NeighborWatcher:基于程序家族关系的附加恶意手机应用检测方法研究

张 焕[1],武建亮[1],唐俊杰[1],班 涛[2],俞 研[3],郭山清[1,4],王利明[5],胡安磊[5]
(1.山东大学计算机科学与技术学院,山东济南 250100;2.日本信息与通信技术研究所,日本东京 1848795;
3.南京理工大学计算机学院,江苏南京 210094;4.山东省软件工程重点实验室,山东济南 250100;
5.中国互联网络信息中心,北京 100010)

摘 要: 经过对多个手机恶意应用程序的分析,发现其与被感染程序所属家族的不同版本在程序语义方面存在很大的相似性,并且这种相似性与原家族中不同版本之间的相似性有很大不同.基于该事实,本文借助于分层聚类技术,针对函数的调用图,提出了一种基于程序家族关系的恶意手机应用检测方法并构建了一个 NeighborWatcher 系统.实验结果表明当每个程序家族都含有四个以上的成员时,NeighborWatcher 系统对附加恶意应用的检测率可以达到 92.86%.

关键词: 附加恶意应用程序; 方法调用图; 家族聚类; 手机安全
中图分类号: TP309 **文献标识码:** A **文章编号:** 0372-2112 (2014)08-1642-05
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2014.08.029

NeighborWatcher: Detecting Piggybacked Smartphone Applications with Their Family Members

ZHANG Huan[1], WU Jian-liang[1], TANG Jun-jie[1], BAN Tao[2], YU Yan[3], GUO Shan-qing[1,4], WANG Li-ming[5], HU An-lei[5]
(1. *Department of Computer Science and Technology, University of Shandong, Jinan, Shandong* 250100, *China*; 2. *Japanese Institute of Information and Communications, Tokyo* 1848795, *Japan*; 3. *Department of Computer, Nanjing University of Science and Technology, Nanjing, Jiangsu* 210094, *China*; 4. *Shandong Provincial Key Laboratory of Software Engineering, Jinan, Shandong* 250100, *China*; 5. *China Internet Network Information Center, Beijing* 100010, *China*)

Abstract: Through the analysis of some mobile malwares, we found that malware is similar with its original application in semantics of the program, and the similarity is different with the similarity between other members of the family. Based on this fact, by means of hierarchical clustering technology for the function call graph, we propose a program based on family relationships to detect the malicious mobile applications and build a system named as "NeighborWatcher". Experimental results show that when each family contains four or more members, the detection rate of Piggybacked application can reach 92.86%.

Key words: piggybacked application; call function graph; family clustering; mobile security

1 引言

Nielsen 最近的调查显示,在美国手机操作系统中,安卓已达到了 51.8%的市场份额,远远超过了苹果的 iOS 系统(34.3%)和 RIM 的 Black-berry 系统(8.1%)[1].Lookout[2]最近的一份报告指出,安卓应用市场的增长速度是苹果应用市场的 3 倍.大部分专家将这一现象归结于安卓系统的开源性及其相对简单的检查机制,简单的检查机制为用户带来方便的同时也给攻击者以可乘之机.攻击者可以利用已有的工具[3]向手机应用软件植入恶意代码,利用市场传播感染用户手机.在此,我们将这种植入恶意代码的攻击行为称为附加攻击(Piggybacked),将这种被植入恶意功能的应用程序称为附加恶意应用(piggybacked 应用).由于从互联网上发现的手机恶意应用大多属于附加恶意应用,这使得检测此类恶意应用成为一项迫在眉睫的工作.

目前对于附加恶意应用检测的研究已取得了一些成果,如 DNADroid[4]和 PiggyApp[5].DNADroid 主要是通过建立程序依赖图来检测恶意应用的工具,但其是建立在 Google Player 上所有程序是绝对安全的基础上的.

收稿日期:2013-04-17;修回日期:2013-12-31;责任编辑:赵克
基金项目:国家自然科学基金(No.61173139, No.61303243);山东省科技攻关计划(No.2010GGX10117);互联网基础技术开放实验室基金(No.K201206007)

PiggyApp 设计了一个概念验证原型来检测恶意软件,避免了对 Google Player 的依赖.我们的工作也从降低该依赖性出发,通过对多个附加恶意应用实例进行分析,我们发现附加恶意应用在保留原应用功能的同时,多通过改变原程序中的文件、图片、语言或者注入一段恶意代码来实现,这使得其与该程序家族不同版本的程序在语义表达,如函数调用图等上存在高相似性,基于此,本文提出了一种基于程序家族关系的附加恶意应用检测方法并实现了一个称之为 NeighborWatcher 的系统,实现了 92.86%的恶意软件检测率.

2　相关工作

近年来,针对面向对象语言和函数式语言,多种函数调用图的构造算法已被提出,包括 Grove[6] 提出的一种通用参数方法.Grove 使用点阵理论模型证实了一个程序的调用图越准确耗时越多.针对 Java 语言,IBM 研发了静态分析库 WALA[7],其可通过分析指针来解决构造函数调用图时所面临的动态调用问题.

在程序的家族分析方面,Gupta[8] 等利用反病毒厂商所提供的数据研究了恶意程序的演化过程.Dumitras[9] 等通过构造恶意程序的演化过程发现了一些著名的恶意程序的变种.Lindorfer[10] 等调查了一个恶意程序的不同版本的演化过程.Jiyong Jang 等开发的 ILINE[11] 系统研究了构造二进制程序族谱过程中所面临的四个基础问题.

在恶意软件识别方面,随着手机应用的大幅增加,大量基于安卓手机平台的恶意软件检测工作应运而生.例如 DNADroid,通过比较程序依赖图来检测克隆代码,以此来发现被植入恶意代码的程序.由 Chaudhuri 等人提出的 SCanDroid[12],通过静态分析安卓应用程序的方法来帮助识别安卓平台下的隐私泄漏问题.

3　基于家族聚类的附加恶意应用检测方法

图 1 是 NeighborWatcher 的系统结构图.从图中可以看出,该系统一共包括四个部分:预处理、二进制程序语义分析、程序家族分类和恶意程序检测.

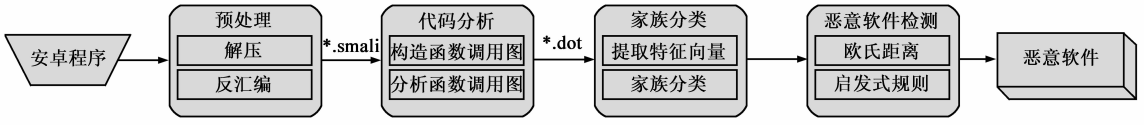

图1　Neighbor Watcher系统的结构图

预处理主要包括对 APK 程序的解压和反汇编两部分.二进制程序分析部分实现了对反汇编结果的语义分析,提取出了函数调用图.程序家族分类部分实现对所获得的函数调用图进行正则化,然后利用分层聚类算法完成了对程序家族关系的重建.恶意程序检测部分则利用定义的规则实现了附加恶意应用的检测.

3.1　反汇编

APK 是安卓应用的后缀,我们之后用 APK 表示安卓应用程序.由于不能得到每个 APK 所对应的源程序,在构建方法调用图前我们需要先对 APK 进行反汇编操作.在此,我们选择使用 apktool 进行反汇编.通过 apktool,每个 APK 都会得到一些以 smali 为后缀的文件.这些文件中包含着一些类 Dalvik 指令的代码,我们称为 smali 代码,这些代码将成为构造函数调用图的基础.

3.2　函数调用图构造

函数调用图由两部分组成,一个是代表方法的点,一个是代表方法间调用关系的边.简单方法调用图的构建可首先遍历所有 smali 文件,找到每个文件中的函数,即可得到所有点(每个点的信息包括:函数名、父类、所属类、被调用函数列表、调用的函数列表等),接着分析所有 INVOKE-like 语句来获得函数调用图的边.

Android 应用的特殊性使如上所述的构造函数调用图的方法并不完善.一方面,类似于 java 语言,安卓语言中的继承结构使调用情况不明确,对此,NeighborWatcher 参考 Woodpecker[13] 中的方法,根据继承关系构成一个层次类,当出现不确定的调用时,NeighborWatcher 会通过匹配参数类型、个数等来确定调用图的所有可能边.另一方面,函数调用图中的点只包括开发者自己定义的函数,即系统函数并没出现在调用图中,而在实际情况中,系统函数不仅频繁出现,而且直接影响调用图的连通性,解决这一问题最简单的办法就是将所有的系统函数都用一个节点表示,当出现非开发者定义的函数时,将被认为是系统方法.具体方法见算法 1.

算法 1　构建函数调用关系

输入:r 代表被函数 f 调用的函数集合;
　　d 代表调用函数 f 的函数集合;
　　s 代表系统函数.

算法:

```
function FINDRELATIONSHIP
    r←θ;
    d←θ;
    for each instruction i do
        if i is INVOKE-like then
            m←class + method;
            if m is the system methed
                m←s;
            ConstructRelationship(m, r, d);
```

```
            end if
            if m dese NOT appearance in its class AND appearance in its superclass then
                m←superclass + method;
                ConstructRelationship(m, r, d);
            end if
            if m's class is Native type then
                m←subclass + method;
                ConstructRelationship(m, r, d);
            end if
        end if
    end for
end function
function ConstructRelationship(m, r, d)
    if r does NOT contain function m AND d does NOT contain function m then
        r←r∪m;
        d←d∪m;
    end if
end function
```

算法 1 描述了构建函数调用图方法中最重要的部分.算法中 FINDRELATIONSHIP()函数遍历所有指令找到 INVOKE-like 指令.分析 INVOKE-like 指令得到被调函数 m,接下来需要分析 m 的情况.情况一,m 是否为系统函数,如果所有函数列表中都没有该函数,则是系统函数,直接让函数 f 与系统函数间建一条边即可;情况二,m 所在类中不包含该函数,则判断 m 是否是其父类中的函数,如果是则重新定义函数 m 的名字,即由原来的 *class* + *method* 变为 *superclass* + *method*;情况三,若函数 m 所属类的类型为 NATIVE,则需要在函数 m 的所有子类中寻找该函数,同情况二一样需要改写函数 m 的名字.边的增加则由函数 ConstructRelationship(m, r, d)实现,构建边之前先判断边是否已经存在,避免重复.

3.3 家族聚类

家族聚类主要实现将函数调用图相似的 APK 文件聚集在一起,而图匹配问题是一个 NP-hard 问题.本文借鉴 Polymorphic[14] 中提到的一种有效的近似求解方法,即通过统计方法调用图中同构 k 点子图的个数将图用一个特征向量来表示,这个特征向量中每个元素为方法调用图中同构 k 点子图的个数.得到每个 APK 的特征向量后,我们使用聚类工具 CLUTO[15] 进行聚类,最终形成一个参考数据集.

3.4 检测附加恶意程序

附加恶意程序与其原程序所在家族的成员有很高相似度,所以其在家族聚类时会被看作与原程序隶属于同一个家族,但其多表现为在原有程序上进行局部插入,基于此,首先选出一个家族中相似度小于某一阈值的程序 A 与 B,并结合其它的一些启发式的信息,包括 INTERNET 权限等实现对附加恶意应用的判别.这里,两程序的相似性由两程序对应向量间的欧氏距离决定.

4 性能分析

4.1 数据集

从 2012 年 12 月 8 日到 2012 年 12 月 29 日,我们从 N 多、安极、木蚂蚁和安卓市场共下载了 76,269 个安卓应用软件,大小共计 442GB.

由于 apktool 自身的局限性,8,976 个 APK 反汇编失败,因此最终共得到 67,293 个有效向量.为了更好的体现家族性,减少个别程序对家族分类的影响,我们选择家族成员数在 4 个以上的 44,385 个 APK.这种选择方式并不影响本算法在实际环境中的应用效果,因为攻击者在选择附加对象时,大多选择比较流行的程序,而这类程序一般会存在 4 个以上的版本.

4.2 家族聚类结果评估

为了对家族聚类的结果进行评估,分析参考聚类结果与实际聚类结果的相似度,我们定义了两个数值:Precision 值和 Recall 值.我们用(R_1, R_2, …… R_r)来表示参考聚类结果,用(C_1, C_2, …… C_c)来表示实际聚类结果,其中 R_i 或者 C_i 表示一个家族.

Precision 值用来验证聚类的正确性,即将不同功能的应用聚类到不同家族的性能,其表达式如下,其中 n 代表应用程序总数:

$$Precision(C_i) = \max(|C_i \cap R_1|, |C_i \cap R_2|, \cdots, |C_i \cap R_r|) \quad (1)$$

$$Precision = \frac{1}{n} \sum_{i=1}^{c} Precision(C_i) \quad (2)$$

Recall 值用来测量将类似功能的应用聚类到同一家族的性能. *Recall* 表达式如下:

$$Recall(R_i) = \max(|C_1 \cap R_i|, |C_2 \cap R_i|, \cdots, |C_c \cap R_i|) \quad (3)$$

$$Recall = \frac{1}{n} \sum_{i=1}^{r} Recall(R_i) \quad (4)$$

我们随机选取了 72 个家族 368 个 APK 来测试算法的性能.表 1 列出了当 k 取不同值时家族聚类的时间消耗.从表中我们可以看出,当 k 大于等于 12 时,消耗的时间会达到 114.70 秒,显然该取值使得本算法无法应用于大规模的 APK 分析,所以,本文取的 k 值要小于 12.图 2 显示了家族聚类结果,黑线代表 Precision 值,灰线代表 Recall 值.横坐标表示 k 的不同取值.从图 2 中可以看出,当 $k=8$ 时,Precision 值和 Recall 值都可以达到 80% 以上.综合图 2 和表 1 的结果,我们选择 $k=8$.

为了更直观的表示聚类效果,我们定义相似值(Simi)来比较两个应用程序的相似性.我们以应用程序 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 为例,分别用向量 $\boldsymbol{A}(A_1, A_2 \cdots A_r)$ 和向量 $\boldsymbol{B}(B_1,$

$B_2,\cdots B_r$)表示,相似值定义如下:

表 1　处理程序的平均时间消耗

k 的值	时间消耗(s)
4	16.428
6	16.544
8	22.857
10	43.183
12	114.170

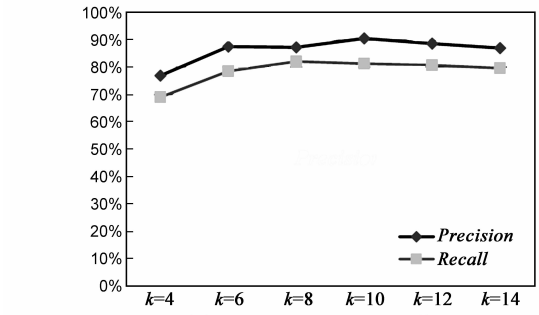

图2　*Precision* 和 *Recall* 值

$$\mathrm{Simi}(\boldsymbol{A}/\boldsymbol{B}) = \frac{\sum_{i=1}^{r}\min(A_i, B_i)}{\sum_{i=1}^{r}\max(A_i, B_i)} \quad (5)$$

图 3 展示了两个实例.在图 3(a)中,一共有 3 个家族,同一家族用同种形状的点表示,可见上面三个为同一家族,中间 3 个为同一家族,最后一个为同一家族.从图中可以看出,同一家族间的两成员的 Simi 值通常会比不同家族中两成员的 Simi 值大.

图 3(b)展示了另外一个例子.在图中看到,中间家族的成员与底层家族的成员也有很高的相似度,但它们的包名不同,此时我们猜测其中一个家族是另一家族重新打包后的产物.通过手动观察其中两个应用程序的界面、功能、权限等信息,我们发现底层家族确实是附加恶意程序,而其原程序正是来自中间的家族.

4.3　恶意程序检测

为测试对附加恶意应用的识别能力,先从大约 1000 个恶意程序中随机选取 52 个进行实验.结果显示,如果我们规定每个家族成员个数大于 2 个,那么这 52 个恶意程序都可以找到其原程序;如果我们将每个家族的最少成员数定义为 3 个,可以找到 36 个;如果每个家族的最少成员数为 4,我们找到了 28 个恶意程序对应的原程序.

图 4 显示了除掉那些误报、漏报等不准确的情况恶意程序的正确检测率.在图 4 中,横坐标代表每个家族最少成员数,纵坐标代表恶意程序检测率.可见当家族最少成员数达到 4 时,恶意程序的准确检测率会高达 92.86%,并且随着家族最少成员数的增加,这一结果会更高,可见漏报误报的情况对检测率影响不大.

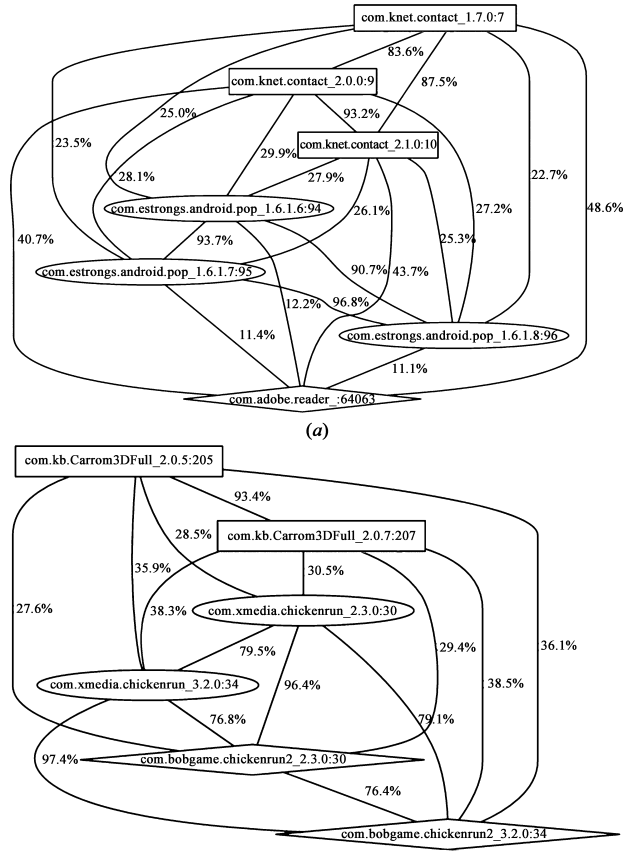

图3　相似图

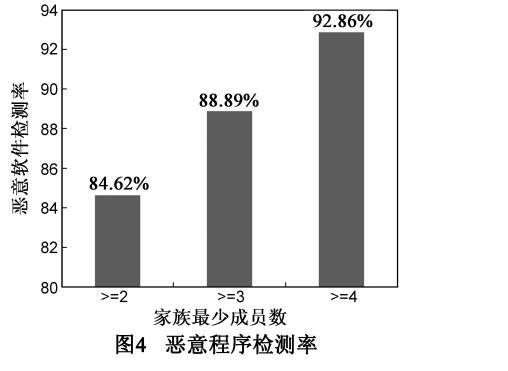

图4　恶意程序检测率

4.4　实现细节及时间消耗

NeighborWatcher 由预处理、二进制程序语义分析、程序家族分类和恶意程序检测四个部分组成,共包括 2,365行 java 代码和 1,459 行 C 代码.反汇编使用免费 APK 反汇编工具 apktool;接下来的两步由 java 语言和 C 语言实现;最后的检测由 java 语言实现.

在时间消耗方面,反汇编是时间消耗最大的部分,平均每个程序需要 9s,构建函数调用图大概需要 4s 的时间,将 368 个应用程序聚类到 100 个家族共耗费了 22.857s.

5 不足

尽管本文所提出的方法取得了很好的实验结果,但仍有许多不足需要改进.

(1)目前方法依赖于静态反汇编技术,而随着手机应用程序自保护机制的流行,将来会有越来越多的程序不能被反汇编,为此,将来需要考虑进一步引入动态分析的技术来改善本方法.

(2)NeighborWatcher 系统摒弃了官方应用程序都是安全的这一假设,而是以同一家族中不同版本间有很高相似度作为假设进行构建.事实上,有些版本会跟其他版本有很大差别,这会对实验结果产生很大影响.为此我们将来在家族聚类时,尝试加上软件的其它信息分析,如包名等,使聚类结果更准确.

6 结论

当前,Android 免费应用越来越多,恶意软件的发展趋势也日趋明显.针对 Android 环境下的恶意应用多借助于附加攻击技术实现,本文基于函数调用图,利用分层聚类技术提出了基于程序家族关系的恶意手机应用检测方法并构建了 NeighborWatcher 系统.实验结果表明,当每个程序家族都含有四个及以上的家族成员时,对附加恶意应用的检测率可以达到 92.86%.

未来,我们将从如下两个方面进一步完善所提出的方法:(1)针对目前方法难以分析使用了自保护机制的程序,将尝试引入动态分析技术来提高可分析对象的覆盖率;(2)针对程序的语义刻画不够精确,尤其是缺少对 Intent 等的分析与处理这一问题,未来将进一步引入污点追踪技术来完善本方法.

参考文献

[1] Nielsen. Who is winning the u. s. smartphone battle? [EB/OL]. http://blog. nielsen. com/nielsenwire/onlinemobile/who-is-winning-the-u-s-smartphone-battle/. 2011-03-03.

[2] Lookout. App genome report[CP]. https://www. mylookout. com/, 2011-02-16.

[3] apktool[CP]. http://code. google. com/p/android-apktool/, 2012-12-14.

[4] J Crussell, C Gibler, et al. Attack of the clones: Detecting cloned applications on android markets[A]. ESORICS 2012[C]. Berlin: Springer, 2012. 37 – 54.

[5] W Zhou, Y Zhou, et al. Fast, scalable detection of "piggybacked" mobile applications [A]. CODASPY'13 [C]. NewYork, USA: ACM, 2013. 185 – 196.

[6] D Grove, C Chambers. A framework for call graph construction algorithms[J]. ACM Trans Program Lang Syst, 2001, 23(6): 685 – 746.

[7] T J Watson libraries for analysis wala[EB/OL]. http://wala. sourceforge. net/, 2011-07-17.

[8] A Gupta, P Kuppili, et al. An empirical study of malware evolution[A]. COMSNETS 2009[C]. Piscataway: IEEE, 2009. 1 – 10.

[9] T Dumitras, I Neamtiu. Experimental challenges in cyber security: A story of provenance and lineage for malware[A]. CSET'11[C]. Berkeley, CA, USA: USENIX Association, 2011. 9 – 9.

[10] M Lindorfer, A Di Federico, et al. Lines of malicious code: Insights into the malicious software industry[A]. ACSAC'12[C]. New York, NY, USA: ACM, 2012. 349 – 358.

[11] Jiyong Jang, Maverick Woo, et al. Towards Automatic Software Lineage Inference[A]. USENIX Security'13[C]. Berkeley, CA, USA: USENIX Association, 2013. 81 – 96.

[12] A P Fuchs, A Chaudhuri, et al. SCanDroid: Automated security certification of android applications[R]. Maryland: Department of Computer Science, University of Maryland, 2009.

[13] Z W Michael Grace, et al. Systematic detection of capability leaks in stock android smartphones[A]. Proceedings of the 19th Annual Network and Distributed System Security Symposium[C]. San Diego, CA: NDSS Symposium, 2012.

[14] C Kruegel, E Kirda, et al. Polymorphic worm detection using structural information of executables[A]. Recent Advances in Intrusion Detection[C]. Berlin: Springer, 2006. 207 – 226.

[15] George Karypis. Cluto[CP]. http://glaros. dtc. umn. edu/gkhome/cluto/cluto/overview, 2006-10-18.

作者简介

张 焕 女,1989 年生于河南安阳.山东大学计算机科学与技术学院硕士研究生.研究方向为信息安全.

E-mail: inzhanghuan@163.com

郭山清 男,1976 年生于山东滨州.山东大学计算机科学与技术学院副教授;山东省软件工程重点实验室.研究方向为信息安全.

E-mail: guoshanqing@sdu.edu.cn