第 36 卷 第 8 期 计 算 机 工 程 2010 年 4 月
Vol.36 No.8 Computer Engineering April 2010
·安全技术· 文章编号：1000—3428(2010)08—0147—03 文献标识码：A 中图分类号：TP309.2

基于面向方面编程的 J2EE 源代码保护

李媛媛
(天津工业大学计算机科学与软件学院，天津 300160)

摘 要：分析类文件加密技术在保护 J2EE 应用时遇到的动态编译问题和服务器检测问题，指出这 2 个问题都是由字节码文件格式遭破坏而引起的。利用面向方面编程技术的连接点模型，通过制作一个骨架类解决上述问题，从而扩展加密技术的适用范围，增强对 J2EE 源代码的保护。

关键词：加密；J2EE 源代码；面向方面编程；连接点模型；源代码保护

J2EE Source Code Protection Based on Aspect-Oriented Programming

LI Yuan-yuan
(School of Computer Science & Software Engineering, Tianjin Polytechnic University, Tianjin 300160)

【Abstract】 This paper analyzes two issues of dynamic compilation and server detection, when encryption technology on class files is applied to protect J2EE applications, which are both due to the broken format of the encrypted classes files. With the joint point model provided by Aspect-Oriented Programming(AOP) technology, the two problems can be solved by making the special class named as sketch class. And it enlarges the range of encryption to protect J2EE source code.

【Key words】 encryption; J2EE source code; Aspect-Oriented Programming(AOP); joint point model; source code protection

1 概述

随着 Java 语言的普及和应用，越来越多的软件使用 Java 进行实现。同时，如何保护软件的知识产权，即保护源代码不被泄露成为一个亟需解决的问题。但由于 Java 语言的灵活性，其源代码的保护变得十分困难。借助于反编译可以很容易地将源代码从其二进制类文件中提取出。所以，任何一个恶意用户都可以利用反编译工具，如 Jad，对软件进行逆向工程攻击。例如，破解商业软件中授权保护的程序段进而随意分发未经授权的拷贝。

2 主流的保护技术

当前，针对 Java 语言的源代码保护主要有代码混淆和类文件加密 2 种。

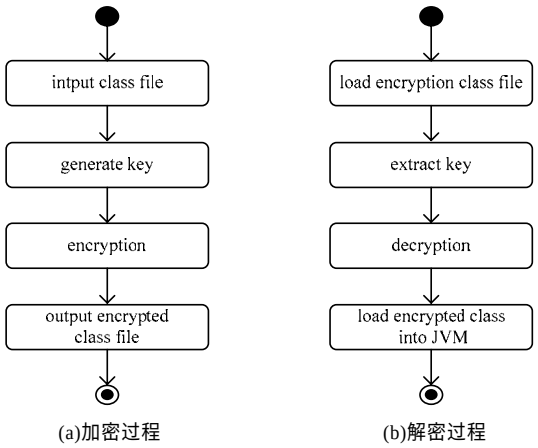
2.1 代码混淆[1]

代码混淆包括名称混淆、控制流混淆、字符串加密混淆等若干技术。名称混淆是用一些无意义或者难以阅读的字符串替代程序中出现的变量名、类名、方法名等，几乎所有的混淆器都支持该功能。有的混淆器使用自定义的标识符替代 Java 语言保留的标识符，使反编译后的程序无法重新编译。更高级的混淆器使用基于控制流的混淆，通过对二进制的类文件进行精心的处理，使反编译后程序的控制流程很难被跟踪和控制。对于程序中重要的字符串信息，有的混淆器可以对其进行加密处理，并在运行时动态解密。

通过混淆可以降低反编译后代码的可读性，甚至无法重新编译，从而有效地阻止了逆向工程等攻击。但最新研究结果表明，使用面向方面编程(Aspect-Oriented Programming, AOP)技术的连接点模型和字节码导入方法可以使上述混淆失效，这种攻击甚至不需要对软件实施反编译和重新编译。另一方面，对于那些经验丰富的攻击者，混淆后的代码并不是没有任何破绽可寻。

2.2 类文件加密

类文件加密方案采用加密整个 class 文件的方法来保护 Java 源代码。当 JVM 需要载入这些加密后的类时，可以选择通过自定制 ClassLoader 技术或 JVMTI 接口技术，先解密类文件，然后载入 JVM 中。图 1 描述了类文件的加密与解密过程。需要注意的是，加密的对象是整个类文件，而解密过程只发生在 JVM 加载 class 文件时。从图 1 可以看到，在加解密过程中，有 2 个重要的过程：密钥生成与密钥提取。一旦这 2 个过程被攻击者破解，所用的保护方案就会随之失效。



(a)加密过程 (b)解密过程

图 1 类文件的加密与解密过程

作者简介：李媛媛(1980－)，女，讲师、硕士，主研方向：网络安全
收稿日期：2009-11-15 **E-mail**：li_yuanyuan2008@yahoo.com.cn

2.2.1 定制 ClassLoader 技术

JVM 每次装入类文件时都需要 ClassLoader 对象，这个对象负责把新的类装入正在运行的 JVM。JVM 给 ClassLoader 一个包含待装入类(比如 java.lang.Object)名字的字符串，然后由 ClassLoader 负责找到类文件，装入原始数据，并把它转换成一个 Class 对象。所以，可以通过定制 ClassLoader，使其在类文件装入时进行解密。这也是该技术的核心。但是文献[2]给出了针对这种技术的攻击方法。

2.2.2 JVMTI 接口技术

JVMTI 是一套编程接口，常用于实现开发工具和监视工具。它提供了一种监视 JVM 状态以及控制 JVM 执行的方法。

JVMTI 为那些访问 JVM 状态的工具提供了一套完整的接口，其中包括调试、跟踪、线程分析及覆盖分析等。可以使用其中的 ClassFileLoadHook 事件对类文件进行解密。

当一个加密的类文件被载入 JVM 中时，其二进制字节会发送给代理，即 JVMTI 的客户端。然后 ClassFileLoadHook 接管工作，这些工作可以自定义。本文使用 JNI 技术[3]实现 ClassFileLoadHook 接口，在其中完成对密文的解密，然后把控制权交给 JVM，由它装载解密后的类文件。

使用纯 JNI 接口技术的方法可以达到同样的效果，那时可以用 C 语言或其他语言调用本地接口完成类文件的加解密工作。文献[4]对这种方法进行了详细的描述。事实上，JVMTI 接口是对这种方法的一种规范、封装和扩展。

2.2.3 类文件加密技术存在的问题

因为定制 ClassLoader 技术存在缺陷，所以使用 JVMTI 接口加密方式对整个 Class 字节码进行加密。在桌面应用中其运行良好。但如果对 J2EE 应用进行加密，就会产生 JSP 页面的编译问题以及部署过程中的类格式检测问题。JSP 页面的编译过程如图 2 所示，整个编译过程大致可分为 5 步：

(1)客户端向服务器端发送 JSP 页面请求。

(2)服务器端根据 JSP 页面生成对应的 Servlet 源文件。

(3)服务器端编译 Servlet 源文件。

(4)服务器端执行 Servlet 字节码。

(5)服务器端向客户端返回执行结果并以 HTML 页面形式展示给用户。

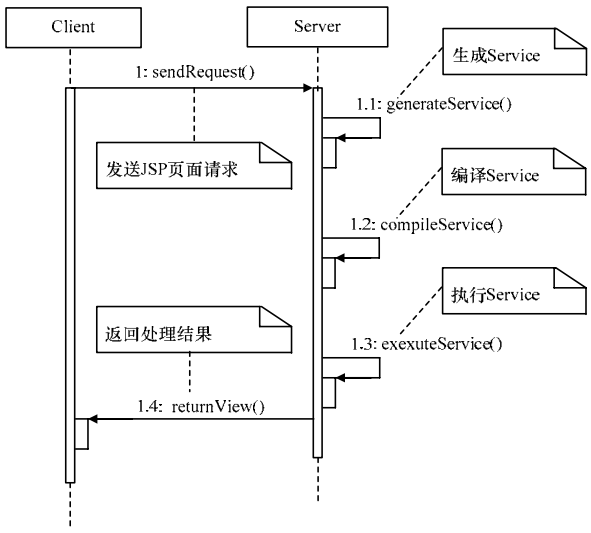


图 2 JSP 页面的编译过程

由此可见，编译问题的产生是因为在 JSP 页面中引用了组件中的类，而此时加密使得字节码文件的格式被破坏了，所以在编译 Servlet 源文件时类加载器无法找到相应的字节码文件。

类格式检测问题的表现形式因服务器的不同而不同，如 JBOSS 服务器会抛出 NonClassFormatException，而 Weblogic 服务器会抛出 ClassNotFoundException，其原因是加密后的字节码文件的格式被破坏了。

因此，导致这 2 个问题的根本原因都在于字节码文件的格式被破坏。本文借助 AOP 来解决这 2 个问题。

3 AOP[5]的引入

3.1 概述

AOP 的主要目的是模块化那些横切软件组件的关注点，即横切关注点问题。在 AOP 中，一个程序是由许多功能模块和一些方面模块组成的，后者封装了横切的关注点。一个方面由两部分组成：(1)切入点(pointcut)，定义了程序运行中的一些点，并指定了何时何地在这些运行点上对其他模块进行横切。(2)通知(advice)，指定了程序执行到切入点时所要执行的代码片段。换句话说，在程序的执行过程中，通知被织入(weave)功能模块中。AspectJ 是对 Java 语言的一种无缝扩展语言。借助于一个强大的连接点模型，它提供了丰富的切入点类型。其中，典型的连接点为方法调用、方法执行和字段访问。在 AspectJ 中，有 3 种类型的通知：before, after 和 around。前 2 种通知分别在被截获的方法之前和之后执行。相比之下，第 3 种通知更加精细，通过调用特殊的内置方法 proceed()继续执行被截获的方法，也可以简单地绕过该方法执行，甚至可以自定义程序段来替换被截获方法的行为。

AspectJ 的编译器为 ajc，它把 Java 编译器集成在内。它在编译过程中，先使用 Java 编译器生成字节码，然后把方面织入其中。从 1.1 版本开始，ajc 支持字节码的直接织入，而不需要提供目标源码。

3.2 解决方案

借助于 AspectJ 这一强大的工具，可以解决上文提到的 2 个问题。这 2 个问题本质上是由加密后的字节码文件格式被破坏造成的，所以，有必要在加密前对所有的类进行预处理，即制作一个与之对应的骨架类，其所处的包及类名不做改动，只保留所有的方法定义及必要的字段，而略去方法的具体实现，具体制作规则如图 3 所示。然后把加密后的密文作为一个字段插入该骨架类中。最后用这个骨架类替换原来的类。不过应当注意的是，在处理结束之后就无法对骨架类加密，另一方面，因为该骨架类仅是一个空壳文件，所以即使该类被反编译，也不会泄露程序的任何信息。并且该骨架类是格式良好的，可以实现 JSP 页面编译及软件的部署。

源类		骨架类
包名		保留
类名		保留
字段		保留必要字段
构造函数		设空
方法的返回类型	boolean	返回 false
	void	void
	int, float, long, double, short, byte, character	0
	其他类型	null

图 3 骨架类制作规则

例如有如下的类：

```
public class DemoClass {
    private int number;
    private String string;
    public DemoClass(int number) {
        this.number = number;
    }
    public int getNumber() {
        return number;
    }
    public String getString() {
        return string;
    }
    public void setNumber(int number) {
        this.number = number;
    }
}
```

那么根据上述制作规则，生成的骨架类如下：

```
public class DemoClass {
    private int number;
    private String string;
    public DemoClass(int number) {
    }
    public int getNumber() {
        return 0;
    }
    public String getString() {
        return null;
    }
    public void setNumber(int number) {
    }
}
```

3.3 骨架类的制作

借助 AspectJ 制作骨架类的步骤如下：

(1)编写一个类 Main，其方法 read(String)依次读入要保护应用的所有类。

(2)当调用 Main 实例的 read(String)方法时，AspectJ 会捕捉到方面 sketch 的切入点 stub，从而执行与之对应的 around 通知。

(3)在 around 通知捕获到 read 方法的参数后，便将该参数传递给 make 方法。

make 方法是骨架类的生成程序。它先根据读入的类文件的路径生成该类的一个 Class 对象，然后获取其所有的方法及构造函数。再按图 3 中的规则对其进行改造，生成源类的骨架，其中略去了类中所有的方法实现。最后 make 方法把加密后的字节码文件的密文作为一个字段写入该骨架类中，输出该骨架类并替换源字节码文件。

以下代码片段显示了上述思想：

```
public aspect sketch {
  …
  pointcut stub(String path) : call(void read(String))
&&args(path)&&target(Main);
    void around(String path) : stub(path) {
        make(path);
    }
    public void make(String path) {
      …
  //生成对应的 Class 对象
  Class cls = getClass(path);
  //获取类的方法
  Method[] m = cls.getDeclaredMethods();
  //获取类的构造函数
  Constructor[] con = cls.getDeclared
  Constructors();
  //处理构造函数
  dealConstructor(m);
  //处理方法
  dealMethod();
  //以字段方式插入到 Class 中
  insert();
  //输出骨架类并替换源 Class 文件
  write();
  …
```

其中，make 方法需要借助 Java 语言的反射机制或第三方软件，如 javassist。由于其实现较为容易，且与本文内容不太相关，因此略去其完整实现，仅以伪代码表示。

这样，借助于生成的骨架类就能解决上文提到的 2 个问题，并扩大了原有方案的适用范围，增强了对 J2EE 源代码的保护。

4 结束语

本文介绍了 Java 源代码保护的研究现状，指出各种方法的不足，并得出一个初步结论：采用 JVMTI 接口的加密技术相比之下更具优势。同时给出该技术的局限性，即无法有效地保护 J2EE 应用。然后借助于 AOP 的思想来解决当前加密技术在保护 J2EE 应用时遇到的 2 个问题。其核心思想是，制作一个骨架类，然后将加密后的字节码文件嵌入其中，最后用该骨架类覆盖源字节码文件，不仅解决了上述 2 个问题，还把当前加密技术的应用领域扩展到 J2EE 的应用上。

参考文献

[1] Low D. Protecting Java Code via Code Obfuscation[J]. ACM Crossroads, 1998, 4(3): 21-23.

[2] Roubtsov V. Cracking Java Byte-code Encryption[Z]. [2009-02-11]. http://www.javaworld.com/javaqa/2003-05/01-qa-0509-jcrypt.html.

[3] 李亚东, 夏雨佳, 席裕庚. 基于 JNI 的跨平台软件设计[J]. 计算机工程, 2000, 26(9): 87-89.

[4] 鲍福良, 徐 洁, 方志刚. 改进的 Java 类文件保护方法[J]. 计算机工程, 2009, 35(1): 93-94.

[5] Kiczales G, Lamping J, Mendhekar A, et al. Aspect-Oriented Programming[C]//Proceedings of the European Conference on Object-Oriented Programming. [S. l.]: Springer-Verlag, 1997.

编辑 张 帆