# 基于加密和即时解密的软件保护

刘志成 周伯生

(北京航空航天大学 计算机学院, 北京 100083)

**摘 要**: 通过加密来保护软件,是防止盗版者通过逆向工程理解程序的一种有效方式. 传统加密保护方式,运行时解密开销大,又由于采用一次性解密方式,易受到攻击而直接从存储器中获得解密后的程序映像. 提出了即时解密技术,摒弃了传统的一步解密的方式,在运行时通过即时解码器以函数为单位一步步揭开隐藏在软件中的秘密. 仅对一次运行中必须的部分解密,使得攻击者难以获得完整的解密后程序,难以篡改并绕过未解密部分中的软件保护功能,增强了抗攻击性. 在运行效果上,由于解密分散进行,就解决了启动延时长的问题. 用基于一次一密加密技术,与对称密钥加密算法相比,进一步减少了运行时开销,使得该技术在目前处理器上可以流畅运行. 该方法增强了基于加密的软件保护的实用性,增大了破解的难度.

**关 键 词**: 软件工程; 数据安全; 保护

**中图分类号**: TP 309.7

**文献标识码**: A **文 章 编 号**: 1001-5965(2006)06-0721-04

## Software encryption and just-in-time decryption for software protection

Liu Zhicheng Zhou Bosheng

(School of Computer Science and Technology, Beijing University of Aeronautics and Astronautics, Beijing 100083, China)

**Abstract**: Software protection via encryption is an effective way to prevent software pirate from understanding the program by means of reverse engineering. Traditional software protection method via encryption has obvious runtime overhead and start up delay because of its one-time decryption prior to execution. Moreover, fully decrypted code can easily be obtained from memory. A novel approach, called just-in-time(JIT) decryption, was proposed. JIT decryption differs from traditional decryption method in that it decrypts only one function in a time and reveal the secret of the software step by step. The runtime JIT decipherator only decrypts called functions in a certain run of the program. The uncalled functions remain a secret for an attacker, where self-checking code can be hidden. So it is hard for an attacker to understand the entire program and thus hard to remove all the protecting code. Because decryption is scattered, the start up delay no longer exists. One-time pad based cipher further reduce the runtime overhead as compared with symmetric algorithms. JIT decryption makes encryption based software protection more applicable in practice and harder to attack.

**Key words**: software engineering; security of data; protection

软件盗版是指非法拷贝并转售软件. 盗版给整个软件产业带来巨大的损失. 盗版者最基本的技术手段就是通过逆向工程来理解程序,进而对软件进行非法的篡改,用来去掉软件中的合法性自检程序片断等软件保护手段. 为了增加盗版者进行逆向工程的难度,提出了混淆技术(obsfucation),抗反汇编技术(anti-disassembling),反调试技术(anti-debugging). 然而,没有任何一种技术在理论上是不可攻破的. 软件保护的意义在于增加盗版者的法律风险或者增加工程上非法复制软件的技术难度. 简单的抗反汇编技术,通过在目标文件中增加无用的随机字节来迷惑反汇编器,相对容易破解. 使用加密技术来保护软件,对程序进行加密,运行前自动解密. 这样由于可执行程序的磁

**收稿日期**: 2004-04-15

**作者简介**: 刘志成(1978-),男,河北景县人,博士生, lzc@buaa.edu.cn.

盘映象中保存的是加密后的程序,反汇编器由于无法获得程序的明文就无法进行反汇编操作. 然而使用加密技术进行软件保护,面临着2个必须解决的难题[1]:①加密后的软件要在运行时自行解密,带来运行时开销,对大型的可执行程序更为严重,有较长的启动延时;②程序最终要转化成处理器能够理解的二进制代码流,一方面要对处理器暴露这些信息,另一方面对攻击者隐藏这些信息,这是一个两难问题. 通常攻击者对被加密软件的攻击并不是针对密码算法的攻击,而是通过逆向工程手段理解软件的自引导程序片断,找到密钥进行解密,或者,直接利用自引导程序解密之后形成的存储器中的映像.

提出的基于退化的一次一密乱码本的软件加密技术和即时解密技术,在很大程度上解决了上述2个问题,增强了加密保护的实用性,增大了破解的难度. 本文的其余部分对这种新颖的软件保护方法进行介绍.

## 1 退化的一次一密乱码本加密

一次一密乱码本(one-time pad)使用一个大的不重复的真随机密钥字母集,这个密钥字母集被称为乱码本,最初用来为电传打字机的明文字母加密. 文献[2]介绍了使用一次一密乱码本对二进制数据加密的推广. 由二进制数字组成的一次密乱码本代替由字母组成的一次一密乱码本,用异或代替明文字母加法即可. 解密时只需对照一次一密乱码本再次异或即可. 在使用加密方法对软件进行保护时,解密是在软件运行时自动完成的,所以解密时间要占用软件的运行时间. 用一次一密乱码本,解密的操作仅仅是一个简单的异或操作,与标准密码算法相比,解密时间有大幅提高.

标准的一次一密乱码本加密,被应用到高度机密的应用当中,从理论上说如果无法得到一次一密乱码本,就无法破解. 一次一密乱码本加密用来保护软件,并不追求这种加密强度,因为如果无法解密的话,也同样无法得到可执行的程序. 乱码本必须以某种形式存在于软件当中,以便用于自身的解密,同时还有防止攻击者找到密钥解密软件. 这个两难问题,似乎永远无法解决. 然而,寻求的软件保护方法并不是追求其在理论上不可攻破,而是追求在实际的破解过程中尽可能地增加破解的难度和工作量,使其在工程上成为不可能.

退化的一次一密乱码本,是一个足够大的二进制密钥池 $K$,设其大小为 $N$ 个字节,每个字节的编址为0,1,…,$N$-1. 这个二进制密钥池中的密钥为 $N$ 个介于[0,255]之间的随机数. 对软件的加密保护,以程序中的函数为单位,对每个函数体分别加密,并且在密钥池中随机选取不同的密钥. 密钥的选取过程如下,生成一个[0,$N$-1]之间的随机数 $x$,以 $K[x]$ 作为密钥的第一个字节,根据被加密方法体的二进制编码长度,选取$K[x]$的后续字节,如果到达密钥池的结尾,则从密钥池的开始位置继续选取,直到密钥长度等于被加密方法体的二进制编码长度. 设程序中共有方法 $F$ 个,即使攻击者清楚加密保护算法的原理,但是如果无法准确获得每个方法的密钥,而尝试破解软件需要进行 $F\times N$ 次试验,而且每次试验结果是否正确很难判断,从而无法自动完成. 由于 $N$ 是足够大的数,并且在一个实际的软件中 $F$ 一般也很大,这就使破解加密的尝试在实际的工程中不可行. 对于软件的保护并不依赖于算法的保密性是本算法的一个良好性质.

软件加密过程如下,从软件的入口函数开始,以函数为单位递归进行:①为当前处理函数随机选取加密密钥,在全局数据结构"已加密函数表"中标记该函数已被加密处理,并记录加密密钥. ②在函数体之前为函数添加一个跳板(跳板的作用和实现在第2节中介绍). ③扫描当前处理函数的每一条指令,对于函数调用做特殊处理,为其增加传递隐式参数的指令. 这个隐式参数为被调用函数的加密密钥. 如果被调用函数已经在此之前被加密保护,则加密密钥可以从全局数据结构"已加密函数表"中获得. 对于被调用函数还没有被加密的情况,递归调用本算法首先处理被调用函数. ④对函数体的指令逐条扫描结束后,使用①中产生的密钥对函数体使用乱码本加密.

退化的一次一密乱码本就以某种形式存在于加密后的程序当中. 可以以明文的形式,也可以以使用某种加密算法加密后的密文形式存在. 不管怎样,使用纯软件的方法无法绝对的保守乱码本的秘密,即使以密文形式保存也可以通过逆向工程获得加密乱码本的密钥从而解密获得其明文. 本算法的健壮性不依赖于保守乱码本的秘密,由于每个方法采用不同密钥加密,即使获得乱码本也难以轻易破解. 这与传统的使用一个密钥加密软件,并且依靠保守密钥的秘密来防范破解的方法是本质不同的.

对于乱码本嵌入程序中一个有效的改进是,利用硬件锁中的存储功能来加密保存乱码本(关

于本算法与硬件锁保护的正交性,在第 3 节中论述). 这种做法继承了传统保护方法的思路,对乱码本也进行了保护,可进一步增加破解难度. 但与纯软件的保护方法相比增加了保护成本. 作为一种候选的策略可根据实际工程的需要进行取舍.

## 2　即时解密技术

传统的基于加密的软件保护方法,在软件自引导的时候,一次性对软件进行解密. 这种方法的一个显著弱点是解密后的软件以明文的形式存在于主存当中,很容易让攻击者通过调试器等工具直接从主存获得解密后的软件.

采用的解密技术被称为即时解密技术(just-in-time decryption). 每个函数当第 1 次被调用的时候才进行解密操作,解密结束之后交处理器执行. 本文借鉴了在 Java 虚拟机和程序设计语言领域的即时编译技术 JIT(Just-In-Time Compilation)[3-4]. 每个加密后的函数体,前面都添加了一个跳板,调用即时解码器. 即时解码器负责在运行时解密每个函数,解密后的函数体仍在主存中原来的位置,只不过前面的跳板已经被打上补丁去掉,当第 2 次执行的时候直接执行解密后的函数体,无须再次解密,如图 1 所示.

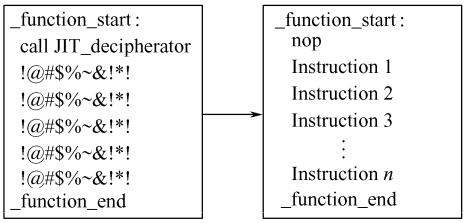



图 1　函数体在即时解密前后的变化

用于解密的具体密钥(密钥在乱码本中的起始位置)并不以明文的形式在代码中存在,而是由被解密函数的调用者在调用时作为一个隐式参数来提供. 所以只有解密了一个函数的某一调用者(caller)才可以进一步解密被调用的函数(callee).

即时解密技术一步步揭开隐藏在代码中的秘密. 在软件的执行过程中,没有使用到的软件片断的逻辑不会被揭秘,这就避免了攻击者一次性通过调试器等工具获得解密后应用程序在主存中的映像. 现代软件的功能是强大而复杂的,一个软件系统可以完成若干项主要功能,软件的一次运行仅使用其中一部分功能. 并且软件中有很大一部分的逻辑是用来处理各种异常和例外情况的,在通常情况下这些代码并不运行. 还可以在软件中人为地设置各种时间触发的事件. 例如,程序启动 1 h 后触发一个事件;或者,软件安装 10 d 后触发一个事件,等等. 这就保证了一个攻击者,即使多次运行软件也很难揭开软件中所有的秘密. 未被解密的代码中,就可以隐藏用于软件合法性自检和自毁的程序片断. 这些程序片断如果不被盗版者成功的去除,盗版的非法拷贝的使用价值就会大大降低.

现代软件是庞大的,即时解密技术仅仅对一次运行中需要的代码进行解密,对没有运行到的代码并不解密,这又进一步减少了解密所需的时间. 与传统的一次性解密方法相比较,解密的操作分为多次进行,也有效地减小了软件启动的时间. 由于采用退化的一次一密乱码本加解密,解密的运算仅仅是简单的异或操作,即使对于交互性应用也感觉不到明显的延时.

## 3　正交性与限制

软件的保护并不依赖于一项单独的技术,而往往在实际的工程中联合使用多项保护技术,增加破解的难度. 如果 2 项保护技术可以同时使用,则称它们是正交的.

1) 软件水印与指纹. 软件水印[5-6]通过把相关的信息,如软件的版权信息,秘密地嵌入到软件当中来证明软件所代表的知识产权的归属. 软件指纹[5-6]与软件水印相似,不同的是软件指纹在每个不同的软件发布中秘密嵌入不同的信息,软件指纹技术可以有效地帮助追踪盗版的源头. 这些技术增加盗版者的法律风险. 软件水印和指纹技术一般在源码层次上实现,水印的检出不依赖于对目标代码的操作,这些方法可以与软件加密保护联合使用,并且可以有效增强一些方法的抗攻击性.

一些较高级的水印技术依靠编译器的支持,对目标代码进行等价转换,将水印隐藏于代码当中,加密技术妨碍了水印的检出,往往不能和这些水印技术联合使用.

2) 软件混淆. 软件混淆[6]增加了逆向工程的难度,使得攻击者即使获得了软件的代码也难于分析清楚软件的逻辑. 软件混淆与软件加密保护联合使用,极大地增加了软件破解的难度. 这 2 种软件保护方法是正交的,在工程中可以先实施混淆,对混淆后的结果进行加密保护.

3) 抗篡改技术. 为了防止盗版者对软件进行非法篡改,提出了软件抗篡改技术(tamper-proofing). 文献[7]中的方法通过对目标代码的自检,防止攻击者对软件的篡改. 与软件的加密保护和

即时解密技术联合使用,可以增强检测器的隐蔽性. 但是由于二者都涉及对软件目标代码自身的操作,不可简单的结合,需要对它们进行较为复杂的联合设计.

4) 硬件锁保护. 基于硬件锁的软件保护,使得在理想情况下复制软件的难度,等同于复制硬件的难度. 但是如果不与基于软件的保护相结合,攻击者很容易通过篡改软件来绕过硬件锁的保护. 软件的加密保护和即时解密技术,增加了逆向工程难度,使攻击者无法理解软件,进而无法进一步篡改软件,这项技术与硬件锁保护是正交的.

5) 加密保护的使用限制. 软件的加密保护和即时解密技术,通过软件运行时对自身的自动修改(即时解密),实现了对用户的透明,可以广泛的应用在传统的 C/C++等语言开发的应用中. 但是在虚拟机(Java 和. Net 平台)上执行的软件,由于不具备自身修改的能力,不能使用这项技术进行保护.

## 4 实验结果

为了验证基于退化的一次一密乱码本的软件加密技术和即时解密技术在运行时开销方面的优势,将其与传统的软件加密保护方法进行了比较. 选择不同大小的应用程序映像,以考察不同规模应用程序应用这项技术的可行性. 对于使用的加密算法,采用了本文中建议的退化的一次一密乱码本加密,标准 DES,3DES. 而没有采用基于公开密钥的 RSA 等算法,这是因为软件实现的 RSA 算法比 DES 算法要慢 100 倍[2],运行时开销过大.

使用 AMD Athlon XP 2500+ 处理器 512 M DDR 存储器的计算机为试验平台,DES、3DES 算法采用 cryptlib 的高效软件实现,所得试验数据列于表 1 中. 其中,DES,3DES,OTP 分别对应采用相应算法一次性解密的总时间,其后的列对应着采用 JIT 方式解密一个函数的平均时间.

**表 1 一次性和 JIT 方式的运行时时间开销**

| 程序规模/MB | 运行时时间开销/ms | | | | | |
|---|---|---|---|---|---|---|
| | DES | DES-JIT | 3DES | 3DES-JIT | OTP | OTP-JIT |
| 2.13 | 47 | 0.111 | 141 | 0.329 | 15 | 0.035 |
| 11.6 | 265 | 0.094 | 767 | 0.272 | 93 | 0.033 |
| 64.4 | 1 485 | 0.068 | 4 156 | 0.191 | 501 | 0.023 |

对于现代的微处理器,其计算能力已很强大. 对于对称密钥加密和一次一密乱码本加密的解密时间均有不错的实验结果. 对于较小的应用程序,即使用 DES/3DES 进行加密并一次解密也不会带来大的启动时延. 对于较大的应用程序,用 DES/3DES 进行加密并且一次解密就会略感时延,这就体现出一次一密乱码本比 DES、3DES 分别快约 3 倍、9 倍的优势. 使用即时解密更能大大缩短启动时延,用上述 3 种算法启动时时延均不明显. 解密每个函数的时间根据采用的加密算法而变化,以一次一密乱码本最短. 对于交互式应用,用一次一密乱码本加密 JIT 解密方式,运行中无停滞感. 对于对称密钥加密,如用 JIT 解密,运行时偶有停滞感,这是集中解密造成的时延.

## 5 结束语

盗版给整个软件产业带来巨大的损失. 从技术上对软件进行保护,有着重要的工程应用价值. 提出的基于加密的软件保护和即时解密技术,增强了传统的加密保护技术的抗攻击性,减少了一次性启动延时. 并且,由于正交性使得这种方法可以和其他软件保护技术联合使用,对软件进行多重保护. 由于采用一次一密乱码本加密,即使在当前处理器上同样运行流畅. 可以预见,随着处理器运算能力的增强,基于对称密钥加密的软件保护和即时解密技术也有较大的可行性.

## 参考文献(References)


[1] Samarati P, Bertino E, Ciampichetti S. Software engineering for security, a roadmap[C]// Proceedings of the Conference on the Future of Software Engineering. New York: ACM Press, 2000: 227-239

[2] Schneier B. 应用密码学,协议,算法和 C 源程序[M]. 第二版. 北京:机械工业出版社,1996:11-13
Schneier B. Applied cryptography, protocols, algorithms, and source code in C[M]. 2nd ed. Beijing: China Machine Press, 1996: 11-13 (in Chinese)

[3] AYCOCK J. A brief history of just-in-time[J]. ACM Computing Surveys, 2003, 35(2): 97-113

[4] Cramer T, Friedman R, Miller T, et al. Compiling Java just in time[J]. IEEE Micro, 1997, 17(3): 36-43

[5] Collberg C, Thomborson C. Software watermarking: models and dynamic embeddings[C]// Principles of Programming Languages. San Antonio: ACM Press, 1999: 311-324

[6] Collberg C, Thomborson C. Watermarking, tamper-proofing, and obfuscation-tools for software protection[J]. IEEE Transactions on Software Engineering, 2002, 28: 735-746

[7] Horne B, Matheson L R, Sheehan C, et al. Dynamic self-checking techniques for improved tamper resistance[C]// Security and Privacy in Digital Rights Management, LNCS 2320. London: Springer Verlag, 2001: 141-159


(责任编辑:张 嵘)