# 一种基于图的程序行为相似性比较方法 *

陈 浩, 王广南, 孙建华
(湖南大学 计算机与通信学院 计算机科学与技术系, 长沙 410082)

**摘 要**: 针对目前的软件盗版现象,在没有软件源代码的情形下提出一种程序相似性的比较方法。该方法是运用程序系统调用之间的参数依赖关系组成依赖图,对程序行为进行描述;在此基础上定义了一种动态程序胎记,用它比较两个功能类似的应用程序。最后的试验数据表明,该方法能够有效地检测出相似程度不一的各组程序之间的相似度,具有一定的可信度和适用性。

**关键词**: 软件剽窃; 图; 系统调用; 动态软件胎记; 相似性
**中图分类号**: TP311.5 **文献标志码**: A **文章编号**: 1001-3695(2010)02-0532-05
doi:10.3969/j.issn.1001-3695.2010.02.036

# Approach for measuring software similarity based on graphs

CHEN Hao, WANG Guang-nan, SUN Jian-hua
(*Dept. of Computer Science & Technology, Institute of Computer & Communication, University of Hunan, Changsha* 410082, *China*)

**Abstract**: View of software piracy, this paper proposed an approach for measuring software similarity without sourcecode. It created dependence graphs to specify relationships between system call arguments for describing program behavior, based on which defined an dynamic software birthmark. It could be used to measure the similarity of two same-purpose applications. Experimental results indicate that the approach is effective in detecting similarity between two programs in groups of varying degrees similar, which proves its certain degree of credibility and applicability.

**Key words**: software theft; graphs; system call; dynamic software birthmark; similarity

## 0 引言

对越来越多的被剽窃的软件无源代码发布,程序动态胎记技术近年来在各项剽窃软件与源程序的相似性检测技术中逐渐成为一个研究热点。它主要是从表示程序运行时的动态行为的信息中提取程序特征,因此无需源代码,并且面对程序模糊或反编译等常见的剽窃痕迹隐藏技术能很好地保存原程序的特征信息[1]。在提取动态胎记的许多方法中,系统调用序列是经常被用到的一个[1~4]。因为其数目有限,且各操作系统的系统调用函数版本间变化非常小,对其进行分析,数量适中稳定;但是许多程序的系统调用序列由于其数量巨大,很多研究者并不考虑其参数而直接使用函数本身在程序运行时的执行次序或者出现的频率作为程序行为特征[1,2,4]。由于其程序特征提取不明显,只能适应于比较两个相似程度很大的程序,并且容易受到程序每次运行时系统调用执行次序差异的影响以及添入其他系统调用等伪装技术的攻击,尽管后者所花费的代价将很大。

程序行为不仅体现在不同系统调用间的先后次序上,即使同一个系统调用由于其所带参数的不同也会让程序表现大不一样,因此准确分析程序行为必须充分考虑系统调用参数。本文对于程序相似性研究的主要贡献在于挖掘系统调用参数之间的依赖关系[5],将这种关系用图来表示,这样能够比较准确地描述程序的行为特征。本文的主要内容有以下几点:

a)程序行为的图形化描述以及基于行为子图的程序特征胎记的提取。

b)程序相似性比较算法及两个程序相似度的定义。

c)性能测评实验对于本文所提出的方法可信度以及健壮性的证明。

## 1 关于程序胎记的一些概念

**定义 1** 程序复制关系[1,2,6~8]。假设 Prog 是一组给定的程序集,令"≡cp"表示程序集 Prog 之间的相等关系,使得对于 $P,Q\in$ Prog,如果 $Q$ 是 $P$ 的一个副本,那么 $P\equiv \mathrm{cp}Q$。有关程序间几种具体的复制关系的详细描述见参考文献[1,2,6,7]。对于程序复制关系有如下属性:

**性质 1** 对于程序 $P,Q\in$ Prog,有反身性 $P\equiv \mathrm{cp}P$,对称性 $P\equiv \mathrm{cp}Q\Rightarrow Q\equiv \mathrm{cp}P$,传递性 $P\equiv \mathrm{cp}Q\wedge Q\equiv \mathrm{cp}R\Rightarrow P\equiv \mathrm{cp}R$。

上述属性直接由程序复制关系而来,另外,如果 $Q$ 是 $P$ 的一个拷贝,那么 $Q$ 的外部实现规范应该与 $P$ 一样。

**性质 2** 假设 spec($p$)为实现程序 $P$ 所遵从的(外部的)规范,那么将有如下推导关系:$P\equiv \mathrm{cp}Q\Rightarrow \mathrm{spec}(P)=\mathrm{spec}(Q)$。

需要注意的是性质 2 的逆命题并不一定成立,不同的程序实现可能会遵从相同的程序规范。接下来,给出程序动态胎记的定义:

**收稿日期**: 2009-05-31; **修回日期**: 2009-06-22 **基金项目**: 国家自然科学基金资助项目(60803130)

**作者简介**: 陈浩(1977-), 男, 湖南岳阳人, 副教授,主要研究方向为分布式系统、流媒体、虚拟化(haochen@aimlab.org); 王广南(1983-), 女, 硕士, 主要研究方向为系统安全; 孙建华(1977-), 女, 副教授, 主要研究方向为网络安全、入侵检测.

**定义 2**　动态胎记[1,2,8]。对给定的程序 $P$、$Q$,输入 $I$ 以及复制关系"≡cp",假设 $f(P,I)$ 为用一种特定的方法从给定输入 $I$ 的程序 $P$ 中提取出的一组特征集,那么 $f(P,I)$ 可以称为在满足如下所有条件时,程序 $P$ 在复制关系"≡cp"下的一种动态胎记。

**条件 1**　$f(P,I)$ 仅仅只从程序 $P$ 本身在给定的输入 $I$ 下运行时得来。

**条件 2**　$P\equiv_{cp}Q\Rightarrow f(P,I)=f(Q,I)$;同样反过来不一定成立。

条件 1 意味着提取胎记并不需要其他信息(如用户自己编写代码)来实现;条件 2 说明胎记 $f(P,I)$ 只是程序相似关系的一种证明,而不是标记。但是如果 $f(P,I)\neq f(Q,I)$,那么可以肯定 $P\neq Q$。这样,能确定 $Q$ 一定不是 $P$ 的一个拷贝。

当定义了一个胎记时,理想情况下,希望它具有如下的性质:

**性质 3**　保留性[1]。如果 $P'$ 是从程序 $P$ 经过任何一种程序变换得到的,那么 $f(P',I)=f(P,I)$。

**性质 4**　区别性[1]。对于程序 $P$ 和 $Q$,以及 spec($P$) = spec($Q$),如果 $P$ 和 $Q$ 是独立实现的,那么 $f(P,I)\neq f(Q,I)$。

从性质 1 和 2 可以看到,性质 1 确切说明胎记的保留属性对于程序变换的抵抗性。可以假设这种程序变换是黑客们试图通过对程序作保留语义的模糊变换而改变胎记以隐藏程序剽窃的痕迹,如程序模糊技术。性质 2 则是说明胎记的区别性,即使程序 $P$ 和 $Q$ 实现时所遵从的外部规范是一样的,但如果它们是独立实现的,则提取的胎记必然不同。一般而言,两个独立的程序细节几乎从来不会完全一样。然而大量的研究表明,不排除 $P$、$Q$ 是只有几十行的小程序,那么尽管 $P$、$Q$ 是独立实现的,但是提取的胎记特征有可能一致[7]。这表明:程序胎记有一定的适应范围,并且从理想意义上,提取的胎记应该完全满足上述的两性质。然而由于程序的各种实现方法和变换,在现实中很难做到提取的胎记同时完美地符合上述两种性质。本文工作是如何有效地提取程序胎记以尽可能地符合这两种性质。

## 2　程序胎记的提取

### 2.1　图的基本概念

图是一种普遍且强大的数据结构,不论对图形作何种转换、旋转,或把它转变成它的镜像图像,图的数学意义仍然不变。因此,本文提出用图来表示系统调用及各参数间的依赖关系。先回顾一下需要用到几个图的基本概念。

**定义 3**　图[9]。本文用一个四元组来表示图,用 $Lv$ 和 $Le$ 分别表示图中节点与边的属性,即 $G=(V,E,U,W)$。其中:

$V$ 为有限的节点集;

$E\leqslant V\times V$ 是边集;

$U$: $V\rightarrow Lv$ 是定义节点属性的函数,

$W$: $E\rightarrow Le$ 是定义边属性的函数。

**定义 4**　子图[9]。给定一个图 $G=(V,E,u,v)$,它的一个子图可以这样表示:$S=(Vs,Es,Us,Ws)$。其中:

$Vs\subseteq V$; $Es=E\cap(Vs\times Vs)$;

$Us$ 和 $Ws$ 则分别为定义节点属性和边属性的函数,如下

$$Us(v)=\begin{cases}U(v) & \text{如果 } v\in Vs,\\ \text{undefined} & \text{否则}\end{cases}$$

$$Ws(e)=\begin{cases}W(e) & \text{如果 } e\in Es\\ \text{undefined} & \text{否则}\end{cases}$$

**定义 5**　最大共同子图[9]。给定图 $G_1$、$G_2$,如果 $G$ 既使是 $G_1$ 的子图也是 $G_2$ 的子图,并且 $G_1$ 和 $G_2$ 中不存在其他子图的节点数比 $G$ 大,那么 $G$ 是 $G_1$ 和 $G_2$ 的一个最大共同子图,本文用 mcs($G_1$,$G_2$)表示。

**定义 6**　基于最大共同子图的图的相似度。给定两个非空图 $G_1$ 和 $G_2$,以及它们的最大共同子图 mcs($G_1$,$G_2$),那么它们之间的距离[9,10]可以计算如下:

$$D(G_1,G_2)=1-|\text{mcs}(G_1,G_2)|/\max(|G_1|,|G_2|)$$

其中:$|G_1|$和$|G_2|$分别表示 $G_1$,$G_2$ 的节点数,$|\text{mcs}(G_1,G_2)|$表示最大共同子图的节点数。那么图 $G_1$ 与 $G_2$ 的相似度可以定义为

$$S(G_1,G_2)=1-D(G_1,G_2)=|\text{mcs}(G_1,G_2)|/\max(|G_1|,|G_2|)$$

假如 $G_1$ 的节点数为 5,$G_2$ 的节点数为 4,最大共同子图节点数为 3,则 $D(G_1,G_2)=0.4$,$G_1$ 与 $G_2$ 的相似度 $S(G_1,G_2)=0.6$。

### 2.2　程序行为的语义描述与图形化建模

不论程序的源代码或编译后的指令有什么样的改变,也不论编译器的版本或者编译的方式是否不同,在 CPU 中运行的代码总是无法改变的。程序在运行时,任何一个涉及到与进程环境交互的行为都需要进程调用相应的系统服务函数[5]。除了前面提到的各种优点,对于笔者的研究工作来说,截取程序系统调用序列的工具或方法很多而且容易实现[1,2,5,11]。从系统调用之间的参数流向关系上可以看到程序的各种动态行为。基于图语义表达上的优越性,本文对这些行为作图形化建模,所得到的每个局部子图体现的是程序的每个局部行为语义。对搜集到的系统调用序列进行分析发现,其在每个程序运行时出现的次序有一定规律,如频繁的文件操作。Linux 下 vsftpd-2.0.6 运行时所截取到的部分系统调用序列代码如下:

```
open("/etc/ld.so.cache",O_RDONLY)=3
fstat64(3,{st_mode=S_IFREG|0644,st_size=98823,…})=0
old_mmap(NULL,98823,PROT_READ,MAP_PRIVATE,3,0)=0xb7ef0000
colse(3)=0
munnap(0xb7eff0000,98823)=0
open("/usr/lib/libgssapi_krb5.so.2",O_RDONLY_=3
read(3,"\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\362\217",…,512)=512
fstat64(3,{st_mode=S_IFREG|0755,st_size=96108,…})=0
old_mmap(NULL,93132,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_DENYWRITE,3,0)=0x8d1000
old_mmap(0x8e7000,4096,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x16000)=0x8e7000
close(3)=0
```

这样,可以对上述系统调用序列进行划分,系统调用 1 ~ 5 可以划分为一个序列子集,表示程序的一个行为,即打开一个文件,取文件状态,对其内容进行内存映射等操作,然后关闭文件,并释放该文件所占用的资源;系统调用 5 ~ 10 则为程序对另一个文件的一系列操作行为。由此,可将系统调用序列按照

程序的局部行为划分为一个个长短不一的序列片段,称之为序列子集。每一个序列子集代表程序的一个行为,然后对每个序列子集进行组图。具体过程如下:

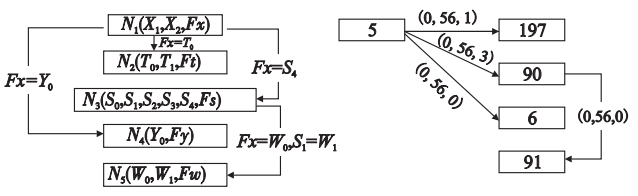
将图的节点表示为每一个系统调用函数,节点属性用操作系统给系统调用分配的序列号(详见文献[12])表示,系统调用参数之间的依赖关系定义为节点之间的边。对边的定义是这样:由于参数的类型各异,通常将这些不同类型的参数间的约束关系按值与参数类型进行划分,定义为边的一种属性。Linux 的参数类型一共有 70 种(详见文献[12]),那么这种边属性一共有 70 种。另外,由于实验平台 Linux 操作系统的特殊性,很多文件操作系统调用的返回值会经常参与到与其他系统调用的依赖关系中,如图 1(b)中 $Fx = T_0$),于是系统调用的返回值也当成一种特殊的参数值进行处理。用 0 表示系统调用的返回值与系统调用的参数值相等,1 表示系统调用的参数值与其他系统调用的参数值相等。对于第二种情况,假定系统调用的参数最多有六个[11],那么考虑到系统调用执行的先后顺序,对于两个系统调用的相等参数对,一共有 36 种可能的组合,而对于第一种情况,则有六种可能的组合。本文用一个三元组$\{X,Y,Z\}$来表示边的属性,$X$ 表示是否返回值与参数值的相等,取值为 0 和 1,$Y$ 表示参数值类型,取值为 0 ~ 69;$Z$ 表示匹配的参数对序数;$Z$ 的取值依赖于 $X$(如果 $X=0$,取值为 0 ~ 5;$X=1$,取值为 0 ~ 35)。图 1 表示了图形化建模的过程。

```
open ("/etc/ld.so.cache", O_RDONLY) = 3
fstat64 (3, {st_mode=S_IFREG|0644, st_size=98823, ···}) = 0
old_mmap (NULL, 98823, PROT_READ, MAP_PRIVATE, 3,
0) = 0xb7ef0000
close (3) = 0
munmap(0xb7ef0000, 98823) = 0
```

(a) 一个系统调用子序列

$n_1$: $X_0$="/etc/ld.so.cache", $X_1$= O_RDONLY, Fx=3;
$n_2$: $T_0$=3, $S_1$={st_mode=S_IFREG|0644, st_size=98823, ···}, Ft=0;
$n_3$: $S_0$= NULL, $S_1$=98823, $S_2$= PROT_READ, $S_3$=MAP_PRIVATE, $S_4$=3, $S_5$=0, Fs=0xb7ef0000;
$n_4$: $Y_0$=3, Fy=0;
$n_5$: $W_0$=0xb7ef0000, $W_1$=98823, Fw=0;
$n_1 \rightarrow n_2$: Fx=$T_0$;
$n_1 \rightarrow n_3$: Fx= $S_4$;
$n_1 \rightarrow n_4$: Fx= $Y_0$;
$n_3 \rightarrow n_5$: Fs= $W_0$, $S_1$= $W_1$;

(b) 参数间的依赖关系




(c) 依赖图　　(d) 行为子图

图1　图形化建模过程

在图 1(d)的程序行为子图中没有标出依赖关系 $S_1 = W_1$,这是因为在系统调用参数依赖关系的分析中,有一部分参数远不如其他的参数重要,这些参数与其他参数之间的依赖关系并不是关心的描述程序行为的最主要的依赖关系。因此针对具体的系统调用对其参数依据重要程度进行适当的删减。另外,为了提取所需要的程序特征胎记,还对截获的系统调用序列进行了其他的处理。所有的操作具体如下:

a) 参数删减。例如获取文件状态的系统调用函数 fstat64[13],参数"3"表示其操作文件的描述符,是最重要的参数;"st_mode = S_IFREG" 等参数表示所获取的文件状态,通常在程序行为中不会当成重要参数传递,除非有专门针对文件状态进行处理的系统调用函数。所以在组图的时候,这部分的参数可以舍弃掉。这样做还可以避免假边依赖[5],即有许多无关的系统调用的整型参数与所关心的依赖关系中的系统调用参数值相等。组图时的数据处理开销也节省许多。对图 1(a)中的序列进行参数删减后的序列如下:

```
open("/etc/ld. so. cache") = 3
fstat64(3) = 0
old_mmap(3) = 0xb7ef0000
close(3) = 0
mumap(0xb7ef0000) = 0
```

从以上序列可以很清楚地看到各系统调用参数之间的主要流向关系,能更加明确地表达出程序的行为。

b)系统调用序列聚类。在组建系统依赖图时,有很多系统调用是对于同一个资源连续的相似操作,如 1 000 次向一个文件中连续地读入 1 Byte,相当于一次往这个文件中读入 1 000 Byte。系统调用中诸如此类的操作都可以聚集为一个节点。对系统调用的聚类操作不仅使得依赖图中的节点数目大幅度减少,降低进行图匹配计算时的内存开销,而且可以得到系统调用序列的一些简单的等价形式,便于在组图时对程序行为进行描述。

c)序列子集的截取与统计。依据前面的分析,将截取到的系统调用序列按照程序的行为动作划分为一个个序列片段,每一个序列片段代表程序的一个局部行为。Linux 系统平台是以文件的形式管理所有资源,即使是硬件设备。所以在一个程序中,大部分的系统调用都是对文件的操作,并且有一定的次序约束。比如在文件操作中,对于同一个文件描述符,close 函数必须在 open 函数之后,网络操作中 receive 函数必须在 send 函数之后等,可以将系统调用序列按一定的程序行为规律进行划分,如图 1(a)所示。然后统计相同的子序列出现次数,组成子序列集,本文中用 $DL$ 表示,$DL = \{(L_1,F_1),(L_2,F_2),\cdots,(L_n,F_n)\}$。其中 $L_i$ 和 $F_i(1 \leqslant i \leqslant n)$ 分别表示每一个子序列和其出现的次数。

## 3　胎记的定义

本文对于胎记定义是这样的:对于前面 $DL$ 中的每一个子序列分别进行组图,得到 $Dg = \{(G_1,F_1),(G_2,F_2),\cdots,(G_n,F_n)\}$,$G_i$ 和 $F_i$ 分别表示每个行为子图以及其数目。对 $Dg$ 中的行为子图两两之间进行图同构的匹配[14],将同构的子图合并为一个,其次数值为两者之和,得到新的集合,用 $Dg' = \{(G_1',F_1'),(G_2',F_2'),\cdots,(G_m',F_m')\}$表示,则 $G_i'$表示程序 $P$ 中的每一种行为子图,$F_i'$表示其出现次数($F_i' = F'(G_i')$),然后依次将 $F_i'(1 \leqslant i \leqslant n)$除以总图数$\sum_{i=1}^{m} F_i'$得到比值 $f_i$。为了区别,本文用 $B(p) = \{(g_1,f_1),(g_2,f_2),\cdots,(g_m,f_m)\}$表示,并将此定义为程序 $P$ 的一个动态胎记。

## 4　程序相似性计算

为了对给定程序 $P$、$Q$ 之间进行相似性比较,进行了如下工作:给定程序 $P$、$Q$ 及其提取的程序胎记 $B(P)$和 $B(Q)$,令 $B(P) = \{(p_1,Pf_1),(p_2,Pf_2),(p_3,Pf_3),\cdots,(p_n,P_{fn})\}$,$B(Q)$ =

$\{(q_1, Q_{f1}), (q_2, Q_{f2}), (q_3, Q_{f3}), \cdots, (q_m, Q_{fm})\}$。基本算法如下:

a)首先找出程序 $P$ 的每一个行为子图 $p_i$(如图 1(d))与程序 $Q$ 的所有行为子图 $q_j$ 之间的最大共同子图,再在 $p_i$ 的所有最大共同子图中选取节点数目最大的作为 $p_i$ 与对应的 $q_j$ 之间的最大共同子图,记为$(p_i, q_j, \mathrm{mcs}(p_i, q_j))(1 \leqslant i, j \leqslant n)$。若 $p_i$ 与 $q_j$ 有多个最大共同子图,任选其中的一个作为代表,使得 $q_j$ 在$\{(p_i, q_j, \mathrm{mcs}(p_i, q_j))\}$中不重复出现。

b)在$\{(p_i, q_j, \mathrm{mcs}(p_i, q_j))\}$集中,若存在$|\mathrm{mcs}(p_i, q_j)| = |\mathrm{mcs}(p_i, q_k)|, 1 \leqslant k, j \leqslant m$ 且 $k \neq j$,则从$\{(p_i, q_j, \mathrm{mcs}(p_i, q_j))\}$集中删去 $\mathrm{mcs}(p_i, q_j)/|q_j|$ 与 $|\mathrm{mcs}(p_i, q_k)|/|q_k|$ 的值较小的,$|q_j|, |q_k|$ 分别表示子图 $q_j$ 与 $q_k$ 中节点个数,得到 $p_i$ 的最大共同子图集 $S_{pi} = \{(p_i', q_j', \mathrm{mcs}(p_i', q_j'))\}$。

c)对于程序 $Q$,重复 a) ~ b)同样的操作,得到 $q_j$ 的最大共同子图集 $S_{qj}$。

d)将$\{S_{pi}\}$与$\{S_{qj}\}$求交集(视$(p_i, q_j, \mathrm{mcs}(p_i, q_j))$与$(q_j, p_i, \mathrm{mcs}(q_j, p_i))$为同一项)得到结果集$\{(p_k, q_k, \mathrm{mcs}(p_k, q_k))\}$,用 $W_{p,q}$ 表示。

e)计算 $W_{p,q}$ 中,依据定义 6,每两图之间的相似度,$S(p_k, q_k) = |\mathrm{mcs}(p_k, q_k)|/\max(|p_k|, |q_k|)$。假设 $W_{p,q}$ 中的元素个数为 $t$,则程序 $P$ 与 $Q$ 的相似度可以定义为

$$S_{p,q} = \left(\sum_{k=1}^{t} \min(P_{fk}/Q_{fk}, Q_{fk}/P_{fk}) \times S(p_k, q_k)/n\right) \times \min\left(\sum_{k=1}^{t} F(p_k)/\sum_{i=1}^{n} F(p_i), \sum_{k=1}^{t} F(q_k)/\sum_{j=1}^{m} F(q_j)\right)$$

其中:$P_{fk}$ 和 $Q_{fk}$ 为 $W_{p,q}$ 中 $p_k$ 与 $q_k$ 在各自程序中出现的几率。

## 5　实验与测评

### 5.1　实验

本文比较了六组功能相似程度不同的程序相似度,采用虚拟机搭建的 Linux 操作系统 FC4 作为实验平台,运用 strace-o 命令对六组运行中的程序进行了跟踪和系统调用搜集,两个程序的行为子图的最大共同子图的计算采用文献[15]中的最大共同子图算法,实验结果如表 1 所示。

表 1　实验结果数据

| 程序名称 | 参与比较子图数($\sum F(p_j')$) | 行为子图数 | 总系统调用总数 | 相似度 |
|---|---|---|---|---|
| KuickShow/Kview | 278/371 | 298/389 | 10631/15631 | 0.597 |
| vsftpd(2.1.0/2.0.3) | 26/30 | 26/30 | 361/374 | 0.852 |
| bzip2/gzip | 1269/1269 | 1274/1274 | 16264/16264 | 1 |
| jar/tar | 5/4 | 19/25 | 223/171 | 0.106 |
| Ksnapshot/scrot | 165/48 | 270/51 | 14844/574 | 0.259 |
| java_make/c_make | 12/4 | 77/45 | 635/998 | 0.041 |

a)KuickShow 和 KView 是 FC4 桌面上的两个看图软件,分别用其打开桌面的一张图文件,并跟踪搜集其系统调用。

b)Vsftpd 2.1.0 和 vsftpd 2.0.3 是两个不同版本的 FTP 软件,分别以匿名身份登录到本地主机,查看主机中的文件。

c)Bzip2 和 gzip 虽是 Linux 下的两个压缩软件,但是不能单独使用,而是通过 tar -xjf 和 tar -xzf 命令分别对 glib-2.19.0.tar.bz2 和 glib-2.19.0.tar.gz 两个压缩文件进行解压,搜集解压过程中的系统调用集。

d)Jar 和 tar 是两个打包软件,但不同的是,jar 是 Java 类文件的一种压缩工具,生成的 jar 文件直接被 Java 运行;而 tar 则是一种文件打包工具,程序实现不同。本文分别用 jar 和 tar 对/application/alcatraz-0.64/GUI 目录下 *.class 文件和/application/etrace-0.8.2/libetrace/目录下的 *.h 文件打包。

e)KSnapshot 和 scrot 是截图软件,但不同的是 KSnapshot 是基于 KDE 桌面环境的一个截图软件,而 scrot 是个基于终端使用的截图小软件。虽然精简小巧,但是两者的功能大同小异,scrot 几乎能完成 KSnapshot 所有的基本功能,分别应用其对桌面、窗口和鼠标截取的区域进行截图操作。

f)Java_make 和 C++_make 分别表示 Java 编译器和C++编译器,实现完全不同。本文应用 javac 和 make 命令对/application/alcatraz-0.64/GUI 目录下的 *.java 文件和/application/etrace-0.8.2/下的所有 C++源代码进行了编译。

### 5.2　分析与总结

1)对于程序相似性的检测

从实验数据可以看到,对于功能一致、实现相同的程序,如 Vsftpd-2.1.0 和 Vsftpd-2.0.3、bzip2 和 gzip,在相同的输入下程序的相似程度比较高。对于实现不同,功能也有很大出入的程序在不同的输入下用本文提出的方法比较则显示出很低的相似度,如 jar 和 tar、java_make 和 C_make,而介于中间的程序,如 Kuickshow 和 KView、KSnapshot 和 Scrot,前者在实现上比较接近,但功能上有部分差异,后者在功能上接近但是具体实现不同,因此前者的相似度要大些。由此可以看到,本文的检测方法对相似程度各异、规模大小不一以及功能繁简程度不一的程序相似程度有较好的体现和证明。这主要是由于本文提取的胎记特征中,行为子图以及其在程序中的出现几率较好地表征了程序局部行为及其在整体程序中的分布,对程序行为特征有较好的保留性,同时也具有良好的区别性。

2)方法的健壮性

由于采用程序局部行为图形建模的方式,提取动态胎记能不受每次程序运行时因进程调度差异或 CPU 并行处理所造成的系统调用序列调用次序差异的影响,并且能够能够抵抗一部分 API 混淆技术[2~4],如添入一部分无关的系统调用或将无约束关系的系统调用次序打乱以隐藏其剽窃的痕迹。

3)未来的研究工作

a)关于相似性阈值的设定。文献[4]中依据文中程序胎记的定义,将检测的经验阈值界定为 0.2,它是依据多次实验数据结果值而设定的。相似度小于阈值 0.2 的两个程序可以视为相互独立的两个程序,相似度大于 0.8 的两个程序可以视为相似的两个程序,属于中间值的不作分类。对于本文提出的方法中相似性界定阈值的设定工作还在研究中,需要更多的实验予以证明。

b)提高方法的健壮性。本文的方法虽然能抵抗一部分 API 混淆技术,但是如果在不破坏程序原有功能的基础上在程序中添加一部分自己定义的代码,如添加一些其他功能,使得触发一部分系统调用序列是行为依赖的子序列,那么将破坏程序中原有行为子图的分布几率,从而对程序胎记造成一定程度的破坏,影响程序相似性的测量准确度。但是这种方法需要比较专业的知识和技能,人力物力的花费将相应增加,那么这种类型的 API 攻击也将大大减少,但不能排除这种情况的可能。为了提高程序胎记的健壮性,在未来的研究中将致力于其他程序胎记特征的挖掘,在对更多的程序进行检测的过程中,将参考进程中资源的访问,对系统调用的参数间关系作更多的挖掘,如程序局部行为之间的联系、系统调用从属功能函数之间

的结构关系、两个进程之间的通信等,这对程序整体行为的描述将更为精确。另外,除了上述的 API 混淆技术,每个程序在启动时的普遍加载动作也会对程序的相似度计算带来一定误差,对于这些问题的研究一并留给将来的研究工作。

## 6 相关工作

关于程序相似性的研究,前人的工作主要集中在程序代码的相似度检测,常见的方法为属性计数法和结构度量法[16~19]。前者只对代码的各种统计属性进行处理,后者则对程序的内部结构进行分析,大多数的程序相似性检测系统将两种方法结合在一起应用,如斯坦福大学的 Moss 系统[20]、德国卡尔斯鲁厄大学的 JPlag[21]、威奇塔州立大学的 Sim[22]、悉尼大学的 YAP3[23]等。其他的源代码检测方法还有代码克隆[24],程序水印、指纹[24],基于编译优化和反汇编的程序相似性检测[25]等。程序胎记技术源于对程序自身各种特征的提取和定义,在程序源代码基础上提取的程序胎记特征主要有初始化类的常量值、方法调用序列、继承结构和程序的类文件等[6,7]。随着软件无源代码的发布,程序相似性研究逐渐趋向于从程序运行时的行为特征中提取证明信息[1~4,8,26~28]。对基于系统调用的程序行为的研究,多用于计算机安全领域的入侵检测方面[11,29],主要是利用系统调用序列不同方面的特性进行建模与分析,判别程序是否正常。其中文献[5]中的内容与本文的研究相似,它是通过比较恶意程序与一组良性程序的依赖图之间的最小相对子图[15],检测出恶意程序异于良性程序的恶意行为。本文基于程序相似性的研究,侧重的是两个程序行为的相似之处,因此比较的是两个行为图之间的最大共同子图。其他将系统调用应用于程序相似性的研究还有日本奈良先端科学技术大学 Tamada 等人[1,2]利用系统调用序列之间的先后次序及其出现的频率作为程序的动态胎记,提出并应用于程序剽窃的检测中。德国萨尔不吕肯大学的 Schuler 等人[4]将系统调用序列用定长的滑动窗口依次截取后的短序列片段定义为程序的动态胎记,这样可以减少巨大的系统调用序列数量。

## 7 结束语

本文首先介绍了程序动态胎记技术的抽象定义及相关属性,结合图形在数据结构上的语义表达优势,将系统调用序列以及其间的参数关系用图形来建模,在此基础上定义了一种基于程序行为子图及其分布的程序动态胎记,并在此基础上提出了一种程序相似性的比较方法。对此方法的实验检测表明,该方法能较好地体现相似程度各异、大小及功能繁简程度不一的程序相似度,具有一定的可信度和适用性。未来的研究工作在组图建模上将考虑更多程序局部之间的行为,这对程序整体行为的描述将更为精确。

**参考文献:**


[1] OKAMOTO K, TAMADA H, NAKAMURA M, *et al*. Dynamic software birthmarks based on API calls[J]. IEICE Trans on Information and Systems, 2006, 89(8):1751-1763.

[2] TAMADA H, OKAMOTO K, NAKAMURA M, *et al*. Dynamic software birthmarks to detect the theft of windows applications[C]//Proc of International Symposium on Future Software Technology. 2004.

[3] SCHULER D, DALLMEIER V. Detecting software theft with API call sequence sets[C]//Proc of Workshop on Software Reengineering. 2006.

[4] SCHULER D, DALLMEIER V, LINDIG C, *et al*. A dynamic birthmark for Java[C]//Proc of the 22nd IEEE/ACM International Conference on Automated Software Engineering. New York: ACM Press, 2007:274-283.

[5] CHRISTODORESCU M, JHA S, KRUEGEL C. Mining specifications of malicious behavior[C]//Proc of the 6th Joint Meeting European Software Engineering Conference and the ACM SIGSOFT International Symposium on Found Ations of Software Engineering. 2007:3-7.

[6] TAMADA H, NAKAMURA M, MONDEN A, *et al*. Detecting the theft of programs using birthmarks, NAIST-IS-TR-2003014[R]. Nara: Nara Institute of Science & Technology, 2003.

[7] TAMADA H, NAKAMURA M, MONDEN A, *et al*. Design and evaluation of birthmarks for detecting theft of Java programs[C]//Proc of the IASTED International Conference on Software Engineering. 2004: 569-575.

[8] MYLES G, COLLBERG C. Detecting software theft via whole program path birthmarks[C]//Proc of the 7th International Conference on Information Security. Berlin: Springer-Verlag, 2004:404-415.

[9] BUNKE H, SHEARER K. A graph distance metric based on the maximal common subgraph[J]. Pattern Recognition Letters, 1998, 19(3-4):255-259.

[10] BUNKE H. Graph matching: theoretical foundations, algorithms, and applications[C]//Proc of Vision Interface Montreal. 2000:82-88.

[11] 朱国强,刘真,李宗伯. 对计算机系统中程序行为的分析和研究[J]. 计算机应用,2005,25(12):2739-2741.

[12] ZHEN Kai-liang. Etrace[CP/OL]. [2008-05-30]. http://www.seclab.cs.sunysb.edu/etrace/.

[13] Linux C function()[EB/OL]. (2003-06-06) [2009-01-30]. http://man.chinaun-ix.net/develop/c&c++/linux_c/default.htm.

[14] VENYO M. The Vflib graph matching library[CP/OL]. (2003-05) [2008-08-30]. http://amalfi.dis.unina.it/graph/.

[15] TING R M H, BAILEY J. Mining minimal contrast subgraph patterns[C]//Proc of the 6th SIAM International Conference on Data Mining. 2006:638-642.

[16] 周高 ,彭四伟. 源代码在线测评系统中剽窃检测技术的研究与实现[J]. 计算机与信息技术,2005(12):85-87.

[17] 陈金宏,刘东升. 程序代码相似度自动度量技术研究综述[J]. 内蒙古师范大学学报,2006,35(4):457-461.

[18] 邓爱萍,徐国梁,肖奔. 基于串匹配方法的源代码复制检测技术研究[J]. 科学技术工程,2007,7(10):63-66.

[19] 邓爱萍,徐国梁,肖奔. 程序源代码剽窃检测串匹配算法的研究[J]. 计算机工程与科学,2008,30(3):62-64.

[20] AIKEN A. Moss: a dystem for detecting software plagiarism[CP/OL]. [2009-04-23]. http://www.cs.berkeley.edu/~aiken/moss.html.

[21] PRECHEL L, MALPOHL G, PHIPPSEN M. Finding plagiarisms among a set of programs with JPlag[J]. Journal of Universal Computer Science, 2002, 8(11):1016-1038.

[22] GITCHELL D, TRAN N. Sim: a utility for detecting similarity in computer programs[C]//Proc of the 30th SIGCSE Technical Symposium on Computer Science Education. 1999.

[23] WISE M. YAP3: improved detection of similarities in computer program and other texts[C]//Proc of the 27th SIGCSE Technical Symposium on Computer Science Education. 1996:130-134.


(下转第 551 页)

实验的方法分别是将两个数据抽取操作并行执行(这两个抽取操作不存在依赖关系),以及将这两个数据抽取操作串行执行,即一个操作结束后才启动另一个操作(以下的分析中将这两种方法简称为并行方法与串行方法)。每一组实验数据都是三次实验数据取平均值的结果。

实验场景一　数据源 S1 位于 SQL Server 数据库中,数据源 S2 位于 Oracle 数据库中,工作数据库位于 SQL Server 数据库中。表 1 是实验数据。

表 1　场景一实验数据

| 数据元组数 | 并行方法/ms | 串行方法/ms | 加速比 |
|---|---|---|---|
| 1 000 | 3 677 | 5 568 | 1.51 |
| 5 000 | 7 896 | 13411 | 1.70 |
| 10 000 | 12 313 | 22 224 | 1.80 |
| 20 000 | 21 943 | 41 833 | 1.91 |
| 50 000 | 53 765 | 104 771 | 1.95 |
| 100000 | 107 365 | 211 786 | 1.97 |

实验场景二　数据源 S1 位于 SQL Server 数据库中,数据源 S2 位于 MySQL 数据库中,工作数据库位于 SQL Server 数据库中。表 2 是实验数据。

表 2　场景二实验数据

| 数据元组数 | 并行方法/ms | 串行方法/ms | 加速比 |
|---|---|---|---|
| 1 000 | 3 843 | 5 416 | 1.41 |
| 5 000 | 7 802 | 13 453 | 1.72 |
| 10 000 | 12 563 | 12 563 | 1.83 |
| 20 000 | 23 396 | 42 894 | 1.83 |
| 50 000 | 55 302 | 103 059 | 1.86 |
| 100 000 | 112 328 | 209 844 | 1.87 |

由上面的实验数据可以看出,本文所提出的算法可以有效提高 ETL 工作流的执行效率。本实验中,并行执行的活动数为 2,理想情况下加速比为 2,实验结果显示实际加速比在 1.4 ~ 2.0,当数据量增大时,加速比也随着增大。这是因为当一个活动所处理的数据量比较小时,迁移数据本身所需要的时间也比较少,而创建线程需要一定的时间开销,此时占用的比例较大;而当处理的数据量比较大时,创建线程所需要的时间占总时间的比例下降,达到了比较好的加速比。ETL 工具所处理的数据量通常情况下都比较大,本文所提出的这个算法所带来时间效率的提高是比较明显的,具有很强的实用性。

## 4　结束语

本文首先介绍了 ETL 的概念模型和逻辑模型,ETL 概念模型主要是建立数据源与数据仓库之间的映射关系,而 ETL 逻辑模型则主要是确定流程中各个活动的执行优先级及其语义。本文通过一个电子商务的例子阐述了建立 ETL 概念模型和逻辑模型的过程,提出了一个算法,分析 ETL 工作流并计算工作流中各个活动的执行优先级,将优先级相同并且相互之间没有依赖关系的活动放在同一个执行阶段,通过创建多个线程并行执行这些活动,提高了 ETL 工作流的执行效率。实验数据表明,该算法与现有的串行算法比较在时间效率方面具有较大优势,并且随着数据量增大,加速比提高,当参与计算的数据量比较大时,加速比可以趋近于理想值。数据仓库环境中的 ETL 工具经常需要处理海量数据,这个算法对于 ETL 工具性能的提高具有较强的实用性。

**参考文献:**


[1] STRANGE K. ETL was the key to this data warehouse's success, Technical Report CS-15-3143[R]. 2002.

[2] VASSILIADIS P, SIMITSIS A, SKIADOPOULOS S. Conceptual modeling for ETL processes[C]//Proc of the 5th ACM International Worshop on Data Warehousing and OLAP. 2002:14-21.

[3] SIMITSIS A, VASSILIADIS P. A methodology for the conceptual modeling of ETL processes[C]//Proc of the Decision Systems Engineering. 2003:305-316

[4] TRUJILLO J, LUJAN-MORA S. A UML based approach for modeling ETL processes in data warehouse[C]//Proc of LNCS, vol 2813. 2003:307-320.

[5] VASSILIADIS P, SIMITSIS A, GEORGANTAS P, *et al*. A generic and customizable framework for the design of ETL scenarios[J]. Information Support Systems, 2005,30(7):492-525.

[6] VASSILIADIS P, SIMITSIS A, SKIADOPOULOS S. Modeling ETL activities as graphs[C]//Proc of the 4th International Workshop on Decision and Management of Data Warehouses. 2002:52-61.

[7] SIMITSIS A, VASSILIADIS P. A method for the mapping of conceptual designs to logical blueprints for ETL processes[J]. Decision Support Systems, 2008,45(1):22-40.

[8] 张忠平,赵瑞珍.基于结构图的 ETL 过程建模方法[J].计算机应用研究,2008, 25(11):3354-3356.

[9] 张旭峰,孙未未,汪卫,等.增量 ETL 过程自动化产生方法的研究[J].计算机研究与发展,2006,43(6):1097-1103.


(上接第 536 页)


[24] BAXTER I D, YAHIN A, MOURA L M D, *et al*. Clone detection using abstract syntax trees[C]//Proc of International Conference on Software Maintance. Washington DC: IEEE Computer Society, 1998:368-377.

[25] COLLBERG C, MYLES G, HUNTWORK A. SandMark: a tool for software protection research[J]. IEEE Magazine of Security and Privacy, 2003, 1(4):40-49.

[26] DALLMEIER V, LINDIG C, ZELLER A, *et al*. Lightweight defect localization for Java[C]//Proc of the 19th European Conference on Object-Oriented Programming. 2005.

[27] MYLES G, COLLBERG C. K-gram based software birthmarks[C]//Proc of ACM Symposium on Applied Computing. 2005:314-318.

[28] LU Bin, LIU Fen-lin, GE Xin, *et al*. A software birthmark based on dynamic opcode n-gram[C]//Proc of International Conference on Semantic Computing. 2007:37-44.

[29] 杨路明,符鹤. 基于系统调用的入侵检测方法研究[J].现代电子技术,2005, 28(4):36-39.

[30] 周立萍, 陈平. 逆向工程发展现状研究[J].计算机工程与设计, 2004, 25(10): 1658-1659.

[31] KAKIMOTO T, MONDEN A, KAMEI Y, *et al*. Using software birthmarks to identify similar classes and major functionalities[C]//Proc of International Conference on Mining Software Repositories. 2006: 22-23.

[32] 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):712-715.