Java 类文件保护方法综述①

Summarization of protect methods for java class file

鲍福良　彭俊艳　方志刚　（浙江大学城市学院信电分院　杭州 310015）

摘要：对 Java 类文件的安全性从编译和反编译的角度进行了深入的分析，综述了现有的各种最新 Java 类文件保护方法及其优缺点。对各种方法进行了进行比较后，提出了自己特有的改进方法。

关键词：java　类文件　反编译　安全

1　引言

编译后的 Java 类文件不是真正的二进制文件，而是一种有格式的中间代码，这就给黑客反编译 Java 类文件提供了可能，反编译后的代码和源代码几乎没有差别。一些作者编写的源代码就会轻而易举地被黑客窃取，一些重要的算法也会泄漏出去。因此 Java 应用程序在源代码上会产生很大的安全问题。近年来，已经有许多公司和 Java 开发人员对 Java 类文件和虚拟机进行了深入的分析，并在此基础上采取了各种方法来保护 Java 类文件，在一定程度上起到了保护 Java 类文件的作用。如本地编译技术、代码隔离技术、代码混淆技术、ClassLoader 加密技术以及数字水印技术等保护方法，但是都有各自的局限性。本文综述了人们在这些方面的研究成果，同时提出了自己的改进方法。

2　Java 类文件的安全

2.1　Java 的编译

开发 Java 应用程序的过程是：首先使用编辑工具编写 Java 的源代码（扩展名为. java 的文件），然后使用编译器（如 jdk 自带的 javac 编译器）编译成虚拟机可执行的类文件（扩展名为. class 的文件）。编译后生成的类文件是一种有格式的中间代码——字节码文件，不能在本地机器上独立运行，只能在 Java 虚拟机里解释执行。

Java 的这一编译过程和 C/C + + 的编译过程有些不同。C/C + + 编译器编译生成的一个对象代码是为在某一特定平台运行的代码，编译器通过查找表将所有变量和方法等符号的引用转换为特定的内存偏移量[1]。而 Java 编译器却不对变量和方法等符号的引用转换为数值引用，也不确定程序执行过程中的内存布局，而是将这些符号的引用信息保留在类文件中，由解释器在运行过程中创建内存布局，然后再通过查找表来确定一个变量或方法所在的地址[3]。

从 Java 类文件的结构及其实际数据可知 Java 类文件保留了源代码文件的大部份信息，如所有的变量和方法等信息。正是由于这个特点，只要在各个平台上实现了各自的 Java 虚拟机，不用修改 Java 应用程序的源代码就可以在各个平台上运行，真正做到跨平台的特性，这也是 Java 能够迅速流行起来的重要原因。

2.2　Java 的反编译

反编译是一个将目标代码转换成源代码的过程[2]。而目标代码是一种用语言表示的代码，这种语言能通过实机或虚拟机直接执行。从本质上说，他需要根据小规模、低层次的行为来推断大规模、高层次的行为。因此，反编译目标代码并不容易，特别是对 C/C + + 编译生成的二进制目标代码。

在 JDK 中，有一个反编译器 javap[9]，利用该工具可以对 Java 类文件进行反编译。经过该工具反汇编后得到的结果并不是源代码，但是通过使用 javap 进行反编译的 Java 类文件可以得到成员变量、方法、行号以及局部变量名等信息[7]。在 javap 工具的基础上，一些反编译工具如 Mocha，WinDis，DjDecompiler 等

① 基金项目：浙江省科技计划项目（2006C31006）

工具可反编译出和源代码几乎一模一样的代码。

3 Java 类文件保护方法

对 Java 类文件进行反编译后得到的代码和原始的源代码几乎没有差别，因此如何保护 Java 类文件不被反编译或者提高反编译的难度已经是 Java 领域的一个重要研究课题。目前针对 Java 类文件的保护方法主要有以下几种方法：本地编译技术、代码隔离技术、代码混淆技术、ClassLoader 加密技术以及数字水印技术等方法。

3.1 本地编译技术

Java 本地编译是指将 Java 应用程序编译成本地应用程序[8]，如 Windows 操作系统上扩展名为 exe 的应用程序。其步骤如下：首先编写 Java 源代码，然后通过 Java 编译器将 Java 源代码编译成 Java 类文件，最后将 Java 类文件编译成真正的本地应用程序。

使用该技术生成的本地应用程序是二进制格式的可执行文件，与在虚拟机中执行的 Java 应用程序相比，可以产生更快的执行速度和更小的内存占用，而且其安全性能也等价于本地可执行应用程序的安全强度，这些对于当今许多应用都很关键[10]。但该方法牺牲了 Java 的跨平台特性，且现有的本地编译器都不够成熟、类支持也不广泛等原因直接影响了该技术的应用。

3.2 代码隔离技术

代码隔离技术是让用户访问不到类文件，如将类文件放在远程的服务器端，客户端通过访问服务器的相关接口来获得服务。这种方法使得黑客无法获得类文件，从而无法进行反编译。而且这种服务接口可以提高系统的可移植性和互操作性，大大降低软件的开发成本。现在通过接口提供服务的标准和协议也越来越多，如 HTTP，RPC，Web Service 等[13]。

代码隔离技术能很好的起到保护 Java 类文件的作用。但是这种技术也有它的局限性和不安全性。首先这种方法只能适合网络环境的客户/服务器结构或者分布式的环境，对 Java 的通用组件 Jar 文件[14]以及客户机运行的应用程序就显得无能为力。其次，需要使用安全机制保护服务器开放接口的使用，服务器的安全成了整个系统安全的焦点。一旦服务器被攻破，Java 类文件就很容易被窃取，接下来的反编译工作是非常容易的，因此后果不堪设想。

3.3 代码混淆技术

代码混淆技术是目前比较成熟和流行的 Java 类文件保护方法[4]，其本质上是一种类文件模糊技术。它的原理就是把类文件重新进行组织，使别人无法轻易的读懂反编译出来的代码，但是处理以后的类文件功能和处理以前的类文件功能在逻辑上是等同的，即运行后能够得到一样的输出结果。

有一些专业的代码混淆工具已经非常出色，如果合理的使用这些工具就可以对自己的产品起到很好的保护作用。但是代码混淆工具也不是万无一失的方法，事实上只要有足够的耐心，这些混淆了的代码还是可以被反编译出来并且能够读懂。特别是在重构技术非常成熟的今天，要替换这些变量名或者函数名还是非常容易的。

3.4 ClassLoader 加密技术

表 1 不同 Java 类文件保护方法的比较

方法	安全等级	方便性	致命缺陷
本地编译技术	高	较好	非跨平台、成熟度不够、不支持复杂应用
代码隔离技术	高	较好	不支持通用组件、对服务器的安全要求高
代码混淆技术	中	较好	易被反编译
ClassLoader 加密技术	中	一般	ClassLoader 易被反编译
数字水印技术	低	一般	不能保护被反编译，只能保证版权

该技术是利用 Java 虚拟机调入 Class 到系统中进行执行的过程。由于虚拟机每次装入类文件时都需要使用一个 ClassLoader 对象，该对象负责把新的类装入到虚拟机中。虚拟机向 ClassLoader 对象提供一个包含待装入类名字的字符串，然后由 ClassLoader 负责找到类文件，并把它转换成一个 Class 对象[12]。利用上述机制我们可以改写 ClassLoader 对象，在装入了原始数据后先进行解密，然后再转换成 Class 对象。由于把原始字节码转换成 Class 对象的过程完全由系统负责，只需先获得原始数据，接着就可以进行包含解密在内的任何转换。

这种保护系统源代码的方法比其他方法更加安全，然而这种加密方法存在着一个严重的漏洞，由于

ClassLoader 的类是使用 Java 编写的，如果把 ClassLoader 函数进行反编译，提取其中解密部分的内容，就可以解密所有加密的类。

3.5 数字水印技术

由于 Java 类文件容易被反编译，从而导致对 Java 程序的未授权使用变得容易。所以在需要证明程序是否非法使用时，数字水印就变得很重要。像在图片声音中嵌入水印一样，在 Java 程序中也能嵌入透明的、安全的和鲁棒性的信息。使用水印技术并不能阻止类文件被反编译，但是可以在需要确认某些程序是否属于剽窃时提供有效的证据。它可以有效地保证开发者对该程序的版权。嵌入的水印对程序的使用者来说是透明的，而对程序的开发者来说，可以轻易地找出未经授权的非法的程序使用[5]。

然而数字水印技术也存在一些不足，比如需要插入额外的代码，需要仔细地编写哑函数及其调用，否则容易被有经验的反编译者识破，从而擦除水印。而且如果同时利用各种 Java 混淆器对带有水印的 Java 文件进行攻击，在有些情况下水印将失效[6]。另外目前的水印算法在提供可靠的版权证明方面或多或少有一定的不完善性，因此寻找能提供完全版权保护的数字水印算法也是一个重要的课题[11]。

4 各种技术的比较及其改进

4.1 各种技术的比较

综上所述，各种 Java 类文件保护方法都有其优势，也有其局限性。现从安全等级、方便性以及致命缺陷等各个方面对上述各方法进行比较，如表 1。

4.2 代码混淆技术的改进

Java 因其具有跨平台性、开放性和高度成熟等特性得到了广泛应用。目前的应用主要包括客户端应用程序和 Web 应用程序。客户端应用程序都是以 Jar 包的方式发布，Web 应用程序都是以 War 包或 Ear 包的方式发布。这些 Jar 包、War 包和 Ear 包都是对 Java 类文件的封装包，目前对这些封装包最流行的保护方法是采用代码混淆技术。

代码混淆技术本质上并不能阻止类文件被反编译，只能干扰黑客阅读反编译后的代码。因此，可以对混淆技术加以改进，加入干扰的条件判断代码，这些条件判断代码不会改变程序的执行流程，但增加的代码可以增加黑客理解反编译后代码的难度。

4.3 ClassLoader 加密技术的改进

ClassLoader 加密技术的问题在于对 ClassLoader 类(Java 语言编写)进行反编译即可获取解密部分的法，从而可以解密其他被加密的 Java 类文件。对把 ClassLoader 的解密函数部分用 C 语言编写，并采用 JNI 技术调用解密函数，黑客就很难对 C 语言编写的解密函数进行反编译，其安全性可以达到本地应用程序的安全强度。

5 结束语

随着 Java 技术的进一步发展，很多企业使用 Java 开发自己的产品，但是发布的 Java 类文件很容易被反编译。本文比较了各种 Java 类文件的保护方法，并提出了对代码混淆技术和 ClassLoader 技术的改进方法。

参考文献

1 Bacon, David F, Susan L. Compiler Transformations for High – Performance Computing[J], ACM Computing Surveys. 1994, 26(4):345 – 420.

2 Behrens B, Levary R. Practical Legal Aspects of Software Reverse Engineering[J], Communications of the ACM. 1998,41(2): 27 – 29.

3 Cifuentes, Cristina, K. John Gough. Decompilation of Binary Programs[J], Software – Practice and Experience. 1995, 25(7):811 – 829.

4 Chan Jien – Tsai, Yang Wu. Advanced Obfuscation Techniques for Java Bytecode[J], Journal of Systems and Software Volume. 2004,71(1):1 – 10.

5 Berghei H. Wartermarking Cyberspace[J], Communications of the ACM. 1997, 40(11):19 – 24.

6 Ichisugi Y. Watermark for Software and its Insertion, Attacking, Evaluation and Implementation Methods [A], Summer Symposium on Programming, IPSJ [C]. 1997:57 – 64.

7 Hanpeter van Vliet. Mocha, the Java Decompiler [EB/OL]. http://www.brouhaha.com/ ~ eric/software/mocha/.

(下转第 102 页)

(上接第 126 页)

8 Martyn Honeyford. Weighing in on Java native compilation [EB/OL]. http://www-900.ibm.com/developerworks/cn/java/j-native/index.shtml.

9 刘劼,JAVA 反编译技术和代码安全[J],现代电子技术,2004,27(10):22-23.

10 冀振燕,java 编译程序技术与 Java 性能[J],软件学报,2000,11(02):173-178.

11 陈晗、赵轶群、缪亚波,Java 字节码的水印嵌入[J],计算机应用,2003,23(9):96-99.

12 张敦华、刘建,Java 动态类加载机制及其应用[J],计算机工程与设计,2004,25(3):432-435.

13 飞天诚信,软件加密原理与应用[M],北京:电子工业出版社,2004.

14 陈刚,基于封装包的 Java 源代码安全保护[J],电子信息对抗技术,2006,21(3):45-48.