178-3182.

[14] Nevill-Manning C G, Witten I H. Compression and explanation using hierarchical grammars[J]. The Computer Journal, 1997, 40(2-3): 103-116.

[15] Larus J R. Whole program paths[C]//ACM SIGPLAN Notices, 1999, 34(5): 259-269.

[16] Fernando Ruiz. Android. FakeInstaller 恶意伪装序. [EB/OL]. (2012-10-12)[2017-02-17]. http: //www. ccidnet. com/2012/1012/4350505. shtml

[17] Tan K M C, Maxion R A. Determining the operational limits of an anomaly-based intrusion detector[J]. IEEE Journal on selected areas in communications, 2003, 21(1): 96-110.