ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.2, February 2007, pp.351−360
DOI: 10.1360/jos180351

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于混沌的软件水印算法框架及实现*

芦 斌+, 罗向阳, 刘粉林

(信息工程大学 信息工程学院,河南 郑州 450002)

A Chaos-Based Framework and Implementation for Software Watermarking Algorithm

LU Bin+, LUO Xiang-Yang, LIU Fen-Lin

(Information Engineering Institute, Information Engineering University, Zhengzhou 450002, China)

+ Corresponding author: Phn: +86-371-63531411, E-mail: stoneclever@gmail.com, http://www.plaieu.cn

Lu B, Luo XY, Liu FL. A chaos-based framework and implementation for software watermarking algorithm. *Journal of Software*, 2007,18(2):351−360. http://www.jos.org.cn/1000-9825/18/351.htm

Abstract: A chaos-based software watermarking framework against several limitations of the existing watermark algorithms is proposed in this paper, in which the anti-reverse engineering technique and chaotic system are combined with the idea of Easter Egg software watermarking. With chaotic system, global protection for the program is provided by dispersing watermark over all the whole code of the program; With the anti-reverse engineering technique, the resistance against reverse engineering is improved. The framework can be implemented under various software and hardware platforms. In this paper, the watermarking framework is implemented under the Intel i386 architecture and the Windows operating system. Then the implementation is taken as an example to analyze the robustness of the watermark framework and the performance degradation of the watermarked program. The results indicate that the watermarking can resist various semantics-preserving transformation attacks and be good tolerance for reverse engineering attacks. The robustness of the algorithm is at a high quality.

Key words: software watermarking; anti-reverse engineering; chaotic system; Easter Egg software watermarking

摘 要: 针对现有软件水印算法中存在的一些不足,将反逆向工程技术和混沌系统与 Easter Egg 软件水印的思想相结合,提出了一个基于混沌的软件水印算法框架.该框架通过引入混沌系统,把水印信息散列编码到整个代码当中,以保护全部代码;通过引入反逆向工程技术来抵抗逆向工程攻击,算法框架与软硬件平台无关.在 i386 体系结构 Windows 平台下实现了该算法框架,并以该实现为例分析了水印的鲁棒性,讨论了水印的嵌入对程序性能的影响.分析表明,该算法可以有效地抵抗各种语义保持变换攻击,对逆向工程攻击具有较好的抵抗性,鲁棒性较高.

关键词: 软件水印;反逆向工程;混沌系统;Easter Egg 软件水印
中图法分类号: TP309 文献标识码: A

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2006AA01Z409 (国家高技术研究发展计划(863)); the He'nan Science Fund for Distinguished Young Scholar of China under Grant No.0412000200 (河南省杰出青年基金); the Science-Technology Project of He'nan Province of China under Grant No.0623021500 (河南省科技攻关项目)

Received 2005-10-26; Accepted 2006-01-24

随着计算机技术的发展,计算机已成为人们工作、学习和生活中不可缺少的部分,而计算机软件正是推动这一发展的重要动力.然而盗版现象日益严重,引起了许多企业和学者的关注[1−3].目前的版权保护主要由软件开发商自己负责,通过加密、序列号、关键文件(key file)、软件狗等方式实施保护[2,4].这些软件保护模式容易遭受破解攻击,难以实现盗版跟踪,而且由于各开发商采用的保护技术千差万别,难以为法律提供统一的依据.此外,软件开发商自己需要分出大量精力来考虑软件版权保护的问题,如果可以像密码体制一样,制定一套可靠的软件保护机制,基于该机制的软件可以得到一定程度的保护,软件开发商则不需要把大量人力物力投入到版权保护上.软件水印技术正是这方面的一种尝试.

软件水印是嵌入到程序中的秘密消息.这些消息应该能够方便、可靠地提取出来,以证明软件的所有权,并且具有在保证程序功能的情况下不能或难以去除该消息的功能.该技术可以提供所有者鉴别、所有权验证、操作跟踪、拷贝控制等服务,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[1,2].

目前,详细而全面地介绍软件水印的论文并不多[1],可以抵抗所有攻击的水印算法几乎没有.文献[5]利用程序基本块的顺序(ordering of basic blocks)编码水印,但只要重排基本块便可破坏水印;文献[6]结合扩频技术,基于指令的频率嵌入水印,从理论上说具有很高的鲁棒性,但数据嵌入率低,实现困难,且可能被代码优化、插入冗余代码等攻击;文献[7]最先提出动态软件水印的概念,并基于指针的混叠效应设计了一种动态图软件水印算法,该算法在堆当中通过图的拓扑结构编码水印信息,比静态水印具有更好的鲁棒性,但如果拓扑指针被修改,水印则不能正确提取;文献[8]把水印信息嵌入到局部变量中,该水印可单独对程序代码中的一个方法(或函数)嵌入水印,甚至可以从程序碎片中提取水印信息,但无法抵抗拆分变量攻击;文献[9]通过代码的动态路径编码水印,并且分别针对 Java 字节码和机器码给出了不同的实现,但构造复杂,实用困难;文献[10]基于多线程程序难调试的特点,通过线程竞争编码水印,但该水印需引入大量线程,将会降低程序的执行效率.概括起来,现有软件水印算法有以下一些不足:(A) 设计水印算法时所假设的水印攻击模型大多局限于自动化攻击(如代码优化、代码迷乱、拆分变量等),而对人工或半人工攻击(如逆向工程攻击)考虑较少;(B) 水印信息只嵌入在程序的某一部分,不能保护软件的所有模块,无法抵抗裁剪攻击;(C) 水印嵌入过程多在源代码中实现,嵌入后需要重新编译,嵌入效率低*;(D) 在水印嵌入过程中,几乎所有工作都由程序开发者完成,水印构造、嵌入复杂,实用性较差.

本文针对以上不足,把反逆向工程技术、混沌系统与传统的 Easter Egg 软件水印的思想相结合,提出一种基于混沌的软件水印算法,简记为 CBSW(chaos-based software watermarking).该算法在保持 Easter Egg 软件水印嵌入提取方便、实用性较强(针对不足(D)),可以有效地抵抗各种语义保持变换攻击等优点的基础上,通过引入混沌系统,把水印信息散列编码到整个软件代码当中,对整个代码实施保护(针对不足(B));通过引入反逆向工程技术,能够较好地抵抗逆向工程攻击(针对不足(A)),增强了水印的鲁棒性.此外,该算法直接在可执行文件中嵌入水印,不需要重新编译(针对不足(C)),提高了嵌入效率.

本文首先提出 CBSW 算法框架(第 1 节、第 2 节):第 1 节阐述 CBSW 的组成结构及各模块之间的相互关系;第 2 节提出 CBSW 的嵌入和提取算法,该算法中混沌替换∂、水印预处理 E 以及混沌散列编码ξ可以根据实际情况具体调整.然后,给出∂,ξ,E 的一种具体算法,并在 i386 体系 Windows 操作系统下实现整个算法框架(第 3 节).最后,以实现中给出的∂,ξ,E 算法为例分析 CBSW 的鲁棒性,并讨论水印的嵌入对程序性能的影响.分析表明,该算法可以有效地抵抗各种语义保持变换攻击,对逆向工程攻击具有较好的抵抗性,鲁棒性较高.

1 CBSW 的组成结构

针对引言中阐述的现有软件水印方案中的一些不足,结合 Easter Egg 软件水印的思想,本文提出了一种基于混沌的软件水印算法.

Easter Egg 水印是一种动态软件水印,是使用最为广泛的软件水印之一[2,7].该水印的基本思想是直接把水

* 软件指纹水印要求每套软件嵌入不同的用户信息,如果在源代码中嵌入水印,则每套软件都必须重新编译,效率问题将成为指纹水印应用的瓶颈.

印检测器(或提取器)加入到软件代码中,通过特定的输入激活水印检测器(或提取器),在检测(或提取)水印后将其通过可视化方式显示给用户.这种可视化行为使水印的检测(或提取)行为成为水印嵌入后程序的正常语义,故可抵抗各种语义保持变换攻击.然而,这种可视化行为也为攻击者提供了突破口,攻击者可以此为线索,使用各种逆向工程技术定位水印,进而将其去除.Easter Egg 水印的另一个缺点是:水印仅嵌入到程序的某一部分,不能保护整个软件代码,没有嵌入水印的那部分代码可能被非法重用.

CBSW 由 4 个基本部分构成:水印信息 W、输入监控模块 C_m、水印解码模块 C_d、反逆向工程模块 C_a.不同于一般的水印算法,该算法不仅把水印信息 W 嵌入到程序中,而且把 C_m,C_d,C_a 以可执行代码形式嵌入到程序中.下面具体阐述 CBSW 水印嵌入代码(C_m,C_d,C_a)的组成及相互关系(如图 1 所示).

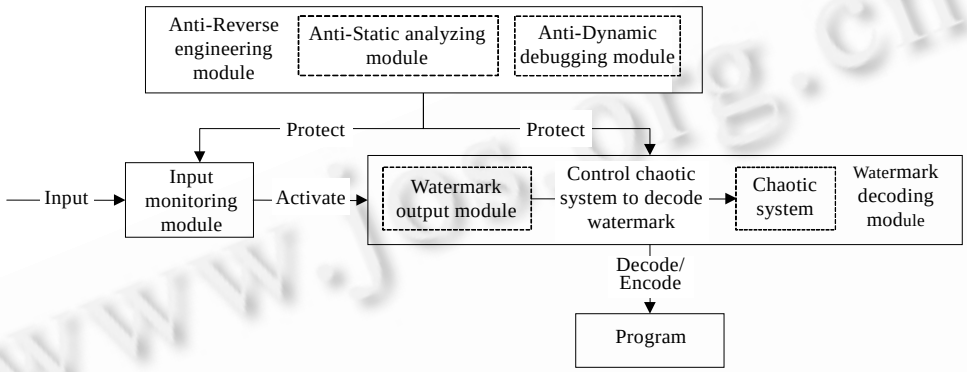

Fig.1 The structure and interrelationship of CBSW embedding code

图 1 CBSW 嵌入代码组成及关系

为了方便描述,给出如下符号定义.设水印嵌入前的程序为 P,P 的可接受输入组成的集合为$\{\alpha_1,\alpha_2,\ldots\}$,$P'=T(P,W,C_m,C_d,C_a)$是嵌入水印后的程序($T$ 为水印的嵌入变换),水印提取过程为 $\tilde{W}=D(\Gamma(P'))$(D 为水印提取变换,Γ是某种代码变换,如果 P'被攻击,则Γ表示该攻击变换;如果 P'未被攻击,则Γ表示恒等变换),如果 $D(\Gamma(P'))\equiv_{cp}W$,则认为该水印算法可以抵抗 Γ 攻击,这里,$\equiv_{cp}$ 是自定义的"相等关系".

输入监控模块 C_m 实现映射 $\psi:\{\alpha_1,\alpha_2,\ldots\}\to\{0,1\}$.当 $\psi(\alpha_i)=1$ 时,启动水印解码模块.$\alpha\in\Sigma(\Sigma=\{\beta|\psi(\beta)=1\},|\Sigma|\neq 0)$称为激活密钥.

为了更清楚地说明水印解码模块,先简单描述水印嵌入的过程(如图 2 所示):首先把 C_m,C_d,C_a 通过混沌替换∂嵌入到程序中,然后对水印信息 W 作预处理($U=E(W)$),再使用混沌散列编码算法ξ把预处理后的水印信息 U 散列到整个代码段(除了被 C_m,C_d,C_a 替换以外的那部分代码).

水印解码模块完成提取水印信息并以可视化行为输出的功能.该模块包括混沌系统模块和水印输出模块.该模块首先使用ξ^{-1} 从程序代码中提取出经过预处理的水印信息(记为 $\tilde{U}$),再对其作逆预处理($\tilde{W}=E^{-1}(\tilde{U})$),水印输出模块把 $\tilde{W}$ 转化成可视化行为 $V_{\tilde{W}}$ 输出.

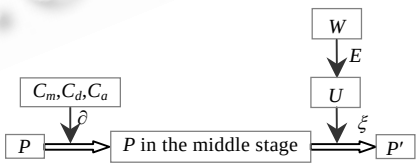

Fig.2 The embedding framework of CBSW

图 2 CBSW 嵌入框架

反逆向工程模块 C_a 为 C_m,C_d 提供抵抗逆向工程攻击的保护.该模块分为反静态分析模块和反动态跟踪模块,分别实现各种反静态分析和反动态跟踪技术.该水印算法抵抗逆向工程攻击的性能依赖于该模块中使用的反逆向工程技术的强弱.

2 CBSW 的嵌入和提取算法

在描述 CBSW 的嵌入和提取算法之前,先介绍其中使用的相关运算:混沌替换∂、水印信息预处理 E 和混沌散列编码ξ.

2.1 水印算法中使用的相关运算

2.1.1 混沌替换∂

混沌替换完成嵌入 C_m,C_d,C_a,并保存被 C_m,C_d,C_a 覆盖后的代码,以备恢复代码时使用.如果直接保存被 C_m,C_d,C_a 覆盖后的代码,容易因攻击者将其私自恢复代码而破坏 C_m,C_d,C_a,所以需要对其作一些特殊处理.一般地,用 c 替换 i,得到 $s=\partial(i,c,G)$,称为混沌替换.其中:c,i 是 8-bit 的整数;G 是数字化混沌系统.替换之后,i 的值等于 c,并保存 s 以备恢复 i.混沌替换的逆过程是用 c 和 s 恢复 i:$i=\partial^{-1}(s,c,G)$.

∂运算可看作是以 c 为密钥,通过 G 加密 i 的过程;∂^{-1} 是以 c 为密钥通过 G 把 s 解密成 i 的过程.如果密钥 c 被篡改,则 i 无法解密,从而防止攻击者对 c 的篡改.一般地,对于集合 $A=\{a_1,a_2,\ldots,a_k\}$使用混沌替换算法替换 $B=\{b_1,b_2,\ldots,b_k\}$,得到:

$$R=\{r_j\}=\partial(B,A,G)=\{\partial(b_j,a_j,G)\},\ j=1,2,\ldots,k \tag{1}$$

其逆过程记为

$$B=\partial^{-1}(R,A,G) \tag{2}$$

注:∂ 实际上也可以使用其他类似过程,这里使用混沌系统,是因为其效率较高.

2.1.2 水印信息预处理 E

水印信息预处理 E 把水印信息 W 编码成 $U(U=E(W))$,其逆过程记为 $W=E^{-1}(U)$.E 把用户期望嵌入的相关信息转化成方便嵌入算法操作,或有利于某种特殊功能(如软件指纹功能)的编码.此外,其他水印预处理过程也可以应用于该过程.

2.1.3 混沌散列编码ξ

混沌散列编码ξ利用混沌序列把水印信息均匀散列在程序的代码段中.设 X 为混沌序列,用该序列把 U 散列编码到代码 I 中,得到$[I',S']=\xi(U,I,X)$.其中:I'为新代码,含有 U 的信息;S'为保存代码,用来恢复原始代码.其逆过程是用 S'和 I'恢复 W 和 I,记为$[I,W]=\xi^{-1}(S',I',X)$.

2.2 CBSW的嵌入算法

现有的大部分软件水印算法的嵌入过程在源代码中进行,该水印算法是在可执行文件中直接嵌入.具体算法如下(图 3 给出了水印嵌入前后可执行文件的变化情况).

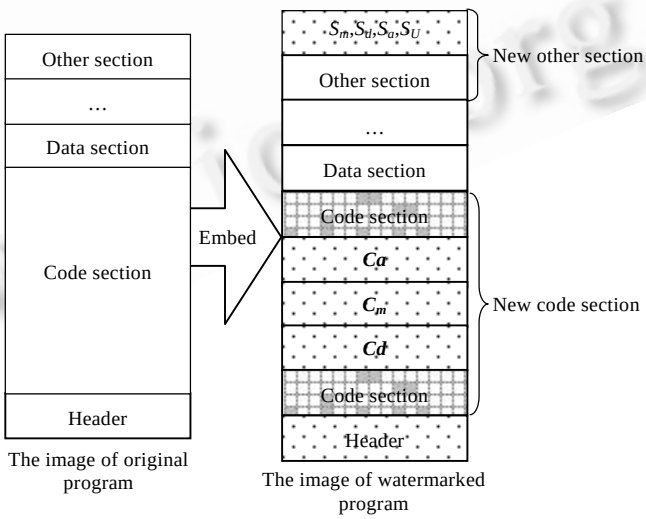

Fig.3 The images of the original program and watermarked program

图 3 水印嵌入前后程序的映像图

1) 给定密钥$\langle K_1,K_2\rangle$(其中:K_1 是激活密钥,提取水印时用于激活水印解码模块;K_2 是产生混沌序列的密钥),

设水印信息 W 长度为 n 字节,记为 $W=\{w_1,w_2,\ldots,w_n\}$.

2) 构造水印解码模块 C_d、反逆向工程模块 C_a,根据 K_1 构造输入监控模块 C_m(具体构造方法在下一节详细讨论).
3) 将 C_m,C_d,C_a 嵌入到程序的代码段,被其覆盖的代码分别记为 I_m,I_d,I_a,经混沌替换得到 $S_m=\partial(I_m,C_m,G)$,$S_d=\partial(I_d,C_d,G)$,$S_a=\partial(I_a,C_a,G)$,其中,G 是数字化混沌系统.
4) 对水印信息 W 作预处理:$U=E(W)$.
5) 根据 K_2 生成混沌序列 X,将 U 用 ξ 散列编码到代码 I 中(I 为整个代码段除去嵌入 C_m,C_d,C_a 的部分,图 3 给出水印嵌入后,水印在可执行文件中的分布情况),得到新代码段 I'和保存代码 $S_U([I',S_U]=\xi(U,I,X))$.

说明:在图 3 中,带小点的部分表示水印嵌入后发生改变的部分;带小方格的部分表示水印嵌入后,水印在代码中的分布情况,其中,灰色表示水印信息,白色表示原来的代码.

把 S_m,S_d,S_a,S_U 保存到可执行文件的尾部,并修改可执行文件头中相应的信息.

2.3 水印的提取

由于该水印算法把水印的提取器嵌入到程序 P 中,因此,水印的提取过程与程序的执行过程混叠在一起.下面,我们通过描述嵌入水印后程序的执行过程来描述水印的提取过程.

1) 程序启动.
2) 反逆向工程模块 C_a 执行.
3) 输入监控模块 C_m 执行,监控用户输入.
4) 生成混沌序列 Y,把散列编码在代码段的水印信息重新组合,并恢复原来的代码 $[I,\tilde{U}]=\xi^{-1}(S_U,I',Y)$.
5) 通过 S_m,S_d,S_a 恢复被 C_m,C_d,C_a 覆盖的代码 $I_m=\partial^{-1}(S_m,C_m,G)$,$I_d=\partial^{-1}(S_d,C_d,G)$,$I_a=\partial^{-1}(S_a,C_a,G)$.
6) 程序继续执行.
7) 当输入与激活密钥 K_1 相匹配时,启动 C_d,并对 $\tilde{U}$ 作逆预处理:$\tilde{W}=E^{-1}(\tilde{U})$.
8) 水印输出模块把水印信息 $\tilde{W}$ 转化成可视化行为 $V_{\tilde{W}}$ 输出.

3 算法实现

本文在 i386 体系 Windows 平台下实现了该算法.本节首先给出实现中使用的混沌系统 G,以及∂,E,ξ的具体算法,然后分别讨论输入监控模块、反逆向工程模块和水印解码模块的具体实现细节,最后针对实现中遇到的问题给出相应的解决方案.

3.1 混沌系统G和∂,E,ξ算法

3.1.1 混沌系统 G

混沌系统在数字化设备上实现时,由于精度总是有限的,不可避免地存在着动力学特性的退化,因此在使用数字化混沌系统时,需要对其做退化补偿.本文使用一维 PWLCM(piecewise linear chaotic map)混沌映射 $G:[a,b]\to[a,b]$,并使用文献[11]提出的退化补偿方案.

3.1.2 混沌替换∂

c,i 是 8-bit 的整数($0\le c,i\le 2^8$),∂可描述如下:

$$s=\partial(i,c,G)=\left\lfloor 2^8\times\frac{G(x,m)-a}{b-a}\right\rfloor\oplus i \tag{3}$$

$$x=\frac{c(b-a)}{2^8}+a\ ,\ m=\left\lfloor\frac{c}{\lambda}\right\rfloor+1 \tag{4}$$

其中:$\oplus$表示二进制异或运算;$G(x,m)$表示混沌 G 以 x 为初值迭代 m 次后 G 的状态;λ为常数,用来调控混沌的迭代次数.其逆过程为

$$i=\partial^{-1}(s,c,G)=\partial(s,c,G) \tag{5}$$

3.1.3 水印信息预处理 E

本文在实现中对水印信息 W 没有作特殊的编码处理,只是将其与混沌序列进行异或运算.把水印信息 W 按 8 比特分割,记为 $W=\{w_1,w_2,\ldots,w_n\}$,将混沌序列变换成 8-bit 整数序列,记为 $Z=\{z_1,z_2,\ldots,z_n\}$,则 E 描述如下:

$$U=\{u_1,u_2,\ldots,u_n\}=E(W)=W\oplus Z=\{w_1\oplus z_1,w_2\oplus z_2,\ldots,w_n\oplus z_n\} \quad (6)$$

其逆过程为 $W=E^{-1}(U)=U\oplus Z=\{u_1\oplus z_1,u_2\oplus z_2,\ldots,u_n\oplus z_n\}$.

3.1.4 混沌散列编码ξ

混沌散列编码ξ利用混沌序列把水印信息均匀散列在程序的代码段中.设 $X=\{x_1,x_2,\ldots,x_n\}$为混沌系统 G 的状态序列,则 $x_j\in[a,b]$, $j=1,2,...,n$.用该序列把 U 散列编码到代码 I 中,得到$[I',S_U]=\xi(U,I,X)$(设 U 的长度为 n 字节, I 的长度为 l 字节,其中,$U=\{u_1,u_2,\ldots,u_n\}$,$I=\{i_1,i_2,\ldots,i_l\}$, $I'=\{i'_1,i'_2,...,i'_l\}$, $S'=\{s'_1,s'_2,...,s'_n\}$).

ξ的具体步骤如下:

1) 初始化:$L\leftarrow l,N\leftarrow n,m\leftarrow\lfloor L/N\rfloor,j\leftarrow 1,d\leftarrow 0$,置 $I'=I$.

2) $r=\left\lfloor m\times\dfrac{x_j-a}{b-a}\right\rfloor+1$, $d=d+r$, $s'_j=\partial(i_d,u_j,G)$, $i'_d=u_j$.

3) 如果 $j=n$,则算法结束;否则,$j\leftarrow j+1$,跳转到 4).

4) $L=L-r,N=N-1$.

5) $m=\lfloor L/N\rfloor$,跳转到 2).

6) 算法结束后, $S'=\{s'_1,s'_2,...,s'_n\}$, $I'=\{i'_1,i'_2,...,i'_l\}$.

其逆过程$[I,U]=\xi^{-1}(S',I',X)$的步骤如下:

1) 初始化:$L\leftarrow l,N\leftarrow n,m\leftarrow\lfloor L/N\rfloor$, $j\leftarrow 1,d\leftarrow 0$,置 $I=I'$.

2) $r=\left\lfloor m\times\dfrac{x_j-a}{b-a}\right\rfloor+1$, $d=d+r$, $i_d=\partial^{-1}(s'_j,i'_d,G)$, $u_j=i'_d$.

3) 如果 $j=n$,则算法结束;否则,$j\leftarrow j+1$,跳转到 4).

4) $L=L-r,N=N-1$.

5) $m=\lfloor L/N\rfloor$,跳转到 2).

算法结束后,$U=\{u_1,u_2,\ldots,u_n\}$,$I=\{i_1,i_2,\ldots,i_l\}$.

3.2 用户监控模块C_m

用户监控模块的功能是监控用户输入.在构造该模块时,把激活密钥 K_1(或$\mu(K_1)$,μ是单向函数)保存到该模块中,当输入α与 K_1(或$\mu(K_1)$)相匹配时,启动水印解码模块 C_d,其实现代码如图 4 所示.本文在算法实现中通过监控键盘扫描码实现该模块.

```
If (α is matched with K1)
    Activate Cd
End if
```

Fig.4 The pseudocode of C_m

图 4 C_m 的伪代码

3.3 反逆向工程模块C_a

从理论上讲,只要有足够的时间和精力,通过逆向工程可以彻底分析任何软件的运行机制,所以,完全抵抗逆向工程攻击是不可能的.该模块设计的原则是延长逆向工程所需的时间,使其在时间上不可行,或使攻击者在逆向工程上耗费的代价大于通过逆向工程所得到的利益.逆向工程技术可以分为静态分析和动态跟踪两类,因此该模块分为反静态分析模块和反动态跟踪模块.

静态分析的基础是反汇编,因此可以从扰乱反汇编入手来抵抗静态分析.一般而言,程序的代码和数据是分离的,反汇编工具就是基于此设计的.基于冯·诺依曼体系的计算机不区分数据和指令,所以,可以采用指令和数

据混合的技术扰乱反汇编.在指令之间加入一些特定数据(花指令),就会扰乱正常反汇编,从而达到抵抗静态分析的目的,文献[4]收集了一些花指令.图 5(a)是汇编语言编写的源代码,第 1,5,6 行是正常的汇编代码,第 2,3,4 行是添加的花指令.图 5(b)是反汇编后的代码,可以看到:反汇编的代码从第 4 行开始出错,未能正常反汇编其后的代码.

```
1          xor   eax,eax
2          jnz   Next
3          jz    Next
4          db    0e8h
5    Next: xor   ebx, 1
6          add   ebx, 2
```

(a)

```
.text:004010D7    xor eax, eax
.text:004010D9    jnz short near ptr loc_4010DD+1
.text:004010DB    jz short near ptr loc_4010DD+1
.text:004010DD    call near ptr 83420465h
.text:004010E2    retn
```

(b)

Fig.5 An example of disturbing data

图 5 花指令示例

本文在实现中把花指令插入到输入监控模块、反逆向工程模块和水印解码模块中.进一步完善的实现还可以把代码加密、压缩等反静态分析技术运用到该模块中,这将可以进一步增强其抗静态分析的能力.

动态跟踪高度依赖于调试工具.模仿杀毒软件杀毒的原理,寻找各调试工具的特征,基于此特征,检测程序是否正在被调试,若正在被调试,则把程序引导到错误的控制流上,阻止调试工具对程序的动态跟踪.本文在算法实现时,基于 SoftICE,Windbg,Ollydbg 的特征实现了反跟踪,实验表明,可以有效地抵抗利用这些调试工具的跟踪.进一步完善的算法实现,可以基于更多的调试器特征实现反动态跟踪.

i386 体系的处理器集成了硬件寄存器,专门提供对调试的支持,一些优秀的调试工具利用这些调试寄存器设计了逆向工程中实用的功能,如 BPM 功能**、硬件断点***等[4].本文在算法实现中通过修改调试寄存器而使调试器的 BPM 功能、硬件断点功能失效.此外,本文在算法实现中还加入了时间敏感代码(一般而言,程序被调试时执行的时间远远大于程序正常执行的时间,据此可以设计时间敏感代码,实现反动态跟踪)、断点检测(检测代码中是否被设置断点,如果设置了断点,则可以修改断点设置位置,使断点设置失败)的反动态跟踪技术.

本文实现了一些反逆向工程技术,值得注意的是,该模块是开放模块,可以添加更多、更有效的反逆向工程技术,从而不断提高水印对逆向工程攻击的抵抗性.

3.4 水印解码模块C_d

水印解码模块包括水印输出模块和混沌系统模块.混沌系统模块实现数字化混沌 G,该混沌系统用于水印提取时的∂^{-1},E^{-1},ξ^{-1} 运算.水印输出模块把提取出的水印信息转化成可视化行为输出.在实现中,本文把水印信息通过对话框形式显示给用户.

3.5 遇到的问题及解决方案

由于该算法在可执行代码中直接嵌入水印,所以,C_m,C_d,C_a 在实现时遇到两个问题:(1) 当各模块被嵌入到不同的可执行文件中以后,各模块代码及数据加载到内存的位置不同,代码在执行过程中无法找到代码及数据的正确位置,必须进行自定位;(2) 嵌入水印后,代码不重新编译,故无法通过编译器和加载器自动找到各模块中使用的 Windows API 的地址,需要各模块自己获取 Windows API 的地址.

```
          call  @F
@@:  pop  ebx
          sub   ebx, offset @B
```

Fig.6 The code of self-location

图 6 自定位代码

代码及数据的自定位可通过 call/pop/sub 指令实现,具体代码(Win32 汇编语言编写)如图 6 所示,ebx 是实际加载地址与设计预期地址的偏差.数据或代码的设计预期地址与 ebx 之和就是其实际地址,即可实现自定位.为了不使自定位代码 call/pop/sub 指令成为特征让攻击者利用而降低水印的安全性,可以使用等价指令替换各条指令,也可以在各条

** SoftICE 的 BPM 技术可以锁定内存单元,当程序读/写该单元时产生中断.

*** 一般的调试软件通过在代码中插入 Int 3 指令来产生中断;硬件中断是使用调试寄存器产生中断.

指令之间添加冗余指令并作相应修改,从而掩盖 call/pop/sub 指令形成的特征,保证自定位代码的隐蔽性.

Windows API 的地址可以通过如下方法获取:

1) 获得 kernel32.dll 的加载基址.Windows 系统提供了线程之间独立的异常处理方法——结构化异常处理.系统把所有异常处理函数挂在一个链表上,链表的最后一个元素是系统默认异常处理函数,该函数在 kernel32.dll 模块中.通过遍历异常处理链表,得到系统默认异常处理函数的地址,从该地址的页边界对齐地址开始,以页粒度向低地址搜寻,便可找到 kernel32.dll 的加载基址[****].
2) 通过 kernel32.dll 的导出表获得 Windows API 函数 GetProcAddress,LoadLibrary 的地址.
3) 通过 GetProcAddress,LoadLibrary 得到任意 DLL 模块导出的任意函数的地址.

4 算法分析

本节以算法实现中给出的∂,E,ξ的具体算法为例,讨论 CBSW 的鲁棒性及水印的嵌入对程序性能的影响.设程序代码的长度为 l 字节;水印信息为 W;C_m,C_a,C_d 的长度分别为 n,l_m,l_a,l_d 字节.

首先分析水印的鲁棒性.设 $R[P]$是 P 的语义,称$\omega\in\Gamma_b=\{\varphi|R[\varphi(P)]=R[P]\}$为语义保持变换,变换前后,程序的语义保持不变.对于该水印算法,由于可视化输出 $V_{\tilde{W}}$ 的存在,其语义 $R[V_{\tilde{W}}]\subseteq R[P']$,再由语义保持变换的定义得到:

$$R[V_{\tilde{W}}]\subseteq R[\omega(P')] \tag{7}$$

式(7)说明,语义保持变换不会破坏 $V_{\tilde{W}}$ 的语义,故该水印可抵抗保持语义变换攻击(如去除死代码、代码优化、代码迷乱、变量拆分、代码块重排序等).

对于逆向工程攻击,该水印算法在反逆向工程模块中实现了反静态分析和反动态跟踪技术抵抗逆向工程攻击.其抵抗逆向工程的性能依赖于所采用的反逆向工程技术的强弱.反逆向工程模块是一个动态的、开放式模块,可以把更新更有效的反逆向工程技术引入到该模块中,从而可以使该算法不断升级,不断提高水印抵抗逆向工程攻击的能力.此外,混沌散列编码(ξ)把水印信息编码到代码中,客观上实现了指令和数据的混合,干扰了反汇编,提高了水印抵抗静态分析的能力.

该算法使用混沌散列编码算法嵌入水印信息,嵌入位置散列在整个代码段中.如果想单独重用软件中的某段代码,其中包含的水印信息将导致这段代码无法正常运行.由于 U 均匀散列在代码(去除被 C_m,C_d,C_a 覆盖的那部分代码)中,则平均每 $\frac{l-l_m-l_d-l_a}{n}$ 字节中包含 1 字节水印信息,所以可被重用代码的平均长度 l_v 满足

$$l_v=\frac{l-l_m-l_d-l_a}{n} \tag{8}$$

如果期望平均可重用代码的长度小于 l_T,则要求水印长度 n 满足 $n\geq\frac{l-l_m-l_d-l_a}{l_T}$.

由于 U 通过混沌散列编码(ξ)嵌入,其位置由混沌序列确定,因此定位水印信息的位置比较困难.此外,混沌替换过程 $s=\partial(i,c,G)$可以看作是以 c 为密钥使用混沌系统 G 对 i 加密的过程,如果攻击者试图篡改 c,则当恢复 $i=\partial^{-1}(s,c,G)$时,i 无法正确解码.对于该水印算法,如果篡改预处理后的水印信息 U,则被其覆盖的代码无法正确恢复,导致程序无法正确运行;如果期望通过篡改水印解码器使水印不能正确解码,则被 C_m,C_a,C_d 覆盖的代码无法正确恢复,也将导致程序无法正确运行.对于确定的混沌系统,把 c 看成密钥,则密钥空间为 2^{l_c} (l_c 为 c 的长度,单位为 bit),在该算法中,密钥空间为 $2^{8(n+l_m+l_d+l_a)}$.

下面讨论水印的嵌入对程序性能的影响.从空间上看,水印的嵌入会导致程序代码的增大,由于嵌入过程基于混沌替换(∂),所以该水印的嵌入会使程序增大 $n+l_m+l_d+l_a$ 字节.从时间上看,嵌入水印的程序运行时间将增加,这是由于程序运行需要先通过 S_m,S_d,S_a,S_U 恢复原来的代码,该过程所消耗的时间不仅与使用的数字化混沌系统的迭代效率有关,而且与 U,C_m,C_a,C_d 的具体内容有关.设所采用的数字化混沌迭代一次的时间为 t,则通过

**** 在 Windows 系统中,模块的加载基址是页对齐的,且前两个内存单元是 77,90(字母“MZ”的 ASCII 码值).

S_m,S_d,S_a 恢复代码所需的时间 T_1 满足

$$(l_m+l_d+l_a)t \le T_1 \le \frac{1}{\lambda}(l_m+l_d+l_a)t\times 2^8 \tag{9}$$

通过 S_U 恢复代码要生成长度为 n 的混沌序列用于 ξ^{-1},该过程中包含了 $\mathcal{O}^{-1}$ 运算,所以其时间 T_2 满足

$$nt+nt \le T_2 \le nt+\frac{2^8}{\lambda}nt \tag{10}$$

则恢复全部代码所需的时间 $T=T_1+T_2$,满足

$$2nt+(l_m+l_d+l_a)t \le T \le nt+\frac{2^8}{\lambda}(n+l_m+l_d+l_a)t \tag{11}$$

如果 U,C_m,C_a,C_d 的内容中 0/1 比例相同(如果近似相等,则得到近似的关系),则该过程的平均时间为

$$\overline{T}=nt+\frac{2^7+0.5}{\lambda}(n+l_m+l_d+l_a)t \tag{12}$$

一般而言,对于某个确定的实现,C_m,C_a,C_d 是确定的,即 l_m+l_d+l_a 为常数.对于选定的混沌系统,t 也为常数,则式(12)可写成

$$\overline{T}=nt\left(1+\frac{2^7+0.5}{\lambda}\right)+\frac{2^7+0.5}{\lambda}(l_m+l_d+l_a)=\sigma_1 n+\sigma_2 \tag{13}$$

其中,σ_1,σ_2 为常数.

由式(8)得出:n 的值越大,可重用代码的平均长度越小,保护强度越大.由式(13)得出:$\overline{T}$ 随着 n 的增大呈线性增加.所以,水印长度 n 应在保护强度和程序的性能退化之间取一个折衷.

5 结 论

针对现有软件水印算法的一些不足,本文把反逆向工程技术、混沌系统与 Easter Egg 软件水印的思想相结合,提出了一种基于混沌的软件水印算法框架.该算法在保持传统 Easter Egg 软件水印嵌入方便、实用性较强,可以有效抵抗去除死代码、代码迷乱、变量拆分等语义保持变换攻击等优点的基础上,通过把水印散列编码到整个代码中,保护了整个代码;水印信息的嵌入位置由混沌序列确定,攻击者定位水印困难.即便攻击者找到水印信息的位置,混沌替换算法亦使其难以在保持程序语义的情况下篡改(或删除)水印信息.同时,该算法的反逆向工程模块是开放的、可扩展的,能够把更新、更有效的反逆向工程技术加入到算法中,所以,该水印算法可不断升级,进而不断增强水印的鲁棒性.此外,只要混沌序列的初值不同,水印的嵌入位置就不同.这使得攻击者难以通过比较多套拷贝来定位水印信息.而且,算法直接在可执行文件中嵌入水印,不需要对源代码重新编译,嵌入效率较高.因此,该算法框架对于软件指纹水印也具有一定的意义.

References:

[1] Zhang LH, Yang YX, Niu XX, Niu SZ. A survey on software watermarking. Journal of Software, 2003,14(2):268−277 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/268.htm

[2] Collberg C, Thomborson C. Watermarking, tamper-proofing, and obfuscation—Tools for software protection. IEEE Trans. on Software Engineering, 2002,28(8):735−746.

[3] Business Software Alliance. 8th Annual BSA Global Software Piracy Study: Trends in Software Piracy 1994−2002. 2003. http://global.bsa.org/globalstudy/2003_GSPS.pdf

[4] Duan G. Encryption and Decryption. 2nd ed., Beijing: Publishing House of Electronics Industry, 2003 (in Chinese).

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

[6] Stern JP, Hachez G, Koeune F, Quisquater JJ. Robust object watermarking: Application to code. In: Andreas P, ed. Proc. of the 3rd Int'l Workshop on Information Hiding. Berlin: Springer-Verlag, 1999. 368−378.

[7] Christian C, Thomborsen C. Software watermarking: Models and dynamic embeddings. In: Aiken A, *et al.*, eds. Proc. of the 26th

Annual SIGPLAN-AIGACT Symp. on Principles of Programming Languages (POPL'99). ACM Press, 1999. 311–324.

[8] Cousot P, Cousot R. An abstract interpretation-based framework for software watermarking. In: Jones ND, Leroy X, eds. Proc. of the 31st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. Venice: ACM Press, 2004. 173–185.

[9] Collberg C, Carter E, Debray S, Huntwork H, Kececioglu J, Linn C, Stepp M. Dynamic path-based software watermarking. In: Pugh W, Chambers C, eds. Proc. of the ACM SIGPLAN 2004 Conf. on Programming Language Design and Implementation 2004, Vol 39. Washington: ACM, 2004. 107–118.

[10] Nagra J, Thomborson C. Threading software watermarks. In: Fridrich J, ed. Proc. of the 6th Workshop on Information Hiding. Berlin, Heidelberg: Springer-Verlag, 2004. 208–223.

[11] Liu B, Zhang YQ, Liu FL. A new scheme on perturbing digital chaotic systems. Computer Science, 2005,32(4):71–74 (in Chinese with English abstract).

附中文参考文献:

[1] 张立和,杨义先,钮心忻,牛少彰.软件水印综述.软件学报,2003,14(2):268–277. http://www.jos.org.cn/1000-9825/14/268.htm

[4] 段钢.加密与解密.第 2 版,北京:电子工业出版社,2003.

[11] 刘镔,张永强,刘粉林.一种新的数字化混沌扰动方案.计算机科学,2005,32(4):71–74.

芦斌(1982－),男,山西灵石人,硕士生,主要研究领域为数字水印,软件工程.

刘粉林(1964－),男,博士,教授,博士生导师,主要研究领域为网络信息安全,信息隐藏技术.

罗向阳(1978－),男,博士生,讲师,主要研究领域为信息隐藏技术.