zenofx.com
The Java People

ClassGuard Quick Start Guide

Version 7.0

Table Of Contents

1   Introduction

ClassGuard is a tool to prevent Java decompiling and for license management of Java applications. The Java class files are encrypted using 128Bit AES encryption. The AES key is generated randomly every time you start the encryption tool.

The decryption is done transparently by a custom class loader. The main part of this class loader is written in C to prevent decompiling and other tampering.

If license management is activated, classes are only decrypted when a valid license is available.

ClassGuard supports Oracle Java and OpenJDK 8, 11, 17 and 21 on Windows and Linux (32 and 64bit Intel platforms). Optionally, OSX (both 64bit Intel and Apple Silicon/ARM) and Linux ARM64 platforms are available. Other Java versions may work, but are not officially supported.

There are three ways of invoking ClassGuard:

- A simple command line interface (CLI)
- Ant tasks
- Maven integration

An evaluation version of ClassGuard is available for download at

https://zenofx.com/classguard/ClassGuard-latest.zip

1.1   Feature list

This version contains the following features:

- Transparent class encryption
- License management (by CLI and GUI)
- Built-in application starter
- Support for multiple jar files
- Ant task
- Maven integration
- Resource encryption
- Tomcat support

2 Release notes

2.1 Upgrade to version 2.5

- ClassGuard now has a built in application starter, including a GUI for requesting licenses. Implementing your own startup code or setting the system class loader is now only necessary in exceptional cases.
- The syntax for setting custom attributes in licenses has slightly changed.

2.2 Upgrade to version 3.0

- ClassGuard now has a GUI for creating licenses.

2.3 Upgrade to version 3.5

- TomcatClassGuard.jar has changed.

2.4 Upgrade to version 4.0

- The package structure of ClassGuard has changed. The complete path of the ClassGuard class is now *com.zenofx.classguard.bootstrap.ClassGuard*. For compatibility reason, a wrapper class with the old package *net.jsecurity.classguard.bootstrap.ClassGuard* exists.
- The ant task definition file is now in *com/zenofx/classguard/bootstrap/antlib.xml*
- Support for Jboss 4 was dropped, as this version is no longer supported by RedHat. Jboss version 5.1 or 6 may be supported in a later version of ClassGuard 4.
- Support for Java 5 on OSX was dropped, since Java 6 is now available for all Intel based Macs.
- Support for the Solaris operating system is not yet available in ClassGuard 4.

2.5 Upgrade to version 5.0

- Support for Java 6 was dropped on all platforms
- Java is supported as Oracle JDK and OpenJDK in versions 7, 8, 9, 10 and 11.

2.6 Upgrade to version 6.0

- Java is supported as Oracle JDK and OpenJDK in versions 7, 8, 9, 10, 11 and 17.
- Tomcat is supported for versions > 8.0. If you need support for Tomcat 7, please contact us.

2.7 Upgrade to version 7.0

- Java is supported as Oracle JDK and OpenJDK in versions 8, 11, 17 and 21.
- Maven is now officially supported with an own repository provided by zenofx.com (see 4.4).
- Support for Java 7 was dropped.

3 First demo application

For your first test, use an application which is completely contained in one jar file. You may create your own application jar file (e.g. using ant or the jar command line utility) or you can use an existing demo application. In this example, we use the SwingSet2 demo from the JDK demo directory.

Encrypt your jar file using

```
java -jar ClassGuard.jar crypt -in SwingSet2.jar -out SwingSet2ClassGuard.jar
   -resource java
```

You may now start your encrypted application using

```
java -jar SwingSet2ClassGuard.jar
```

The SwingSet2 demo now starts up.

4   Encrypting applications

4.1   The Command Line Interface

The ClassGuard CLI ist started using

```
java -jar ClassGuard.jar <action> <options>
```

If started without an action, the license GUI appears. If started with the action *help*, a short description of command line parameters is displayed.

4.2   Encryption by command line

Applications are encrypted using the *crypt* action. The following options are available:

-in my_project.jar	Jar file to encrypt
-out crypted_jarfile.jar	Destination jar file
-include packages	Comma-separated list of packages or class files to encrypt in jar file. Format is *com/example/package1, com/example/package2/Class1.class.* Default is all packages are included.
-exclude packages	Comma-separated list of packages or class files to exclude from encryption in same format. Default is no packages are excluded.
-platform platforms	Comma-separated list of platforms for the destination file. Possible platforms in the evaluation version are *x86_win, x64_win, x64_osx, arm64_osx, x86_linux and x64_linux.*
-cryptkey keyfile	Import encryption key from file
-includelibs	Force inclusion of ClassGuard libs in importkey mode
-war	War mode, useful for web applications
-resource	Comma-separated list of extensions of resources to encrypt. Default is no resource encryption.
-startclass	Class for starting up this jar file. By default, the class defined in the attribute *Main-Class* of *META-INF/MANIFEST.MF* is used.

So encrypting the package *myproject.business* including all property files in your jar file *myproject.jar* (leaving all other packages unencrypted) would look like this:

```
java -jar ClassGuard.jar crypt -in myproject.jar -out myproject_c.jar -include
  myproject/business -resource properties
```

4.3   The ClassGuard crypt ant task

The ClassGuard ant task is defined using

```
<taskdef classpath="ClassGuard.jar"
  resource="com/zenofx/classguard/bootstrap/antlib.xml" />
```

Currently there is one task *crypt* with roughly the same parameters as the crypt command line action. The following options are available:

in my_project.jar	Jar file to encrypt
out crypted_jarfile.jar	Destination jar file. Must not be identical to source jar file.
include packages	Comma-separated list of packages or class files to encrypt in jar file. Format is *com/example/package1, com/example/package2/Class1.class*. Default is all packages are included.
exclude packages	Comma-separated list of packages or class files to exclude from encryption in same format. Default is no packages are excluded.
platform platforms	Comma-separated list of platforms for the destination file. Possible platforms in the evaluation version are *x86_win, x86_osx, x86_linux and x64_linux.*
cryptkey keyfile	Import encryption key from file
includelibs true/false	Force inclusion of ClassGuard libs in importkey mode. Default false.
war true/false	War mode, useful for web applications. Default false.
resource	Comma-separated list of extensions of resources to encrypt. Default is no resource encryption.
startclass	Class for starting up this jar file. By default, the class defined in the attribute *Main-Class* of *META-INF/MANIFEST.MF* is used.

Using the ant task, the above example would look like this:

```
<taskdef classpath="ClassGuard.jar"
   resource="com/zenofx/classguard/bootstrap/antlib.xml" />

<crypt in="myproject.jar" out="myproject_c.jar" include="myproject/business"
   resource="properties" />
```

4.4   The ClassGuard crypt maven mojo

You can use ClassGuard in your maven build by extending your pom.xml file like this:

```
<project>
  …
  <pluginRepositories>
    <pluginRepository>
      <id>zenofx</id>
      <url>https://download.zenofx.com/repository/</url>
    </pluginRepository>
  </pluginRepositories>
  …
```

```
   <plugins>
       <!-- ClassGuard encryption -->
       <plugin>
         <groupId>com.zenofx.maven</groupId>
         <artifactId>classguard</artifactId>
         <configuration>
             …
         </configuration>
         <executions>
           <execution>
             <goals>
               <goal>crypt</goal>
             </goals>
           </execution>
         </executions>
       </plugin>
   </plugins>
<project>
```

The following configuration options are available:

include packages	Comma-separated list of packages or class files to encrypt in jar file. Format is *com/example/package1, com/example/package2/Class1.class*. Default is all packages are included.
exclude packages	Comma-separated list of packages or class files to exclude from encryption in same format. Default is no packages are excluded.
platform platforms	Comma-separated list of platforms for the destination file. Possible platforms in the evaluation version are *x86_win, x86_osx, x86_linux and x64_linux.*
cryptkey keyfile	Import encryption key from file
includelibs true/false	Force inclusion of ClassGuard libs in importkey mode. Default false.
externallibs	Use external libs, save in *target* directory
war true/false	War mode, useful for web applications. Default false.
resource	Comma-separated list of extensions of resources to encrypt. Default is no resource encryption.
startclass	Class for starting up this jar file. By default, the class defined in the attribute *Main-Class* of *META-INF/MANIFEST.MF* is used.
postfix	Postfix for encrypted jar (default unencrypted jar is overwritten)

4.5 Encrypting multiple jar files

Usually ClassGuard generates a new encryption key every time it is started. In some cases, this is inconvenient. An application consisting of multiple jar files must share the same encryption key. In this case, first create the encryption key using

```
java -jar ClassGuard.jar createkey -cryptkey key.txt
```

Then use the created encryption key for crypting each of the jar files. Only one of the jar files should contain the ClassGuard libraries (usually the launcher).

```
java -jar ClassGuard.jar crypt -in file1.jar -out file1c.jar -cryptkey key.txt
   -includelibs
java -jar ClassGuard.jar crypt -in file2.jar -out file2c.jar -cryptkey key.txt
```

You should change the encryption key as often as possible, e.g. for each new version.

An encryption key can also be created using the createkey ant task, e.g.

```
<createkey cryptkey="key.txt" />
```

4.6 Reflection based frameworks

When using reflection based frameworks (e.g. Spring), it may be useful to use ClassGuard as the system class loader. In this case, the automated launch feature of ClassGuard may not be used. Launching your application may look like this:

```
java -cp myproject_c.jar
   -Djava.system.class.loader=com.zenofx.classguard.bootstrap.ClassGuard
   myproject.Start
```

4.7 WebStart

Java WebStart applications are supported by ClassGuard.

A WebStart application has to meet the following conditions:

- Your users need a Java version still supporting WebStart (currently only Oracle's Extended Support version of Java 8)
- All jar files have to be signed (for encrypted jar files, this has to be done **after** encryption)
- The jnlp file has to include the *all-permissions* entry
- For the automated bootstrap mechanism to work, *com.zenofx.classguard.bootstrap.Start* has to be set as main class in the jnlp file

If you need example code or you want ClassGuard to run on OpenWebStart, please contact our support team.

5 Using ClassGuard as a license manager

The license management features of ClassGuard are only available in the full version. The evaluation version contains license management, but all generated licenses expire when your evaluation copy of ClassGuard expires.

5.1 Encrypting your application

For using the license manager, you have to create a product key firstly:

```
java -jar ClassGuard.jar createkey -productkey myproduct.key
```

A product key is unique for one of your products. The resulting key file should be kept secret, as everyone owning this key file may generate licenses for this product.

When encrypting your licensed application, you have to supply your product key and your product name:
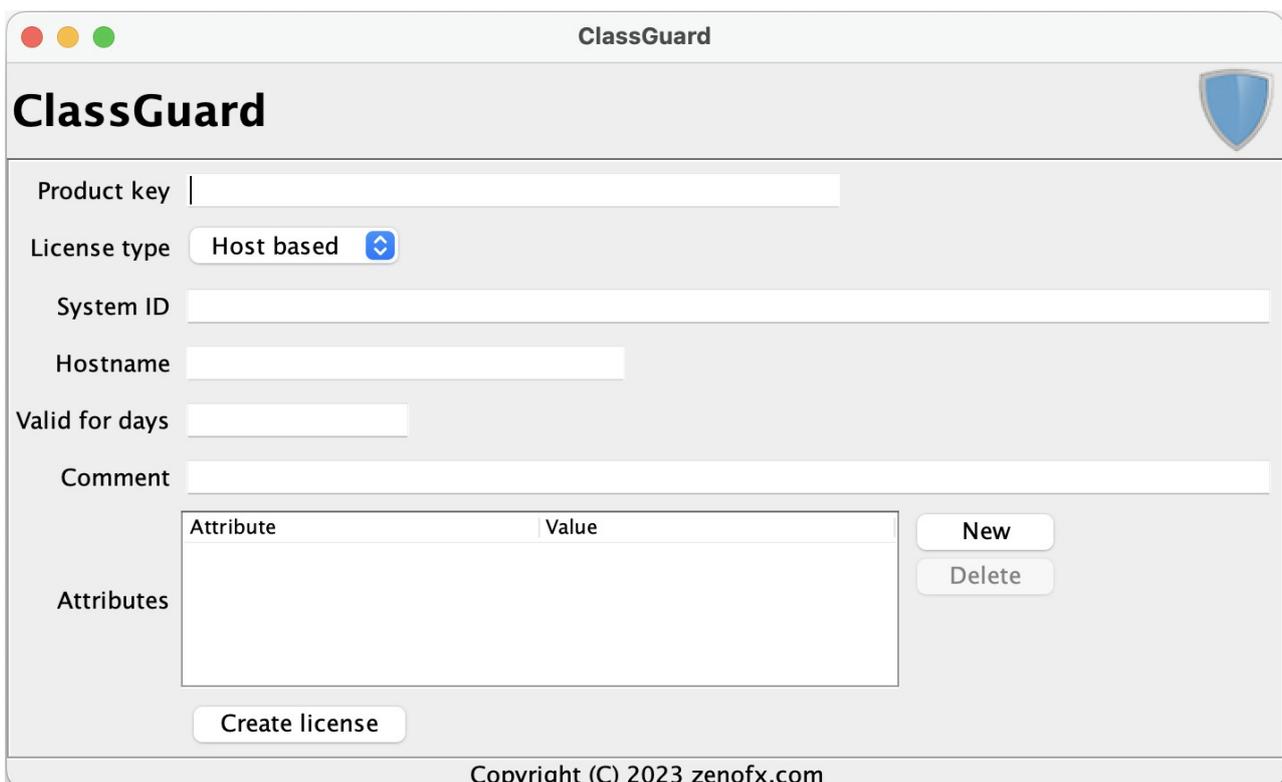
```
java -jar ClassGuard.jar crypt  -productkey myproduct.key -productname
  myproduct -in ... -out ...
```

The resulting application will only run if a matching license is available.

If no license is available, the application starter will ask for a license. This license will be stored in the java registry. On windows, the storage path is in the windows registry   in *HKCU\Software\ JavaSoft\Prefs\net\jsecurity\classguard\bootstrap\<productname>. On Unix and Linux, the file for storage of the license is ~/.java/.userPrefs/net/jsecurity/classguard/ bootstrap/prefs.xml.*

5.2 Creating licenses by GUI

After starting ClassGuard without command line parameters, the license GUI appears.

5.2.1   Creating a product key

A product key may be generated and saved to disk using *File/New product key*. An existing product key can be loaded using *File/Open product key*.

5.2.2   Creating a license

For creating a license, you have to load or create a product key and fill in the system id delivered by your customer. For evaluation licenses, a system id is not necessary. A new license may be created using the *Create license* button.

5.3   *Creating licenses by command line*

Licenses for your product may be created using

```
java -jar ClassGuard.jar license -productkey myproduct.key -licensefile
   license.txt licenseoptions
```

The following options are available:

-type H | U | E	License type host-based/user-based/eval (required)
-productkey keyfile	Create license for this product (required)
-licensefile file	License file for output (required)
-comment comment	License comment
-expire days	License expiration in days
-systemid id	System id delivered by customer (required for type H and U)
-attribute attrib	Custom attribute (key=value), may be used multiple times

5.4   *Implementing your own license GUI or license storage*

If you want to store the license in your own file or you want your own GUI for requesting the license, you have to implement the interface *com.zenofx.classguard.bootstrap.LicenseInterface.* You have to define your own implementation by setting the attribute *ClassGuard-License-Interface* in *META-INF/MANIFEST.MF*.

6 Using ClassGuard in combination with J2EE containers

6.1 Using ClassGuard in combination with Tomcat

As of Version 1.5, ClassGuard supports Tomcat containers. Dynamic starting, stopping and reloading of web applications is also supported.

To use ClassGuard in combination with tomcat, perform these simple steps:

1. Configure your web application for using the ClassGuard tomcat class loader. The easiest way to do so is to put a file "context.xml" into the META-INF directory of your web application, containing the following snippet:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Context>
   <Loader loaderClass="com.zenofx.classguard.support.TomcatClassGuard"/>
</Context>
```

Other ways of configuring the context of your web application will also work (see https://tomcat.apache.org/tomcat-9.0-doc/config/context.html).

2. Encrypt your war file using the -war option, e.g.

```
java -jar ClassGuard.jar crypt -in myapp.war -out my_encrypted_app.war -war
```

All classes contained in WEB-INF/classes will be encrypted.

OR

Encrypt the jar file containing your application logic and put it into the WEB-INF/lib directory of your war file.

```
java -jar ClassGuard.jar crypt -in myapp.jar -out my_encrypted_app.jar
```

Don't use the -war option in this case. It is not possible to combine both methods for one web application.

3. Put the ClassGuard tomcat class loader *TomcatClassGuard.jar* in the *lib* directory of your tomcat installation. TomcatClassGuard.jar does not contain any keys and may be shared between applications and even between different application vendors.

4. Start tomcat and deploy your application as usual.

If you want to protect jsp files, pre-compile them as classes. You can use JSPC to do this (see https://tomcat.apache.org/tomcat-9.0-doc/jasper-howto.html).

The Tomcat class loading mechanism works differently for listeners. They are not loaded by the class loader configured in the application context. Therefore, it is not possible to encrypt listener classes or any classes which are called directly be listeners. You can exclude these classes from encryption using the -exclude parameter.

The `<jsp:useBean>` tag uses dynamic class inspection, which doesn't work with encrypted classes. You can exclude these classes from encryption using the -exclude parameter.