软件水印综述*

张立和+, 杨义先, 钮心忻, 牛少彰

(北京邮电大学 信息安全中心,北京 100876)

A Survey on Software Watermarking

ZHANG Li-He+, YANG Yi-Xian, NIU Xin-Xin, NIU Shao-Zhang

(Information Security Center, Beijing University of Posts and Telecommunication, Beijing 100876, China)

+ Corresponding author: Phn: 86-10-62282715 ext 802, E-mail: zhanglihe@yeah.net

http://www.bupt.edu.cn

Received 2002-05-13; Accepted 2002-09-06

Zhang LH, Yang YX, Niu XX, Niu SZ. A survey on software watermarking. *Journal of Software*, 2003,14(2): 268~277.

Abstract: With the rapid development of software industry, the copyright protection of software product already becomes a very important issue. In this paper, the software watermarking technique is presented in detail. An overview of software watermarking including the taxonomy, the methods of attacking, the current algorithms, and their advantages and disadvantages are presented in this paper. Finally the state of arts and possible new directions of software watermarking are also stated.

Key words: intellectual property right protection; software watermarking; digital watermarking

摘 要: 随着软件产业的迅速发展,软件产品的版权保护已成为一个十分重要的问题.详细介绍了软件水印这种新兴软件版权保护技术,深入分析了软件水印的现状、分类、攻击方法以及已有的各种算法,并分别讨论了这些算法的利弊,最后提出软件水印的下一步发展方向.

关键词: 知识产权保护;软件水印;数字水印

中图法分类号: TP309 **文献标识码**: A

软件水印是近年来出现的软件产品版权保护技术,可以用来标识作者、发行者、所有者、使用者等,并携带有版权保护信息和身份认证信息,可以鉴别出非法复制和盗用的软件产品.计算机网络技术的飞速发展,为数字信息传播提供了极大的便利,但是随之而来的副作用是通过网络传输的数字产品很容易被恶意的个人或团体在未经许可的情况下非法使用,因而网络世界中数字产品版权保护问题成为人们亟待解决的问题.软件水印技术就是在这种情况下产生的.目前在软件版权保护方面,人们主要是通过加密的方式进行,比如:软件狗、Vbox、SoftSENTRY、SecuROM 和 SafeDISC 等[1].最近出现的软件水印则是另一种全新的软件保护措施.

* Supported by the National Natural Science Foundation of China under Grant No.60073049 (国家自然科学基金); the National Grand Fundamental Research 973 Program of China under Grant No.G1999035805 (国家重点基础研究发展规划(973)); the Assisting Project of Ministry of Education of China for Backbone Teachers of University and College (国家教育部高等院校骨干教师资助计划)

第一作者简介: 张立和(1976－),男,山西朔州人,博士生,主要研究领域为信息安全,信息伪装与数字水印.

所谓的软件水印就是把程序的版权信息和用户身份信息嵌入到程序中.美国在软件水印方面已经申请了 4 项专利[2~5].目前,国内外详细而全面地介绍软件水印的论文鲜见,这与多媒体水印的研究热潮形成了鲜明的对比.本文重点对软件水印的分类、攻击方法和现有算法作深入的分析,并讨论了这些算法的利弊,提出了软件水印下一步的发展方向.

1 软件水印的分类

根据水印的加入位置,软件水印可以分为代码水印和数据水印.代码水印隐藏在程序的指令部分中,而数据水印则隐藏在包括头文件、字符串和调试信息等数据中.根据水印被加载的时刻,软件水印可分为静态水印和动态水印.静态水印存储在可执行程序代码中,比较典型的是把水印信息放在安装模块部分,或者是指令代码中,或者是调试信息的符号部分.对于 Java 程序,水印信息也可以隐藏在类文件(包括常量池表、方法表、行号表)的任何部分中.静态水印又可以进一步分为静态数据水印和静态代码水印.区别于静态水印,动态水印则保存在程序的执行状态中,而不是程序源代码本身.这种水印可用于证明程序是否经过了迷乱变换处理.动态水印主要有 3 类:Easter Egg 水印、数据结构水印和执行状态水印.其中,每种情况都需要有预先输入,然后根据输入,程序会运行到某种状态,这些状态就代表水印.下面详细介绍几种水印的特点.

1.1 静态数据水印

数据水印很容易产生和识别,是一种常见的水印.这种水印可以在程序中的一些数据中体现出来,因而很容易被迷乱攻击破坏.比如把所有的数据分解成一系列数据,然后散布到整个程序中,这样代表水印信息的数据也被分解,增加了水印检测的难度;或者用一个产生这些数据的子程序来代替这些数据,这样在程序中就找不到该数据的原型,也就无法检测水印.

1.2 静态代码水印

利用人类视觉和听觉的不敏感性,多媒体水印通常是加在载体上的冗余部分.也可以用相同的方式来构造代码水印,因为目标代码也包含了冗余信息.比如通过调整两条无依赖关系指令的顺序可以嵌入 1bit 的水印信息.IBM 提出了一种把寄存器出入栈的顺序作为水印的方法,同样可以通过排列有 m 个分支的 case 语句的顺序来编码 $\log(m!)$ 比特信息.Davidson[3]描述一种类似的代码水印,它在程序的控制流图的一个基本模块中对软件的序列号进行编码.

许多代码水印都经不起一些简单的水印攻击(比如,调整指令的顺序).既然交换指令的顺序不影响原程序,那么就可以把源代码中所有满足这个条件的指令都交换位置,这样就无法检测到先前加入的水印了.

很多代码迷乱技术能够破坏代码水印.对于 Davidson 的方法,只要能够准确地找到控制流图的基本模块.我们很容易通过插入一个布尔值始终为 TRUE 的条件分支破坏这个基本模块,使水印无法检测.通过迷乱变换,所有静态结构水印都会被破坏.内嵌、循环变换和代码都是常见的优化技术,这些技术也很容易破坏静态代码水印.Moslkowitz[4]提出了一种具有防篡改的水印算法,其基本思想是把关键代码的一部分隐藏在软件的资源(如图标、声音)中,并且程序会不时地从资源中提取出这段代码执行,如果资源被破坏,那么程序就会出错.

静态代码水印更难抵抗语义保持变换攻击.出于安全考虑,Java 程序不能检测自己的代码,比如语句 if (instruction #100!="add") exit()在 Java 语法中是不允许的,虽然在其他语言如 C 中是可能的.但是由于这种语句的特殊性,它要检查指令,而不是可执行程序数据段中的数据,所以很容易找到这种语句在程序中的位置.

总之,尽管静态水印比较简单,但是由于它容易遭到破坏、鲁棒性不好,因而不能得到广泛应用.

1.3 Easter Egg水印

这种水印无须检测,它通过一个输入产生一个输出.比如输入一个字符串,然后屏幕上就显示出版权信息或一幅图像.Easter Egg 水印的主要问题是水印在程序中的位置容易找到,一旦输入正确信息,用 softice 这样的标准调试软件就可以跟踪程序执行情况,进而找到水印的位置,所以这种水印不是很安全.

1.4 动态数据结构水印

这种水印的机制是:输入特定信息激发程序把水印信息隐藏在堆、栈或者全局变量域等程序状态中.当所有信息都输完之后,通过检测程序变量的当前值来进行水印提取.可以安排一个提取水印信息的进程或在调试器下运行程序查看变量取值.

与 Easter Egg 水印不同的是,动态数据水印没有输出,而且水印的提取过程不是封装在应用程序中,因而不容易找到水印在程序中的位置,但是这种水印也经不住迷乱变换的攻击.

1.5 动态执行过程水印

当程序在特定的输入下运行时,对程序中指令的执行顺序或内存地址走向进行编码生成水印.水印检测则通过控制地址和操作码顺序的统计特性来进行.

2 软件水印系统的攻击

任何信息隐藏系统的安全强度都与隐藏信息量、隐蔽性和可靠性密切相关,而且要在这 3 个参量之间进行折衷,比如,被隐藏信息量大必然导致隐蔽性和可靠性的降低.为了评价水印算法的好坏,我们必须清楚水印可能遇到哪些攻击.一般来说,没有一种隐藏系统可以抵抗所有类型的攻击.Bender 在文献[6]中指出,目前的所有多媒体水印技术都存在着缺陷,要达到保护大量隐藏信息抵抗所有试图去掉这些信息的目的是不可能的.只要有足够的时间,一个熟练的软件工程师总能够对任何应用软件进行逆向工程.逆向工程师通过反汇编器或反编译器可以反编译应用程序,然后可以分析它的数据结构和控制流图.这可以手工来做,也可以借助一些逆向工具.这个问题其实很早就出现了,只是一直没有得到软件开发商的重视,因为多数程序都非常大,而且被统一封装起来,要想对它进行逆向工程非常困难,几乎是不可能的,但是现在软件正在以一种越来越容易被反编译和逆向工程的方式发行.比如现在的 Java 字节码和中性分布格式体系.Java 应用程序是以一种独立于硬件的类文件形式发行的,这种类文件含有 Java 源代码的全部信息,而且很容易被反编译.由于大部分运算是通过标准库进行的,类文件相对比较小,容易逆向工程.只要应用程序被反编译过来,程序就一览无遗,这样嵌入到程序中的水印信息就很容易被分析出来,或者对程序进行一些语义保持变换,即使水印信息仍然存在也无法被检测出来.下面我们根据文献[7]对语义保持变换进行了分析总结.语义保持变换主要分为两大类:控制流程变换和数据变换,前者主要影响代码水印的检测,后者主要影响数据水印的检测.

2.1 控制流程变换

这种变换可以改变程序的控制流程.

2.1.1 插入支路

假设基本模块 $S=S_1,...,S_n$(如图 1 所示).在图 1(a)中,插入一个不透明判断条件 P^T,把 S 分割成两部分.判断条件 P^T 是不相关代码,因为它始终为 True.在图 1(b)中,先把 S 分割成两部分,然后把第 2 部分复制成两个版本 S^a 和 S^b,这两个版本其实具有相同的功能,只是在表现形式上不同罢了,根据判断条件 $P^?$ 来选择 S^a 或 S^b.图 1(c)对图 1(b)作了一下修改,把条件 $P^?$ 换成条件 P^T.

2.1.2 增加冗余操作数

假设冗余操作数为 P 和 Q.在语句(1′)中,令 $P=2$;语句(2′)中,令 $P=Q>0$,则表达式 1 和 1′、表达式 2 和 2′具有相同的意思.

(1) $X=Y+Z$; (1′) $X=P*(Y+Z)/2$;

(2) $X=Y+1$; (2′) $X=Y+\log_Q^P$.

2.1.3 模块并行化

在多处理器系统中,并行化可以提高系统的运行和处理效率.这里的模块并行化不是为了提高效率,而是为了扰乱程序原来的流程.我们可以创建一个执行不相关任务的虚构进程与原进程并行,或者将原来前后不相关的几个模块并行处理.

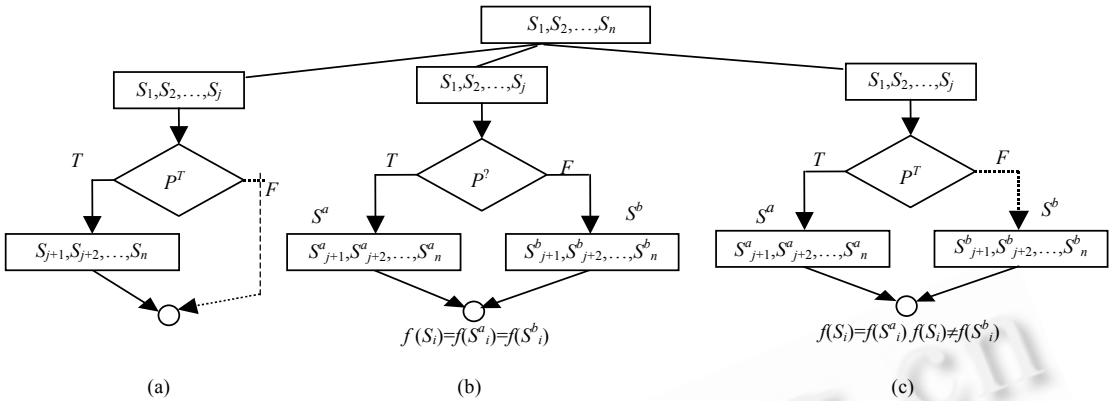


Fig.1 Inserting branch transformation

图 1 支路插入变换

假设 S_1 和 S_2 是两个没有依赖关系的模块(如图 2 所示),(a)是原来程序流程图,(b)是并行化处理后的流程图.

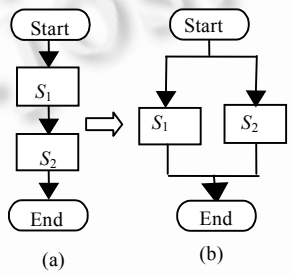


Fig.2 Module parallelized

图 2 模块并行化

2.1.4 简单流程图复杂化

通过附加一些判断条件 P^F 和 Q^F,把原来比较简单的流程图复杂化为 a 和 b,如图 3 所示.

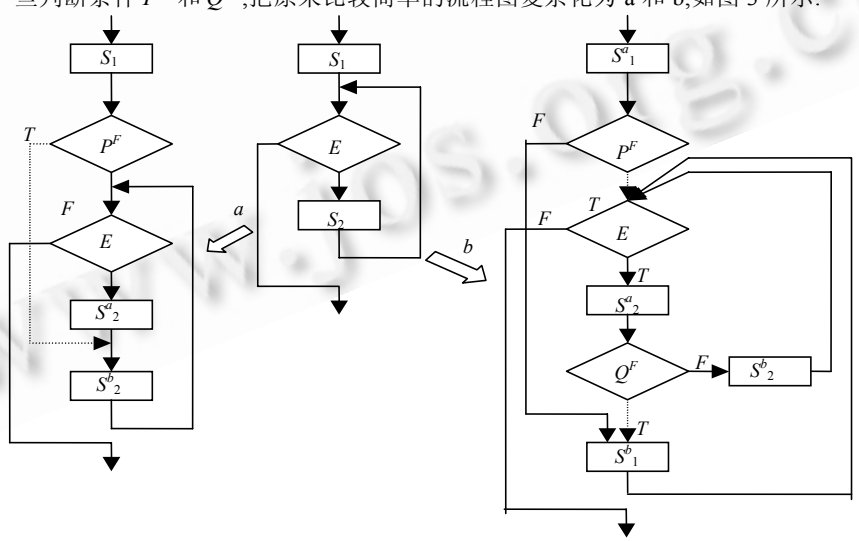


Fig.3 Confusing flow chart

图 3 流程复杂化

2.1.5 循环语句变换

这种变换是改变循环语句的书写方式,将一层循环变为多层循环,将多层循环变为一层循环等.比如:

```
for (int i=0; i<n; i++)                    for (int i=0; i<n; i+=64)
for(int j=0; j<n; j++)          ⇒          for(int j=0; j<n; j+=64)
    a[i][j]=b[j][i];                           for(int m=i; m<min((i+63),n); m++)
                                                   for(int n=j; n<min((j+63),n); n++)
                                                       a[m][n]=b[n][m]
    for (int i=0; i<n; i++)                    for (int i=0; i<n; i++)
{   a[i]=c*b[i];                ⇒                  a[i]=c*b[i];
    d[i]=i*(i−1);                              for ( i=0; i<n; i++)
}                                                  d[i]=i*(i−1);
```

2.1.6 内嵌技术

内嵌(inline)是优化编译的一种方法,它可以改变程序的调用关系.内嵌是将子函数体中的全部内容拷贝到调用该子函数的主函数中,这种优化处理是不可逆的.当程序经过内嵌优化后,在程序中将不会再找到有关该子函数的任何信息.与此相反,还有一种形象地称为轮廓(outline)的技术,它是将主函数中的一部分代码提取出来作为子函数,供主函数调用,这对于优化程序设计很有用.

假设 P, Q 和 R 是子程序, $P_1,P_2,...,P_k$ 和 $Q_1,Q_2,...,Q_m$ 可以是子程序,也可以是一组语句.inline 和 outline 技术如图 4 所示.

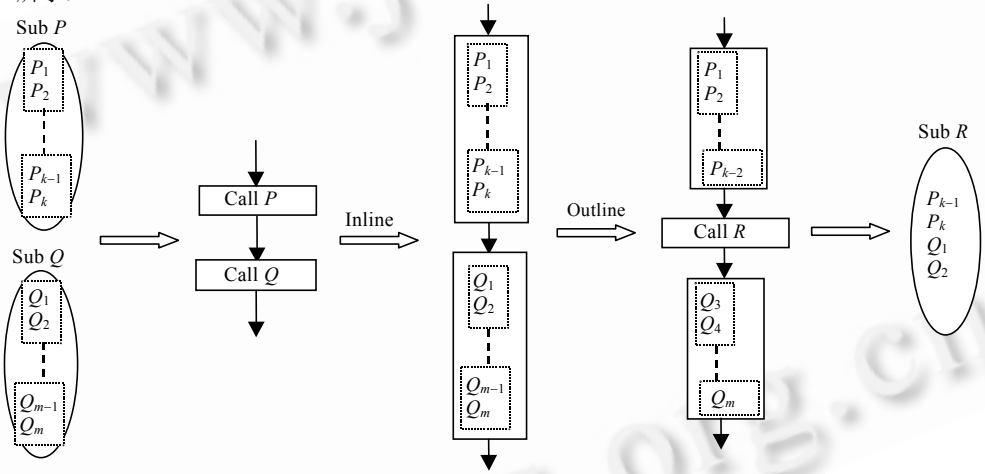


Fig.4 Inline and outline

图 4 内嵌和轮廓

2.2 数据变换

2.2.1 数据编码

比如用 $i'=c_1\cdot i+c_2$ 代替 i,其中 c_1 和 c_2 是常量.假设有下面一段程序,

```
int i=1;                          int j=c1+c2;
while (i<n){                          while(j<n*c1+c2){
…                       ⇒         …
  A[i]=…;                             A[(j−c2)/c1]=…;
  …                                 …
}                                 }
```

显然,这两段程序语义相同,但是变量 i 的表达形式不一样.

2.2.2 改变变量的存储方式和生存周期

在 Java 程序中,一个整型变量可以看做是 Integer 类的一个对象.

```
int i=1;                          Integer i=new Interger(1);
while (i<10){                       while(i.value<10){
…                       ⇒         …
  A[i]=…;                           A[i.value]=…;
  i++                               i.value++
}                                 }
```

在一个函数内部定义的变量是内部变量,它只在本函数范围内有效,也就是说,只有在本函数内部才能使用它,此函数以外是不能使用这些变量的,称其为局部变量.而在函数外定义的变量是全局变量,它可以为本文件中其他函数共用.其生存周期是从定义变量的位置开始到本源文件结束.我们可以把一个局部变量改为全局变量,这不会影响程序的运行.

2.2.3 拆分变量

顾名思义,拆分变量就是把一个变量分成几个变量来处理.在数学中,平面中的一个点可以用两个变量(横坐标和纵坐标)来代表,对点进行的运算都可以用其坐标运算来代替.布尔型变量和其他取值范围有限的变量都可以分解成两个或多个变量.

将一个类型 T 的变量 V 分解成两个类型为 U 的变量 P 和 Q,必须满足以下 3 个条件:

(1) 存在一个函数 f,使得 $f:(p,q)\to v$;

(2) 存在一个函数 g,使得 $g:v\to(p,g)$;

(3) 对 T 类型变量进行的运算能转化为对 U 类型变量进行的运算.

其中 p,q,v 分别是变量 P,Q,V 的取值.下面以布尔型变量为例进行说明.布尔型变量 V 分解成两个整型变量 P 和 Q,$v=f(p,q)=2p+q$,$p=v/2$,$q=v\%2$.当 (p,q) 为 (0,0) 或 (1,1) 时,表示 V=FALSE;当 (p,q) 为 (0,1) 或 (1,0) 时,表示 V=TURE,也可以用 0 或 3 代表 FALSE;用 1 或 2 代表 TURE.这时图 5(b)和图 5(c)分别表示两个布尔变量 A 和 B 的与运算、或运算.图 5(d)是用编程语言来表达 A 和 B 之间的布尔运算.

$g(v)$ p	q	$f(p,q)$ V	$2p+q$
0	0	False	0
1	0	True	1
0	1	True	2
1	1	False	3

(a)

AND(A,B)		A			
		0	1	2	3
B	0	3	0	0	0
	1	3	1	2	3
	2	0	2	1	3
	3	3	0	0	3

(b)

OR(A,B)		A			
		0	1	2	3
B	0	3	1	2	3
	1	1	1	2	2
	2	2	2	1	1
	3	0	1	2	0

(c)

```
(1) BOOL A,B,C;                  (1′) short a1,a2,b1,b2,c1,c2;
(2) A=TRUE;                      (2′) a1=1; a2=0;
(3) B=FALSE;                     (3′) b1=1; b2=1;
(4) C=FALSE;              ⇒      (4′) c1=0; c2=0;
(5) C=A & B;                     (5′) short x=AND(2*a1+a2, 2*b1+b2); c1=x/2; c2=x%2;
(6) C=A | B;                     (6′) short x=OR(2*a1+a2, 2*b1+b2); c1=x/2; c2=x%2;
(7) if (!A)…;                    (7′) short x=2*a1+a2; if ((x= =0) || (x= =3))…;
(8) if (B)…;                     (8′) if (b1∧b2)
```

(d)

Fig.5 Split variable

图 5 拆分变量

与拆分变量相反,还有一种合并变量的变换,这里不再叙述.

上述各种变换都可能影响水印的检测,但它不会影响程序的语义,类似于多媒体水印不影响图像、视频和音频的视觉和听觉效果一样.对于软件水印来说,关键问题是如何准确地将嵌入到应用程序中的水印提取出来.程序经过控制流程变换和数据变换之后,水印数据信息或水印定位信息很可能遭到破坏,导致水印无法检测,或者检测出来的水印面目全非,起不到版权保护的作用.但是,如果从另一个角度来讲,我们也可以利用这些变换设计各种水印.

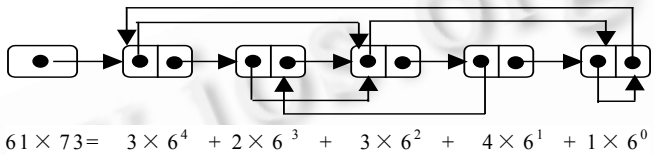
3 现有的软件水印算法

3.1 动态图水印

Collberg 和 Thomborson [8]提出了一种动态图水印技术,即把软件水印隐藏在程序动态建立的图结构拓扑中.由于指针的引用,使得控制动态图结构的那部分代码不易分析,因而很难对图进行保持语义变换,而且证明这种结构是否被篡改比证明代码或纯数据更容易些.这种技术的基本思想是,用图的拓扑结构来代表一个数 N,这个数是两个大素数 P 和 Q 的乘积.在进行水印检测时,先根据图的拓扑结构恢复出 N,由于数论中的大数分解难问题,只有合法用户才能将检测到的 N 分解成大素数 P 和 Q 的乘积,从而证明其合法版权.其中,图的拓扑结构可以由面向对象程序设计中的对象概念作为节点构成.从另一个角度来说,这就是图的编码、枚举问题.

3.1.1 基数 k 编码

图 6 是说明基数 k 编码的循环链表.其中一个指针始终指向下一个节点,另一个指针用来编码从该节点返回原处需要经过的节点数,空指针表示 0,指向自己的指针表示 1,指向下一个节点的指针表示 2,依此类推.



$61\times 73= 3\times 6^4 + 2\times 6^3 + 3\times 6^2 + 4\times 6^1 + 1\times 6^0$

Fig.6 Radix-k coding

图 6 基数 k 编码

3.1.2 枚举编码

下面以树的枚举为例进行说明.假设有树 A 和树 B,定义树大小的比较规则为

(1) 若 A 的深度大于 B,则 $A>B$;若 A 的深度小于 B,则 $A<B$;

(2) 若 A 与 B 深度相等,视 A 与 B 的结点数;

(3) 若 A 的结点数大于 B,则 $A>B$;若 A 的结点数小于 B,则 $A<B$;

(4) 若 A 与 B 的结点数相等,依次讨论 A 与 B 的子树:拥有较大子树的树较大.若当前讨论的子树相等,则讨论 A 与 B 的下一棵子树.

节点数为 6 和深度为 3 的树由大到小排序的情况如图 7 所示.

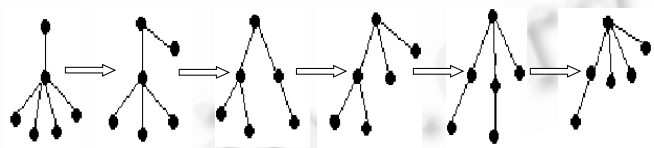


Fig.7 Tree enumerate coding

图 7 树枚举编码

为了防止恶意者通过增加或删除节点破坏图的拓扑结构,使得检测出来的 $N'!=N$,从而无法分解成 P 和 Q,文献[8]提出了一些相应的解决办法以增强水印的抗攻击性.Jens 和 Sowmya 对这种水印算法进行了实验[9].实验表明,嵌入水印后的程序在性能上没有显著下降,代码大小、执行时间和栈空间的使用都和原来相差不大,而且这种水印能够抵抗大多数程序变换攻击,具有良好的鲁棒性.

3.2 虚构条件水印

Akito 和 Hajimu[10]提出一种针对 Java 程序的水印算法.这种算法的基本思想是,在源代码中加入一些冗余代码,这些冗余代码不参与程序的执行.比如有下面一段源代码:

```
for(int k = 2; k <= j; k++)
{
    BigInteger biginteger = BigInteger.valueOf(2*(2*k – 3));
```

```
        biginteger = biginteger.multiply((BigInteger)prod.elementAt(k−1));
        prod.setElementAt(biginteger, k);
    }
```

然后可以在这段代码中的某个位置插入一段代码:

```
    if(opaque condition)
    {
        ….//可以是符合语法规则的任意代码
    }
```

opaque condition 是一个条件语句,但是这个条件的值是确定的.对于上述情况它应该始终为 False,如果插入的代码是

```
    if(!opaque condition)
    {
        …//可以是符合语法规则的任意代码
    }
```

那么这个判断条件的值始终为 True,也就是说不论怎样,if 条件语句括号{}内的语句是不参与执行的.正因为这样,只要符合语法规则地改变括号{}内语句的操作码和操作数,就不会影响原来程序的语义,而这种改变实际上就是嵌入水印的过程,而且这种改变是在字节码文件中进行的.也就是说,在源代码中加入带有不透明条件(opaque condition)的 if 语句后,对程序进行编译生成字节码文件,然后在字节码文件中找到刚才加入的那些代码对应位置,按照规则改变操作码和操作数.因为 Akito 和 Hajimu 提出的这种方法是针对 Java 程序的,而 Java 语言与其他语言(比如 C 语言)的区别在于,Java 经过编译后不能生成可执行文件,而是生成字节码文件,字节码文件必须经过 Java 虚拟机解释后才能执行,在装入字节码文件解释运行之前,Java 有一种字节码校验功能,用于检查这些字节码是否有语法错误,所以如果经过改变后的字节码有语法错误,程序就不能执行了.下面举个简单的例子:

比如括号{}内有一个操作码 iadd(它是“+”运算符对应的字节码),那我们就可以把它变为 isub,imul,idiv 等,因为把加法运算变成减法、乘法、除法(只要分母不为 0)运算都是符合语法规则的,这样,可以对 iadd,isub,imul, idiv 这些操作码进行编码,做成一个码本.

指令	码字
60 iadd	000
64 isub	001
68 imul	010
6c idiv	011
70 irem	100
7e iand	101
80 ior	110
82 ixor	111

如果要嵌入的水印是 010 011 000 111 100 101 000000011…这样一个 0、1 序列,括号{}内遇到的第 1 个操作码是 isub,根据水印序列的前 3 个字符 010 在码本中找到 010 对应的操作码 imul,并代替 isub.接着,如果第 2 个操作码是 ixor,根据水印序列下 3 个字符 011 在码本中找到 011 对应的操作码 idiv,代替 ixor.如此重复,直到遍历所有的水印字符.上面只是简单地改变操作码,还可以改变操作数,情况会稍微复杂一些.检测时,只要遇到 iadd 就译码输出 000,遇到 isub 就输出 001 等,当括号{}内所有的操作码都被译码完毕,输出的 0、1 序列就是水印序列,然后再与原始水印进行相关比较.

用这种方法加入的水印很容易被去掉,这是因为加入的代码不被执行,只要用 softice 这种跟踪调试软件对程序进行多次跟踪,始终没有执行的那些语句很有可能就是加水印的地方,去掉这些语句后,水印就被去掉了.

这种方法只能对 Java 程序进行水印保护,且水印嵌入时需要源代码.这种方法也经不住代码迷乱攻击.为了抵抗代码优化攻击,opaque condition 这个条件语句必须很复杂,但这样又很容易暴露目标,所以此方法的鲁棒性不是很好,而且由于加入的是冗余代码,有可能影响程序的效率.

3.3 矢量提取水印

Julien P.Stern 和 Gael Hachez[11]提出一种基于矢量提取的算法.他们不是把程序代码看做是顺序执行的指令,也没有利用程序的执行流程,而是把它当作统计模型来处理.具体地说,就是把代码中一些单个指令或指令组出现的频率统计出来形成一个矢量 $\vec{c}$,然后根据水印矢量 $\vec{w}$ 改变这些指令或指令组在代码中的出现频率,形成另一个矢量 $\vec{c}' = \vec{c} + \vec{w}$,把水印信息嵌入到代码中.这种算法不需要高级语言程序的源代码,它可以直接对编译后生成的机器指令程序和 Java 字节码程序进行操作,检测时用相关检测方法.这实际上是利用了多媒体水印技术的思想.用这种方法嵌入的水印具有良好的鲁棒性,可以抵抗代码重排,增加冗余代码攻击,因为将不相关的指令颠倒一下顺序以及少量地增加一些冗余代码是不会影响 $\vec{c}'$ 的.如果采用扩频技术,可以进一步提高水印对第 2 种攻击的鲁棒性,这时要想通过改变矢量 $\vec{c}'$ 去除水印信息,必须大量地增加冗余代码,但这样会显著地降低程序的执行效率.这种方法的另一个特点是,检测水印时无须知道水印的确切位置,也不需要源代码,只要知道原矢量 $\vec{c}$ 即可.

4 软件水印的研究现状和发展方向

软件水印是密码学、软件工程、算法设计、图论、程序设计等学科的交叉研究领域.目前主要是新西兰的 Auckland 大学、日本 Osaka 大学在做这方面的研究工作.

由于数字水印技术是近几年来国际学术界才兴起的一个前沿研究领域,处于迅速发展阶段,而软件水印又是其中的一个重要组成部分,因此,掌握其发展方向对软件水印的研究有着重要的指导意义.我们认为,今后软件水印技术的研究和其他多媒体水印技术一样也应该侧重于完善理论,提高水印算法的鲁棒性,建立相关标准.而且软件保护方式的设计应在一开始就作为软件开发的一部分来考虑,列入开发计划和开发成本中,并在保护强度、成本、易用性之间进行折衷考虑,选择一个合适的平衡点.

软件水印应用中安全性是很重要的指标,但是水印的安全性是不能靠保密算法得到的,应该像密码算法一样必须公开,经过公开的研究和攻击来验证其安全性和可靠性.因此软件水印算法必须能抵抗任何攻击,上述几种算法都不能满足这个要求,研究更安全的水印算法仍是软件水印研究的重点之一.

致谢 吕明、朱政荣硕士和北京邮电大学信息安全中心的其他同学和老师对本文提出了很多有益的建议,在此表示感谢.

References:

[1] Kan X. Encryption and Decryption: Software Protection Technique and Complete Resolvent. Beijing: Electronic Engineering Publishing Company, 2001 (in Chinese).

[2] Holmes K. Computer software protection. American Pat 5 287 407. 1994.

[3] Davidson R L, Myhrvold N. Method and system for generating and auditing a signature for a computer program. American Pat 5 559 884. 1996.

[4] Moskowitz SA, Cooperman M. Method for stega-cipher protection of computer code. American Pat 5 745 569. 1996.

[5] Samson PR. Apparatus and method for serializing and validating copies of computer software. American Pat 5 287 408. 1994.

[6] Bender W, Gruhl D, Morimoto N, Lu A. Techniques for data hiding. IBM Systems Journal, 1996,35(3-4):313~336.

[7] Christian C, Thomborsen C, Douglas L. A taxonomy of obfuscation transformations. Technical Report #148, 1997.

[8] Christian C, Thomborsen C. Software watermarking: models and dynamic embeddings. In: Aiken A, *et al*., eds. Proceedings of the 26th Annual SIGPLAN-AIGACT Symposium on Principles of Programming Languages (POPL'99). Association for Computing Machinery Press, 1999. 311~324.

[9] Palsberg J, Krishnaswamy S, Minseok K, Ma D, Shao QY, Zhang Y. Experience with software watermaking. In: Epstein J, *et al*., eds. Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC 2000). New Orleans: IEEE Computer Society Press, 2000. 308~316.

[10] Monden A, Hajimu L, Matsumoto KI, Inoue K, Torii K. Watermarking java programs. In: Lu J, *et al*., eds. Proceedings of the 4th International Symposium on Future Software Technology. Nanjing: Software Engineers Association, 1999. 119~124.

[11] Stern P, Hachez J, Quisquater G, Jean-Jacques. Robust object watermarking: application to code. In: Andreas P, *et al*., eds. Proceedings of the 3rd International Workshop on Information Hiding. Springer-Verlag, 1999.

附中文参考文献:

[1] 看雪.加密与解密——软件保护技术及完全解决方案.北京:电子工业出版社,2001.

IEEE/WIC2003 网上智能与智能代理技术联合国际会议(WI2003/IAT2003)

征 文 通 知

IEEE/WIC2003 网上智能与智能代理技术联合国际会议(WI2003/IAT2003)由 IEEE Computer Society 和国际网上智能协会(Web Intelligence Consortium),国家自然科学基金委员会,中国计算机学会和北京工业大学联合协办,定于 2003 年 10 月 13 日~17 日在北京举行.会议由 IEEE Press 出版正式的论文集,优秀论文将收录于 IOS Press 出版的 Web Intelligence and Agent System: An International Journal 和 World Scientific Publishing 出版的 Annual Review of Intelligent Informatics.

一、征文范围

WI2003 会议征文范围包括(但不限于):

新的 Web 技术;网格计算;智能人网交互;基于智能 web 的商务;知识网络和管理;语义 web;web 智能体;web 信息管理;Web 信息检索;web 挖掘;知识网络与管理;基于智能网络的商务.

IAT2003 会议征文范围包括(但不限于):

分布式智能;可学习和自适应 Agent;数据和知识管理 Agent;面向自组织的计算范例;计算模型、体系结构及其基础;智能 Agent 的应用.

二、征文要求

论文必须未公开发表,并用英文书写.本次会议只接受电子文档(LaTex、MSWord、PostScript、PDF 文件均可).论文格式和投稿要求,详见会议网址.

三、重要日期

征文截至日期: 2003 年 3 月 20 日

录用通知日期: 2003 年 5 月

四、联系方式

联系地址: Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong
(香港浸会大学计算机科学系)

联系人: Dr. Jiming Liu

电话: (852)34117088 传真: (852)34117892

E-mail: jiming@comp.hkbu.edu.hk

会议网址: http://www.comp.hkbu.edu.hk/WI03/ 和 http://www.comp.hkbu.edu.hk/IAT03/