

# HOWTO: Maximize protection of your application against reverse engineering

Article ID: 000023

Last Revised On: 26-Dec-2005

The information in this article applies to:

- Excelsior JET 3.5 and above

## SUMMARY

The document explains how to use Excelsior JET most effectively for the protection of your applications against reverse engineering.

Project files and build/run scripts for this article may be found [here](#).

## MORE INFORMATION

There are a number of [Java decompilers](#) on the market that produce amazingly readable source code, thanks to the high-level nature of Java bytecode and presence of reflection information in class files. As a consequence, there are even more tools called *obfuscators*, which can rename classes, fields and methods consistently throughout your application and possibly change the control flow so as to fool or break the decompilers, while preserving the application's functionality.

An obfuscator can surely make the decompiled source somewhat less comprehensible, but control flow obfuscation negatively impacts performance and can make classes unverifiable.

With Excelsior JET, you ship your Java applications as optimized binary executables. If you also use a name obfuscator to rename classes and members, your application will be as hard to reverse engineer as an equivalent C++ program passed through an optimizing compiler.

## DETAILS

### Application domain

If all classes of your application are available to Excelsior JET and loaded by one of the default classloaders (system or application), it may be fully protected using Ahead-Of-Time (static) compilation.

In the general case, however, you may have to ship some of the original class files. In particular, Excelsior JET may not statically precompile classes that are loaded by a custom classloader except by fusing the JIT cache. And even if you ship a precompiled JIT cache, the original classes must still be available at run time to enable consistency checks.

As a result, if your application heavily relies on custom classloading, Excelsior JET will not be able to protect it against decompilation. One example of such application is Eclipse, which consists of a small launcher that loads numerous plugins using custom classloaders.

Another common case is applications using the Java Security API, which requires the original class files of security providers to be present at runtime. If you are using a third-party security provider other than those included in the Sun JRE, you have to ship its jar with your application.

## Project setup guidelines

Suppose you have already set up an Excelsior JET project file for your application. You can further improve its protection by taking the following measures:

1. Make sure to **compile ahead-of-time as many classes as possible**, so that you do not have to ship them with your application.

If a class is loaded dynamically, but is known statically, that is, it is available at compile time and will be loaded by one of the two default classloaders (system or application) at run time, you can safely precompile it and not include with your application. The only exception are security providers — the Java Security API requires their original jars to be present at run time.

2. If you have to include any classes in the bytecode form, **pack them into the resulting executable** so that they are not immediately visible.

Normally, you would only pack resource files, such as images or audio clips, from application jars. Excelsior JET does not modify the original jars but filters class files out during packing. To pack an entire jar, open your project in the JET Control Panel, go to the **Resources** page, select the jar file(s) containing classes you need to pack and select the **Pack entire jar/zip file** button.<sup>1</sup>

Check the status of other jars on the **Resources** page to make sure you do not unnecessarily pack classes.

3. Add the following options to the Excelsior JET project file for your application:

`+disableclassssaving`

By default, if the property `jet.jit.save.classes` is set, the JIT compiler flushes classes being loaded into its cache directory. This is required for subsequent JIT cache optimization. The `+disableclassssaving` option disables this behavior.

`+disableusagelist`

This option disables generation of the usage list file when the `jet.usage.list` property is set. The usage list contains names of classes and members being accessed dynamically and is only needed for JetPerfect Global Optimizer.

`+disablestacktrace`

With this option set during compilation, stack trace output will not work in the resulting executable even if the `jet.stack.trace` property is set.

**Note:** This option must be used with caution because your application may rely on stack trace information being available at run time.

## Name obfuscation

In order for the Reflection and JNI APIs to function properly, an executable generated by Excelsior JET must contain names of all compiled classes, methods and fields. If exposing these names is not desirable, you can replace them with meaningless character sequences like `a0`, `a1`, `a2`, etc. prior to native compilation using a third-party Java obfuscator.

Of course, you would have to manually disable renaming of classes and members that are accessed via reflection or JNI at run time. It is not possible to determine automatically whether a class or member name must not change in most cases.

---

<sup>1</sup>In Excelsior JET 4.0 use **Pack as a whole** radiobutton. In JET 3.7 and below use **Bind with classes** radiobutton.

**Warning:** Beside renaming, many obfuscators are capable of optimizing bytecode and/or obfuscating control flow. Use of these features may substantially reduce the strength of optimizations implemented by Excelsior JET AOT and JIT compilers or even force them to degrade to the non-optimizing mode. It is also not recommended to let your obfuscator remove unused classes, fields and methods. Ideally, you should set up your obfuscator so that it only changes class and member names and leaves the rest of a class file intact.

After name obfuscation, make sure your program works as expected on the Sun HotSpot JVM before compiling it with Excelsior JET.

We leave it up to you to decide whether to obfuscate control flow in class files that you need to ship with your application for reasons outlined in the section “Application domain” above.

## Sample obfuscation using ProGuard

You do not need to buy any expensive tool to rename classes and members. We have observed good results with the open source [ProGuard](#) obfuscator.

Below is a sample ProGuard configuration file for the SwingSet2 demo, which you may use as a starting point for obfuscating your application.

```
# Input/output files setup:
-injars      SwingSet2.jar
-outjars     ss2.jar
-libraryjars <java.home>/lib/rt.jar
-printmapping SwingSet2.map

# Most importantly, we need to disable bytecode optimization
# and removal of unused classes and members:
-dontoptimize
-dontshrink

# Now, we need ProGuard to preserve the entry point name:
-keep public class SwingSet2 {
    public static void main(java.lang.String[]);
}

# The remaining lines tell ProGuard to not rename
# classes that implement individual demos, as they
# are loaded using Class.forName().
-keep public class ButtonDemo
-keep public class ColorChooserDemo
-keep public class ComboBoxDemo
-keep public class FileChooserDemo
-keep public class HtmlDemo
-keep public class ListDemo
-keep public class OptionPaneDemo
-keep public class ProgressBarDemo
-keep public class ScrollPaneDemo
-keep public class SliderDemo
-keep public class SplitPaneDemo
-keep public class TabbedPaneDemo
-keep public class TableDemo
-keep public class ToolTipDemo
-keep public class TreeDemo
```

## REFERENCES

1. Excelsior JET and IP Protection:  
<http://www.excelsior-usa.com/jetprotection.html>
2. ProGuard obfuscator:  
<http://proguard.sourceforge.net/>