gemalto

Java obfuscator

Pierre.Girard@gemalto.com

Re-Trust quarterly meeting

Meudon, March 21, 2007

Agenda

- Use cases for Java obfuscation in Gemalto
- Functional requirements
- Security requirements
- Design directions

Protect the secure link between an agent and a server

- Secure channel between a card and a host agent on a PC
  - VoIP soft phone / IM client
    - Potential targets: authentication and encryption keys
  - Electronic signature
    - Potential target: unauthorized signature of e-document
- Secure channel between a card and a host agent on a phone
  - J2ME/CLDC/MIDP game + JSR#177 + SIM dongle
- Obfuscation is mandatory to protect the secure link
  - Integrity and confidentiality keys
  - Authentication key of the agent

Protection against software modifications

- New “features” or “customization” by the customer
  - Warranty problem
  - Nightmare for the customer care
- Extraction of value added modules
  - E.g. : SS7 communication module, PKCS#11 module, …
- Enforcement of crypto export regulations
  - Avoid modifications to by-pass key length limitation

Security of embedded software

- Some parts of Java Card platforms are written in Java Card
- These parts may be reused in SDK (card simulators)
- Threats
  - Security mechanisms identification and reverse engineering
  - Bugs or weaknesses identification
  - Preparation and tuning of attacks on real cards

IP protection

- Protection of know-how
  - Security mechanisms
  - Optimizations
  - …
- Protection of secret algorithms
  - Still widely used in telecom industry
  - Third party property

Agenda

- Use cases for Java obfuscation in Gemalto
- Functional requirements
- Security requirements
- Design directions

Obfuscation control

- Precise selection of obfuscation
  - Preserve internal API / module interfaces
    - Debug, reuse, future extension, legacy tools, …
  - Tune the security vs performance trade-off
    - Performance = timing, code-size, power consumption, …
- Integration in the build process
  - Obfuscation project generation (GUI ?)
  - Command line mode (make files)
  - Java API
  - Eclipse / Ant / Maven integration
  - Reporting functionalities
  - Partial build / obfuscation

Java features handling

- Reflection
- Serialization
- RMI
- Beans
- CORBA
- Resources management
- Native methods

Agenda

- Use cases for Java obfuscation in Gemalto
- Functional requirements
- Security requirements
- Design directions

Why a proprietary obfuscator (YAJO) ?

- Flexibility
  - Build integration, configuration, …
- Security
  - Unknown by decompilers / desobfuscators
  - Yes it is security by obscurity !
  - Evaluation of security level
  - Possibility to implement knew / powerful obfuscation algorithms

Classical transformations classification

- Layout Obfuscation
  - Remove debug information
  - Change identifier names
- Data Obfuscation
  - Change the way data is stored or encoded in the program
- Control Obfuscation
  - Change the way the program runs
- Preventive Obfuscation
  - Try to find weaknesses in current deobfuscators / decompilers to make them crash

Agenda

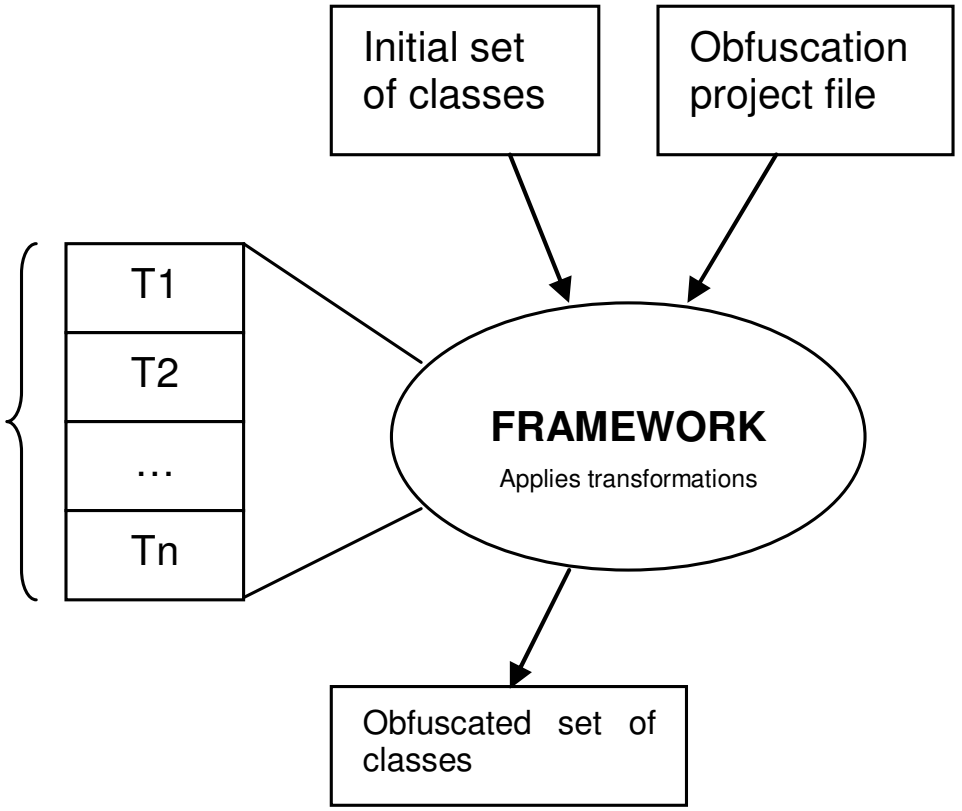
- Use cases for Java obfuscation in Gemalto
- Functional requirements
- Security requirements
- Design directions

Main design principles

- Separate obfuscation framework and transformations
- Pluggable transformation modules that can be developed and added separately
  - Transformations should be more than only obfuscating transformations (for ex : code optimisation tool / metrics / reporting)
- The obfuscation process can be integrated into build process
  - Logging, error codes, command line tool, public API

Overview

Pluggable transformations



Framework design

- Application
- Obfuscation policy and selections
- Core obfuscation engine

Application

- Load application files accessible from a classpath
- Provide enumerators on the application classes
- 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, selection* and *rules* in a policy
- *Profile* = set of transformations
- *Selection* = set of application nodes to be transformed
- *Rule* = a selection and a set of transformations to be applied on this selection

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

Conclusion

- Still in the early stage of design
- A set of transformations need to be selected
- Security / performance need to be evaluated