ClassLoader 加密技术改进研究

徐首泽[1]，金瓯[1,2]，贺建飚[1,2]
[1]中南大学信息科学与工程学院，湖南长沙(410082)
[2]湖南省金融货币识别与自助服务平台工程技术研究中心，湖南长沙(410082)
E-mail：xushouze2006@163.com

摘　要：ClassLoader 加密技术是 Java 当中用的比较广泛的代码保护技术，本文分析了 ClassLoader 加密技术的原理，发现并分析了现有方法存在的漏洞，同时根据这个漏洞的特点对原有方法进行了的改进。

关键字：类加载器；Java 本地接口；混淆

1. 引言

Java 源代码编译后生成的不是真正的二进制文件，而是一种有自己格式的中间代码，这种代码极易被反编译，而且反编译后的代码和源代码几乎没有差别。因此一些重要的算法或是专利会泄漏出去。近年来，已经有许多公司和开发人员对 Java 类文件和虚拟机进行了深入的分析，产生了一些 Java 类文件的保护方法，如代码隔离技术、本地编译技术、代码混淆技术、ClassLoader(类加载器)加密技术以及数字水印技术等保护方法[1]，但是都有各自的局限性。目前使用比较多的是 ClassLoader 加密技术，然而 ClassLoader 加密技术存在一个致命的漏洞，程序员可以绕过加密的 ClassLoader 和加密的 Java 类文件直接得到解密后的 Java 类文件，使加密完全失去其作用。本文分析这个漏洞存在的原因，同时提出了自己的改进方法。

2. ClassLoader 加密原理

Java 类文件加密技术是借助于 Java 中的类加载机制[2]。Java 虚拟机通过一个称为 ClassLoader 的对象来装载类文件的字节码，它根据类名在 classpath 中找到该类的文件，读取文件，并把它转换成一个 Class 对象。ClassLoader 可以由 Java 程序员自己定制，并借助于 ClassLoader 对需要保护的 Java 类文件进行加密。加密的方法可以由用户自己定，如 PGP，RSA， MD5 等，加密后的 Java 类文件就无法被反编译，起到了保护的作用。加密后的文件在被 ClassLoader 读取之后，先对其进行解密，然后才能转换成一个 Class 对象，否则虚拟机无法将其转换成 Class 对象。

其具体步骤大体分三步：

- 将需要保护的 class 文件加密，加密算法可以任意选择。
- 定制自己的 ClassLoader。自己定制的 ClassLoader 需要继承自 java.lang.ClassLoader 类，然后覆盖其 findClass 方法或 loadClass 方法即可。在这两个方法中需要从 classpath 中读取已经加密文件，然后对这些加密的文件进行解密，最后调用父类中的 defineClass 方法得到一个 Class 对象，这是最为关键的一步。主要步骤如下：

```
protected Class findClass(String name) throws ClassNotFoundException {
    ……
    byte[] b = null，classBytes=null;
    b = loadClassData(name);                    //从环境变量中读取要加载的经过加密的class文件数据
    classBytes = decrypt(classBytes);           //解密
    ……
    return defineClass(name, classBytes, 0, classBytes.length); //关键的一步，得到需要的Class对象
}
```

- 最后在自己定应用程序中使用自己的 ClassLoader 加载器。如：

```
MyClassLoader    myClassLoader = new MyClassLoader(null); //创建自己的类加载器
Class myClass = myClassLoader.loadClass("mycrptClass");       //加载经过加密的类
```

在上面自定义的 ClassLoader 中包含有关键信息 findClass()及 decrypt ()，尤其是 decrypt()函数，包含了解密信息。由于自定义的 ClassLoader 本身不是被加密的，也是用 Java 写的，因此它可能成为最先攻击的目标。如果相关的解密算法被攻克，那么被加密的类也很容易被解密了。因此，必须想办法保护才行。Java 类文件保护分析与研究[3]中使用 Obfuscation 方法，即代码混淆[4,5]的办法来保护，也有的地方用 JNI[6]来实现 decrypt()函数的方法。

3. Classloader 加密的漏洞

以上方法对 Java 类文件进行了加密，若自定义的 classloader 是绝对安全的——即无法通过反编译[7] Classloader 类文件得到解密算法，那么是否被加密的类文件就无法被反编译呢？事实上正好相反，程序员可以跳过自定义的 Classloader，无需知道解密算法就可以轻而易举的获得解密后的 Java 类文件，得到了类文件后借助于普通的反编译工具就可以得到被加密类的源代码。

破解方法：

- 首先下载 JDK 的源代码，找到 java. lang. ClassLoader 类的源文件。
- 改写 defineClass(String name, byte[] b, int off, int len, ProtectionDomain protectionDomain)方法，在最后(return c 之前)加入类似下面功能的代码：
-

```
if (isAncestor(getSystemClassLoader().getParent())) {
        final File parentDir = new File("c:/TEMP/classes/");
        File dump = new File(parentDir, name.replace('.', File.separatorChar)
                + getClass().getName()+".class");
        dump.getParentFile().mkdirs();      FileOutputStream out = null;
        try {
            out = new FileOutputStream(dump);      out.write(b, off, len);
        } catch (IOException ioe) {//异常处理
        } finally { //释放资源
        }
}
```

- 重新编译 java.lang.ClassLoader 源文件，并用新生成的 ClassLoader.class 类文件替换掉 JAVA_HOME/jre/lib/rt.jar 中的类文件。
- 重新运行原先的含有加密文件的程序，则在 C:/TEMP/classes/目录中会生成使用自定义 Classloader 加载的已经解密的 Java 类文件，使用普通反编译工具即可得到加密过的源代码。

漏洞存在原因：

Java 虚拟机(JVM)定义了三种类加载器，分别为：启动（Bootstrap）类加载器，标准扩展（Extension）类加载器，系统（System）类加载器。启动类加载器是用本地代码 (C/C++语言实现) 实现的，它负责将 JAVA_HOME/jre/lib 下面的类库加载到内存中。这些类库是 Java 运行时所依赖的类库，其中最重要的一个类库是 rt.jar。他几乎包括了 Java 运行时的核心的类库，例如前一节中讲到的类 java.lang.ClassLoader 类就在其中。查看此类的源代码可以发现，他总共定义了四个可被子类调用的 defineClass()函数：

```
protected final Class    defineClass(byte[], int, int)
protected final Class    defineClass(String, byte[], int, int)
protected final Class    defineClass(String, byte[], int, int, ProtectionDomain)
protected final Class    defineClass(String, ByteBuffer, ProtectionDomain)
```

再进一步分析可知，这四个函数最终调用的都是：

protected final Class defineClass(String, byte[], int, int, ProtectionDomain)

这个函数调用本地方法写的 defineClass0 函数得到一个 Class 实例。此函数中的第二个参数，是一个字节数组，这个字节数组就是 Java 类文件的数据。自定义的类加载器中先将加密的类文件解密然后再调用这个函数获得一个 Class 实例。所以调用此函数是的第二个参数一定是一个非加密的字节码数据，把这个字节码数据重新写入到其他文件就得到了字节码的内容。在示例代码中，把这些文件存放到了 C:\EMP\classes\目录中，这样只要反编译这些类，加密的类的源代码就其而易举的得到了。

4. ClassLoder 加密方法的改进

从上面的分析得知，加密的 Class 类文件内容还是很容易得到。究其原因是在自定义的 ClassLoder 中调用了父类中的 defineClass()函数，由于 Java 的开放性，很容易对其进行修改。这表明，只要不直接调用超类的 defineClass()函数就不是那么容易得到解密后的 Java 类文件了，有两种方法可以对其进行改进。

4.1 采用 JNI(Java 本地接口)

采用本地方法实现 loadClass (String name)函数，不调用 java.lang. ClassLoder.defineClass()函数，修改 java.lang. ClassLoder.defineClass()方法就会无效，起到了保护作用。

Java 本地接口(JNI)中提供一个方法用于将字节码转换成 Class 对象的函数 DefineClass(), 此函数是用 C 语言实现的，不易被反编译。所以在自定义的 ClassLoder 中用本地方法重新实现 loadClass 方法：

public **native** Class loadClass(String name)throws ClassNotFoundException；

将用本地方法实现的函数制作成动态链接库，在调用前使用 System.load ()或 loadLibrary()函数将其载入。

4.2 改写加载器 java.exe

当要运行 Java 程序的时候是通过 java.exe 这个工具(在 windows 平台上)来启动 JVM 的，分析 java.exe 的源代码发现他首先初始化 JVM 的运行环境，启动 JVM，然后从运行环境中读取含有主函数(main)的类，并通过 JNI 接口将这个类转换成 java 对象传给 java 虚拟机运行。再进一步可以发现源代码中是通过 LoadClass(JNIEnv *env, char *name)函数来加载这个类的，此函数中调用 FindClass(JNIEnv *env, char *name)来得到一个实例。如果我们要装载已加密过的 JAVA 程序，显然直接调用 FindClass 函数是不行的，那么，我们可以读取加密的 Class 文件，然后通过 env 变量调用 defineClass 方法获得实例。

```
static jclass LoadClass(JNIEnv *env, char *name){
    FILE *in;   long   length, i;   char *cc;   int   x;
     if(strcmp(buf,"MyLoader")==0) {      /*如果装载的类是加密的 MyLoader*/
          if (GetApplicationHome(javapath, sizeof(javapath))){
               sprintf(javaloader, "%s\\MyLoader.class", javapath);
          }
          if ((in = fopen(javaloader, "rb")) == NULL)
          {
               fprintf(stderr, "Cannot open input file.\n");
               return (jclass)0x0f;
          }
          /*读出加密的 class 文件*/
```

```
            fseek(in, 0L, SEEK_END);
            length = ftell(in);
            fseek(in, 0, SEEK_SET);
            cc = MemAlloc(length);
            fread((void*)cc,length,1,in);
            fclose(in);
            /*解密算法*/
            ……
            /*将解密后的 class 字节码转换成 class*/
            cls = (*env)->DefineClass(env, buf, 0, cc, length-1);
           free(cc);
      }else{
       cls = (*env)->FindClass(env, buf);
     }
      ……
     }
         return cls;
}
```

5. 结束语

本文分析了ClassLoader加密技术的原理，针对此方法存在的漏洞，提出了两个改进的方法。虽然这两个改进方法对于java的可移植性产生了一定的影响，但影响并不是很大，只要稍微修改一点就可以移植，对于需要保密的程序具有可操作性。这两种方法在Windows 平台上都经过测试，效果良好，起到了很好的保护作用。

参考文献

[1] 鲍福良,彭俊艳,方志刚.Java 类文件保护方法综述[J].计算机系统应用.2007 年第 6 期.
[2] By Li Gong, Gary Ellison, Mary Dageforde.Inside Java 2 Platform Security: Architecture, API Design, and Implementation, Second Edition[M].Addison Wesley.
[3] 汪霞,刘岚.java 类文件保护分析与研究[EB/OL].http://www.paper.edu.cn/paper.php.
[4] Michael R. Batchelder.Java bytecode obfuscation [M] .School of Computer Science McGill University, Montr´eal.January 2007.
[5] Michael Batchelder, Laurie Hendren.Obfuscating Java: the most pain for the least gain.Sable Technical Report [J].2006.5.
[6] Sheng Liang. The Java Native interface: programmer's guide and specification [M].Addison Wesley.
[7] Behrens B, Levary R. Practical Legal Aspects of Software Reverse Engineering[J].Communications of the ACM. 1998, 41(2): 27-29.

Study of improving the method of ClassLoader encryption

Xu Shouze, Jin Ou, He Jianbiao
School of Information science & Engineering, Center South University, Changsha, Hunan (410083)

Abstract
The Java language is an object-oriented and middle-code language so that the software wrote by Java is easily being pirated. This article analysis one of methods to protecting the java bytecode—ClassLoader encryption method and find out that the method is deficient. Finally, designs two effective and practical methods to protect Java software from being pirated and decompiles.
Keywords: ClassLoader; JNI; Obfuscation

作者简介：
徐首泽：中南大学信息科学与工程学院在读硕士；
金瓯：中南大学信息科学与工程学院教授；
贺建飚：中南大学信息科学与工程学院副教授。