Java obfuscator: implementation

Pierre.Girard@gemalto.com

Re-Trust quarterly meeting

Villach, March 11, 2008

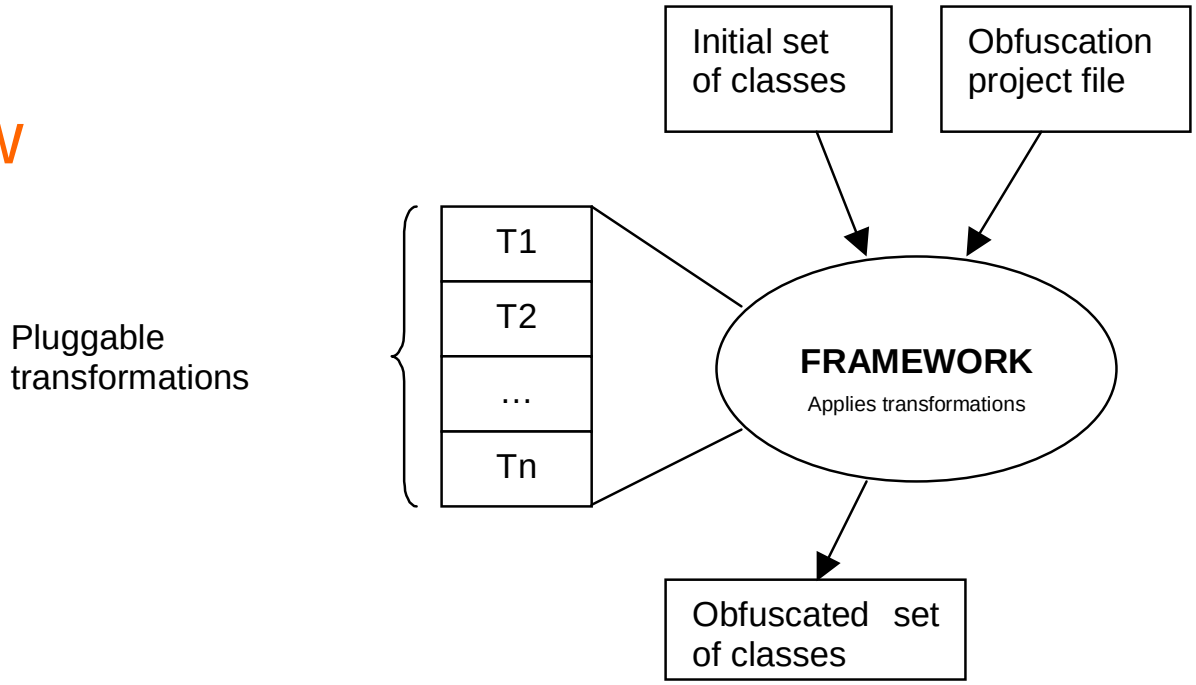
Introduction

- Java obfuscator design presented one year ago
  - Use cases for Java obfuscation in Gemalto
  - Functional requirements
  - Security requirements
  - Design directions
- Under implementation now

Agenda

- Reminder on design
- Obfuscation framework
- Transformations
- Next steps

Overview



- Take an application in entry (Jar files and/or directories)
- Apply transformations on this application following an obfuscation policy
- The resulting obfuscated application has the same behaviour (as experienced by the user) as the initial application

Features

- Obfuscation framework independent of the transformations
  - transformations can be developed separately and are pluggable
  - transformations should be more than only obfuscating transformations (for ex : code optimisation tool)
- The obfuscation process can be integrated into build process
  - logging, error codes, command line tool, API

Framework design

- Application
- Obfuscation policy and selections
- Core obfuscation engine

Application

- Load application files accessible from a classpath
- Provide enumerators on the application classes
  - classes with a given name (or all classes)
  - classes from a given package (or classes from all packages)
  - class with a fully qualified name
  - ...
- During the obfuscation process, modify the state of the application according to modifications requests issued by the obfuscating transformations
- Save the obfuscated application

Obfuscation policy

- Definition of *profiles* and *rules* in a policy
- *Profile* = set of transformations with eventually local parameters
- *Rule* = a selection and a set of transformations to be applied on this selection
- *Selection* = properties an application node (package, class method or field) must have to be selected
  - name, from-package, from-class, qualified-name, modifiers, extends, implements

Selections

- A selection is able to select application nodes
  - Tells the application to enumerate classes with their name and/or package name or qualified name
  - Within this enumeration, return only the nodes which properties match those required by the selection
- A selection is of type package, class, method or field

Obfuscation policy samples

- Profiles definition : set of transformations with their parameters

```
<profile name=«profile1»>
  <transformation name=«T1» />
  <transformation name=«T2»>
    <parameter name=«p1» value=«X» />
  </transformation>
</profile>
```

- Rules : profiles application on application nodes

```
<rule profile=«profile1»>
  <select … />
</rule>
```

```
<rule>
  <transformation name=«T1» />
  <transformation name=«T2»>
    <parameter name=«p1» value=«X»/>
  </transformation>
  <select … />
</rule>
```

Selections samples ...

All classes from all the application packages

<select type=«package»/> or <select type=«class»/>

All classes from all packages below com.gemalto

<select type=«package» name=«com.gemalto»/> or

<select type=«class» from-package=«com.gemalto»/>

All public final classes implementing Remote from all the application packages

<select type=«package» modifiers=«public final» implements=«java.rmi.Remote»/>

The class which qualified name is com.gemalto.components.a.C1

<select type=«class» qualified-name=«com.gemalto.components.a.C1»/> or

<select type=«class» from-package=«com.gemalto.components.a» name=«C1»/>

… Selections samples

All classes extending java.lang.String

```
<select type=«class» extends=«java.lang.String»/>
```

All public synchronized methods from the classes named C1 from all packages

```
<select type=«method» from-class=«C1» modifiers=«public synchronized»/>
```

All methods named getName from all classes named C1 from all packages below com

```
<select type=«method» name=«getName» from-package=«com» from-class=«C1» />
```

All methods named getName from the com.gemalto.components.a.C1 class

```
<select type=«method» qualified-name=«com.gemalto.components.a.C1getName»/>
```

The public fields named name from all classes from all packages below com

```
<select type=«field» name=«name» from-package=«com» />
```

Transformations ...

- Applied in their priority order
- User friendly name (correspondence user friendly name - transformation class name to load them dynamically)
- Applicable on packages and/or classes and/or methods and/or fields depending on the interface(s) they implement
  - check the transformations are applicable on the selections as defined in the obfuscation policy

… Transformations

- Local or global application. Does a transformation require a preparation before application and a propagation after application on the whole application ?
  - ex : class name changes must be reflected back on the whole appli
- Parameters (setters as Java Beans : name-value parameters)

Transformations lifecycle

All transformations should extend the ObfuscatingTransformation abstract class

ObfuscatingTransformation
(from spi)

getObfuscableClass(name : String) : ObfuscableClass
getName() : String
getPriority() : int
init(classFinder : ApplicationClassFinder, tracer : Tracer) : void
classSetup() : void
classTearDown() : void
instanceSetup() : void
instanceTearDown() : void
getTracer() : Tracer
findClass(qualifiedName : String) : ObfuscableClass

Transformations targets

All transformations should implement at least one of the XXObfuscatingTransformation interfaces to define on which type of nodes they are applicable

PackageObfuscatingTransformation
(from spi)

afterApplyPackage() : ModificationsRequest
applyPackage(aClass : ObfuscableClass) : ModificationsRequest
beforeApplyPackage() : ModificationsRequest

ClassObfuscatingTransformation
(from spi)

afterApplyClass() : ModificationsRequest
applyClass(aClass : ObfuscableClass) : ModificationsRequest

FieldObfuscatingTransformation
(from spi)

afterApplyField() : ModificationsRequest
applyField(aField : ObfuscableField) : ModificationsRequest
beforeApplyField() : ModificationsRequest

MethodObfuscatingTransformation
(from spi)

afterApplyMethod() : ModificationsRequest
applyMethod(aMethod : ObfuscableMethod) : ModificationsRequest
beforeApplyMethod() : ModificationsRequest

Obfuscation engine (transfos lifecycle)

initialisation

Preparation (*)

application

propagation (*)

destruction

```
For all transfos Ti order by priority {
    Ti.classSetup()
    For all rules Rj in the policy file containing Ti {
        new Ti0; Ti0.init(); Ti0.instanceSetup();
        If Ti needs a preparation {
            Ti0.beforePrepare();
            for all the appli° classes : Ti0.prepare();
            Ti0.afterPrepare();
        }
        Ti0.beforeApply();
        For all application nodes selected by Rj {
            Ti0.applyXX();
        }
        Ti0.afterApply();
        If Ti needs a propagation {
            Ti0.beforePropagate();
            for all the appli° classes : Ti0.propagate();
            Ti0.afterPropagate();
        }
        Ti0.instanceTearDown();
    }
    Ti.classTearDown();
}
```

(*) optional step

Traces, events

- **Logging** component to log debug, info, warnings, errors, fatal errors
  - log4J provider
- **Tracer** component for the transformations
  - front end to the logging
- Events sending based on the ‘events builder - events listeners model’
  - notification on the obfuscation process steps (preparation, application and propagation on an application node)

Integrated transformations under implementation

change-name(-reflection)	names obfuscation on packages, classes, methods and fields (with reflection handling)
optimize-constant-pool	remove unused entries in the class constant pool
remove-debug-info	remove classes and methods debug information
add-try-catch	add nested try-catch blocks into methods (should make decompilation fail)
extend-jump	add code and conditional jumps into methods using opaque predicates
cipher-strings	applicable on methods

Conclusion

- First release of an industrial obfuscation framework
- Next steps: more transformations integration
  - Innovative ones !
  - Secret ones !
- Next steps: framework enhancement
  - Support for incremental builds (handle patches)
  - Obfuscation reports
  - Definition of standard profiles
  - More powerful semantics for the obfuscation policy (for example using regular expressions)
- GUI to define the policy and to start an obfuscation process

Thank you !