Java 程序混淆技术综述

王建民[1),2),3)] 余志伟[1),2),3),4)] 王朝坤[1),2),3)] 付军宁[1),2),3)]

[1)](清华大学软件学院 北京 100084)

[2)](清华信息科学与技术重点实验室 北京 100084)

[3)](信息系统安全教育部重点实验室 北京 100084)

[4)](清华大学计算机科学与技术系 北京 100084)

摘 要 软件混淆技术已经广泛应用于抵制逆向工程和重组工程.文中从混淆技术的历史发展角度对现有的混淆技术理论、算法、攻击模式和评估进行了综述,将Java程序混淆算法分为类内混淆和类间混淆两个类别,并对其中的各类算法进行详尽的阐释.最后在现有工作的基础上,展望了软件混淆技术未来的发展与研究方向.

关键词 程序混淆;软件水印;防篡改;软件版权保护

中图法分类号 TP309 **DOI号**: 10.3724/SP.J.1016.2011.01578

A Survey on Java Program Obfuscation Techniques

WANG Jian-Min[1),2),3)] YU Zhi-Wei[1),2),3),4)] WANG Chao-Kun[1),2),3)] FU Jun-Ning[1),2),3)]

[1)](*School of Software, Tsinghua University, Beijing* 100084)

[2)](*Tsinghua Laboratory for Information Science and Technology, Beijing* 100084)

[3)](*Key Laboratory for Information System Security of Ministry of Education, Beijing* 100084)

[4)](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract Obfuscation techniques have been widely applied in the defenses of reverse engineering and re-engineering attacks. From the view of the development of obfuscation, we briefly discussed the principles, algorithms, different kinds of attack approaches and evaluating standard. Java program obfuscation algorithms can be divided into two types: One is obfuscation within a class; the other is obfuscation between classes. Finally, based on the survey of these obfuscation techniques, the future research of Java program obfuscation is also stated.

Keywords obfuscation; software watermark; temp-proofing; software copyright protection

1 引 言

随着计算机软件的广泛使用,软件的安全问题严重威胁着软件产业的发展,主要表现为:软件攻击者获得试图攻击的软件备份之后,成功地破解软件.现有的调试和编辑工具,使直接检查或修改二进制程序代码变得更加容易.因此,通常情况下攻击者会重组或破解软件,导致软件的使用控制机制失效.破解后的软件可以非法向大众分发,更加助长了软件盗版问题.

Java程序由于平台无关性得以在Internet上迅速传播和使用.但同时,Java语言的这一特性也带来了保护知识产权方面的新问题.Java语言为了支

收稿日期:2010-10-25;最终修改稿收到日期:2011-07-04.本课题得到国家自然科学基金(61073005,60803016)、国家"九七三"重点基础研究发展规划项目基金(2009CB320706)、国家"八六三"高技术研究发展计划项目基金(2009AA043401)和清华信息科学与技术重点实验室学科交叉基金项目资助.**王建民**,男,1968年生,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域包括数据管理与信息系统、云环境中非结构化数据管理技术、业务过程与产品生命周期管理、软件保护与系统安全技术. E-mail: jimwang@tsinghua.edu.cn. **余志伟**,男,1985年生,博士,主要研究方向为软件混淆、软件水印、防篡改等软件保护技术. **王朝坤**,男,1976年生,博士,副教授,主要研究方向为数据管理、软件保护技术. **付军宁**,女,1985年生,硕士,主要研究方向为软件保护技术与云计算.

持平台无关性，采用了 Class 这种字节码文件格式①. 经过预编译之后，Java 源代码指令转化为字节码，而后在虚拟机上解释执行. 由于 Java 字节码的设计是为了使语言更简洁、更具有平台独立性和网络灵活性，它的指令集相对简单通用，每一个类编译成一个单独的文件，Class 文件保留方法变量名称等大量语义信息. 这些特点都导致 Java 字节码更容易被反编译为 Java 源代码. 攻击者通过反编译和反汇编技术，获得软件的全部或部分源代码，从而获取关键信息如核心算法、秘密信息等为自己所用. 目前已有许多 Java 的反编译工具，如 Jad②、Mocha[1] 等，都是常用的针对 Java 字节码文件的反编译器. 因此，防止针对 Java 字节码文件的静态分析和反编译攻击的要求，显得更加迫切.

混淆技术可以被理解成为是一种特殊的编码技术，目前被广泛地应用于软件知识产权保护领域. 通常所说的混淆是指对拟发布的应用程序进行保持语义的混淆变换，使得变换后的程序和原来的程序在功能上相同或相近，但是更难以被静态分析和逆向工程所攻击，如图 1 所示.

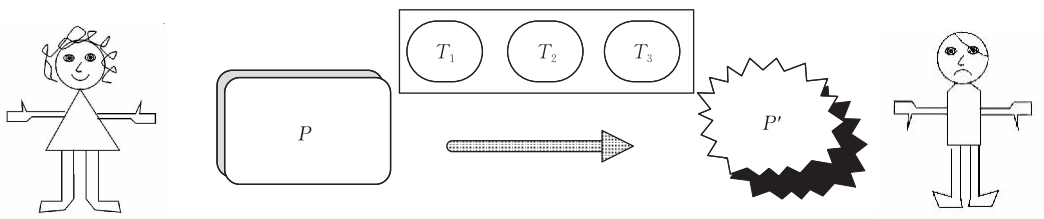

图 1　软件混淆技术应用

本文从混淆技术的发展过程角度，对现有的混淆技术进行综述. 首先，在第 2 节介绍 Collberg 和 Thomborson 等人[2]最早提出的混淆技术的形式化定义；第 3 节介绍混淆算法的分类、各类算法的原理及其实现方法；第 4 节介绍针对混淆算法的理论研究以及攻击模式；第 5 节阐述针对混淆算法的评估度量；最后总结并展望混淆技术的前景.

2　软件混淆技术基本原理

软件③混淆技术是在 1997 年由新西兰 Auckland 大学的 Collberg 和 Thomborson 等人[2]首先提出的. 混淆技术属于一种重要的软件保护技术，其本质是利用程序等价变换的方法，将原始程序转变为语义等价、难以被理解或反编译的程序. 同时，他们还阐述了对于混淆算法的分类及其评估方法，并分别就每一类混淆技术提出了若干算法. 这些理论及算法对混淆技术的发展起到了提纲挈领的作用.

在 Collberg 和 Thomborson 提出的理论中，混淆技术可以视为一种高效的程序变换技术，它将原始程序 P 转换成新的程序 P'. P' 与 P 相比具有相同的外部行为，并且代码安全性能更强. 其形式化定义如下：

设 T 是从原始程序 P 到目标程序 P' 的一个变换，$P'=T(P)$. 如果该变换满足下列条件，就称 $P'=T(P)$ 为一个软件混淆变换：

① 若 P 无法结束或者以错误的状态结束，则 P' 可结束也可不结束；

② 若 P 结束，则 P' 也必须结束并且产生和 P 相同的输出结果；

③ 对于完成某特定的计算任务，P' 比 P 多消耗的运行时间在一个有限的范围内；

④ 攻击者将 P' 恢复为 P 所耗费的时间大于将 P 转换至 P' 的时间.

其中，P 为未经混淆的程序，即原始程序，而 P' 为混淆后的程序. 对于任意的合法输入，P' 与 P 输出相同的处理结果，即满足程序语义等价性.

3　软件混淆技术的分类

Java 文件经过编译生成类文件. 类文件分为不同的字段，包括魔数(magic)、版本(version)、常量池(constant pool)、访问标识(access flags)、(this)类、(super)类、接口(interfaces)、域(fields)、方法(methods)和属性(attributes).

根据混淆算法针对的对象不同，可以将混淆分为类内混淆和类间混淆.

类内混淆指混淆的作用范围在某个类文件内部，基于字节码的类内混淆作用对象主要是上述类

① The Java Virtual Machine Specification. http://java.sun.com/docs/books/jvms/
② Jad. http://www.kpdus.com/jad.html
③ 为描述方便，本文对“程序”和“软件”不作区分.

文件结构方法(methods)中的 code 字段.

类间混淆指混淆作用于两个或者两个以上的类文件,实现的方式主要包括类合并以及类拆分等.

3.1 类内混淆

类内混淆实现的方式主要包括数据混淆、控制流混淆、切片混淆和针对特定工具的混淆.

3.1.1 数据混淆

数据混淆的原理是通过对常量、变量和数据结构这些程序的基本组成元素进行修改的方式,增大攻击者进行逆向工程的难度.数据混淆包括:变量存储和编码混淆、变量聚合混淆、顺序调整混淆、词法混淆以及移除注释和调试信息混淆.

3.1.1.1 变量存储和编码混淆

(1)分裂变量混淆

分裂变量混淆[2-3]将较为简单的数据结构或数据类型(例如 int、boolean 等)分解为一些变量的组合,从而达到隐藏原始数据的效果.

(2)将静态数据转换为与程序相关的数据

以一个静态的字符串为例,可以通过混淆变换将其转化为一个函数或一段子程序,从而在程序执行的过程中通过调用生成相应的字符串[2].这种变换的实现并不困难,但应用时需要考虑以下这对矛盾的制约因素:① 在常见的应用程序中都包含大量的静态数据,如果对所有的静态数据都进行混淆,将导致代码的执行代价和传输代价显著增加;② 如果仅对关键的静态数据进行混淆,则为反混淆者寻找关键数据提供明显的提示.所以,如何把握①和②的平衡点,是需要在实践中解决的问题.

(3)改变编码方式混淆和改变变量生命周期混淆

改变编码方式混淆隐藏了真实的数据,使原程序更加复杂[2].下面是一个改变编码方式混淆的例子:用表达式 $i'=c\times i+u$ 替代整型变量 i,其中 c 和 u 为整型常数.

原程序:int $i=1$; while($i<1000$){…$A[i]$…$i++$}.

混淆后的程序:int $i=11$; while($i<8003$){…$A[(i-3)/8]$…$i=i+8$}.

此外,也可以把局部变量改变为全局变量,从而改变变量的生命周期.

(4)变量替换

改变变量的类型,例如将一个整型变量更改为一个整型对象.这种操作的目的在于,利用 Java 的自动回收垃圾机制及时清理一些变量的使用信息,从而增大逆向工程的难度[2].

3.1.1.2 变量聚合混淆

(1)合并变量混淆

合并变量混淆算法的原理是:在保持程序语义等价的前提下,将两个或两个以上的数值变量 V_1,V_2,…,V_n 合并为一个变量 V_m[2].例如,两个 32 位整型变量可被合并为一个 64 位整型变量.为了增加混淆变换的弹性,可以考虑在程序中添加不影响原始变量值的伪操作.

(2)数组重构混淆

数组重构混淆有多种实现方式,主要包括数组分裂变换(将一个数组拆分为两个子数组)、数组合并变换、数组折叠变换和数组辗平变换[2].为了获得更好的效果,可以将某些上述变换组合使用.

3.1.1.3 顺序调整混淆

程序员在进行程序设计时,通常会把逻辑上相关的数据放在物理上相邻的位置.因此可以通过顺序调整把逻辑上相关的数据分散在物理上不相邻的位置,从而增大攻击者分析程序的难度.顺序调整混淆主要有重新调整实例变量顺序混淆、重新调整方法顺序混淆、重新调整数组顺序混淆[2].

3.1.1.4 词法混淆

词法混淆的原理是通过对函数和变量的名称进行扰乱,使其违背见名知义的软件工程原则[4].这类混淆变换大多针对 Java 语言设计,其原理是依照 Java 虚拟机规范(The Java Virtual Machine Specification)中有关类文件结构的规定,对常量池中存储类、域、方法和变量名字的"CONSTANT-Utf8-info"类型数据项加以混淆.

词法变换混淆具有单向的性质,且无额外的执行代价,在保护知识产权的实践中得到了广泛的应用.目前的 Java 字节码混淆器和混淆编译器大多支持这一功能,可见词法变换具有良好的可行性.为了提高混淆算法的隐蔽性,De A R 和 Van L O[5]提出了使用标识符交换来进行词法变换的方法,其思想是将标识符作交换,而不是将其更改成毫无意义的名字.标识符交换包括交换变量名、交换函数名和交换类名 3 个步骤.使用标识符交换进行词法变换提高了算法的隐蔽性,同时也保留了标识符,这使得该混淆算法不再具有单向性,为反混淆提供了可能性.

Chan 等人[4]在 2002 年提出了一种具体的基于修改和重用标识符的词法混淆算法.其基本的思想是尽可能多地对标识符进行重用,通过对标识符的多次重用来迷惑攻击者.此外,这种算法将复杂的标识符用简单的标识符代替,因此具有减小字节码文件体

积的作用.

3.1.1.5　移除注释和调试信息混淆

程序的注释和调试信息在一定程度上为攻击者理解程序语义提供了便利，去除此类信息可以有效阻碍攻击者对程序的理解. 移除注释和调试信息混淆的主要实现方法是去除注释和调试信息等源代码格式化信息，以及将原有标识符替换为无意义的标识符. 这是一种单向的变换，源文件在经过混淆之后无法恢复[3]. 该混淆方法操作简单，但是强度较差，因为移除的调试信息量较小. 这种移除信息的混淆方法对程序运行的时间和空间复杂度几乎没有影响[2]. 混淆器 Crema① 是一个典型的应用实例.

3.1.2　控制流混淆

程序的控制转换过程的信息是追踪定位程序状态的重要线索，如何保护这部分信息也是软件保护中很重要的一个环节. 控制流图(Control Flow Graph，CFG)是程序可能执行流程的图形化表示，它可以用来描述程序的控制转换. 一个程序可以被分成由一系列无分支的代码组成的基本代码块，这些基本块作为控制流图的结点，而图的边即为各个基本块之间可能的跳转关系. 控制流混淆的目的就是改变或复杂化程序的控制流，使程序更难以破译. 控制混淆可采用的手段很多，比如应用不透明谓词增加伪造分支、加入可导致反编译错误的指令(例如在 Java 字节码中添加 goto 语句等)、将一段代码转换为内联函数调用等[2].

3.1.2.1　控制计算混淆

(1)通过不透明谓词加入不会被执行的分支语句

对于一串语句 S，利用不透明谓词可以构造出两种变换[6]. 第 1 种是增加恒真或恒假的不透明谓词，构造出一条不会被执行的分支，该分支上可以没有语句，也可以有和 S 不同的语句；第 2 种是利用时为真时为假的不透明谓词，令它的两个分支上的语句都与原语句 S 相同. 其中第 2 种混淆变换能够较好地抵抗动态分析.

(2)循环条件插入变换

通过不透明谓词使一个循环的终止条件变得更为复杂，可以达到对循环进行混淆的目的[2]. 两种可能的变换原理如图 2 所示. 其中 f 和 g 是判断谓词 k，j 的表达式.

(3)将可化简的控制流转换为不可化简的控制流

通过将一个结构化的循环变换为含有多个入口的循环，可以将可化简的控制流转换为不可化简的控制流[2]. 其原理如图 3 所示，主要是利用不透明谓词加入的假分支，生成循环交叉，从而使控制流图难以被反编译.

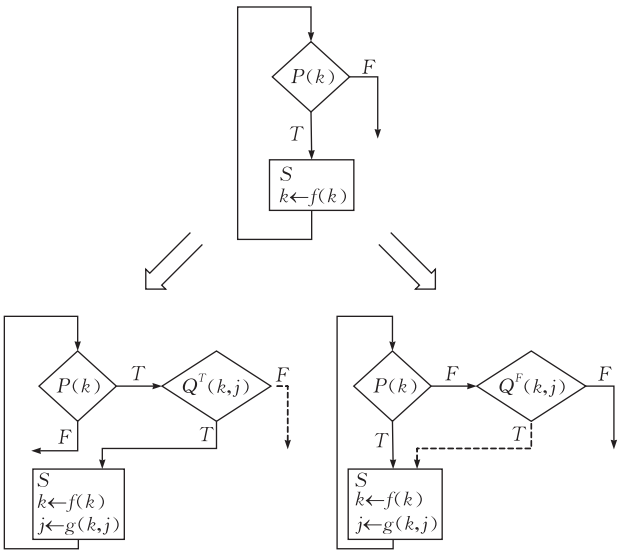

图 2　循环插入方式

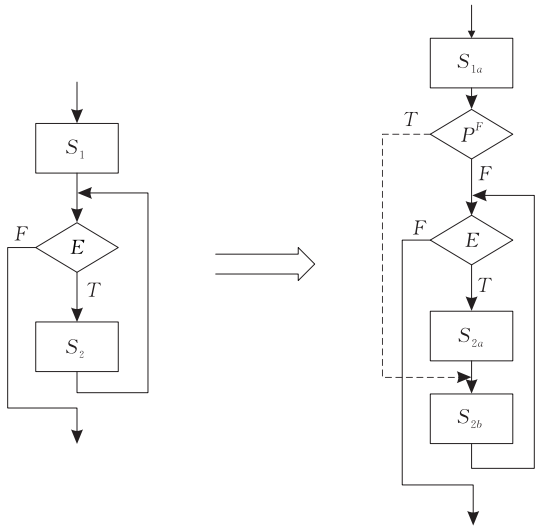

图 3　可化简的控制流图转换为不可化简的控制流图示意

(4)并行化代码

并行化代码是编译器优化程序性能的重要方法之一[2]. 这一技术可以用于混淆程序控制流，从而增大攻击者理解程序的难度. 具体地，有两种实现方式：① 在程序中添加伪造的进程代码(dummy process)；② 将一组串行程序并行化.

使用并行化代码算法，需要注意代码中的数据依赖关系，否则可能破坏程序的语义等价性.

3.1.2.2　控制聚合混淆

通过方法内联变换和将某些代码片段转换成为

① Van V H. Crema-The Java obfuscator. http://web.inter.nl.net/users/H.P.van.Vliet/crema.html, 1996

子程序,可以改变程序自身的控制流图.这种混淆方法是单向的,具有较好的鲁棒性[2].

(1)方法交叉调用和方法克隆

其主要思想如图 4 所示,合并两个调用方法的参数列表和主体,并增加一个参数或全局变量(图 4 中的 int V),用以声明执行时具体应当调用哪一个方法.方法克隆的目的是混淆程序的方法调用关系,克隆的方法应与原方法看似区别但又具有相同的语义[2].

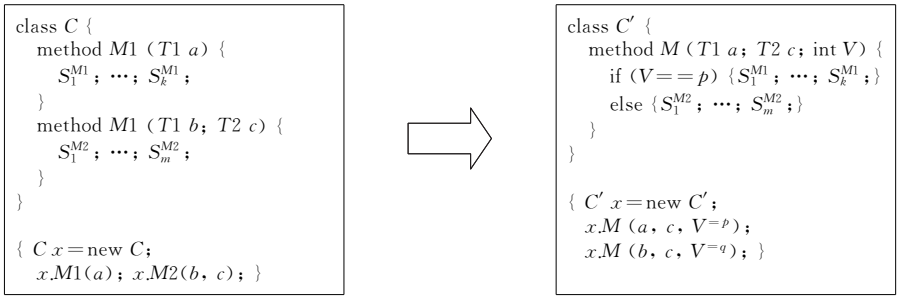

图 4 方法交叉调用

对于控制混淆如何抵抗自动化分析工具的攻击,也有不少研究.Wang[7]提出了通过更改跳转指令实现控制混淆的一种方法,使得编译时候每一个跳转指令的目标都是未知的.Chow 等人[8]提出了一种相似的方法,通过一个有限状态自动机来控制程序的跳转语句.

根据以上的控制混淆算法理论,Low[9]在 1998 年提出了针对 Java 代码的控制混淆算法实现.对于不透明谓词的设计,Majumdar 等人[6]在 2006 年为分布式系统的混淆补充了一种分布式不透明谓词.应用该不透明谓词可以有效地抵抗多种自动静态分析的攻击.

3.1.3 切片混淆

切片通常是用来帮助理解程序的,而混淆的目的是使程序更难以被理解.Drape 等人[10-11]提出了切片混淆算法,使得混淆过的程序能够更好地对抗切片分析攻击.

定义 Slice(P, S, V)用来标记程序 P 在状态 S 时,对应的变量集 V 的一个后向切片(backwards slice),指令 out 表明 V 中变量的输出.

图 5 中显示了一个包含有变量 i,x,y 的程序,当输出为 y 时的程序切片:黑体部分为切片时可观测到的变量值.此时变量 x 不在观测范围内.

切片混淆(slicing obfuscation)算法的主要思想就是尽可能多地将多个变量的值放入到切片的观察范围之内,增加使用切片分析程序的攻击者的困难程度[11].切片混淆的主要方法有:增加恒假谓词、变量编码和增加循环变量.增加恒假谓词是在恒假谓词的假分支上增加令 x 与 y 相关的函数;变量编码是在不改变语义的情况下将 y 的表达式重新编码为与 x 相关的表达式;增加循环变量是在循环变量中添加与 x,y 相关的变量.图 6 至图 8 分别描述了这几种方法的实现.

```
int sumprod (int n) {
int i=0;
int x=0;
int y=1;
while (i<n) {
  i++;
  x=x+i;
  y=y*i; }
out(x);
out(y); }
```

图 5 原始程序 out(y)时的切片

```
int bogus (int n) {
int i=0;
int x=0;
int y=1;
while (i<n) {
 i++;
 if (i<5||x<y)
 x=x+i;
 else y=x*i;
 y=y*i; }
out(x);
out(y);}
```

图 6 增加伪造分支

```
int varEnc (int n) {
int i=0;
int x=0;
int y=1;
while (i<n) {
 i++;
 x=x+i;
 y=(y+i-x)*i+x; }
out(x);
out(y-x); }
```

图 7 变量编码

```
int addVar (int n) {
int i=0;
int x=0;
int y=1;
int j=2;
while (i<n && j>0) {
  i++;
  x=x+i;
  y=y×i;
  j=j+y−x; }
out(x);
out(y); }
```

图 8　增加循环变量

Majumdar 等人[12]从单词计量、结果求和、定位时间等角度验证了切片混淆的方法能够增强程序抵抗切片分析攻击的能力.

3.1.4　针对特定工具的混淆

针对特定工具的混淆是针对自动化的反编译和反混淆工具提出的混淆方法，旨在阻止这类自动化工具的使用，或者增加其使用难度和代价[2]. 该类混淆方法与前面提到的 3 种类别的混淆，最大的不同之处在于，前 3 种混淆技术主要是针对程序的阅读者，即针对的是人；而后者针对的是自动化的反混淆以及反编译工具.

针对特定工具的混淆可以通过与其他的混淆技术综合使用从而达到更好的鲁棒性[2]. 在实际操作中，可以在循环体内部加上伪造的无用变量，形成数据依赖，增加反混淆的自动工具分析难度. 例如在程序的某处使用了反向循环的混淆，为了防止这一变换被反编译器分析破解，可以采用图 9 所示方法：在这个例子中，增加了 B 数组，for 循环的前后向顺序会对 B 数组的各个元素产生影响，从而形成数据依赖，阻止反编译器变换循环的方向. 这一混淆方法的弹性，取决于添加伪造数据的复杂度.

```
For(i=0; i<=10;i++)
{
   A[i]=i;
}
```

⇨

```
int B[50];
For(i=10; i>=10;i−−)
{
   A[i]=i;
   B[i]+=i×i/5;
}
```

图 9　增加伪造的数据依赖

例如，针对 Mocha[1] 这一反编译工具，我们可以采用这样的措施造成它反编译失败：在每个方法的每一条返回语句之后增加额外的指令. 在不影响程序行为的前提下，造成 Mocha 的崩溃. Batchelder 等人[13]在 2007 年较为详细地阐述了如何利用 Java 字节码与 Java 源代码之间指令规范的不同，在字节码的级别进行可以导致反编译失败的控制流混淆. 例如利用 jsr-ret 指令取代普通的 if-goto；针对模式匹配的反编译器，将 load 指令提到 if 指令之前；在子类构造函数中对父类的调用进行外包等等.

3.2　类间混淆

Sosonkin 等人[3]在 2003 年提出了 3 种分别基于类的合并、类的拆分以及类型隐藏的类间混淆算法.

3.2.1　类合并

类合并算法的主要思想是合并两个或多个类各自包含的变量和函数，根据需要重命名变量或函数标识符[3]. 若待合并的类中有标识符相同的变量或非构造函数，则将其改为不重复的新标识符；若待合并的类中有标识符及参数都相同的构造函数，无法随意更改标识符，则合并后为其中一个增加伪造的参数；若待合并的两个类之间有继承关系，则合并后增加一个布尔型的私有变量用于区分标识符相同的函数，图 10 是类合并的一个例子.

```
Original classes:
class A {
   private int i;
   public A() {
   i=5;
   }
   public boolean m(){
      return i <0;
   }
}
class B extends A {
   public B() {
      super();
      i=10;
   }
   public boolean m(){
      return i <10;
   }
}
class C {
void n() {
   A a;
   if (…) {
      a=new A();
   } else {
      a=new B();
   }
   a.m();
   }
}
```

Obfuscation ⇨

```
Obfuscated classes:
class AB {
   private int i;
   private boolean isA;
   public AB() {
      i=5;
      isA=true;
   }
   public AB(int j) {
      this();
      i=10;
      isA=false;
   }
   public boolean m(){
      if (isA) {
      return i <0;
      } else {
      return i <10;
      }
   }
}
class C {
   void n() {
      AB a;
      if (…) {
         a=new AB();
      } else {
         a=new AB(38);
      }
      a.m();
   }
}
```

图 10　一个类合并算法的实现

3.2.2　类拆分

类的拆分算法首先分析判断待拆分的类内部的继承和方法调用关系[14]，观察它们是否能够进行拆

分.对于可拆分的类,主要方法是将一个类 C 拆分为 C1, C2 两个类,且 C2 是 C1 的子类,并确保 C 中的每个方法和变量,在 C2 中均包含或由 C1 继承,图 11 是类拆分的一个例子[3].

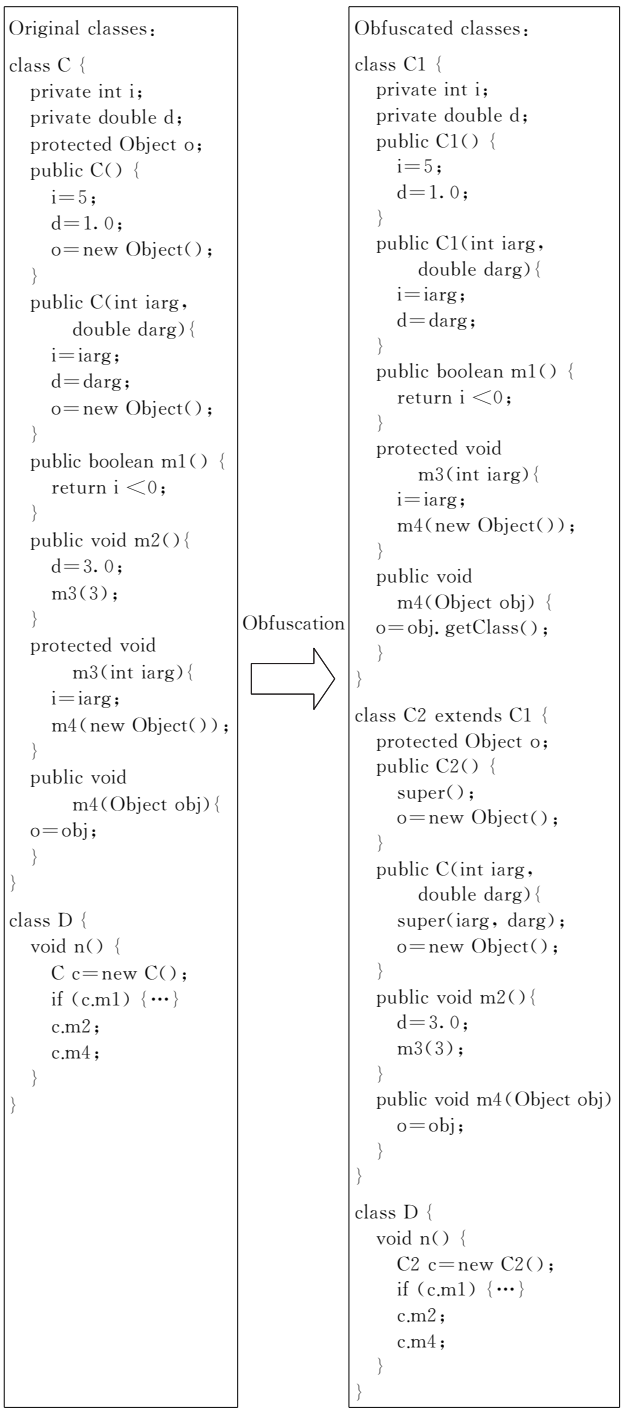

图 11 一个类拆分算法的实现

3.2.3 类型隐藏

类型隐藏算法是利用 Java 中的接口来声明待混淆类的变量和方法,而原先的类则实现这些接口,具体实现无需改动[3].算法的关键部分是每一个接口只随机包含一个待混淆类中公开的方法的子集.

除类合并算法之外,另外两种混淆算法对程序运行时间的影响较小.

4 软件混淆算法的攻击模式

4.1 混淆算法的理论研究

Barak 等人[15]曾经证明,如果把混淆的过程看作是一个"虚拟黑箱"(virtual black box),即要求被混淆过的程序的输入输出应当完全相同于原程序.满足这一性质的混淆器是不可能实现的.然而"虚拟黑箱"的要求过于严格,在较宽泛的限制中,混淆器是可以实现的.

Lynn 等人[16]在 Barak 的证明基础上给出关于程序混淆第一个"积极的"结果:提出了几种针对复杂的访问控制函数的混淆方法,并进行了验证. Wee 等人[17]也针对"虚拟黑箱"理论,阐述了关于点函数的混淆,证明在黑箱条件下对点函数混淆的存在性.关于混淆技术理论,还存在较大的研究空间.

4.2 混淆算法的攻击模式

4.2.1 针对数据混淆算法的攻击

假定 T 是对程序 P 的一个单向的混淆变换,当且仅当从原始程序 P 除去某些信息后,无法通过混淆后的程序 P'恢复出 P[18].词法变换是最典型的不可逆混淆算法.虽然对于经过词法变换的程序进行攻击不可能恢复程序的原貌,但只要理解程序各个模块的含义就可能对程序产生威胁.例如根据无法被混淆的系统 API 名称等关键字,攻击者可以推测出该模块的大致功能.

Cimato 等人[19]提出基于字节码中的标识符修正的反混淆方法.该方法可以用于攻击标识符重命名的混淆算法.

4.2.2 针对控制流混淆算法的攻击

目前,针对控制流混淆算法的主要攻击方法是动态分析. Udupa 等人[20]指出了动态分析对大部分混淆算法的攻击作用,并阐述攻击模型.对于控制流混淆而言,变换后生成的程序中若存在始终不执行的分支,通过动态分析就能找到对破解程序有用的信息.针对控制流混淆,还有黑盒测试攻击[21],该方法通过对程序进行黑盒测试,了解各个类及其函数的功能,从而获取攻击者需要的信息.这种方法对大多数的混淆变换均能加以攻击.同时,该方法也存在着一定的局限性:黑盒测试缺乏自动分析工具,需要依靠大量的人力来完成分析工作.同时, Udupa 等

人[22]还提出将静态分析与动态分析相结合的反混淆方法.

Lakhotia 等人[23]提出可以用抽象栈图的方法检测混淆后程序中的调用关系. Chen 等人[24]提出使用面向方向的语言 AspectJ 的字节码工具破解混淆算法，该方案可以修正混淆后的程序的方法. 表 1 是对上述典型攻击方式的简单阐释.

表 1　针对程序混淆的典型攻击

攻击方法	针对混淆算法
基于字节码中的标识符修正方法	数据混淆(标识符重命名)
动态分析；静态分析；静态和动态分析结合	控制流混淆
黑盒测试	控制流混淆
抽象栈图	控制流混淆
使用 AspectJ 分析	数据混淆；控制流混淆

5　混淆算法的评估

上述混淆算法的效用如何，需要一个对混淆算法进行评价的标准. 文献[2]中提出了 3 个评估混淆算法的指标，分别为程序增加的复杂度即强度(potency)、算法抵抗机器攻击的能力即弹性(resilience)以及由于对代码转换而带来的额外开销(cost).

强度(potency)表征混淆算法为程序所增加的复杂度以及理解的困难程度.

程序主要有 7 类属性决定它的复杂程度，分别是程序长度、循环复杂度、嵌套复杂度、入度出度复杂性、数据流复杂度、数据结构复杂度和面向对象的特性(包括方法个数、继承树深度等指标). 混淆算法对程序这些属性的值提升越多，它的强度性能越好.

弹性(resilience)表征混淆算法抵抗攻击的能力.

弹性由两个部分组成：一部分是攻击者为了攻克算法，设计并实现一个相应反混淆器所需花费的时间；另一部分是反混淆器对混淆算法进行反混淆所花费的开销[24]. 弹性区别于强度的主要特点是针对自动化的反混淆工具，而强度针对的是程序的阅读者.

开销(cost)衡量的是混淆算法给程序带来的额外开销.

开销包括有两个方面：一方面是在对程序混淆时所花费的开销；另一方面是混淆后的程序相对于原始程序执行时所增加的时间复杂度以及空间复杂度.

良好的混淆算法应当使其强度和弹性最大化，同时最小化开销. 表 2 对上述混淆算法做了一个综合的评估.

表 2　混淆算法 3 项指标评估

作用范围	混淆类别		强度(高/中/低)	弹性(好/中/弱)	开销(高/中/低)
类内	数据混淆	变量存储和编码混淆	低	好	低
		变量聚合混淆	中	弱	低
		顺序调整混淆	低	弱	中
		词法混淆	中/低	好	低
		移除注释和调试信息混淆	高	中	低
	控制流混淆	控制计算混淆	高	好	高
		控制聚合混淆	中/低	弱	低
		控制顺序混淆	低	好	低
	切片混淆		高	中	中
	针对特定工具的混淆		中	好	低
类间	基于类变换的混淆	类合并	高/中	好/中	高
		类拆分	高/中	好/中	低
		类型隐藏	中	中	低

基于上述评估指标，Naeem 等[25]对现有的一些反编译器和混淆器的性能进行了评估，针对前者有 Jad、SourceAgain①，针对后者则主要包括 JBCO[26]以及 KlassMaster②.

2007 年，Anckaert 等人[27]提出了基于软件复杂度的度量方法用以评估混淆和反混淆的质量. 这里的度量指标包括 4 个：程序代码、控制流、数据和数据流.

2008 年，Ceccato 等人[28]提出一种新的评估方法：根据攻击者理解和更改混淆后代码的难易程度来度量混淆算法的质量. 这种方式与上述方式最大的不同之处在于考虑了攻击者这一因素，通过比较测试者(程序员)对混淆和未混淆的代码进行攻击时的表现，来经验性地评估混淆算法. 实验得到的结果是，混淆以后的代码与未混淆代码相比，前者被攻击

① Source Again. http://www.ahpah.com/

② Zelix KlassMaster—The second generation Java Obfuscator. http://www.zelix.com/klassmaster

成功的概率是后者的 1/7. 可见,混淆算法可以显著增加测试者理解代码的难度.

2009 年,Tsai 等人[29]提出了用于评估和分析控制流混淆算法的框架. 借助基于图的表示方法,许多现有的控制流变换都可以转化为原子操作表示. 同时,他们还提出了一种对混淆变换效果定量分析度量方法,帮助评估控制流混淆的弹性以及开销代价.

6 总结与展望

混淆技术是一种可用于抵制逆向工程和重组工程、对软件知识产权进行保护的程序变换技术. 使用混淆技术虽然会使代码的效率有一定程度的降低,但是它的实现代价相对较小,因而在近年引起了广泛的关注[2,25,30-33]. 由于混淆技术不改变程序功能的特性,并且有些词法混淆算法甚至可以减小原程序的体积,因此混淆技术得以在保护移动代码方面得到广泛的应用. 受移动平台资源的限制,今后混淆技术的研究方向一方面要加大混淆的力度,增加攻击者反编译的难度,另一方面也要考虑降低混淆算法对目标程序的运行负担. 文献[26,32]探讨了混淆技术在移动代理上的应用.

目前保护软件知识产权的新技术主要有 3 种:混淆技术、软件水印技术和防篡改技术[25]. 这 3 种技术各有特长,将它们互相结合使用,可以给予目标软件更为可靠的保护,这也是目前研究的一个重要方向.

混淆技术的主要目的是增加攻击者对程序进行反编译的难度和代价,因而可以用于提高防篡改技术的保护力度[8,34-36]. 此外,混淆技术可以增强软件水印的鲁棒性,使软件系统的安全性更强. 例如不透明谓词除了上文提到的用于控制流混淆之外,也经常用于在程序中添加软件水印[37-38]. 孙光等人[39]提出将程序的编译混淆,而不是混淆程序本身. 但是该方法还有若干问题需要考虑,比如加入水印后的软件执行效率问题,不同水印算法与混淆算法结合的顺序等.

最后,对于混淆算法的评估、正确性验证以及如何研制高效可靠的混淆算法,也是将来的发展方向之一.

致 谢 在此,我们向对本文的工作给予支持和建议的同行,尤其是奥克兰大学的 Thomborson 教授给予的指导和帮助表示感谢!

参 考 文 献

[1] Van V H. Mocha, the Java Decompiler. http://www.brouhaha.com/~eric/software/mocha/, 1996

[2] Collberg C, Thomborson C, Low D. A Taxonomy of Obfuscating Transformations. Department of Computer Science, University of Auckland: Technical Report 10, 1997

[3] Sosonkin M, Naumovich G, Memon N. Obfuscation of design intent in object-oriented applications//Proceedings of the Digital Rights Management Workshop. Washington, DC, USA, 2003: 142-153

[4] Chan J T, Yang W. Advanced obfuscation techniques for Java bytecode. Journal of Systems and Software, 2004, 71(1-2): 1-10

[5] De A R, Van L O. Stealthy obfuscation techniques: Misleading the pirates. Department of Computer Science University of Twente Enschede, The Netherlands, 2003

[6] Majumdar A, Thomborson C. Manufacturing opaque predicates in distributed systems for code obfuscation//Proceedings of the 4th International Conference on Information Security. Hobart, Tasmania, Australia, 2006: 187-196

[7] Wang C, Hill J, Knight J et al. Software tamper resistance: Obstructing static analysis of programs. University of Virginia: Technical Report CS-2000-12, 2000

[8] Chow S, Gu Y, Johnson H et al. An Approach to the Obfuscation of Control-Flow of Sequential Computer Programs//Proceedings of the 4th International Conference on Information. Malaga, Spain, 2001: 144-155

[9] Low D. Java Control Flow Obfuscation[M. S. dissertation]. University of Auckland, 1998

[10] Drape S, Majumdar A, Thomborson C. Slicing aided design of obfuscating transforms//Proceedings of the International Computing and Information Systems Conference (ICIS 2007). Melbourne, Australia, 2007: 1019-1024

[11] Majumdar A, Drape S, Thomborson C. Slicing obfuscations: Design, correctness, and evaluation//Proceedings of the 2007 ACM Workshop on Digital Rights. Alexandria, VA, USA, 2007: 70-81

[12] Majumdar A, Drape S, Thomborson C et al. Metrics-based evaluation of slicing obfuscations//Proceedings of the 3rd International Symposium on Information Assurance and Security. Manchester, United Kingdom, 2007: 472-477

[13] Batchelder M, Hendren L. Obfuscating Java: The most pain for the least gain//Proceedings of the Compiler Construction. Braga, Portugal, 2007: 96-110

[14] Snelting G, Tip F. Understanding class hierarchies using concept analysis. ACM Transactions on Programming Languages and Systems (TOPLAS), 2000, 22(3): 540-582

[15] Barak B, Goldreich O, Impagliazzo R et al. On the (Im)possibility of Obfuscating Programs//Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology. Santa Barbara, California, USA, 2001: 1-18

[16] Lynn B, Prabhakaran M, Sahai A. Positive results and techniques for obfuscation//Proceedings of the Eurocrypt'04. Interlaken, Switzerland, 2004: 20-39

[17] Wee H. On obfuscating point functions//Proceedings of the 37th Annual ACM Symposium on Theory of Computing. Baltimore, MD, USA, 2005: 523-532

[18] Wang Xiao. Research and implementation of JAVA bytecode control flow obfuscator [M. S. dissertation]. Tsinghua University, Beijing, 2009(in Chinese)
(王潇. JAVA 字节码控制流混淆器的研究与实现. 清华大学, 北京, 2009)

[19] Cimato S, De S A, Petrillo U F. Overcoming the obfuscation of Java programs by identifier renaming. Journal of Systems and Software, 2005, 78(1): 60-72

[20] Udupa S K, Debray S K, Madou M. Deobfuscation: Reverse engineering obfuscated code//Proceedings of the 12th Working Conference on Reverse Engineering. Pittsburgh, PA, USA, 2005: 45-54

[21] Hohl F. Time limited blackbox security: Protecting mobile agents from malicious hosts//Lecture Notes in Computer Science 1419. Berlin: Springer-Verlag, 1998: 92-113

[22] Udupa S K, Debray S K, Deobfuscation M M. Reverse Engineering Obfuscated Code//Proceedings of the 12th Working Conference on Reverse Engineering. Pittsburgh, PA, USA, 2005: 45-54

[23] Lim H, Park H, Choi S et al. A method for detecting the theft of Java programs through analysis of the control flow information. Information Software Technology, 2009, 51(9): 1338-1350

[24] Chen K, Chen J B. On instrumenting obfuscated java bytecode with aspects//Proceedings of the 2006 International Workshop on Software Engineering for Secure Systems. Shanghai, China, 2006: 19-26

[25] Naeem N A, Batchelder M, Hendren L. Metrics for Measuring the Effectiveness of Decompilers and Obfuscators//Proceedings of the 15th IEEE International Conference on Program. Banff, Alberta, Canada, 2007: 253-258

[26] Majumdar A, Thomborson C. On the use of opaque predicates in mobile agent code obfuscation//Proceedings of the ISI 2005. Altanta, GA, USA, 2005: 648-649

[27] Anckaert B, Madou M, De S B et al. Program obfuscation: A quantitative approach//Proceedings of the 2007 ACM Workshop on Quality of Protection. Alexandria, VA, USA, 2007: 15-20

[28] Ceccato M, Di P M, Nagra J et al. Towards experimental evaluation of code obfuscation techniques//Proceedings of the 4th ACM Workshop on Quality of Protection. Alexandria, VA, USA, 2008: 39-46

[29] Tsai H Y, Huang Y L, Wagner D. A graph approach to quantitative analysis of control-flow obfuscating. IEEE Transactions on Information Forensics and Security, 2009, 4(2): 257-267

[30] Collberg C S, Thomborson C. Watermarking, tamper-proffing, and obfuscation: tools for software protection. IEEE Transactions on Software Engineering, 2002, 28(8): 735-746

[31] Naumovich G, Memon N. Preventing piracy, reverse engineering, and tampering. IEEE Computer, 2003, 36(7): 64-71

[32] Wu J, Zhang Y, Wang X et al. A scheme for protecting mobile agents based on combining obfuscated control flow and time checking technology//Proceedings of the Conference on Computational Intelligence and Security. Harbin, Heilongjiang, China, 2007: 912-916

[33] Collberg C, Nagra J. Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection. New Jersey, USA: Addison-Wesley Professional, 2009

[34] Ertaul L, Venkatesh S. Novel obfuscation algorithms for software security//Proceedings of the 2005 International Conference on Software Engineering Research and Practice (SERP'05). Las Vegas, Nevada, USA, 2005: 209-215

[35] Aucsmith D. Tamper resistant software: An implementation//Proceedings of the 1st International Workshop on Information Hiding. Cambridge, UK, 1996: 317-333

[36] Ge J, Chaudhuri S, Tyagi A. Control Flow Based Obfuscation//Proceedings of the Digital Rights Management Workshop. Alexandria, VA, USA, 2005: 83-92

[37] Arboit G. A method for watermarking Java programs via opaque predicates//Proceedings of the International Conference on Electronic Commerce Research (ICECR-5). Montreal, Canada, 2002: 1-8

[38] Myles G, Collberg C. Software watermarking via opaque predicates: Implementation, analysis, and attacks. Electronic Commerce Research, 2006, 6(2): 155-171

[39] Sun Guang, Sun Xing-Ming, Yang Rong, Huang Hua-Jun. A obfuscating compile framework for watermark. Science Technology and Engineering, 2005, 5(10): 656-660 (in Chinese)
(孙光, 孙星明, 杨蓉, 黄华军. 可嵌入水印的混淆编译框架. 科学技术与工程, 2005, 5(10): 656-660)

WANG Jian-Min, born in 1968, Ph. D., professor, Ph. D. supervisor. His research interests include data management and information system, unstructured data management in cloud environment, service processing and lifecycle management, software protection and security system techniques.

YU Zhi-Wei, born in 1985, Ph. D. candidate. His main research interests include software obfuscation, watermarking and temper proofing techniques.

WANG Chao-Kun, born in 1976, Ph. D., associate professor, master supervisor. His main research interests include data management and software protection techniques.

FU Jun-Ning, born in 1984, M. S.. Her main research interests include software protection techniques and cloud computing.

Background

This paper mainly discusses the existing Java program obfuscation techniques. This research work is supported by the National Natural Science Foundation of China (Nos. 61073005, 60803016), the National Basic Research Program of China (No. 2009CB320706), the National High Technology Research and Development Program of China (No. 2009AA043401), and the Tsinghua National Laboratory for Information Science and Technology (TNLIST) Cross-discipline Foundation. With the rapid development of computer technologies, software is becoming more and more popular in our daily life, which introduces the security issues such as reverse engineering and re-engineering attacks. Java program is subject to these attacks due to its platform independence. Accordingly, obfuscation technique has great significance in software security research and development. In recent years, program obfuscation as one of the key skills for software protection attracts much attention from researchers both at home and abroad. This technique aims at preventing program from unauthorized modification and reverse engineering, by obfuscating the program. This paper generally introduces two categories of Java program obfuscation techniques: one is obfuscation within a class, and the other is obfuscation between classes. And then different obfuscation algorithms are classified according to these two categories. Moreover, the theory, implementation and evaluation of these algorithms are elaborated to illustrate the usages of them. At last, we conclude this paper by giving future research directions of program obfuscation techniques.