一种基于线程关系的软件水印算法

许金超,曾国荪
(同济大学计算机科学及技术系,上海 201804;嵌入式系统与服务计算教育部重点实验室,上海 201804)

摘 要: 针对目前基于线程顺序的软件水印算法存在的隐藏信息量小效率不高的不足,提出了一种新的基于线程间关系的软件水印算法.其主要思想是通过修改程序的源代码控制程序运行中线程间的相互关系,从而在线程关系中隐藏软件水印.文中给出了关系和关系矩阵等概念的形式化定义,描述了软件水印的嵌入和提取过程.文章对该算法的不易觉察性和数据率进行分析比较,总结了有针对性的攻击方式,并对攻击下软件水印的安全性进行实验验证.

关键词: 数字版权;软件水印;线程关系;水印算法
中图分类号: TP391 **文献标识码:** A **文章编号:** 0372-2112 (2012) 05-0891-06
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.05.005

A Software Watermarking Algorithm Based on Threads Relation

XU Jin-chao, ZENG Guo-sun
(*Department of Computer Science and Technology, Tongji University, Shanghai* 201804, *China*;
The Key Laboratory of Embedded System and Service Computing, Ministry of Education, Shanghai 201804, *China*)

Abstract: The existed thread-based software watermarking algorithm can only hide a small amount of information and have low efficiency. Aimed to this status, this paper proposes a new software watermarking algorithm based on the relationship between threads. The main idea is to control the interactive relationship between threads which hides software watermarking by modifying the program's source code. This paper gives a formal definition of the concepts such as relation and relation matrix, and a detailed description of the software watermark embedding and extraction process. We also analyze the date rate and fidelity of the algorithm. Various kinds of attacks used in the experiment, results indicate this algorithm has high resistance.

Key words: digital copyright; dynamic software watermarking; threads relation; watermarking algorithm

1 引言

长久以来,知识产品、人类创新、数字版权的保护一直是一项艰巨的任务.网络时代,数字产品的分发变得愈加容易,版权保护的现状更是不容乐观.随着软件产品市场份额的增加,软件保护成为一个迫切的问题.软件水印技术是所有软件保护的研究中较新的一个领域,它将标志版权的秘密信息嵌入到要保护的软件文件中达到保护的目的,这些秘密信息不易被察觉,并且难以清除,在需要的时候可以提取出来证明该文件的版权所有.

相对于其他软件保护技术,对于软件水印技术的研究起步较晚,理论基础薄弱,已有的算法在种类和数量上也较少.1996年,Davidson[1]等提出了第一个软件水印算法,算法通过重排程序的基本块嵌入水印.其后 Qu[2]等提出了寄存器分配算法,算法基于图着色问题.Stern[3]等提出了扩频算法,算法把程序看作向量,在此基础上做微小的改动嵌入软件水印.Arboit[4]提出了透明谓词算法,软件水印嵌入在判断语句和不执行的代码中.Cousot[5]提出了抽象解释器框架算法,软件水印分散到程序执行过程中的整数局部变量的值中,文献[6]提出的循环软件水印也属于抽象解释的范畴.Collberg[7]等人提出了动态路径算法,算法通过改变程序运行时的分支语句嵌入软件水印.除此之外,基于图论的软件水印算法是软件水印的一个重要研究方向.Venkatesan[8]等提出了第一个称作 VVS 的基于图论的软件水印算法,这是一种静态水印算法,软件水印编码成控制流程图的形式嵌入在程序中.Collberg[9]等提出了名为 CT 算法的动态图算法,其软件水印嵌入在程序运行时生成的堆中的数据结构中.

收稿日期:2011-04-19;修回日期:2011-08-06
基金项目:国家 973 重点基础研究发展规划(No.2007CB316502);国家自然科学基金(No.90718015);NSFC-微软亚洲研究院联合资助(No.60970155);教育部博士点基金(No.20090072110035);上海市优秀学科带头人计划(No.10XD1404400)

Nagra[10]等给出了线程软件水印算法(TBW 算法, Thread-Based Watermarking),该算法利用同步使得程序中的线程按照一定的顺序执行,其中每 3 个线程的顺序表示 1bit 信息.此算法的一个显著优点是能够抵抗许多语义保持变换攻击[11],同时对程序大小和运行时间的影响保持在一个可以容忍的范围内.但是该算法存在效率低下的缺陷.为了保证算法可以提供足够的水印容量,该算法需要大量的线程,且利用同步来来保证大量线程的顺序执行其实现过程较为繁琐,从而导致该算法很难嵌入图像等形式的大数据量的软件水印.

本文吸取了线程可以提供高安全性、易控制的特点,提出了一种高效率的基于线程间关系的软件水印算法,算法通过修改源代码控制线程间关系,利用线程关系嵌入和提取软件水印.算法克服了 TBW 算法必须使用冗余线程,并保持大量线程的顺序执行导致效率下降的缺点,能够充分利用程序中已有的线程,并只需考虑两个线程之间的关系,提高了效率.分析和实验说明,本算法在保持 TBW 算法优点的同时,简化了实现过程,极大的提高了软件水印容量.

2 线程关系矩阵

令 P 表示一个软件程序,定义 $P=\{t_1,t_2,t_3,t_4,\cdots\}$,其中 $t_1,t_2,t_3,t_4,\cdots$是 P 中的所有线程.

定义 1 在程序 P 的运行过程中,对于程序中线程 $t\in P$ 和 $r\in P$.如果线程 t 的执行会对线程 r 的执行产生直接影响,则称 t 和 r 有关系,记作 $t\rightarrow r$.反之,若 P 的整个运行过程线程 t 的执行都不会影响线程 r 的执行,则称线程 t 和线程 r 没有关系,记作 $t\not\rightarrow r$.

程序 1 Java 线程间的关系示例

```
public class ThreadA extends Thread {
    ThreadB threadB;
    protected ThreadA() {
        super();
    }
    public void run() {
        threadB = new ThreadB();
        threadB.wait();
        this.sleep(1000);
        threadB.notify();
    }
    public static void main(String [] args) {
        ThreadA threadA = new ThreadA();
        threadA.start();
    }
}
public class ThreadB extends Thread {
    protected ThreadB() {
        super();
    }
    public void run() {
        BlockB();
    }
}
```

例如一个线程控制另一个线程挂起或者唤醒等,则此时存在控制线程到被控制线程的关系;再如文献[10]中的一个线程需要按顺序等待另一个线程执行,则此时存在被等待线程到等待线程的关系.这些操作可以方便的在源代码层次控制.例如,程序 1 中两个线程运行后,threadA 会对 threadB 调用 wait 和 notify 方法,而对自己使用 sleep 方法,因此有 threadA →threadB 和 threadA →threadA.并且不难发现,这三条语句只是影响程序的运行时间,并不会损害程序最终的功能.

在不同的平台下,线程间关系的实现并不相同.例如:Java 下可以选择线程中的同步机制.在 Windows 下可以通过互斥锁、临界段等方式实现,或者通过消息机制.根据不同的平台,监控不同的目标记录线程间的关系.线程间的所有关系组成了线程关系图.

定义 2 线程关系图是一个有向图$(\boldsymbol{E},\rightarrow)$,其中 $\boldsymbol{E}$ 是线程集合,$x\rightarrow y(x\in\boldsymbol{E},y\in\boldsymbol{E})$表示存在一个从线程 x 到线程 y 的关系.

定义 3 线程关系矩阵是线程关系图的存储表示,设图 $\boldsymbol{G}=(\boldsymbol{E},\rightarrow)$,其中 $\boldsymbol{E}=\{v_1,v_2,\cdots,v_n\}$,则 $\boldsymbol{G}$ 的线程关系矩阵 $\boldsymbol{A}$ 是按如下定义的一个 n 阶方阵:

$$\boldsymbol{A}=(a_{ij})_{n\times n}:a_{ij}=\begin{cases}1, & v_i\rightarrow v_j\\ 0, & \text{else}\end{cases}$$

3 基于线程关系的软件水印算法

3.1 算法概述

本文提出的算法在嵌入时需要拥有程序的源代码,在此基础上,通过增加和修改源代码操控线程间的作用关系.软件水印算法基本思想是将软件水印嵌入在程序运行时生成的线程关系矩阵中.

嵌入过程共分为四个主要步骤.首先,预处理原始软件水印和宿主程序.其次,监控程序执行,生成线程关系矩阵,并作一些必要的处理,第三,将软件水印嵌入线程关系矩阵,最后,根据修改后的线程关系矩阵修改程序源代码.图 1 给出了流程简图.

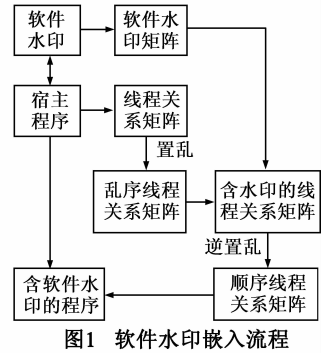


图1 软件水印嵌入流程

提取过程相对简单,它可分为两个主要步骤:首先,监控含软件水印程序的运行得到线程关系矩阵,然后从线程关系矩阵中提取软件水印.图 2 给出了提取算法的流程示意图.

3.2 软件水印的嵌入

设 P 是拥有源代码的程序,I 为程序 P 运行时的秘

密输入,设 P 在输入 I 下运行时共有 n 个线程运行.限定嵌入到多线程程序 P 中的软件水印 W 是宽高相等的二值图像,设其高宽都为 h.嵌入的详细步骤如下:

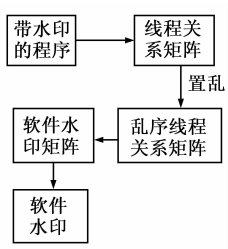


图2　软件水印提取流程

(1)在嵌入软件水印之前,需要先对软件载体和软件水印进行前期处理.

若$\lceil n/2\rceil > h$,则在软件水印 W 对应的二值图像的右侧和下方增加随机噪点,使其高和宽为$\lceil n/2\rceil$,除此以外在程序 P 的源代码中加入 0 个或 1 个生成不执行实际任务的空线程的代码,使修改之后的程序 P' 在 I 下运行时执行的总线程数为$2\lceil n/2\rceil$.

若$\lceil n/2\rceil \leqslant h$,则在程序 P 的源代码中添加代码使其生成不执行实际任务的冗余线程,使得修改之后的程序 P' 在 I 下运行时执行的总线程数为 $2h$.

令 $m = \max\{h, \lceil n/2\rceil\}$,则 P' 拥有 $2m$ 个线程;处理之后的软件水印图像的大小为 $m \times m$.则图像数据可以直接对应一个 $m \times m$ 阶矩阵 $\boldsymbol{M}$,其中矩阵中的每一个元素为 0 或 1.

(2)在输入 I 下运行编译后的程序 P',记录执行后的线程交互关系并按照程序中各个线程开始执行的先后次序对线程进行排序,设这 $2m$ 个线程按顺序用 $t_1, t_2, \cdots, t_{2m}$ 表示.如图 3 所示排列这 $2m$ 个线程,按照定义 3,根据记录的线程关系在矩阵中填入数据,最后得到的线程关系矩阵记为 $\boldsymbol{N}$.

$$\begin{array}{c|cccc} & t_1 & t_2 & \cdots & t_{2m} \\ t_1 & b_{11} & b_{12} & \cdots & b_{12m} \\ t_2 & b_{21} & b_{22} & \cdots & b_{22m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{2m} & b_{2m1} & b_{2m2} & \cdots & b_{2m2m} \end{array}$$
图3　线程关系矩阵$\boldsymbol{N}$及其划分示意图

按照图 3 所示方式依次以两个线程为单位对线程关系矩阵 $\boldsymbol{N}$ 进行划分得到矩阵 $\boldsymbol{B} = \{\boldsymbol{B}_{ij}\}\ m \times m$,其中 $\boldsymbol{B}_{ij}$对应于下式中的 2×2 矩阵.

$$\boldsymbol{B}_{ij} \leftrightarrow \begin{bmatrix} b_{2i-1\,2j-1} & b_{2i-1\,2j} \\ b_{2i\,2j-1} & b_{2i\,2j} \end{bmatrix} \tag{1}$$

其中 b_{ij}是线程关系矩阵 $\boldsymbol{N}$ 中的元素.

对矩阵 $\boldsymbol{B}$ 进行 Arnold 置乱变换,设置乱后的矩阵为 $\boldsymbol{B}'$.

(3)2×2 矩阵 $\boldsymbol{B}_{ij}$共有 16 种可能情况,其中 $\boldsymbol{B}_{ij}$中 1 的个数为奇数的情况共有 $C_4^1 + C_4^3 = 8$ 种,$\boldsymbol{B}_{ij}$中 1 的个数为偶数的情况也共 $C_4^0 + C_4^2 + C_4^4 = 8$ 种.对于普通程序来说,线程的依赖关系不会十分紧密,矩阵 $\boldsymbol{B}_{ij}$中四个元素都为 1 的情况出现的概率非常小,可以忽略不计.

矩阵 $\boldsymbol{B}'$和线程水印矩阵 $\boldsymbol{M}$ 有着同等的维度,$\boldsymbol{B}$ 中元素 $\boldsymbol{B}_{ij}$和 $\boldsymbol{M}$ 中元素 a_{ij}在各自矩阵中的位置相同.如果 a_{ij}为 1,则将其对应的 2×2 矩阵 $\boldsymbol{B}_{ij}$中的 1 的个数修改为奇数,否则修改为偶数.$\boldsymbol{B}_{ij}$有 50% 的概率无需修改,而修改至多将 $\boldsymbol{B}_{ij}$中一个 0 元素改为 1 即可实现.

对上述修改后的矩阵 $\boldsymbol{B}'$进行逆 Arnold 变换,由式(1)可知,此时线程关系矩阵 $\boldsymbol{N}$ 被对应的修改,记修改后的线程关系矩阵为 $\boldsymbol{N}'$.

(4)按照步骤(3)的描述可知,修改后的线程关系矩阵 $\boldsymbol{N}'$较之 $\boldsymbol{N}$ 至多有 m^2 个元素从 0 改为了 1.根据修改前后每个不一致的元素可以找出该元素对应的两个线程 t_i、t_j,不妨设 $\boldsymbol{N}'$中有 $t_i \rightarrow t_j$ 的关系,在 P 的源代码中找到对应的线程代码块 Block_i、Block_j,则此时需要在 Block_i、Block_j 中增加语句,产生 t_i 到 t_j 的关系.这一过程中需要注意加入的语句不能影响程序原来的功能,不能造成死锁.通过控制插入的语句可以保证程序变换后仅在线程间造成延迟,而不对程序的功能有任何影响.程序 1 已经给出了一种关系控制方式示例,文献[10]中给出了另一种控制线程关系的详细的描述.

3.3　软件水印的提取

提取算法已知嵌入算法中最终嵌入的软件水印的高 m,根据给定的输入 I 从含软件水印的程序 P^w 中提取出软件水印 W',其步骤如下:

(1)利用和嵌入过程同样的输入 I 运行程序 P^w,监控并记录程序中参与运行的所有线程和线程关系,设这一过程共监测到 m' 个线程存在.按照线程启动顺序对 m' 个线程加以排序,以排序后的线程作为纵横坐标,类似图 3 根据记录的线程关系生成 $m' \times m'$ 阶线程关系矩阵 $\boldsymbol{N}$.

若 $m' < 2m$,则在在原来的线程顺序末尾添加 $2m - m'$ 个线程,并记录添加的线程与前面的 m' 个线程没有任何关系.

若 $m' > 2m$,则按照下述准则删除其中的 $m' - 2m$ 个线程.优先考虑删除 $\boldsymbol{N}$ 中所在行对应行和列全为 0 的线程,其次删除 $\boldsymbol{N}$ 中所在行和列中除行列相交处之外全为 0 的线程,再次删除 $\boldsymbol{N}$ 中所在行或所在列至多有一个 1 的线程,不存在上述情况下才随机选择线程进行删除.

经过上述操作后的 $2m \times 2m$ 阶矩阵记作 $\boldsymbol{N}'$.

(2)按照嵌入过程步骤 2 中的方式将 $\boldsymbol{N}'$划分为 m^2 个 2×2 矩阵,对每个 2×2 矩阵计算其中 1 元素的个数,如果个数为奇数则将该 2×2 矩阵用 1 代替,个数为偶数则记为 0,得到 $m \times m$ 阶矩阵 $\boldsymbol{N'}_W$.对得到的矩阵 $\boldsymbol{N'}_W$ 进行和嵌入过程相同的 Arnold 置乱变换,得到软件水印矩阵 $\boldsymbol{M}_W$.以软件水印矩阵 $\boldsymbol{M}_W$ 中的每一个元素作为像素点数据得到的二值图像即是要提取的软件水印 W'.

4 实验结果和分析

本文实验基于 3 个开源 Java 程序:TTT、JFIG 和 SciMark.TTT 是一个非常小的井字棋游戏,约有 100 行左右的代码,大部分代码都会参与执行;JFIG 是一个图形编辑器,它有约 23000 行代码,但是大部分代码不会执行;SciMark 是一个命令行程序,有着大约 1300 行代码,其中约 5%的代码执行超过 5 万次.在本文选定的输入下,它们的文件大小和运行时间如表 1 所示.

表 1 实验用到的三个程序的文件大小和运行时间

(实验环境:Pentium(R) 4 CPU 2.40GHz with JDK 1.6)

测试程序	文件大小	运行时间
TTT	0.3kb	30s
JFig	734kb	600s
SciMark	136kb	26s

4.1 不易觉察性

不易觉察性指的是指的是嵌入软件水印后对于观察者的不可察觉程度,这通常用嵌入软件水印后的程序在文件大小、运行时间等较嵌入软件水印之前的程序改变程度来衡量.

图 4 给出了嵌入不同大小的软件水印后对程序大小的影响,图 5 给出了嵌入同样大小的软件水印后对程序运行时间的影响.软件水印分别是 5 种不同大小的二值图像,宽高分别是 1×1,2×2,4×4,8×8,16×16.由图 4 可以看出,当嵌入的软件水印较大时,会增大程序文件的大小.对嵌入的代码进行分析可知,文件大小的增长更多的是由添加的无实际功能的冗余线程代码较多造成的,对于程序内本身线程较多的情况下程序文件大小的增长则不显著.

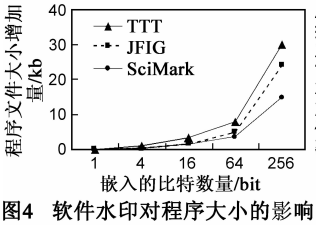


图4 软件水印对程序大小的影响

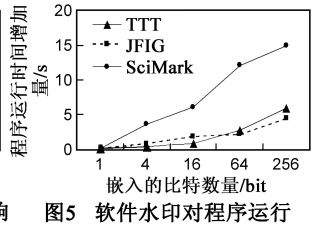


图5 软件水印对程序运行时间的影响

图 5 可以看出,嵌入的数据越多,程序中的线程越多,程序的运行时间就越长.由于算法中采用文献[10]中的同步机制,嵌入软件水印较多时,线程间的等待事件发生频繁,不利于程序中线程的并发运行,导致运行时间存在一定的延迟.

和 TBW 算法相比,在嵌入数据量较小的时候(<40bits),本算法嵌入软件水印前后对宿主程序大小和运行时间的影响比 TBW 稍小,但并未表现出明显的优势.然而随着软件水印数据量的增大,TBW 需要插入的线程增多,其同步将变得非常复杂,这导致 TBW 算法较难嵌入更多的软件水印,文献[10]只给出嵌入 50bits 以下信息的实验数据.由于本文给出的算法只需要在任意两个线程间至多维持二次同步,因此实现较为简洁,由同步导致的对文件大小和执行时间的影响将比 TBW 算法有明显的改善.

4.2 数据率

根据算法嵌入过程的描述可知嵌入 m^2 个比特需要 $2m$ 个线程,即每个线程的效率是 $m/2$ 比特.因此程序内的线程数量愈多,每个线程便越有效率.对于线程比较多的程序,不加入额外的前程即已经拥有较高的容量.随着多核处理器的流行,目前的多线程程序应用越来越多.在云计算成为热门领域的今天,云环境下许多程序都维持一个线程池,同时运行着数十数百的线程,这些都为嵌入较大的软件水印提供了良好的载体.

由于本算法所能嵌入软件水印的大小取决于程序中线程的数目,对于自身线程较少的程序,在知道程序源代码的情况下,可以通过在源代码中添加只产生线程交互关系而不执行实际任务的冗余线程代码达到扩展软件水印容量的目的.例如对于一个拥有 7 个线程的程序,只需要增加 3 个冗余线程就可以嵌入 25bits 的软件水印.而嵌入同样的 25bits 信息,TBW 算法至少需要增加 18 个冗余线程[10].可以看出,本文的算法具有更高的效率,能适用于大容量的软件水印信息的嵌入.

4.3 抗攻击分析

攻击者往往使用攻击使得软件水印难以识别,软件水印算法必须能提供一定的抗攻击能力.

(1)常规攻击

Sandmark[12]是 Arizona 大学开发的 Java 平台下软件水印分析工具,它使用了 Java 下常见的各种攻击方式,攻击描述和实验结果给出的对本算法的影响如表 2 所示.

表 2 给出的程序变换攻击方式并不改变程序本来的功能.原程序和受攻击后的程序对于相同的输入输出保持不变,即这些攻击为语义保持变换攻击.因为这些攻击并未对软件中的线程关系造成影响,所以和 TBW 算法一样,不会影响软件水印的提取,实验也说明了这一点.

因为本文的算法不会受到上述混淆方式的影响,所以可以将嵌入软件水印后的程序进行混淆处理,使得攻击者无法理解线程间的关系,不至于有针对的攻击已嵌入软件水印的程序.

(2)针对线程关系的攻击

有效地攻击应该是针对线程间的关系进行攻击,如调整线程启动顺序、修改线程关系、增加冗余线程、删除无用线程等.为了测试本文提出的算法在针对线

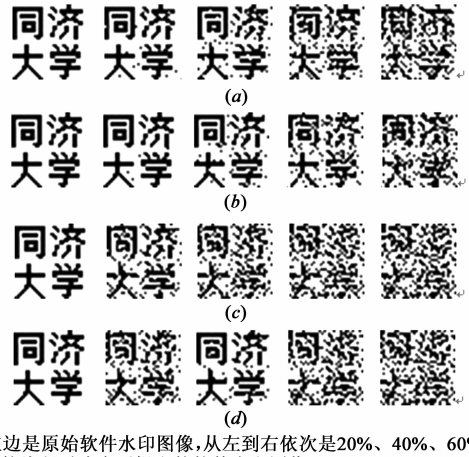
程关系的攻击下的抵抗性,进行了实验.实验中的软件水印选择 32 × 32 的二值图像.因为 TTT 程序在嵌入软件水印的过程中加入了大量仅用来标识线程关系而不执行实际功能的线程,故改变这些加入的线程不会轻易破坏程序的正确运行.为了方便实验的进行,软件水印载体选择 TTT 程序.

表 2　SandMark 中的攻击对本文的软件水印的影响

攻击方式	攻击描述	能否成功提取
数组分解	将一维数组分解为多维数组	能
常量重排	重新排序常量池中的常量	能
类重组	将没有共同方法的两个类组成一个类	能
冗余代码注入	在类中插入冗余代码使之遍布整个类	能
插入透明预测	将每个布尔值用透明预测代替	能
重载方法	将类中不同的方法改成同样的名字	能
公开类中变量或方法	将类中变量或方法公开	能
重命名寄存器	将局部变量分配到不同的寄存器	能
参数重排	重排方法中的参数顺序	能

(ⅰ)改变线程执行顺序

由于程序中线程间可能存在依赖关系,例如某个线程依赖于上一个线程返回的结果,此时无法轻易改变这两个线程间的顺序.在对整个程序不能彻底了解的情况下,改变线程执行顺序会导致未知的风险.



最左边是原始软件水印图像,从左到右依次是20%、40%、60%、80%的线程受攻击后提取的软件水印图像

图6　针对线程的攻击在不同的强度下对提取的影响
(软件水印:32×32二值图像 宿主程序:TTT)

图 6(*a*)给出了算法在受到改变线程执行顺序的攻击方式下的实验情况,可以看到在改动范围较小的时候,并不影响软件水印的正确识别.

(ⅱ)改变线程关系

改变线程间的关系,同样需要对线程间的作用有着清醒的认识,才能发现那些线程间的关系是必需的,那些是多余的,才能正确的修改线程关系而不至于导致程序错误.

图 6(*b*)给出了算法在受到改变部分线程关系的攻击方式下的实验情况.可以看出,本算法对对这种攻击方式有着良好的抵抗能力.

(ⅲ)添加/删除线程

添加线程会导致算法在提取过程中要删掉一个线程.图 6(*c*)给出了算法在受到添加线程的攻击方式下的实验情况.由于仅仅添加一个和其他线程缺少交互的冗余线程将会在提取过程中被删掉,所以实验中还要更改添加的线程与其他线程的交互关系,即图中实验结果实际上是添加线程攻击和更改线程关系攻击的结合.

在不理解其中某个线程的具体作用也不知道该线程和其他线程有无交互就将其删掉是非常危险的.图 6(*d*)给出了算法在受到删除线程的攻击方式下的实验情况.

相对于前两种攻击方式,线程添加/删除攻击有着明显的破坏作用.然而添加线程需要大幅度的改动字节码生成线程代码并使生成的线程在适当时间运行,还需要能够控制其他线程来产生交互关系;而删除线程操作不适用于本身线程较多的程序,这些因素都增加了攻击的难度.

5　结束语

本文提出了一种新的基于多线程关系的软件水印算法,给出软件水印的前期处理、嵌入算法和提取算法的详细过程.实验表明,与现有的线程软件水印算法[10]相比,本文提出的利用程序中的线程关系嵌入软件水印的算法除了对各种针对线程的攻击有着较好的抵抗能力之外,而且实现更为简洁,对原程序的影响较小,并有着更高的嵌入效率.本文提出的算法思想不仅可以基于线程的基础上嵌入水印,而且可以基于程序中的其他元素,例如窗口、代码块等等都可以采用类似的方法嵌入软件水印.目前给出的算法仍有一些步骤可以改进,如软件水印在线程关系矩阵中的嵌入位置,软件水印在线程关系矩阵中的映射关系都可以以一种更有效率的方法给出,对此的进一步的研究将是今后一段时间的工作方向.

参考文献

[1] W Zhu, C Thomborson, F Wang. A survey of software watermarking[A]. Proceedings of 2005 International Conference on Intelligence and Security Informatics[C]. Atlanta: Springer, 2005.454 – 458.

[2] G Qu, M Potkonjak. Analysis of watermarking techniques for graph coloring problem[A]. Proceedings of 1998 International Conference on Computer Aided Design[C]. New York: ACM Press, 1998.190 – 193.

[3] J Stern, G Hachez, et al. Robust object watermarking: application to code[A]. Proceedings of the Third International Workshop on Information Hiding[C]. Dresden: Springer, 1999. 368 – 378.

[4] G Arboit. A method for watermarking java programs via opaque predicates: Implementation, analysis, and attacks[A]. Proceedings of the Fifth International Conference on Electronic Commerce Research[C]. New York: ACM Press, 2006. 155 – 171.

[5] P Cousot, R Cousot. An abstract interpretation-based framework for software watermarking[A]. Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages[C]. New York: ACM Press, 2004. 311 – 324.

[6] M D Preda, R Giacobazzi, E Visentini. Hiding software watermarks in loop structures[A]. Proceedings of the 15th International Symposium on Static Analysis[C]. Heidelberg: Springer, 2008. 174 – 188.

[7] C Collberg, E Carter. Dynamic path-based software watermarking[A]. Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation[C]. New York: ACM Press, 2004. 107 – 118.

[8] R Venkatesan. A graph theoretic approach to software watermarking[A]. Proceedings of the 4th International Workshop on Information Hiding[C]. London: Springer-Verlag, 2001. 157 – 168.

[9] C Thomborson, J Nagra. Tamper-proofing software watermarks [A]. Proceedings of the Second Australasian Information Security Workshop[C]. Darlinghurst: Australian Computer Society Press, 2004. 27 – 36.

[10] J Nagra, C Thomborson. Threading software watermarks[A]. Proceedings of 6th International Workshop on Information Hiding[C]. Heidelberg: Springer, 2004. 208 – 233.

[11] J Nagra. Threading Software Watermarks[D]. Auckland: The University of Auckland, 2007.

[12] C Collberg. SandMark 3.4.0 homepage[OL]. http://sandmark.cs.arizona.edu, 2011-3-18.

[13] C Collberg. Software watermarking: models and dynamic embedding[A]. Proceedings of Symposium on Principles of Programming Languages[C]. New York: ACM Press, 1999. 311 – 324.

[14] Lu Z, Li S Z. Multipurpose watermarking algorithm for secret communication[J]. Chinese Journal of Electronics, 2006, 15 (1): 79 – 84.

[15] 郭萌, 张鸿宾, 魏磊. 二值图像中的数据隐藏算法[J]. 电子学报, 2009, 37(11): 2409 – 2415.
Guo Meng, Zhang HongBin, Wei Lei. Data hiding in binary image[J]. Acta Electronica Sinica, 2009, 37 (11): 2409 – 2415. (in Chinese)

[16] 钮心忻, 杨义先. 信息隐写与隐写分析研究框架探讨[J]. 电子学报, 2006, 34(12A): 2422 – 2424.
Niu XinXin, Yang YX. Study on the frame of information steganography and steganalysis[J]. Acta Electronica Sinica, 2006, 34(12A): 2422 – 2424. (in Chinese)

作者简介

许金超 男, 1982 年生于山东临沂. 博士研究生. 研究方向为软件保护、信息安全.
E-mail: xxujinchao@126.com

曾国荪 男, 1964 年生于江西吉安, 教授, 博士生导师. 研究方向为可信软件、信息安全.