Exercise: Information Flow Control

Software Security

Steffen Helke
Chair of Software Engineering

26th November 2018

Brandenburgische
Technische Universität
Cottbus - Senftenberg

Objectives of today's lecture

- ➜ Repetition: How to make *Bell-LaPadula model* more flexible? How to express security policies in JiF?
- ➜ Understanding and applying analytical techniques to detect undesirable information flows in a program code of JiF
- ➜ Being able to implement security policies for small code examples based on JiF (Java + Information Flow)

Repetition: Bell-LaPadula Model (BLP)

Repetition: Bell-LaPadula Model (BLP)

1. How to make BLP more flexible?

- ➜ The security class of an object/subject can be change during life time under certain rules (*high watermark principle*)
- ➜ Example: A process starts with low security class and is upgraded when accessing objects of a higher security class, iff the new class is not greater than the watermark

2. How to bypass BLP with the help of covered channels?

- ➜ Storage Channels:
  - Process of high security class transmits confidential information by actions with the hard disk drive, e.g. using the read head
  - Process of low security class is able to monitor these actions
- ➜ Timing Channels:
  - Data transmission by measuring the runtime of processes

JIF: Java + Information Flow

JIF: Java + Information Flow

General Remarks

- Development 1997 at the Cornell University
- Security-typed programming language that extends Java
- Based on a *Decentralized Label Model* (DLM)

Language Features

- Labels for *Integrity* & *Confidentiality*
- Policy specification using *Principals*
- Hierarchical relationships between principals can be specified (*acts-for relation*)
- Declassification of labeled objects with respect to confidentiality & integrity is supported.

JIF-Homepage: http://www.cs.cornell.edu/jif/

Decentralized Model of Security Labels

Confidentiality Rules

- Notation: $u \rightarrow p$
- Owner u trusts the reader p not to give the information to unauthorized persons

Integrity Rules

- Notation: $u \leftarrow p$
- Owner u trusts the writer p not to destroy or damage the information

```
// only Alice and Bob are allowed to read a
int{Alice -> Bob} a;

// only Alice is allowed to write b
int{Alice <- Alice} b;

// combined read and write permissions
int{Alice -> Bob; Alice <- Alice} c;
```

Example: Assignment Operator of JIF

Example for Confidentiality

```
// only Alice and Bob are allowed to read a
int{Alice -> Bob} a;

// only Alice is allowed to read b
int{Alice -> Alice} b;

a = b;
b = a;
```

Example: Assignment Operator of JIF

Example for Confidentiality

```
// only Alice and Bob are allowed to read a
int{Alice -> Bob} a;

// only Alice is allowed to read b
int{Alice -> Alice} b;

a = b; // not allowed!
b = a; // allowed, because readers(b) ⊆ readers(a)
```

Example: Assignment Operator of JIF

Example for Integrity

```
// only Alice and Bob are allowed to write a
int{Alice <- Bob} a;

// only Alice is allowed to write b
int{Alice <- Alice} b;

a = b;
b = a;
```

Example: Assignment Operator of JIF

Example for Integrity

```
// only Alice and Bob are allowed to write a
int{Alice <- Bob} a;

// only Alice is allowed to write b
int{Alice <- Alice} b;

a = b; // allowed, because writers(b) ⊆ writers(a)
b = a; // not allowed!
```

JiF Exercises

1 My first JiF Program

Objective

➜ Get a first impression of how to handle security labels in JiF

Tasks

➜ Generate two principals *Alice* and *Bob*

➜ Declare integer variables and define assignments using these variables

➜ Apply confidentiality and integrity policies separately and together

➜ Define a label for a variable in such a way that the value can be assigned by any other variable

Generalization: Assignments

What does the JIF compiler check for the assignment

```
a = b;
```

?

➜ Main target: $sc(b)$ is less restricted than $sc(a)$
i.e. $sc(b) \sqsubseteq sc(a) := sc(b) \sqsubseteq_{\mathbf{C}} sc(a) \wedge sc(b) \sqsubseteq_{\mathbf{I}} sc(a)$

➜ Consists of the following subgoals:

1. b has a greater or equal number of readers than a
   $sc(b) \sqsubseteq_{\mathbf{C}} sc(a) := readers(b) \supseteq readers(a)$,
2. b has a lower or equal number of writers than a
   $sc(b) \sqsubseteq_{\mathbf{I}} sc(a) := writers(b) \subseteq writers(a)$

➜ This results in the following partial order relation with a bottom and top element $\{\_ \text{ -> } \_ \text{ ; } * \text{ <- } *\} \sqsubseteq \ \ldots \ \sqsubseteq \{* \text{ -> } * \text{ ; } \_ \text{ <- } \_\}$

Note: The JIF-operator $*$ represents *no* principals and the JIF-operator _ represents *all* principals, $sc(x)$ is the security label of the variable x

How to deal with implicit information flows?

- Analysis of implicit information flows is implemented using a *pc label* (*program-counter label*)
- What is a suitable security label for the variable *b*?

```
class Test {
   int {Bob -> Alice,Bob} a = 0;
   int { ... } b;

   public void f {} () {
     if (a == 0) {
        b = 4;  // check the pc label
     }
   }
}
```

Implicit information flows and method calls

Question: What is problematic with a method call?

```
class Test {
   int {Bob -> Alice,Bob,Steffen} a = 0;
   int {Bob -> Bob} b;

   public void f {} () {
     if (a == 0)
        setB();
   }
   private void setB () {
     b = 4;
}}
```

➜ Value assignment in the method setB() must be checked for all possible contexts in which this method is potentially called

Implicit information flows and method calls

- Implicit information flows for method calls are handled in JIF by so-called *begin labels*
- A *begin label* defines the *upper bound* for all pc labels at which the method can be called

```
class Test {
   int {Bob -> Alice,Bob,Steffen} a = 0;
   int {Bob -> Bob} b;

   public void f {} () {
     if (a == 0)
        setB();
   }
   private void setB { ... } () {
     b = 4;
}}
```

➜ What is a suitable begin label for the method *setB* ?

JiF Exercises

2 Task: Refactoring *Extract Method*

Objective

➜ Getting familiar with *begin labels* of methods

Tasks

➜ Consider for the information flow only the start labels of the methods

➜ Generate the begin labels in two different ways: First in the most restrictive variant and then in the most general one, such that in the latter case the method could be called in other contexts too

2 Task: Refactoring Extract Method

- In the following program, redundant code is to be extracted into an independent method ➜ Refactoring *Extract Method*
- Which signature (*begin label*) should be used to ensure that the security policy is preserved after restructuring?

```
class Refactoring {
   int {Bob -> Alice,Bob,Steffen} a = 0;
   int {Bob -> Bob} b = 1;
   int {Bob -> Alice,Bob} c;

   public void f {} () {
    if (a == 0) {
        b = 4;
        c = 3;
    }
    if (c == 1) {
        b = 4;
        c = 3;
    }}}
```