SPIIRAS

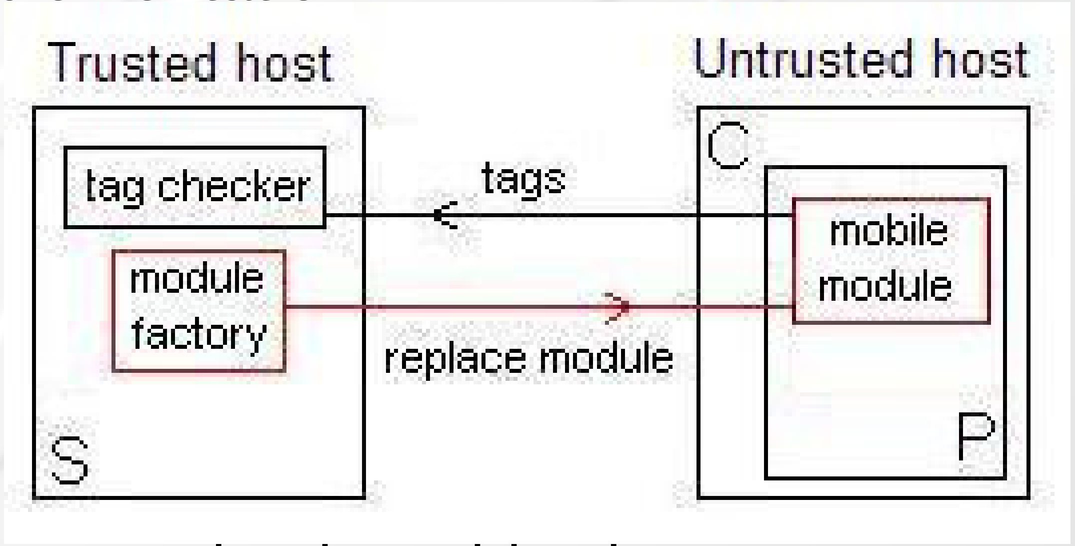
Review of aspect oriented approaches and their use in RE-TRUST for mobile module implementation

Vasily Desnitsky and Igor Kotenko

Computer Security Research Group,
St. Petersburg Institute for Informatics and Automation of Russian Academy of Sciences

Goal

- The goal is to build replacement mechanism for mobile module implementation

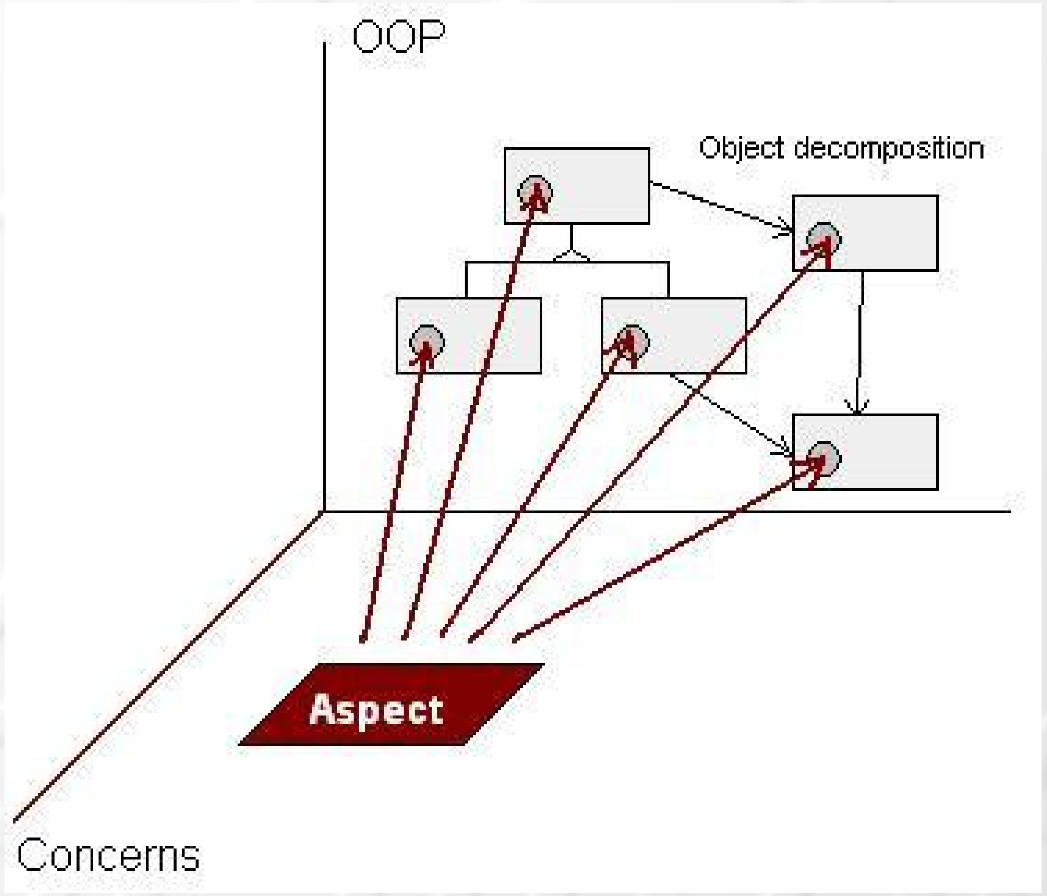


- Replacement mechanism objectives
  - Interpenetrate module to the program
  - Hiding module to prevent from its extraction by an adversary

Introduction to AOP (1)

- Crosscutting concern
  - Scattered and **tangled** code of some behavior feature can’t be expressed in generalized function presentation
- Separation of concerns
- Aspect-oriented programming
- Aspect
- AOP Frameworks
- Visual means

Introduction to AOP (2)



Main AOP notions

- Aspect
- Advice
  - before, after, instead
- Joinpoint
- Pointcut
- Aspect weaving
- Inter-type declaration

Pointcut definition

- Joinpoint types
  - Metod calls
  - Loop's beginning
  - Field access and assignments
  - Exception handlers including exception catch and throw
- Pointcut specification means
  - Name-based way
    - Direct definition of joinpoints by full class name, specific parameter types, modifiers, ect.
    - Wildcard matching / regular expressions
  - Attribute / annotation based way

Subtasks

1. Dynamic aspect loading into application
2. Representation of code of monitor and tag generator in the form of aspect collection

Kinds of AOP Approaches

- Three types of AOP Frameworks are different by the aspect weaving mechanism
  - Compile-time AOP
  - Load-time AOP
  - Runtime AOP

$$\left.\begin{array}{l}\text{Load-time AOP}\\ \text{Runtime AOP}\end{array}\right\} \boxed{\text{dynamic AOP}}$$

Load-time approach (1)

- Advice code is loaded as classes, libraries, assemblies, etc. during runtime
- Each new module's version is a new class collection
- Pointcuts are specified **no later than** application load

- Two possible strategies
  - **Total 'hook'** ***(joinpoint stubs)*** **weaving** – it leads to the 'empty hook problem'
  - **Minimal (Actual) hook weaving**

Load-time approach (2)

- Drawbacks of load-time approach
  - Inalterability of joinpoint location
  - Aspects are located in the memory as single units which can be tampered with by a malicious user
  - Old module version classes unload problem

Runtime AOP approaches (1)

- Aspect weaving and unweaving at runtime without having to stop application
- To specify pointcuts at runtime

Runtime AOP approaches (2)

- An approach using **Debugger** Interface
- Joinpoints implementation by debug events and breakpoints
- Externally advice execution to the application
- Drawbacks:
  - Need to **suspend** and **resume** executing application
  - Necessity to run application in ***debug mode***
  - It’s easy to detach Debugger by a malicious user

Runtime AOP approaches (3)

- **JIT** (Just-in-time) approach
- The alterations take place when the JIT compiler compiles the byte-code into a **native code**
- To apply the **Minimal Hook Strategy** for a native code

- Advantage
  - Aspects are tightly integrated with the application

Representation of the code of monitor and tag generator in the form of aspect collection

- What program entities could we verify?
  - Methods
  - Objects

- Possible techniques
  - To check method's input parameters and return value using before/after advices
  - To check method's body by means of access/assignation/modification advices to the objects
  - Inter-type-declaration

AOP Frameworks

- Opportunities
  - Load-time approach + *Total Hook Weaving* strategy
    - .NET: JAsCO.NET
    - Java: JAsCO, *JAC*
    - *C++: DAO C++*
      - meta-object data *(about classes and methods of application)*
      - *aspect matching expression*
  - Runtime JIT approach
    - Java: Prose

Conclusion (1)

- The aim is to construct a **criteria** which allow to determine if given AOP Framework can be used to implement Re-Trust replacement mechanism

Conclusion (2)

- List of requirements
  - Runtime pointcut specification
  - Runtime advice code weaving/unweaving
  - Aspects should be executed within the main application process
  - Aspects should be hidden in the application
  - AOP engine should be embedded into runtime environment
  - Atomic weaving