Jif: Java + Information Flow

Steve Zdancewic
University of Pennsylvania
Andrew Myers, Lantian Zheng, Nate Nystrom
Cornell University

Confidential Data

- Networked information systems:
  - PCs store passwords, e-mail, finances,...
  - Businesses rely on computing infrastructure
  - Military & government communications

- Security of data and infrastructure is critical
  [Trust in Cyberspace, Schneider et al. ’99]
Problems in Practice

- CartManager leaks personal info of millions...
- BJ’s wholesale club leaks 1000’s of credit cards...
- ...

Eli Lilly, major pharmaceutical company leaks the names of 669 Prozac users. “Oops. Which is essentially what the company said when it found out about the error.” --ComputerUser magazine

Technical Challenges

- Software is large and complex
- Security policies are complex
  - Requires tools & automation
- Existing mechanisms are crucial, but
  - OS: Coarse granularity of access control
  - Cryptography must be applied appropriately
High-level Policy

- HIPAA
- Sarbanes-Oxley
- Penn State
- Visa
- ...

---

High-level policy

- “As in all medical matters these tests and services are confidential in accordance with Pennsylvania Act 148”
  --from Penn State Policy AD43
Low-level security

• “As in all medical matters these tests and services are confidential in accordance with Pennsylvania Act 148”

```java
void checkHIV(Patient id) {
    MedRecord mr = getRecord(id);
    if (mr.tests.HIVpositive) {
        out.print("HIV pos.");
    }
    ...
}
```

Jif: Java+Information Flow

[Myers, Nystrom, Zdancewic, Zheng]

• Java
  – With some restrictions
• Information Flow Policy Language
  – Principals and Labels
  – Principal Hierarchy (delegation)
  – Confidentiality & Integrity constraints
  – Robust Declassification & Endorsement
  – Language features (i.e. polymorphism)
**Benefits**

- Explicit, fine-grained policies
- Program abstractions
- Regulate end-to-end behavior
  - Information Flow vs. Access Control
- Tools: increased confidence in security

**Information-flow Policy**

- Downloadable financial planner:

  ![Diagram](network.png)

- Access control insufficient
Noninterference

[Reynolds ’78, Goguen&Meseguer ’82,’84]

- Private data does not interfere with network communication
- Baseline confidentiality policy

Principals

- Principals: users, groups, etc.
  - Express constraints on data usage
  - Distinct from hosts
    - Alice, Bob, etc. are principals
  - Jif runtime represents principals as Java classes
Decentralized Labels

[Myers & Liskov '97, '00]

- **Simple Component** `{owner: readers}`
  - `{Alice: Bob, Eve}`
    “Alice owns this data and she permits Bob & Eve to read it.”

- **Compound Labels**
  - `{Alice: Charles; Bob: Charles}`
    “Alice & Bob own this data but only Charles can read it.”

---

Label Lattice

Labels higher in the lattice are more restrictive.
Integrity Constraints

• Specify who can write to a piece of data
  – {Alice? Bob} “Alice owns this data and she permits Bob to change it.”

• Both kinds of constraints
  – {Alice: Bob; Alice?}

Extended Types

• Jif augments Java’s types with labels
  – int{Alice:Bob} x;
  – Object{L} o;

• Subtyping
  – Inherited from the ≤ lattice order

• Inference
  – All Jif expressions have labeled types
  – Programmers may elide types
Implicit Flows

int\{Alice:\} a;
int\{Bob:\} b;

PC Label

\{\}

\{\}^\{Alice:\}=\{Alice:\}

if (a > 0) then {
\{\}

b = 4;
}

\{\}

\{Alice; Bob:\}

To assign to variable with label X, must have PC ≤ X.
**Function Calls**

```plaintext
int{Alice:} a;
int{Bob:} b;

PC Label

...{Alice:}=\{Alice:\}

if (a > 0) then {
    f(4);
}

Effects inside the function can leak information about the program counter.
```

**Method Types**

```plaintext
int\{L_1\} method\{B\} (int\{L_2\} arg) : \{E\}
    where authority(Alice)
    {
        ...
    }
```

- Constrain begin and end PC labels
  - To call PC ≤ B
  - On return PC ≤ E
- May include where clauses to specify
  - Authority (set of principals)
  - Caller’s Authority
Richer Security Policies

- More complex policies:
  "Alice will release her data to Bob, but only after he has paid $10."

- Noninterference too restrictive
  - In practice programs do leak some information
  - Justification lies outside the model (i.e. cryptography)

Declassification

```
int{Alice:} a;
int Paid;
...  // compute Paid
if (Paid==10) {
  int{Alice:Bob} b = declassify(a, {Alice:Bob});
  ...
}
```

“down-cast" int{Alice:} to int{Alice:Bob}
Robust Declassification

{Alice:} a;
{Alice?} Paid;
... // compute Paid
if (Paid==10) {
    int{Alice:Bob} b = declassify(a, {Alice:Bob});
    ...
}

Alice needs to trust the contents of paid.

Introduces constraint PC ≤ {Alice?}

First Class Labels & Principals

• Two new primitive datatypes:
  • principal
    – Can be bound to different users at run time
    – Programmer can ask whether p actsfor q
  • label
    – A value that can be used as a dynamic tag
    – If x is a label value then {*x} is the type
    – Can use switchlabel(l) to examine run-time labels
Parameterized Classes

- Jif allows classes to be parameterized by labels and principals
  - Code reuse
  - e.g. Containers parameterized by labels

- class MyClass[\text{label L}]\{ 
  \text{int}\{L\} x;
\}

Unix cat in Jif

```java
public static void main(String[] args) {
    String filename = args[0];
    final principal p = Runtime.user();
    final label lb;
    lb = new label{p:};
    Runtime[p] runtime = Runtime.getRuntime(p);
    FileInputStream[*lb] fis = runtime.openFileRead(filename, lb);
    InputStreamReader[*lb] reader = new InputStreamReader[*lb](fis);
    BufferedReader[*lb] br = new BufferedReader[*lb](reader);
    PrintStream[*lb] out = runtime.out();
    String line = br.readLine();
    while (line != null) {
        out.println(line);
        line = br.readLine();
    }
}
```
Caveats

• No threads
  – Information flow hard to control
  – Active area of research (still preliminary)
• Timing channels not controlled
  – Explicit choice for practicality
  – See Agat ’01 for alternatives
• Other differences from Java
  – Some exceptions are fatal
  – Restricted access to some System calls

Language-based Security

• Denning ’75, ’77
• Reynolds ’78
• Smith & Volpano ’96—’01
• Abadi, Banerjee, Heintz, and Riecke ’99
• Sabelfeld & Sands ’01,...
• Honda & Yoshida ’01, ’02
• Pottier et al. ’01, ’02
• Banerjee & Naumann ’02
• Many others ...


27

28
Jif Project Status

- Complete implementation of Jif.
- Implemented several thousand LOC
  - Mostly small test cases
  - Port of Java’s io package
  - Hashtable implementation
  - Tax simulation (~300 LOC)
  - Battleship program (~300 LOC)
  - Poker (~5000 LOC)
  - Mail client (~5000 LOC)

Jif Project Home

www.cs.cornell.edu/jif