

目标代码混淆技术综述

李 勇, 左志宏

(电子科技大学 计算机科学与工程学院, 四川 成都 610054)

摘 要: 逆向工程领域的进步, 一方面提升了软件分析能力, 另一方面, 给软件安全带来更大的挑战。目标代码混淆技术是对软件进行保护的一种有力手段, 能够有效地阻挡对软件的恶意分析。文中从逆向分析的角度出发介绍了目标代码混淆技术的分类及几种典型的目标代码混淆技术的实现及混淆效果。

关键词: 目标代码混淆; 逆向工程; 反汇编

中图分类号: TP309

文献标识码: A

文章编号: 1673-629X(2007)04-0125-03

An Overview of Object - Code Obfuscation Technologies

LI Yong, ZUO Zhi-hong

(Coll. of Computer Sci. and Eng., Univ. of Electronic Sci. and Techn. of China, Chengdu 610054, China)

Abstract: Advancement in reverse engineering field upgrades the ability to analyze software, and on the other hand, it brings more challenge to software security. As a powerful means to protect software, object - code obfuscation can obstruct malicious software analysis effectively. From the point of reverse analysis, this article introduces classification of object - code obfuscation and some typical technologies of object - code obfuscation.

Key words: object - code obfuscation; reverse engineering; disassembly

0 引 言

计算机软件大都是以可执行二进制代码的形式来发布, 这种方式不便于阅读, 有助于保护软件开发者的智力成果。然而, 逆向分析能力的进步产生了一些采用逆向工程技术的工具, 借助于这些工具可以将可执行的机器码转换成可以阅读的汇编代码, 虽然比起阅读源代码来说, 难度大了些, 有一定汇编基础和经验丰富的人仍然可以了解程序思路, 使得未经授权的使用、篡改等成为可能。对软件的保护可以从法律和技术两个方面来进行: 法律方面, 就是通过行政立法的手段来对软件的知识产权进行保护; 技术方面, 则主要可以从以下四个角度来实施: 一是通过许可认证的方式, 这种方式目前应用可说是非常广泛, 很多软件都是通过序列号等方式来防止盗版, 包括微软公司的产品系列也采用这种方式。二是通过信息隐藏的形式来实施保护, 最为典型的就加壳技术的应用, 通过对原数据加密的方式来隐藏软件的“真面目”, 除此之外, 数字水印也属于此类, 通过以某种形式隐藏在文件中的特定数据来

标识所有者信息、版权信息等。三是通过网络在服务端运行, 这样避免全部代码落到终端用户手里, 以此来保护软件。四是通过代码混淆来增加软件分析和篡改的难度来实施保护。这四类方法并不是相互排斥的, 事实上, 现在很多软件都是结合多种方法来对软件进行保护。

目标代码混淆就是指对已编译连接好的可执行目标代码进行某种处理, 使得处理后的代码更难以理解。这种处理必须满足下面四个条件^[1~3]:

- 处理前后目标代码在执行功能上没有变化。
- 理解和逆向分析处理后的目标代码要更难或耗费更多的时间(相比较处理前的目标代码)。
- 从处理后的代码还原到处理前的代码是非常困难的。
- 处理后的代码在运行效率上和处理前的代码尽可能接近(一般处理后的代码的运行比处理前的要耗费更多的时间)。

1 混淆技术介绍

对软件的非法篡改需要理解软件开发的一些编程思路, 特别是某些关键地方的判断, 所以通常都需要借助反汇编工具从目标代码分析出相应的汇编代码, 然

后阅读汇编代码来分析。目标代码混淆的目的就是对这一过程的阻挠,使得完成这个过程要付出足够的代价、成本(时间,精力等),以此来达到保护软件的目的。可以将这一过程分为两个步骤:第一步就是从目标代码反汇编出汇编代码;第二步就是阅读分析反汇编出来的汇编代码来寻找突破点。针对这两个步骤,目标代码混淆技术大致上可以分为两类:一类是反-反汇编混淆,用来增加反汇编的难度,使得反汇编工具不能正确地将目标代码反汇编成汇编代码;另一类是指令/控制流混淆^[4],用来增加理解、分析反汇编代码的难度。下面分别介绍这两类技术。

1.1 反-反汇编混淆技术

反汇编可以分为两种,即静态反汇编和动态反汇编^[5]。静态反汇编,是指打开一个静态的可执行的目标代码文件,将里面的机器码翻译成汇编指令;动态反汇编,则是将一个可执行的目标代码文件载入内存运行,在运行的过程中捕捉运行指令翻译成汇编指令。静态反汇编可以一次获取整个文件的指令信息,比较完整。动态反汇编只能获取程序的部分指令,因为一个程序的运行只是对一次特定的输入数据的计算的过程,并不能遍历程序的所有指令。静态反汇编所耗时间与文件大小成正比,而动态反汇编所耗时间与指令多少(循环要展开来计)成正比。一般来说,前者所用的时间要比后者的少很多,从效率上优于后者。这里要讨论的反-反汇编混淆技术主要是针对静态反汇编。动态反汇编^[6]主要是在运行时插入 int 3 中断指令,然后读取内存信息来获取程序的执行指令,如果程序采取内存映像完整性检验则可对抗动态反汇编,因为被插入了中断指令,此时再对程序的内存映像进行 CRC 校验就会发现程序内存被篡改,立即退出就可避免被动态反汇编。也可以采用判断父进程的方法,Windows 平台下的 PE 加载器就是 explorer,如果发现父进程不是它即可判断是被其它调试器加载的。这些技术不属于目标代码混淆的范畴。

静态反汇编主要采取两种方式来扫描指令并进行翻译,一种是线性扫描,另一种是递归扫描^[5,7]。线性扫描就是从可执行文件的代码节开始,先判断操作码,再判断指令长度及操作数,然后从紧接着的地址开始判断下一个指令的操作码,直到结束。递归扫描是当扫描到跳转指令时,停止对当前地址的扫描,跳到跳转指令所指向的地方扫描,完了之后再回到跳转指令的下一地址接着扫描,直至结束,这种方法能够较好地实现代码与数据相分离^[8]。

1.1.1 不完整指令混淆技术

线性扫描的速度非常快,但缺点就是容易产生错

误的指令翻译。因为如果在指令流中碰到数据,很容易误解析为某种指令机器码,而指令都是有一定长度的,一条指令错了,紧跟着的指令由于字节错位也会解析错误。针对这种缺点,可以采用一种不完整指令混淆方法。这种方法就是在程序中插入不完整的指令的。譬如,只有操作码,没有操作数,或者操作数不完整,当反汇编器遇到这个指令时,自然会读取后面的数据使之成为一个完整的指令,从而使得紧跟着的指令也不能被正确地识别。但是这种不完整指令还必须满足一个条件,就是它不能被运行到,不然就会出错。一个简单的办法就是将这种指令插在一个无条件跳转指令的后面,这样满足条件,但是程序中这种无条件跳转指令可能不多,必须想办法人为地创造出无条件跳转指令。下面是一种将条件跳转指令转变成无条件跳转指令的示例:

```
...
jxx addr;假设 jxx 是某种条件跳转指令
...
转换后成为:
...
! jxx bbb;! jxx 是与 jxx 条件相反的跳转指令
jmp addr;无条件跳转指令出现了
.....;这儿可以插入不完整指令
bbb:
...
```

在上面的无条件跳转指令和标记 bbb 之间就是满足条件的插入不完整指令的地方,这样不影响程序的正常运行。

1.1.2 假分支混淆技术

递归扫描因为是依照程序控制流来进行扫描,可以避免遇到嵌在指令流中的数据,以及上面提到的在无条件跳转指令后添加的不完整指令。递归扫描的优点在于把握程序的控制流,但是这一点也可以成为弱点。因为程序中很多跳转都是以计算结果为根据的,而计算结果又依赖于输入的数据,因此,对于大多数的跳转,反汇编程序只能假设这每种分支都是可能的。正是基于这一点,可以将无条件跳转转变成为有分支的条件跳转,而事实上其中一个分支是永不可到达的,是一个假分支,因为构造出的条件可以只有一个计算结果,永为真或永为假。在这个假分支里,就可以插入任意的跳转,甚至可以设计出循环跳转,使得递归扫描找不到出口,一层又一层地进入到分支里直至反汇编程序死掉。

这类混淆因为是针对反汇编算法而设计的,因此混淆效果比较好,库伦(Cullen Linn)在文献[5]中经测试得出,反汇编出的指令错误率能达到 93%,函数错

误率能达到83%(分别是反汇编出的有误的指令占总指令数目的比率和有误的函数占函数数目的比率)。

1.2 指令/控制流混淆技术

除了增加反汇编的难度,尽量使目标代码不能被正确地反汇编出来以外,还可以通过指令/控制流混淆使得反汇编出来的代码难以理解,难以理顺其逻辑思路,藉此来达到软件保护的目的^[5,9]。这一类技术又可以根据是否插入新的指令来划分为不插入指令的代码混淆和插入指令的代码混淆两种。

1.2.1 指令顺序重排(不插入指令)

指令顺序重排是一种简单的混淆技术,它主要是改变一些相互独立的指令的执行序列来达到混淆的目的^[10]。在程序指令序列中,存在着一些相对独立的指令,即前面的指令的运算不影响后面的指令的运算,但是也在一定程度上反映着开发人员的思路,改变它们的顺序能够在一定程度上增加理解程序的难度。下面简单地举个例子,假设有如下基于intel x86处理器的汇编指令:

```
mov eax, 1;ecx=1
inc eax ;eax=eax + 1
mov edx,[esi];edx=[esi]
imul edx,ebx;edx=edx * ebx
mov ecx,ebx;ecx=ebx
imul ecx,ebx;ecx = ecx * ebx
```

上面的指令序列可以分为三组,分别对三个寄存器做操作,互不影响,编程惯例是逐个处理,混淆后,如下:

```
mov ecx,ebx
move ax,1
imul ecx,ebx
mov edx,[esi]
inc eax
imul edx,ebx
```

将人的正常思维扰乱,特别是混淆后指令间的距离拉开的时候。这种方法只是将指令顺序打乱,指令数量上没有发生变化。能够进行指令重排的指令对必须满足几个条件:其一,是相互之间不存在计算上的依赖,不然改变顺序后影响计算结果;其二,中间不存在分支(跳转指令);其三,中间的指令还不能存在其它跳转指令的目的。

1.2.2 插入指令混淆

绝大部分指令/控制流的混淆都需要插入新的指令或修改原有指令。关于这类混淆的具体实现方法也是非常之多。下面举个变换跳转指令的方法,先将一个地址压栈,然后调用ret指令,程序将跳转到先前入栈的地址处,不仅可以用来替换跳转指令,也可以将平

凡的顺序指令变得复杂^[6],示例如下:

```
mov eax,1
mov ebx,2
mov ecx,3
mov edx,4
```

混淆前的示例是非常简单的几条顺序执行的语句,混淆后指令如下:

```
push offset jump2 ;jump2 addr -> push stack
push offset jump1 ;jump1 addr -> push stack
..... ; insert any junk code
ret ; pop stack -> jump1
jump1:
    move ax,1
    mov ebx,2
    ret ;pop stack -> jump2
jump2:
    mov ecx,1
    mov edx
```

从上面的例子可以看到,通过两个简单的ret语句,能够实现将程序的控制流变得复杂,增加阅读、分析的难度。而且在地址jump1入栈和第一个ret语句之间还可以插入其它的垃圾指令来干扰。

这一类混淆技术的混淆效果很难准确地计量,克尔贝格(Christian Collberg)在文献[4]中提到从四个方面来衡量:potency(人工理解程序的难度),resilience(对抗反混淆工具的有效性),cost(采取混淆技术所付出的代价,主要是程序的执行时间上的耗费)以及经验上的调查(就是找不同层次的人来对混淆后的代码进行实际分析)。实验结果表明上面的混淆技术在一定程度上增加了目标代码的分析理解的难度,但在对整个程序混淆的一般性算法以及针对反混淆工具的效果上还需要进一步研究^[10]。

2 结束语

目标代码混淆技术虽然能够增加软件非授权篡改的难度,但是也不能够做到绝对地禁止软件被攻破。软件保护还需要结合多种技术,譬如加壳技术等,除了技术层面的保护手段外,法律上的保护也是非常需要和必要的,只有完善的法律保护加上不断进步的技术保护才能更好地保护计算机软件,维护知识产权。

参考文献:

- [1] Collberg C, Thomborson C. Watermarking, tamper-proofing, and obfuscation - tools for software protection[J]. IEEE Trans Software Eng, 2002, 28: 735 - 746.
- [2] Hada S. Zero Knowledge and Code Obfuscation[C]//ASIAC-

(下转第157页)

和任务相关的空间数据库;第二步,使用一些有效空间挖掘算法计算对象之间的空间连接,从而获得一个候选谓词集合;第三步,对第二步中所得到的谓词集合中的每一个谓词计算其支持度,并且将那些支持度小于最小支持度的谓词删除;第四步,对谓词集合进行进一步精化以确定准确的空间关系;第五步,以第四步所得的候选集作为输入,生成空间关联规则。

(4)空间趋势分析。空间趋势^[10]指离开一个给定的起始空间对象时,非空间属性的变化情况。例如,当离城市中心越来越远时经济形势的变化趋势。其分析结果可能是正向趋势、反向趋势、或者没有趋势。一般在空间数据结构和空间访问方法之上分析空间趋势,需要使用回归和相关的分析方法。由于空间对象自身的特殊性,传统的回归模型可能并不合适。例如,传统的线性回归模型($y = X\beta + \epsilon$)对空间对象就不适用,需要使用空间自回归 SAR 模型: $y = \rho W y + X\beta + \epsilon$ 。

在实际应用中,常常要综合运用上述方法。另外,空间数据挖掘方法要与常规的数据库技术充分结合,数据挖掘利用的技术越多,结果的精确性越高。

4 结束语

数据挖掘通过归纳推理的方法发现和建立模型并预测未来。传统的数据挖掘是针对关系和事务型数据,以数据独立作为挖掘前提,而空间数据挖掘不同,研究的对象可能要受到邻近对象的影响。空间数据库中的空间数据对象保留了各个对象间的空间关系,如空间对象间的拓扑结构、距离和方向等信息,空间数据

挖掘的算法中体现了这种关联特性,如聚类、分类、空间关联等数据挖掘方法。

参考文献:

- [1] Shekhar S, Chawla S. Spatial Databases[M]. 北京:机械工业出版社,2004.
- [2] 邱凯昌. 空间数据挖掘和知识发现的理论与方法[D]. 武汉:武汉大学,1999.
- [3] 李德仁,王树良,史文中,等. 论空间数据挖掘和知识发现[J]. 武汉大学学报:信息科学版,2001,26(6):491-499.
- [4] 毛克彪,田庆久. 空间数据挖掘技术方法及应用[J]. 遥感技术与应用,2004,17(4):199-204.
- [5] 王海起,王劲峰. 空间数据挖掘技术研究进展[J]. 地理与地理信息科学,2005,21(4):33-38.
- [6] 吴信才,刘少雄. 基于邻接关系的空间数据挖掘[J]. 计算机工程,2003,28(7):89-91.
- [7] 毛克彪,覃志豪. 空间数据与 GIS 集成及应用研究[J]. 测绘与空间地理信息,2004,27(7):14-17.
- [8] Fayyad. Advances in Knowledge Discovery and Data Mining[M]. Menlopark, CA: AAAI/MIT Press, 1996.
- [9] Ester M, Frommelt A, Kriegel H P. Spatial Data Mining: Database Primitives, Algorithms and Efficient DBMS Support[J]. Data Mining and Knowledge Discovery, 2000, 18(4): 193-216.
- [10] Ester M, Kriegel H P, Sander J. Spatial Data Mining: a Database Approach[C]//In: Scholl M V eds. Proceedings of The 5th International Symposium on Spatial Databases. Berlin: Springer-Verlag, 1997.
- [11] Han Jiawei. Data Mining: Concepts and Techniques[M]. 北京:机械工业出版社,2004.

(上接第 127 页)

- RYPT'2000 - Advances In Cryptology. Kyoto, Japan: International Association for Cryptologic Research, 2000: 443-457.
- [3] Wang C, Hill J, Knight J, et al. Software tamper resistance: Obstructing static analysis of programs[R]. Technical Report CS-2000-12. Virginia: Department of Computer Science, University of Virginia, 2000.
- [4] Low D. Java Control Flow Obfuscation[D]. Auckland: Department of Computer Science, University of Auckland, 1998.
- [5] Linn C, Debray S. Obfuscation of Executable Code to Improve Resistance to Static Disassembly[C]//In 10th ACM Conference on Computer and Communications Security(CCS). Washington DC: [s. n.], 2003: 290-299.
- [6] 段钢,王勇. 软件加密技术内幕[M]. 北京:电子工业出版社,2004.
- [7] Schwarz B, Debray S K, Andrew G R. Disassembly of Executable Code Revisited[C]//In Proc. IEEE 2002 Working

- Conference on Reverse Engineering(WCRE). [s. l.]: IEEE Computer Society, 2002: 45-54.
- [8] Theiling H. Extracting safe and precise control flow from binaries[C]//In Proc. 7th Conference on Real-Time Computing Systems and Applications (RTCSA). Cheju Island, South Korea: [s. n.], 2000: 23-30.
- [9] Szor P. The Art of Computer Virus Research and Defense[M]. Boston, USA: Addison-Wesley Professional, 2005.
- [10] Wroblewski G R. General Method of Program Code Obfuscation[C]//Proceedings of the International Conference on Software Engineering Research and Practice (SERP). Las Vegas, USA: [s. n.], 2002: 56-75.
- [11] Collberg C, Thomborson C, Low D. A Taxonomy of Obfuscation Transformations[R]. Technical Report # 148. New Zealand: Department of Computer Science, University of Auckland, 1997.