基于函数调用图的二进制程序相似性分析

孙 贺[1],吴礼发[1],洪 征[1],徐明飞[2],周胜利[1,3]
SUN He[1], WU Lifa[1], HONG Zheng[1], XU Mingfei[2], ZHOU Shengli[1,3]

1.解放军理工大学 指挥信息系统学院,南京 210007
2.中国人民解放军73680部队
3.浙江警察学院 计算机与信息技术系,杭州 310000
1.Institute of Command Information System, PLA University of Science and Technology, Nanjing 210007, China
2.Unit 73680 of PLA, China
3.Department of Computer and Information Technology, Zhejiang Police College, Hangzhou 310000, China

SUN He, WU Lifa, HONG Zheng, et al. Research on function call graph based method for similarity analysis of binary files. Computer Engineering and Applications, 2016, 52(21):126-133.

Abstract: The existing methods of analyzing similarities of binary files based on function call graphs are generally ineffective when dealing with obfuscated programs. A novel analytic hierarchy method based on sub-graph matching is proposed to handle the problems. The proposed method regards sub-graph as the minimal testing unit and tests the similarity of binary files through three steps. The method uses the weighted average strategy to calculate the similarity according to the similarity of each sub-graph. The test results show that the method is more stable with sound similarity results and more efficient compared with the existing methods.

Key words: static analysis; function call graph; similarity analysis; analytic hierarchy method

摘 要:现有基于函数调用图的程序二进制文件相似性分析方法在分析经混淆处理的复杂程序时存在准确度低的问题。针对该问题提出了一种基于子图匹配的层次分析方法。以子图为最小检测单元,分层检测各个子图的相似度;再依据各个子图的相似度,采用加权平均策略计算程序二进制文件的相似度。实验结果表明,该方法抗干扰能力强,能够有效应用于恶意程序家族分类及新病毒变种检测,且具有较高的检测效率。

关键词:静态分析;函数调用图;相似性分析;层次分析
文献标志码:A **中图分类号:**TP393 doi:10.3778/j.issn.1002-8331.1412-0316

1 引言

二进制文件相似性分析是一种程序静态分析方法,在软件窃取检测、代码剽窃检测等应用中起到不可替代的作用,为解决很多安全问题提供了重要依据[1]。根据赛门铁克公司的互联网安全报告[2],大多数新发现的恶意程序由已知病毒的源码或二进制文件衍生而来。使用程序二进制文件相似性分析方法,将未知样本与已知恶意程序进行相似性分析,可以发现新的病毒变种。程序间相似度还可以作为将恶意程序分类的依据,用以研究病毒家族的特征。但是,由于加壳、代码混淆等恶意程序反制方法的更新,二进制文件相似性分析的准确性及其相关应用的有效性面临着更大的挑战[3-4]。

现有二进制文件相似性分析方法包括比特流比对[5]、控制流图比对[6-8]、API调用序列比对[9]、函数调用图比对[3,10-12]等。函数调用图比对方法兼顾了程序的指令级信息和更高层次的内部特征,能够更好地应对恶意程序中常见的反检测手段。

一些研究人员提出了由函数间相似性和图结构相

基金项目:江苏省自然科学基金(No.BK2011115)。

作者简介:孙贺(1990—),男,硕士研究生,研究领域为逆向工程,E-mail:qldx-s@163.com;吴礼发(1968—),男,博士,教授,博导,研究领域为网络安全;洪征(1979—),男,博士,副教授,硕导,研究领域为网络安全;徐明飞(1989—),男,助理工程师,研究领域为可信计算;周胜利(1982—),男,博士研究生,工程师,研究领域为信息安全。

收稿日期:2014-12-23 **修回日期:**2015-02-08 **文章编号:**1002-8331(2016)21-0126-08
CNKI**网络优先出版:**2015-02-11, http://www.cnki.net/kcms/detail/11.2127.TP.20150211.1449.030.html

似性确定程序间相似性的分析方法，研究的重点在于函数间相似性计算和函数调用图同构问题。文献[3]提出函数调用图的图编辑距离的概念，解决大规模恶意程序样本集的恶意程序家族分类问题；文献[10]提出基于二分图匹配的相似性度量算法，解决恶意程序家族分类问题；文献[11]提出基于最大共同边数的度量算法，检测新病毒样本。这些文献的技术路线是先根据函数内部特征判断节点是否匹配，再根据函数调用关系判断边是否匹配，最后根据边的匹配情况判断函数调用图的相似性。其中“匹配”指两者(两个节点或两条边)可以判定为相同。这种技术路线存在以下问题：

第一，指令混淆等反制手段使得函数间相似度计算存在误差，且两个函数是否匹配需要根据设定的阈值来确定。这使得基于函数内部特征的函数间相似性计算方法无法准确判定函数是否匹配。

第二，根据边的匹配情况计算相似度的方法无法应对函数拆分、垃圾调用等简单、常用的混淆手段。且图匹配问题属于NPC问题，需要使用近似算法求解。

针对现有方法的不足，本文在函数调用图的基础上提出基于子图匹配的层次分析方法AHMSM(Analytic Hierarchy Method based on Sub-graph Matching)，进行二进制文件相似性分析。

2　相关概念

本文在传统的四元组函数调用图定义中添加了索引，用于提高函数调用图中不同类型节点的查找速度，具体如定义1。

定义 1(函数调用图) 函数调用图 $G=(V_G, E_G, \rho_G, \Phi_G, \gamma_G)$，是由五元组构成的有向图，$V_G$ 是节点集，每个节点对应一个函数；E_G 是边集，是函数调用关系的全集，$E_G \subseteq V_G \times V_G$，若存在从节点 f_1 到 f_2 的边，则说明函数 f_1 调用了函数 f_2；ρ_G 是节点的属性集，包括“确定匹配”(Confirmed Match，CM)、“相似匹配”(Similar Match，SM)等，用于记录节点状态，初始状态下所有属性值均为null；Φ_G 是从边集 E_G 到节点序偶集合上的函数；γ_G 为索引，根据索引能够快速完成对函数调用图中特定信息的查找。

依据高级编程语言中函数的类型，将函数调用图中的函数分为静态库函数、系统动态链接库函数、本地动态链接库函数和本地函数。根据函数名称可以判断两个静态库函数或系统动态链接库函数是否为同一函数。

为方便描述，参考树结构的相关定义，令函数调用图中入度为0的节点为函数调用图的根节点，出度为0的节点为函数调用图的叶子节点；边 (a, b) 表示函数 a 和函数 b 存在直接调用关系，称 a 为 b 的父节点，b 为 a 的子节点，(a, b) 为节点 a 的出度边、节点 b 的入度边；如果存在从节点 a 到节点 b 的一条多跳路径，则表示函数 a 和函数 b 存在间接调用关系，称 a 是 b 的祖先，b 是 a 的后裔。叶子节点可能为静态库函数、系统动态链接库函数或没有任何函数调用的函数。

函数调用图的子图是本文相似度计算的最小单位，其概念与图结构中子图的概念一致，如定义2。“$\vdash$”符号表示弄真运算，“$\vdash R$”表示逻辑表达式 R 为真。

定义 2(子图) 在函数调用图 G_1 中给定一个节点 a，删除 a 的所有入度边，将 a 的入度置为0，此时以 a 为根节点的函数调用图 G_2 为 G_1 的子图，当且仅当 $\vdash \forall s(s \in G_2 \rightarrow isAncestor(a, s)) \wedge E_{G_2} \subseteq E_{G_1}$ 其中，$isAncestor(a, s)$ 表示 a 是 s 的祖先。

下文中使用的函数调用序列和邻叶节点概念，定义如下。

定义 3(函数调用序列) $Seq(f)=\{f_1, f_2, \cdots, f_n\}$ 是函数 f 的函数调用序列，当且仅当公式(1)成立。

$$\forall f_i(f_i \in f.decent) \wedge \forall f_i, f_j(f_i \in Seq(f) \wedge f_j \in Seq(f) \wedge i<j \rightarrow Offset(f_i) < Offset(f_j)) \quad (1)$$

其中，$f.decent$ 表示在节点 f 的每条深度优先路径上各取一个后裔节点构成的集合；$Offset(g)$ 表示函数 g 的被调用位置相对于二进制文件代码段起始位置的偏移量。在函数调用图中节点与函数具有双射关系，故函数调用序列亦为节点调用序列。

定义 4(邻叶节点) 一个节点 f 为邻叶节点，当且仅当公式(2)成立。

$$\forall t(t \in Seq(f) \rightarrow isBasic(t)=\text{true}) \quad (2)$$

其中，当 t 为叶子节点时，$isBasic(t)$ 返回true；否则返回false。

相似度计算相关定义如下。

定义 5(相似度) 设源程序为 A，目标程序为 B，则 A 与 B 的相似度为 A、B 的函数调用图中等价的部分占 A 的比例，其中等价指在程序功能上可互相替换。

鉴于代码混淆和等价调用替换的检测算法比较复杂，在函数调用图的基础上提出“确定子图”和“相似子图”的概念，如定义6和定义7。

定义 6(确定子图) 以函数 f 为根节点的子图为一个确定子图，当且仅当公式(3)成立。

$$\begin{aligned}&\forall p(f \in G_{\text{src}} \wedge p \in f.descendant \rightarrow \\ &\quad \exists t(t \in G_{\text{dest}} \wedge FuncMatch(f, t)=\text{true})) \wedge \\ &\quad \exists g(g \in G_{\text{dest}} \wedge FuncMatch(f, g)=\text{true}) \wedge \\ &\quad \nexists q(q \in f.father \wedge \exists g(g \in G_{\text{dest}} \wedge \\ &\quad FuncMatch(q, g)=\text{true}) \wedge \\ &\quad \forall r(r \in q.descendant \rightarrow \\ &\quad \exists t(t \in G_{\text{dest}} \wedge FuncMatch(r, t)=\text{true})))\end{aligned} \quad (3)$$

其中，$f.descendant$ 表示函数 f 后裔的集合；$f.father$ 表

示函数 f 父节点的集合；$FuncMatch(f,g)$ 为确定子图的函数间匹配函数。

定义 7（相似子图）以函数 f 为根节点的子图为一个相似子图，当且仅当公式（4）成立。

$$\forall p(p\in BasicSeq(f)\rightarrow p['CM']=\text{null})\wedge \nexists q(q\in p.father\wedge\forall r(r\in BasicSeq(q)\rightarrow r['CM']=\text{null})\wedge rate(BasicSeq(f))<rate(BasicSeq(q))) \tag{4}$$

其中，$BasicSeq(f)$ 表示函数 f 的叶子节点调用序列；$p.father$ 为函数 p 的父节点集合；$rate(BasicSeq(f))$ 表示函数 f 的叶子节点调用序列中匹配的叶子节点的个数占该序列所有叶子节点总个数的比例。

3　基于子图匹配的层次分析方法

AHMSM的输入为两个程序二进制文件，首先从程序二进制文件中抽取函数调用图，记为源程序函数调用图 G_{src} 和目标程序函数调用图 G_{dest}，目的是分析 G_{dest} 与 G_{src} 相似的部分占 G_{src} 的比例，先使用较为简单的算法识别确定子图；再使用复杂算法消除混淆手段的影响，识别相似子图；之后检测相似子图中的等价调用替换，并计算“相似子图”间的相似度；最后计算两个程序整体的相似度。其整体框架如图1所示。

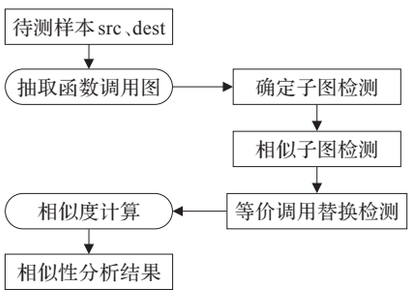

图1　相似性分析方法整体框架

相似性分析方法整体框架减少了处理代码混淆和等价调用替换算法的输入规模，提高了检测效率；EDTW（Enhanced Dynamic Time Wrapping）算法完成了相似子图检测，提高了抗干扰能力；子图检测确定了等价调用替换的查找范围，提高了等价调用替换检测的效率和识别精度。

3.1　函数调用图的抽取

函数调用图的抽取存在平台相关性，本文以Windows平台为例进行说明。首先综合使用PEiD、TrID、SysPack等工具对程序二进制文件脱壳[3]；之后使用深度优先搜索方式扫描脱壳后的程序二进制文件，构造函数调用图。构造过程从程序入口点开始，遍历二进制文件中的所有函数[12-13]；根据PE文件中导入表信息和收集到的系统动态链接库文件列表确定动态链接库函数的类型，将本地动态链接库函数当作本地函数处理，解决本地动态链接库函数的函数名非全局一致的问题；采用隐式系统函数调用检测技术降低函数的漏检率[14]；采用环路检测算法识别递归调用，保证构造的函数调用图为有向无环图。

3.2　确定子图检测

确定子图包括相似度为100%的子图和相似度为0%的子图，分别表示两个子图匹配和不匹配。其中，“匹配”和“不匹配”由文献[3]中4.3.1节中“节点是否匹配”的三个标准来衡量，相应的评价算法记为函数 $FuncMatch$；函数 f 与 g 匹配时 $FuncMatch(f,g)$ 返回true，否则返回false。节点的“确定匹配”属性用于标记节点的状态，如果判断以该节点为根的子图为确定子图，则标记该节点的“确定匹配”属性为“total_match”或“non_match”。

确定子图检测的主要工作是快速识别两个函数调用图 G_{src}, G_{dest} 中的确定子图，减小相似子图检测中复杂算法的输入规模。确定子图检测的处理流程如下：

步骤 1　设两个函数调用图的邻叶节点调用序列为 s_1, s_2，从中筛选出满足公式（5）的邻叶节点调用序列作为待检序列 s_{src}, s_{dest}。

$$s_{src}\subseteq s_1\wedge s_{dest}\subseteq s_2\wedge\forall r(r\in s_{src}\vee r\in s_{dest}\rightarrow (r['CM']=\text{null}\wedge\forall t(t\in r.son\rightarrow t['CM']=\text{null}\vee t['CM']=total\_match))\vee\forall t(t\in r.son\rightarrow t['CM']=\text{null}\vee t['CM']=non\_match)) \tag{5}$$

其中 $r.son$ 表示函数 r 的子节点的集合；$t['CM']$ 表示节点 t 的“确定匹配”属性的值。

如果 $s_{src}=\varnothing$ 或 $s_{dest}=\varnothing$，说明不再有其他未检测的确定子图，执行步骤4；否则执行步骤2。

步骤 2　使用 $FuncMatch$ 函数计算 s_{src}, s_{dest} 中各个函数间的匹配关系，记录在辅助矩阵 A 中，如果 $A(i,j)=\text{true}$，表示 s_{src} 中的函数 i 与 s_{dest} 中的函数 j 匹配，则记录函数 i、j 的匹配关系，标记节点 i、j 的“确定匹配”属性为“$total\_match$”；如果 $\forall j(A(i,j)=\text{false})$，表示没有 s_{dest} 中的函数 j 与 s_{src} 中的函数 i 匹配，则标记函数 i 的“确定匹配”属性为“non_match”。

步骤 3　将两个函数调用图中被标记为“$total\_match$”或“non_match”的节点设置为虚拟确定子图叶子节点，记录以该节点为根节点的子图中总共包含的函数的个数和指令条数。将虚拟确定子图叶子节点当作叶子节点处理，执行步骤1。

步骤 4　删除虚拟确定子图叶子节点的所有出度边，并输出两个样本的带虚拟确定子图叶子节点的函数调用图。

经过上述步骤处理，两个函数调用图中的确定子图都已转化为“确定匹配”属性为“$total\_match$”或“non_match”

的虚拟确定子图叶子节点。节点内记录了原图中以该节点为根的确定子图中所有指令的条数。

3.3　相似子图检测

函数调用序列匹配与信号匹配具有相似的时序特征，考虑采用DTW算法[15-17]作为叶子节点调用序列的匹配算法。该算法在语音信号匹配问题中效果显著[16]，文献[17]亦将其作为多径轮廓的匹配算法。但是，受到代码混淆手段的影响，仅执行一次DTW算法无法完全识别匹配序列。本文提出了增强的DTW算法（EDTW算法），该算法在DTW算法基础上结合Neighbor-Biased（偏邻居节点）原则[3,18]检测匹配的叶子节点序列。最后根据定义7识别相似子图。

相似子图检测的输入为两个样本的带虚拟确定子图叶子节点的函数调用图，目标是识别函数调用图中的相似子图。“相似匹配”属性用于记录算法执行过程中节点的状态。具体流程如下：

步骤1　在函数调用图的所有叶子节点的调用序列中去除虚拟确定子图叶子节点，得到两个样本的待检序列 $s_{\text{src}}, s_{\text{dest}}$。

步骤2　执行EDTW算法，设第 i 轮的输入序列为 s_1, s_2，执行 $DTW(s_1, s_2)$ 得到的匹配序列为 s，则第 $i+1$ 轮执行DTW算法的输入为 $s_1 - s$、$s_2 - s$。从初始 $i=1$ 到 s 为空停止。第 i 轮结束，标记该轮匹配到的节点的“相似匹配”属性值为 i。遍历基本项序列 s_{src}，将满足公式(6)的节点 t 标记为匹配的点：

$$\exists s(|Offset(s)-Offset(t)|=1 \wedge s['SM']= t['SM'] \vee |s['SM']-t['SM']|=1) \quad (6)$$

其中 $Offset(s)$ 表示 s 在输入序列中的位置；$s['SM']$ 表示节点 s 的“相似匹配”属性。之后循环遍历 s_{src}，如果与节点 j 毗邻的节点均被标记为匹配，且节点 j 的“相似匹配”属性不为空，则标记节点 j 为匹配的节点；如果没有满足条件的节点 j 存在，则停止循环。将所有标记为匹配的点的“相似匹配”属性置为1，其余点的“相似匹配”属性置为0。在 s_{dest} 中同步标记节点的“相似匹配”属性值，记录 $s_{\text{src}}, s_{\text{dest}}$ 中节点的匹配关系。

步骤3　根据节点间的匹配关系和函数调用关系识别相似子图。执行步骤如下：

(1)提取源函数调用图的邻叶节点调用序列 s 中满足 $\forall r(r \in s_i.\text{son} \rightarrow r['CM']=\text{null})$ 的序列 t，记录 t 中每个节点的子节点个数 n_1 和“相似匹配”属性为1的节点的个数 n_2，置“相似匹配”属性为 n_1/n_2。

(2)按上述方法递推地计算 s 序列中每个节点的父节点及其他祖先的“相似匹配”属性值（任何一个节点都不是虚拟确定子图叶子节点的祖先），当父节点的“相似匹配”属性值小于子节点的“相似匹配”属性值，或不再有非虚拟确定子图叶子节点的祖先存在时，停止对当前递推路径的检测。当所有递推路径都检查完毕后，用各条路径上“相似匹配”属性最大的节点构成集合 T。

(3)新建虚拟相似子图叶子节点 N，对于任意 x、i，将边 (x, T_i) 变为 (x, N) 和 (N, T_i) 两条边，在 N 中记录以 T_i 为根的相似子图中的叶子节点个数、叶子节点匹配情况和子图内所有指令的条数等信息。将以 T_i 为根的相似子图中的叶子节点调整为 T_i 的子节点，删除以 T_i 的其他子节点为根节点的子图。

步骤4　输出带虚拟相似子图叶子的函数调用图。

常用的程序混淆手段包括代码混淆、垃圾调用、等价调用替换等。区分源程序和目标程序的相似度检测方法已经去除了垃圾调用的影响；上述相似子图的检测去除了代码混淆的影响。但考虑到二进制代码还可能存在利用不同代码实现相同功能的情况，将实施等价调用替换检测，进一步提高相似性分析精度。

3.4　等价调用替换检测

如果库函数 A 和库函数 B 实现了相同的功能，但函数名称不同，则称 B 是 A 的等价调用替换。例如 *printf* 和 *printf_s*、*ReadFile* 和 *ntReadFile*、*CreateDirectory* 和 *CreateDirectoryEx* 等。

等价调用替换检测根据节点匹配关系、偏邻居节点思想[3,18]和等价调用替换库，预测并判定相似子图中存在的等价调用替换，之后将判定为等价调用替换的叶子节点标记为匹配。具体步骤如下：

步骤1　获得相似子图的根节点集合 T，使用偏邻居节点方法[3,18]预测 T 中节点所在的相似子图是否存在等价调用替换；如果 t 为疑似等价调用替换，则在等价调用替换库中查询所有与 t 等价的调用，记为 $\{b_1, b_2, \cdots, b_n\}$，执行步骤2；否则执行步骤4。

步骤2　根据源程序节点和目标程序节点的匹配关系，在目标程序函数调用图的叶子节点调用序列 s_{dest} 中应用内容匹配算法，检测是否有 b_i 存在，如果存在，则标记 t 与 b_i 匹配。

内容匹配算法流程如下：根据节点匹配关系得到 t 所在的叶子节点调用序列在 s_{dest} 中对应的序列 t'。设 t' 在 s_{dest} 中第一个出现的位置为 i，最后一个出现的位置为 j，则抽取 $s_{\text{dest}}[(i+j)/2, (i+3j)/2]$ 序列中未标记为匹配的叶子节点调用序列 s'；使用LCS算法[19]在 s' 中查找 b_i。

步骤3　如果 T 中不再有未处理的疑似等价调用替换，则执行步骤4；如果有，则执行步骤2，检测下一节点。

步骤4　整理经由以上所有步骤得到的函数调用图 G_{src}，在虚拟相似子图叶子节点中记录该相似子图中共包含的指令条数和“相似匹配”属性值等信息，以供相似性计算使用。

等价调用替换检测能够有效发现二进制文件中存在的等价函数调用替换,防范利用等价函数实施的文件变异。

3.5　二进制文件整体相似性计算

经过上述步骤的分析,源程序的函数调用图在节点类型上分为叶子节点、虚拟确定子图叶子节点、虚拟相似子图叶子节点,以及调用它们的本地函数等几部分构成,图2用树形结构描述了经过分析的函数调用图的构成。

其中,虚拟确定子图叶子节点分为*total_match*和*none_match*两类,相似性分别为100%和0%。虚拟相似子图节点 f 的相似度为 f 的叶子节点调用序列中匹配的个数除以总个数。标记为匹配的叶子节点的相似度为100%,其他叶子节点的相似度为0%。

设本地函数 L 的子节点调用序列为 $l_1, l_2, \cdots, l_m$,对应的相似度为 $s_1, s_2, \cdots, s_m$,则 L 的相似度为:

$$\frac{\sum_{i=1}^{m} N(l_i)\times s_i}{\sum_{i=1}^{m} N(l_i)} \tag{7}$$

其中,$N(l_i)$ 是以 l_i 为根节点的子图中包含的所有指令的条数。

根据公式(7)可以递归地求得根节点的相似度,即样本的整体相似度。

4　实验结果分析

在Windows平台上使用Python语言实现AHMSM,利用IDA Pro插件IDAPython实现函数调用图的抽取。设计了验证性实验和对比性实验来检验AHMSM对程序二进制文件相似性的检测效果。

4.1　验证性实验结果分析

实验主要验证AHMSM是否能有效应用于恶意程序家族分类及新病毒变种检测。基于相似度的程序家族分类要求AHMSM能够有效区分恶意程序家族内部样本间的相似度和恶意程序家族间不同样本的相似度;基于程序二进制文件相似度的新病毒检测要求AHMSM能够发现新的恶意程序变种与已知恶意程序具有较高的相似性。

4.1.1　恶意程序家族分类能力验证

正确对恶意程序家族分类要求分类方法对家族内部和家族间样本间的相似度有较强的区分能力。本文使用Virus.Win32.Sality、Virus.Win32.Champ、Virus.Win32.Trats、Virus.Win32.Mkar四个恶意程序家族的样本,测试AHMSM对恶意程序样本间相似度的区分能力。

首先使用Virus.Win32.Sality恶意程序家族的6个样本进行实验,结果如表1。Sality是著名的恶意程序家族,通过感染“.exe”和“.scr”文件进行病毒传播,能够自动识别并感染可移动设备中的文件;内置木马下载器,能够在后台静默下载安装恶意程序。

表1　Sality恶意程序家族内变种的相似度　%

	Sality	Sality.a	Sality.b	Sality.c	Sality.d	Sality.e
Sality	100.00	96.42	97.93	96.04	98.41	94.67
Sality.a	85.59	100.00	63.15	98.68	75.53	72.55
Sality.b	79.25	88.58	100.00	87.36	61.07	69.39
Sality.c	85.40	99.28	62.32	100.00	73.81	75.36
Sality.d	81.87	79.39	69.12	79.57	100.00	77.14
Sality.e	72.65	63.44	72.29	63.89	68.23	100.00

其中Sality样本为反混淆处理过的病毒样本,这里将其当作原始病毒样本。表中第 a 行第 b 列位置的数据代表以版本 a 为源程序,版本 b 为目标程序计算得到的 a、b 两个二进制文件的相似性。可以看到,所有以Sality样本为源程序的相似度值都非常高(超过90%);其他家族内变种间的相似度值也达到55%以上。

但是,在对恶意程序家族没有深入了解和分析的情况下,很难得到该家族的无混淆版的源程序。对于其他不了解的恶意程序家族,可能无法出现表1中第一行相似度非常高的情况。为进一步验证AHMSM对恶意程序家族的分类能力,使用AHMSM对Virus.Win32.Champ、Virus.Win32.Trats、Virus.Win32.Mkar三个恶意程序家族计算样本间相似度,样本中不包括去混淆的版本。AHMSM检测结果如图3所示。

其中 x、y 轴均代表恶意程序的名称,z 轴表示以 x 轴对应程序为源程序,以 y 轴对应程序为目标程序的

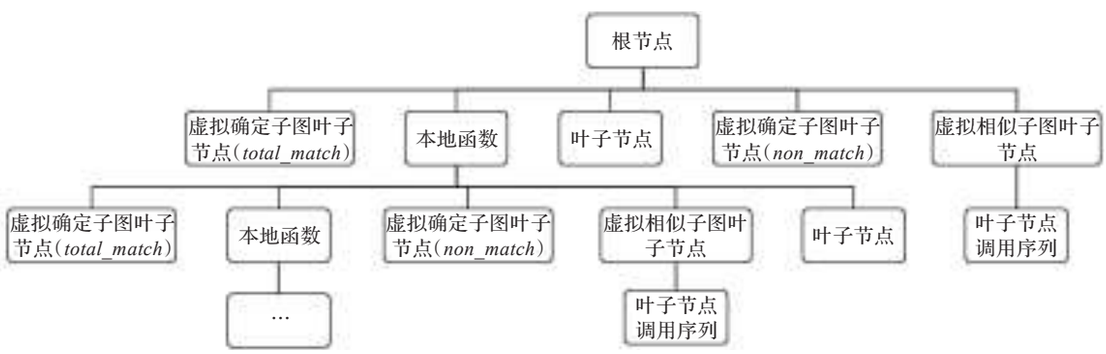

图2　函数调用图的构成

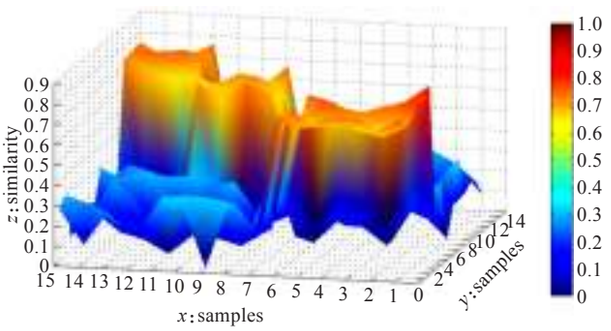

图3　三类恶意程序的版本间相似度计算结果

两样本间相似度。x、y轴从1开始，每个数字代表一个程序样本，分别为Champ、Champ.5430、Champ.5447、Champ.5464、Champ.5477、Champ.5495、Trats.a、Trats.b、Trats.c、Trats.d、Trats.e、Mkar.a、Mkar.b、Mkar.c、Mkar.d，程序样本的名称前缀均为“Virus.Win32.”。

从图3中可以看出，1号样本(Virus.Win32.Champ)与Virus.Win32.Champ家族内部各个样本间的相似度值高于其他值(色标中颜色对应值更接近1)。这说明与Sality恶意程序样本的相似度计算结果类似，Champ样本本身可以认为是基本没有经过混淆处理的源程序。Virus.Win32.Trats、Virus.Win32.Mkar家族分别看到连续的比较高的峰值，说明实验样本中存在恶意程序家族的源程序的样本。这个结果与“新变种大多基于已有程序样本改编而来[3]”的结论一致。从图中可以看出，三个恶意程序家族内部样本间的相似性明显高于恶意程序家族间样本的相似度。说明AHMSM能够较好区分恶意软件家族间和家族内部各个样本的相似度。

4.1.2　新病毒变种检测能力验证

使用文献[4]中提到的SDBot样本进行对比实验。首先手工对该样本进行混淆和免杀处理[20-21]，得到SDBot.a、SDBot.b、SDBot.c、SDBot.d、SDBot.e、SDBot.f等6个变种。各个版本间大致存在如下关系：SDBot.a在源样本的基础上进行指令混淆；SDBot.b在SDBot.a基础上进行控制流混淆；SDBot.c在SDBot.b基础上进行函数调用混淆和等价调用替换；SDBot.d在源样本基础上进行函数调用混淆和等价调用替换；SDBot.e在SDBot.d基础上进行控制流混淆；SDBot.f在SDBot.e基础上进行指令混淆(免杀过程比较复杂，还需要很多其他的辅助操作)。使用Macafee、Norton360、Kaspersky三种杀毒软件(截止至2014.11.25最新的病毒库)进行检测。使用AHMSM计算各个变种与SDBot源样本的相似度。结果如表2所示。

从实验结果可以看出，AHMSM对6个变种的相似性度量值均超过90%。

4.2　对比实验结果分析

基于文献[4]给出的SDBot代码片段，使用AHMSM与文献[3]、[4]做对比。该代码片段给出了一个具体的代码混淆例子，使用C语言代码展示，比二进制文件更便于对比分析。将文献[4]中图1(a)程序称为函数A，图1(b)程序称为函数B。

4.2.1　与文献[4]Wookon系统的对比分析

文献[4]实现的Wookon系统利用动态污点传播分析方法识别恶意行为之间的控制依赖关系和数据依赖关系，依据这两个关系进行相似性比较。实验表明Wooken能够准确识别出A、B两个函数具有100%的相似度。AHMSM使用静态分析方法，依据函数调用关系进行相似性比较。令A为源程序中的函数，B为目标程序中的函数。AHMSM计算A、B间相似度的过程如下：首先根据文献[3]的方法计算得出指令的编辑距离相似比小于85%，判定以A、B为根节点的子图不属于确定子图。对A、B的叶节点调用序列使用EDTW算法，执行结果如表3所示。

其中“1”表示在该轮次的DTW中函数被识别为匹配，“0”表示识别为不匹配；“()”内的数字为“相似匹配”属性值；“—”表示该函数已经标记为匹配，无需进行其他操作。如第1行第1列的“1(1)”代表函数“*InternetOpenUrl*”在第一轮DTW检测中被标记为匹配。

由表3可得，经过EDTW算法处理，只有函数WriteFile未被匹配。之后进入等价调用替换检测。查询等价调

表2　与杀毒软件的比较结果

	SDBot	SDBot.a	SDBot.b	SDBot.c	SDBot.d	SDBot.e	SDBot.f
Macafee	√	√	×	×	√	×	×
Norton360	√	√	√	√	√	√	√
Kaspersky	√	√	√	×	√	×	×
AHMSM	100%	99.31%	99.02%	98.31%	98.59%	96.67%	96.32%

表3　EDTW算法执行结果

	InternetOpenUrl	CreateFile	GetTickCount	memset	InternetReadFile
第一轮DTW	1(1)	0	1(1)	1(1)	1(1)
第二轮DTW	—	1(2)	—	—	—
	WriteFile	sprintf	sprintf	GetTickCount	CloseHandle
第一轮DTW	0	1(1)	1(1)	1(1)	1(1)
第二轮DTW	0	—	—	—	—

用替换库，*WriteFile* 可以替换为 *CreateFileMapping*、*MapViewOfFile* 调用序列，对函数B执行内容匹配算法，发现存在该调用序列，因此 *WriteFile* 函数被标记为匹配。最终得到该本地函数的相似度值为100%。

本实验说明，本文提出的依据子图匹配的层次分析方法能够对未加壳程序达到与文献[4]相同的准确率。但是，由于静态检测方法的局限性和等价调用替换库的完备性限制，AHMSM的整体检测效果不如Wookon的效果，例如AHMSM检测加壳病毒的能力有限，如果自动化程序脱壳失败则需要人工干预，否则无法进行相似性分析等。但是，Wookon的检测复杂度较高，例如文献[4]中提到的Bagle样本，Wookon系统的检测需要6 min39 s，而在测试主机配置相当的条件下，AHMSM的检测仅需43 s。因此Wookon不适用于进行本地检测；而AHMSM达到了与Wookon相近的检测精度，且检测时间短，更加适用于本地检测。

4.2.2　与文献[3]的对比分析

将AHMSM与SMIT系统[3]作比较，两者均使用函数调用关系计算程序二进制文件间的相似度。依然使用文献[4]给出的SDBot代码片段。对函数B进行改造，如图4所示。

由于库函数没有变化，Wookon系统显然可以正确识别出图4中的函数与函数A的相似度为100%；AHMSM中EDTW算法的结果与表3相同，函数封装不对AHMSM的工作造成影响，因此AHMSM也可以正确识别两个片段的相似度为100%。而SMIT系统使用的图编辑距离技术要将函数A的函数调用图转换为图4中程序的函数调用图。按照文献[3]的描述，转换过程需要在函数A的函数调用图上插入5个节点、8条边，并删除1个节点、3条边，如表4所示。

点和边的插入操作都增大了图编辑距离的值，使得本来相同的两段程序出现了17个点边操作的误差。因此AHMSM的抗干扰能力强于SMIT系统。此外，使用SMIT系统计算两个二进制文件的时间复杂度为 $O(2t+t^2+t^2\times m^2+(2t)^2+(2t)^3)$ [3,10]，而AHMSM最坏情况下的时间复杂度为 $O(n^2\lg n)$，故AHMSM在效率上也比SMIT略有优势。

5　总结与展望

本文提出的AHMSM相似性检测方法以函数调用图的子图为基础，按照先易后难的顺序分层检测不同相似程度的子图，提高了检测效率；提出EDTW算法，能够有效应对代码混淆的影响；提出内容匹配算法检测等价

```
1:  charfbuff[512];
2:  …
3:  fh=InternetOpenUrl(internetHandle,dl.url,NULL,0,0,0);
4:  if(fh!=NULL)
5:    f=CreateFile(dl.dest,GENERIC_WRITE,0,NULL,CREATE_ALWAYS,0,0);
6:  if(f<(HANDLE)1){
7:    return0;
8:  }
9:  total=1;
10: start=GetTickCount();
11: do{
12:   memset(fbuff,0,sizeof(fbuff));
13:   InternetReadFile(fh,fbuff,sizeof(fbuff),&r);
14:   WriteFile(f,fbuff,r,&d,NULL);
15:   total=total+r;
16:   if(dl.update!=1)
17:     sprintf(threadDescriptions[dl.threadnum],
          "filedownload(%s-%dkbtransferred)",dl.url,total/1024);
18:   else
19:     sprintf(threadDescriptions[dl.threadnum],
          "update(%s-%dkbtransferred)",dl.url,total/1024);
20: }while(r>0)
21: speed=total/(((GetTickCount()-start)/1000)+1);
22: CloseHandle(f);
23: }
```

(a)改造后的主体代码

```
1:  charfbuff[512];
2:  …
3:  fh=CreateFile(dl.dest,GENERIC_WRITE,0,NULL,CREATE_ALWAYS,0,0);
4:  hFileMapping=CreateFileMapping(f,NULL,PAGE_READWRITE,0,0,NULL);
5:  pCurrentPointer=MapViewOfFile(hFileMapping,FILE_MAP_WRITE,0,0,0);
6:  temp=CreateFile("C:\\test.txt",GENERIC_READ,0,OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL,0);
7:  if(f<(HANDLE)1){
8:    return0;
9:  }else{
10:   fh=InternetOpenUrl(internetHandle,dl.url,NULL,0,0,0);
11:   if(fh==NULL)
12:     return0;
13: }
14: total=1;
15: start=GetTickCount();
16: do{
17:   ReadFile(fh,fbuff,sizeof(fbuff),&r);
18:   memset(fbuff,0,sizeof(fbuff));
19:   InternetReadFile(fh,fbuff,sizeof(fbuff),&r);
20:   memcpy(pCurrentPointer,fubff,r);
21:   pCurrentPointer=(void*)((DWORD)pCurrentPointer+r);
22:   total=total+r;
23:   if(dl.update!=1)
24:     sprintf(threadDescriptions[dl.threadnum],
        "filedownload(%s-%dkbtransferred)",dl.url,total/1024);
25:   else
26:     sprintf(threadDescriptions[dl.threadnum],
        "update(%s-%dkbtransferred)",dl.url,total/1024);
27: }while(r>0)
28: speed=total/(((GetTickCount()-start)/1000)+1);
29: CloseHandle(f);
30: }
```

(b)添加的函数代码

图4　改造的SDBot代码

表4　函数调用图转换操作明细

操作	明细
添加节点	GetCurrentPointer，WriteFILE，CreateFileMapping，MapViewOfFile，ReadFile
删除节点	WriteFile
添加边	(A，GetCurrentPointer)，(A，WriteFILE)，(GetCurrentPointer，CreateFileMapping)，(GetCurrentPointer，MapViewOfFile)，(GetCurrentPointer，CreateFile)，(WriteFILE，ReadFile)，(WriteFILE，memset)，(WriteFILE，InternetReadFile)
删除边	(A，CreateFile)，(A，InternetReadFile)，(A，WriteFile)

调用替换，具有较强的抗干扰能力。测试结果表明，本文方法能够准确计算程序二进制文件的相似性，同时具有较高的识别效率。下一步将以该方法为基础，研究恶意程序家族聚类方法和基于相似性比对的漏洞挖掘方法。此外，提高方法对加壳程序的检测能力，也是一个值得研究的方向。

参考文献：

[1] Silvio Cesare B I T, Info M.Software similarity and classification[D].Vitoria：Deakin University，2013.

[2] Symantec Corp.Symantec global Internet security threat report，volume XII[EB/OL].[2014-10-12].http：//www.symantec.com/.

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

[4] 杨轶，苏璞睿，应凌云，等.基于行为依赖特征的恶意代码相似性比较方法[J].软件学报，2011，22(10)：2438-2453.

[5] Kolter J Z，Maloof M A.Learning to detect malicious executables in the wild[C]//Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining，2004：470-478.

[6] Carrera E，Erdélyi G.Digital genome mapping-advanced binary malware analysis[C]//Virus Bulletin Conference，2004：187-197.

[7] Dullien T，Rolles R.Graph-based comparison of executable objects(English Version)[C]//SSTIC，2005：1-3.

[8] Briones I，Gomez A.Graphs，entropy and grid computing：automatic comparison of malware[C]//Virus Bulletin Conference，2008：1-12.

[9] Ye Y，Wang D，Li T，et al.IMDS：Intelligent Malware Detection System[C]//Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining，2007：1043-1047.

[10] 刘星，唐勇.恶意代码的函数调用图相似性分析[J].计算机工程与科学，2014，36(3)：481-486.

[11] Shang S，Zheng N，Xu J，et al.Detecting malware variants via function-call graph similarity[C]//2010 5th International Conference on Malicious and Unwanted Software(MALWARE)，2010：113-120.

[12] Wu L，Xu M，Xu J，et al.A novel malware variants detection method based on function-call graph[C]//Conference Anthology，2013：1-5.

[13] 付文，赵荣彩，庞建民，等.PE可执行程序中Main函数的定位技术[J].计算机工程，2010，36(16)：47-48.

[14] 付文，赵荣彩，庞建民，等.隐式API调用行为的静态检测方法[J].计算机工程，2010，36(14)：108-110.

[15] Salvador S，Chan P.Toward accurate dynamic time warping in linear time and space[J].Intelligent Data Analysis，2007，11(5)：561-580.

[16] Keogh E J，Pazzani M J.Derivative dynamic time warping[C]//SDM，2001：5-7.

[17] Wang J，Katabi D.Dude，where's my card? RFID positioning that works with multipath and non-line of sight[C]//Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM，2013：51-62.

[18] He H，Singh A K.Closure-tree：an index structure for graph queries[C]//Proceedings of the 22nd International Conference on Data Engineering，ICDE'06，2006.

[19] 王映龙，杨炳儒，宋泽锋，等.基因序列相似程度的LCS算法研究[J].计算机工程与应用，2007，43(31)：45-47.

[20] Borello J M，Mé L.Code obfuscation techniques for metamorphic viruses[J].Journal in Computer Virology，2008，4(3)：211-220.

[21] Moser A，Kruegel C，Kirda E.Limits of static analysis for malware detection[C]//Computer Security Applications Conference，ACSAC 2007，2007：421-430.

(上接56页)

[24] Maji P K，Roy A R，Biswas R.An application of soft sets in a decision making problem[J].Computers and Mathematics with Applications，2002，44：1077-1083.

[25] Chen D G，Tsang E C C，Yeung D S，et al.The parameterization reduction of soft sets and its applications[J].Computers and Mathematics with Applications，2005，49：757-763.

[26] Kong Z，Gao L Q，Wang L F，et al.The normal parameter reduction of soft sets and its algorithm[J].Computers and Mathematics with Applications，2008，56：3029-3037.

[27] Maji P K，Roy A R，Biswas R.Fuzzy soft sets[J].The Journal of Fuzzy Mathematics，2001，9：589-602.

[28] Roy A R，Maji P K.A fuzzy soft set theoretic approach to decision making problems[J].J Comput Appl Math，2007，203：412-418.

[29] Feng F，Li C X，Davvaz B D，et al.Soft sets combined with fuzzy sets and rough sets：a tentative approach[J].Soft Computing，2010，14：899-911.

[30] Cornelis C，Deschrijver G，Kerre E E.Implication in intuitionistic fuzzy and interval-valued fuzzy set theory：construction，classification，application[J].International Journal of Approximate Reasoning，2004，35：55-95.

[31] Gong Z T，Zhang X X.On characterization of fuzzy soft rough sets based on a pair of border[J].Fundamenta Informaticae，2015，137：1-35.