C#项目如何做好源码保护？

原文

在安全领域，源代码安全一直是所有软件厂商重视的安全问题，如果源代码保护不好，就很容易被破解软件程序和算法实现，特别是金融行业，客户端软件源代码保护极为重要，本文循序渐进的讲述了C#项目中源代码保护机制。

反编译概述

编译是利用编译程序从源语言编写的源程序产生目标程序的过程，是把高级语言变成计算机可以识别的2进制语言，计算机只认识1和0，编译程序把人们熟悉的语言换成2进制程序。编译型的计算机语言有C/C++，java，C#，Pascal/Delphi等,C++/Delphi直接将源语言编译成特定的处理器硬件平台对应的指令代码，这种处理方式优点是程序执行速度快，缺点是跨平台特性差。java和C#比较特殊，java/C#编译阶段先将源语言编译成字节码/中间语言，而不是编译成与某个特定的处理器硬件平台对应的指令代码，字节码/中间语言是与运行平台无关的，在执行阶段由JIT Compiler(Just-in-time Compiler)即时编译器编译成特定平台指令的程序并执行，这种处理方式优点是跨平台特性强，但程序执行速度慢。

反编译是编译的逆过程，即把目标程序变成高级语言源代码。由于C++/Delphi语言生成的可执行程序是特定平台机器指令代码，反编译只能将可执行程序转为汇编代码，汇编语言就很难转为高级源语言，所以C++/Delphi的反编译程序(如eXeScope,DEDE等)不是很成熟。而java/C#语言编译的是中间语言，没有直接生成特定平台的机器代码，由于跨平台的需要，字节码/中间语言中包括了很多源代码信息，如变量名、方法名，并且通过这些名称来访问变量和方法，这些符号带有许多语义信息，很容易被反编译成源代码。针对java的反编译工具有JAD，JD等，C#的反编译工具有Reflector，ILSpy等。

程序源代码安全

在软件系统中程序分布在客户端和服务端，服务端的可执行程序对外部来讲是不可获取的，所以服务端的安全不用考虑源代码安全，而客户端的程序是安装客户端电脑上的，能获取到可执行程序，如果客户端程序是用C++/Delphi开发的，只能反编译出汇编语言，所以很难获取程序和算法实现，如果客户端程序是用java/C#语言开发的，就很容易反编译源程序，从而破解程序和算法实现，所以安全问题随之而来。Java一般用于服务端程序开发，很少用于桌面客户端程序开发，C#语言广泛的用于客户端程序开发，所以C#客户端程序的保护尤为重要。

对于java/C#语言产生的程序保护机制主要是混淆/加壳。混淆就是编译的程序进行重新组织和处理，使得处理后的代码与处理前代码完成相同的功能，而混淆后的代码很难被反编译，即使反编译成功也很难得出程序的真正语义。混淆将代码中的所有变量、函数、类的名称变为简短的英文字母代号，在缺乏相应的函数名和程序注释的情况下，即使被反编译也难以阅读。加壳是利用特殊的算法，对EXE、DLL文件里的资源进行压缩、加密。这个压缩之后的文件，可以独立运行，解压过程完全隐蔽，都在内存中完成。针对java混淆/加壳工具有Obfuscator、Zelix KlassMaster、Cinnabar Canner等，针对C#混淆/加壳工具有Dotfuscator、.NET Reactor等。

持续集成概述

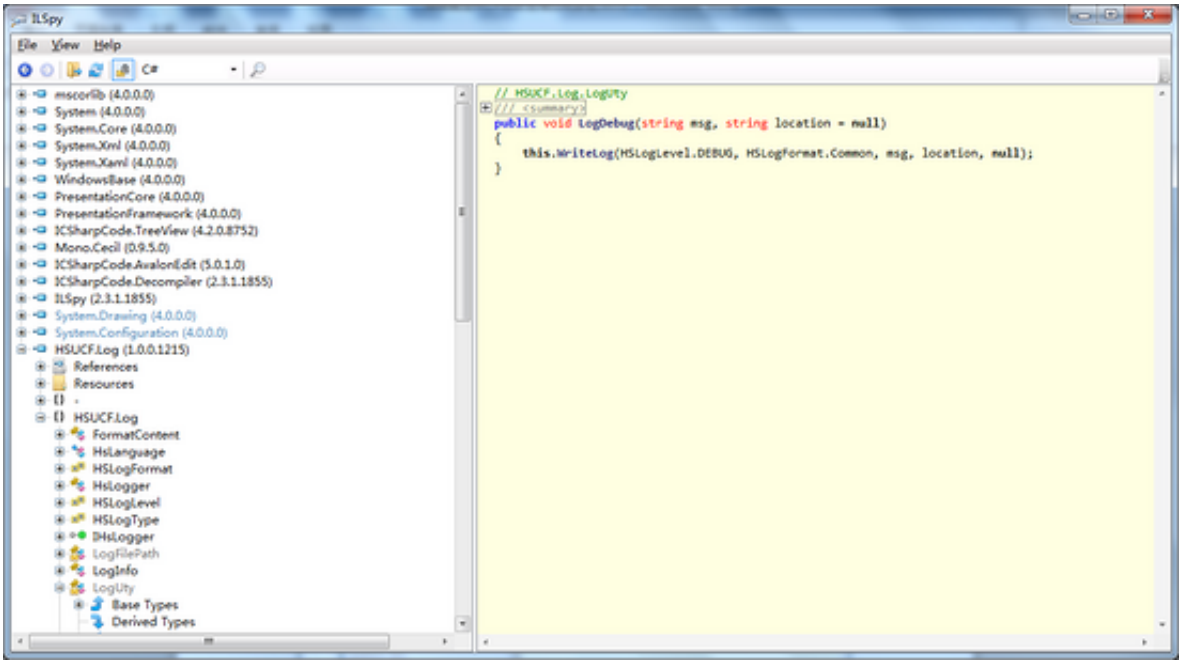
持续集成是一种软件开发实践，通过每个子项目每天至少集成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译，发布，自动化测试）来验证，从而尽早地发现集成错误。早集成，频繁的集成帮助项目在早期发现项目风险和质量问题，如果到后期才发现这些问题，解决问题代价很大，很有可能导致项目延期或者项目失败。所以很多大项目开发管理都用到持续集成，持续集成的项目有可能每天发出很多包。持续集成项目管理工具有Jenkins,Buildbot,Travis CI,Strider,Go等，一般持续集成工具都部署在Window系统上。

在集成过程中，发布运行在Windows上的C++/Delphi/C#程序可调用脚本直接编译出dll、exe等目标程序，运行在linux上的C++程序可先把代码拷贝到linux编译出so文件、可执行文件，再把编译好的so文件、可执行文件拷贝回持续集成发布包中，java语言开发的项目在集成中用maven或者ant项目管理直接编译。 如果持续集成的项目中有用到C#开发客户端程序的子项目，发出的包从源代码安全角度考虑必须混淆加壳后再发出。

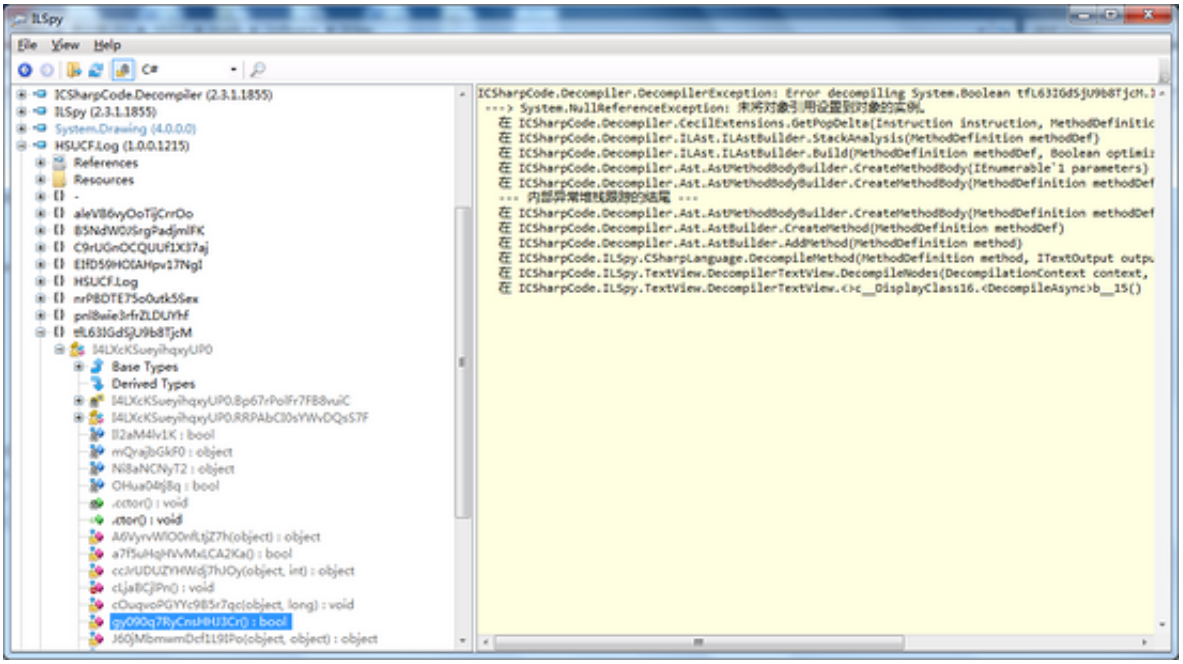
持续集成中集成混淆加壳发包

由于持续集成中发包都是使用脚本程序自动编译产生的，所以对C#项目混淆加壳工具的使用，最好要支持脚本调用，C#混淆加壳工具有很多，如Maxcodev、Dotfuscator、NET Reactor等。综合比较.NET Reactor界面操作方便，支持批处理脚本加壳，这在持续集成项目中自动编译自动混淆加壳很重要，所以本文以.NET Reactor工具作为混淆/加壳工具。

Dll混淆加壳前用ILSpy反编译如下图所示(所有源代码均能看到)：

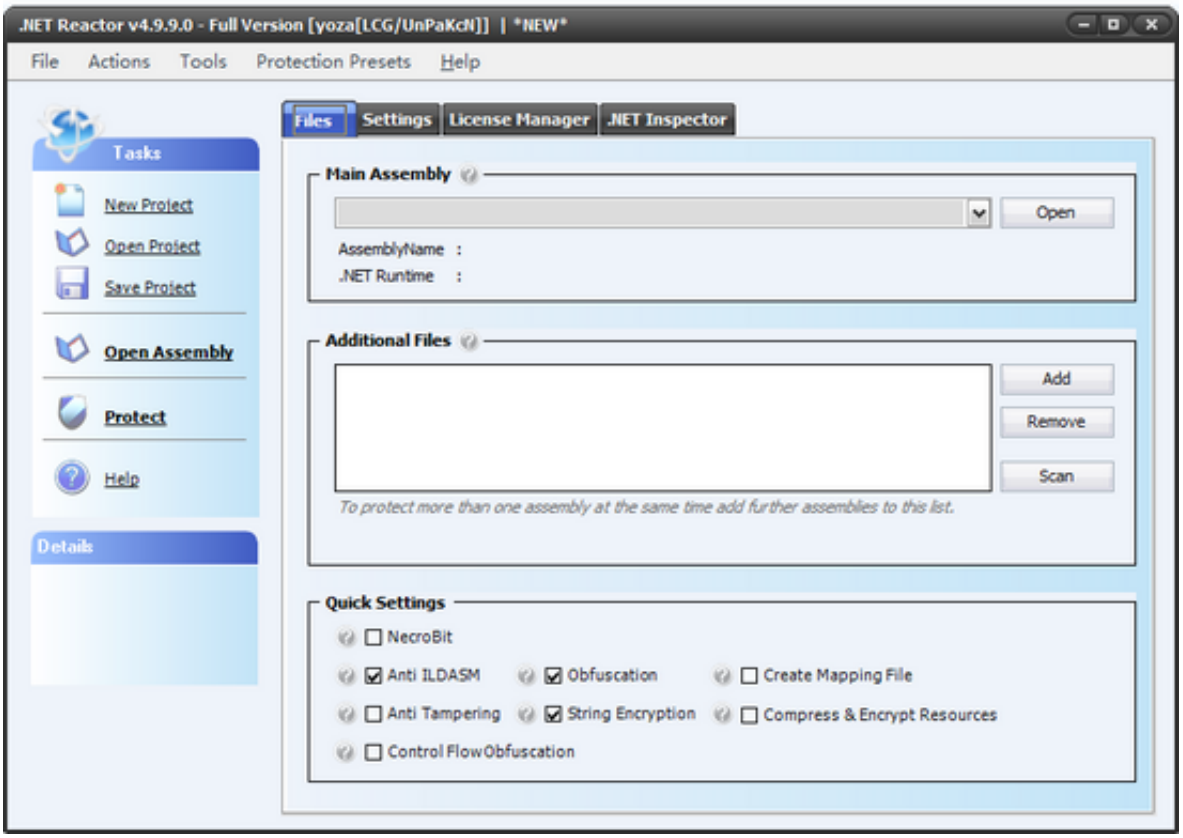


Dll混淆加壳后用ILSpy反编译如下图所示(反编译源代码出现异常)：

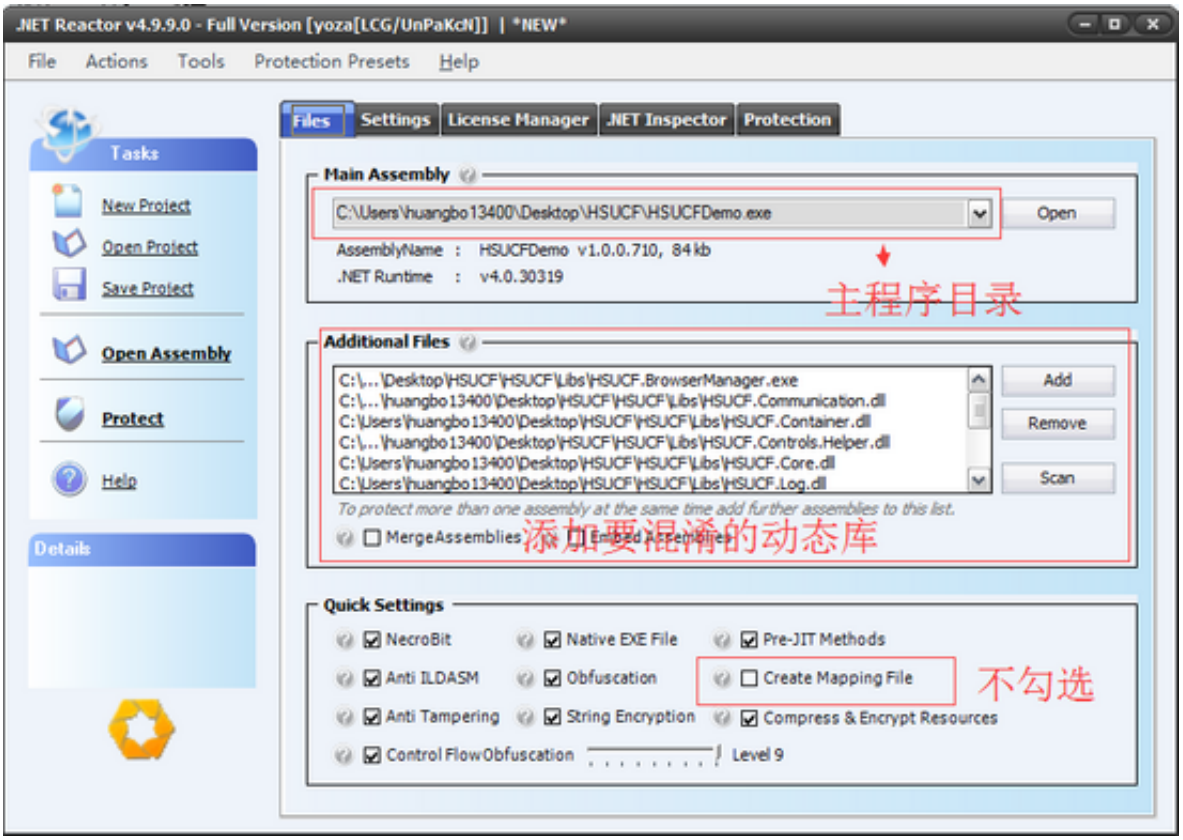


以下是在持续集成中嵌入混淆/加壳脚本步骤：

1、启动Reactor。

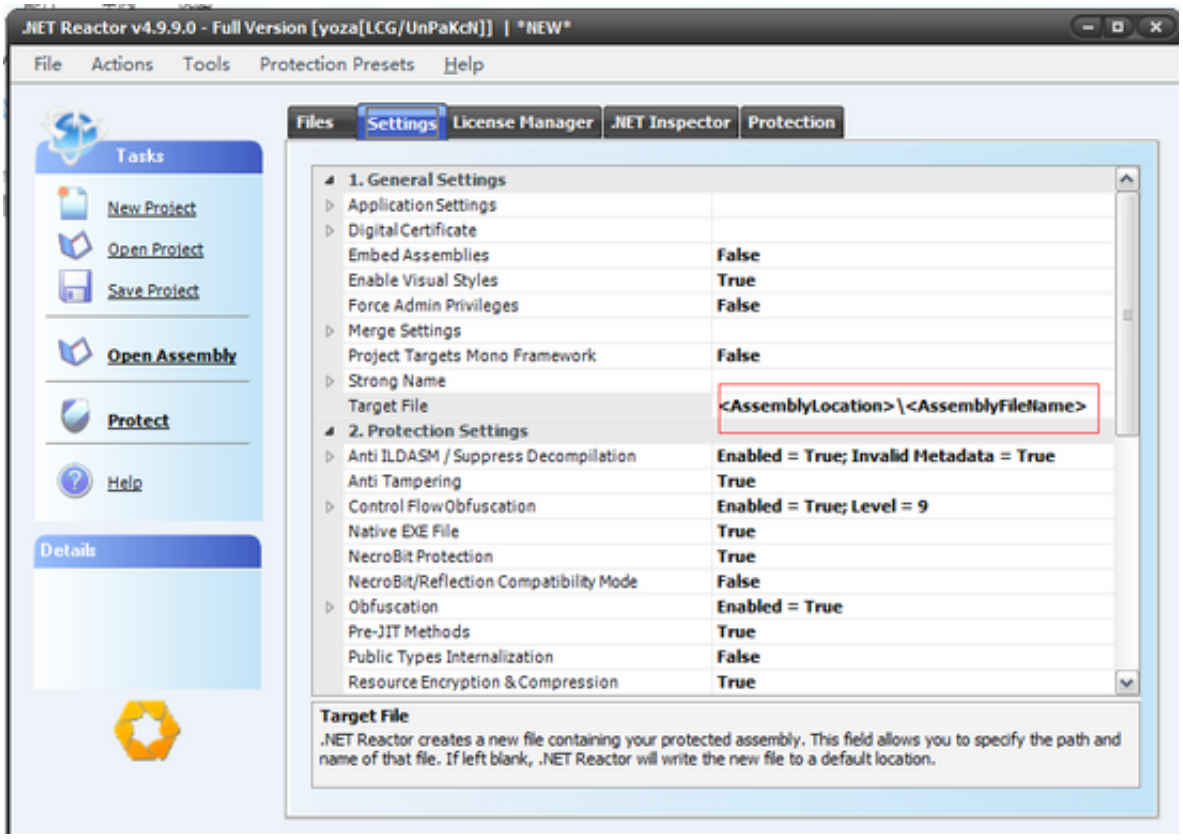


2、添加主程序，要混淆的动态库，勾选混淆设置（注意:不勾选映射文件）

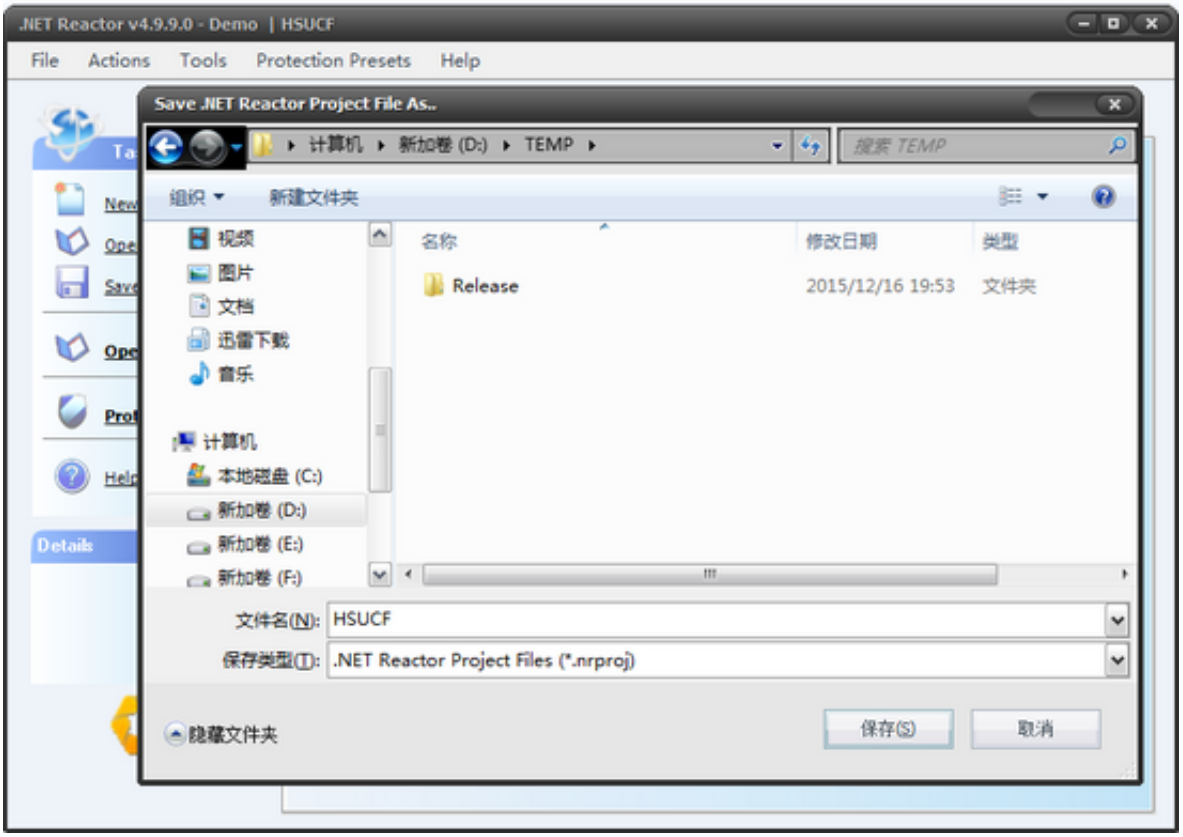


3、设置混淆后目标文件目录，不要使用默认设置，默认设置每个Dll都会在Dll目录下创建子目录存放混淆后的Dll，混淆后要逐个拷贝，

设置Target File为当前目录<AssemblyLocation>\<AssemblyFileName>，混淆后Dll覆盖混淆前的Dll。



4、保存工程文件File->Save Project As。



5、拷贝破解版混淆dot NET_Reactor.exe到工程文件目录，如果购买了正式版，安装后会自动把dotNET_Reactor安装目录添加到系统环境变量，就可以直接使用，不用拷贝破解版。

6、在Reactor工程 文件目录下创建批处理脚本文件Protect.bat。

```
@ECHO OFF
echo 开始混淆

rem 如果不使用 masterkey，使用下面这条命令，替换工程文件名
dotNET_Reactor -project "HSUCF.nrproj"

rem 如果要使用 masterkey，使用下面这条命令
rem dotNET_Reactor -project "HSUCF.nrproj" -masterkeyfile
"masterkey.mkey"

echo 混淆完成

pause
```

7、如果要使用 masterkey，在工程文件目录创建masterkey文本文件并把masterkey保存到文件，在批处理脚本中使用以下命令：

```
dotNET_Reactor -project "HSUCF.nrproj" -masterkeyfile
"masterkey.mkey"
```

8、执行bat批处理文件 就可以混淆加壳，但在持续集成中，要在编译脚本中嵌入执行Protect.bat批处理文件。

```
@echo off
set VS="%VS120COMNTOOLS:Tools\=IDE\devenv.exe%"

rem 持续集成中编译过程
echo 正在编译
del /S/Q/F compile.log 1>nul 2>nul
%VS% HSUCF_Demo.sln /rebuild "Debug|Any CPU" /out compile.log
IF %ERRORLEVEL% NEQ 0 GOTO ERROR
echo 编译完成

rem 持续集成中混淆过程
echo 开始混淆
rem 如果不使用masterkey，使用下面这条命令，替换工程文件名
dotNET_Reactor -project "HSUCF.nrproj"

rem 如果要使用masterkey，使用下面这条命令
rem  dotNET_Reactor  -project  "HSUCF.nrproj"  -masterkeyfile
"masterkey.mkey"
echo 混淆完成
:ERROR
echo 编译失败!

Pause
```

总结

本文从反编译原理出发，讲述了程序源代码的安全保护问题，以及在大型项目开发中持续集成自动调用混淆加壳程序的使用方法。