如何有效防止Java程序源码被人偷窥？[4] PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/244/2021_2022__E5_A6_82_E4_BD_95_E6_9C_89_E6_c104_244673.htm 四、应用实例 前面介绍了如何加密和解密数据。要部署一个经过加密的应用，步骤如下： 步骤1：创建应用。我们的例子包含一个App主类，两个辅助类(分别称为Foo和Bar)。这个应用没有什么实际功用，但只要我们能够加密这个应用，加密其他应用也就不在话下。 步骤2：生成一个安全密匙。在命令行，利用GenerateKey工具(参见GenerateKey.java)把密匙写入一个文件：% java GenerateKey key.data 步骤3：加密应用。在命令行，利用EncryptClasses工具(参见EncryptClasses.java)加密应用的类：% java EncryptClasses key.data App.class Foo.class Bar.class 该命令把每一个.class文件替换成它们各自的加密版本。 步骤4：运行经过加密的应用。用户通过一个DecryptStart程序运行经过加密的应用。DecryptStart程序如Listing 6所示。 【Listing 6：DecryptStart.java，启动被加密应用的程序】以下是引用片段：import java.io.*. import java.security.*. import java.lang.reflect.*. import javax.crypto.*. import javax.crypto.spec.*. public class DecryptStart extends ClassLoader { // 这些对象在构造函数中设置， // 以后loadClass()方法将利用它们解密类 private SecretKey key. private Cipher cipher. // 构造函数：设置解密所需要的对象 public DecryptStart(SecretKey key) throws GeneralSecurityException, IOException { this.key = key. String algorithm = "DES". SecureRandom sr = new SecureRandom(). System.err.println("[DecryptStart: creating cipher]"). cipher =

Cipher.getInstance(algorithm). cipher.init(Cipher.DECRYPT_MODE, key, sr). } // main过程：我们要在这里读入密匙，创建DecryptStart的 // 实例，它就是我们的定制ClassLoader。 // 设置好ClassLoader以后，我们用它装入应用实例， // 最后，我们通过Java Reflection API调用应用实例的main方法 static public void main(String args[]) throws Exception { String keyFilename = args[0]. String appName = args[1]. // 这些是传递给应用本身的参数 String realArgs[] = new String[args.length-2]. System.arraycopy(args, 2, realArgs, 0, args.length-2). // 读取密匙 System.err.println("[DecryptStart: reading key]"). byte rawKey[] = Util.readFile(keyFilename). DESKeySpec dks = new DESKeySpec(rawKey). SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES"). SecretKey key = keyFactory.generateSecret(dks). // 创建解密的ClassLoader DecryptStart dr = new DecryptStart(key). // 创建应用主类的一个实例 // 通过ClassLoader装入它 System.err.println("[DecryptStart: loading " appName "]"). Class clasz = dr.loadClass(appName). // 最后，通过Reflection API调用应用实例 // 的main()方法 // 获取一个对main()的引用 String proto[] = new String[1]. Class mainArgs[] = { (new String[1]).getClass() }. Method main = clasz.getMethod("main", mainArgs). // 创建一个包含main()方法参数的数组 Object argsArray[] = { realArgs }. System.err.println("[DecryptStart: running " appName ".main()]"). // 调用main() main.invoke(null, argsArray). } public Class loadClass(String name, boolean resolve) throws ClassNotFoundException { try { // 我们要创建的Class对象 Class clasz = null. // 必需的步骤1：如果

类已经在系统缓冲之中 // 我们不必再次装入它 clasz = findLoadedClass(name). if (clasz != null) return clasz. // 下面是定制部分 try { // 读取经过加密的类文件 byte classData[] = Util.readFile(name ".class"). if (classData != null) { // 解密... byte decryptedClassData[] = cipher.doFinal(classData). // ... 再把它转换成一个类 clasz = defineClass(name, decryptedClassData, 0, decryptedClassData.length). System.err.println("[DecryptStart: decrypting class " name "]"). } } catch(FileNotFoundException fnfe) // 必需的步骤2：如果上面没有成功 // 我们尝试用默认的ClassLoader装入它 if (clasz == null) clasz = findSystemClass(name). // 必需的步骤3：如有必要，则装入相关的类 if (resolve amp. clasz != null) resolveClass(clasz). // 把类返回给调用者 return clasz. } catch(IOException ie) { throw new ClassNotFoundException(ie.toString()). } catch(GeneralSecurityException gse) { throw new ClassNotFoundException(gse.toString()). } } } 对于未经加密的应用，正常执行方式如下： % java App arg0 arg1 arg2 对于经过加密的应用，则相应的运行方式为： % java DecryptStart key.data App arg0 arg1 arg2 DecryptStart有两个目的。一个DecryptStart的实例就是一个实施即时解密操作的定制ClassLoader.同时，DecryptStart还包含一个main过程，它创建解密器实例并用它装入和运行应用。示例应用App的代码包含在App.java、Foo.java和Bar.java内。Util.java是一个文件I/O工具，本文示例多处用到了它。完整的代码请从本文最后下载。 五、注意事项 我们看到，要在不修改源代码的情况下加密一个Java应用是很容易的。不过，世上没有完全安全的系统

。本文的加密方式提供了一定程度的源代码保护，但对某些攻击来说它是脆弱的。 虽然应用本身经过了加密，但启动程序DecryptStart没有加密。攻击者可以反编译启动程序并修改它，把解密后的类文件保存到磁盘。降低这种风险的办法之一是对启动程序进行高质量的模糊处理。或者，启动程序也可以采用直接编译成机器语言的代码，使得启动程序具有传统执行文件格式的安全性。 另外还要记住的是，大多数JVM本身并不安全。狡猾的黑客可能会修改JVM，从ClassLoader之外获取解密后的代码并保存到磁盘，从而绕过本文的加密技术。Java没有为此提供真正有效的补救措施。 不过应该指出的是，所有这些可能的攻击都有一个前提，这就是攻击者可以得到密匙。如果没有密匙，应用的安全性就完全取决于加密算法的安全性。虽然这种保护代码的方法称不上十全十美，但它仍不失为一种保护知识产权和敏感用户数据的有效方案。