Language-Based Enforcement of Privacy Policies

Katia Hayati and Martín Abadi

University of California, Santa Cruz

May 28, 2004

From privacy wishes to enforcement

- Users may want privacy guarantees:
  - Where their private data goes
  - How it is used
  - How long it is kept ...
- Some providers advertise privacy policies, but policies:
  - are subject to malice or error (and errors are easy)
  - don’t mean much without enforcement

Approaches to enforcement

- Dynamic monitoring
  - DPM, RM from IBM (Bohrer et al, Hill and Fritz)
- Formal reasoning on a model (Dreyer and Olivier, Lategan and Olivier)
- Automated proofs relating different privacy policies (Backes, Pfitzmann and Schunter)

A case for language-based enforcement

- Applies to actual code
- Can be (mostly) static
- Helps programmers reason about privacy
- Provides privacy documentation for system interfaces
- Supports code auditing

Using information-flow control

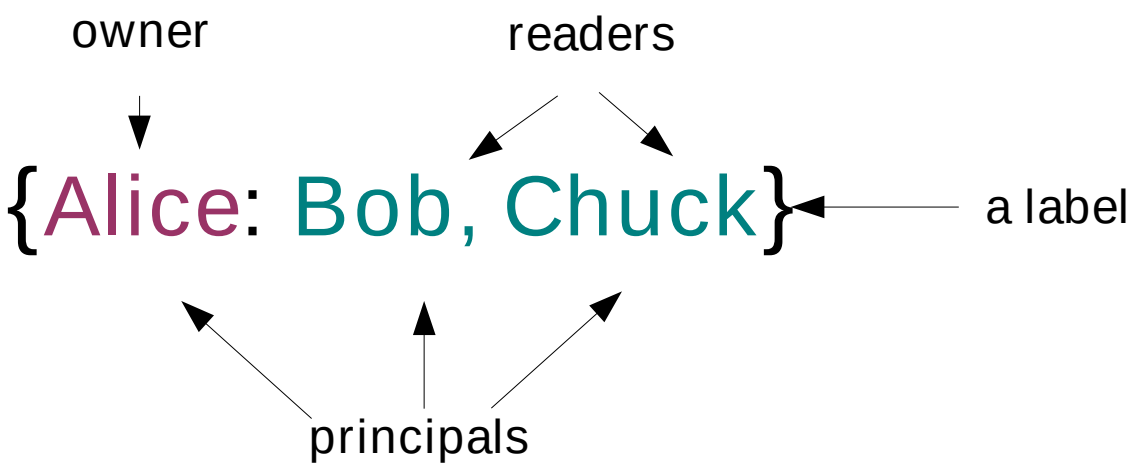
- Information-flow control is mature
- It can be used for guaranteeing secrecy and integrity properties
  - e.g., “low” subjects do not learn anything about “high” data
- Some of it is at the language level, with type systems, even for sophisticated languages (ML, Java)
- It seems relevant, but does it work?

Does it work?

- We ground our work on P3P
  - Well-defined
  - Provides a checklist of important privacy properties
- We focus on three aspects of privacy:
  - Basic control of information leaks
  - Purposes
  - Retention
- We use Jif, an extension of Java with information flow types

Jif in a nutshell

- (Mostly) static type checking
- Variables are annotated with labels
- The owner of a piece of data can give it a less restrictive label via *declassification*



Basic control of information leaks

```
int{Private: } credit_rating = 3;
int{Public: } rebate = 0;

if (credit_rating > 5) {
        rebate = 10;
}
else {
        rebate = 5;
}
```

ERROR: public information allowed to depend on private data

The acts-for relation

- Principals can be ordered with an acts-for relation
  - If Alice acts-for Bob then Alice can do everything Bob can
  - Acts-for is reflexive and transitive
- In the previous example, Private should have access to everything Public does (but not vice versa)

Using acts-for

```
int{Private: } credit_rating = 3;
int{Public: } rebate = 0;

actsFor(Private, Public) {
        if (rebate > 0) {
                credit_rating++;
        }
}
```

acts-for relationships may change at runtime, so check is necessary

no error

Purposes

- Purpose is a central privacy notion
  - A purpose should be interpreted as an “upper bound”
  - Data can be collected for more than one purpose
  - Purposes may have subpurposes (not in P3P)
- We model purposes with Jif principals

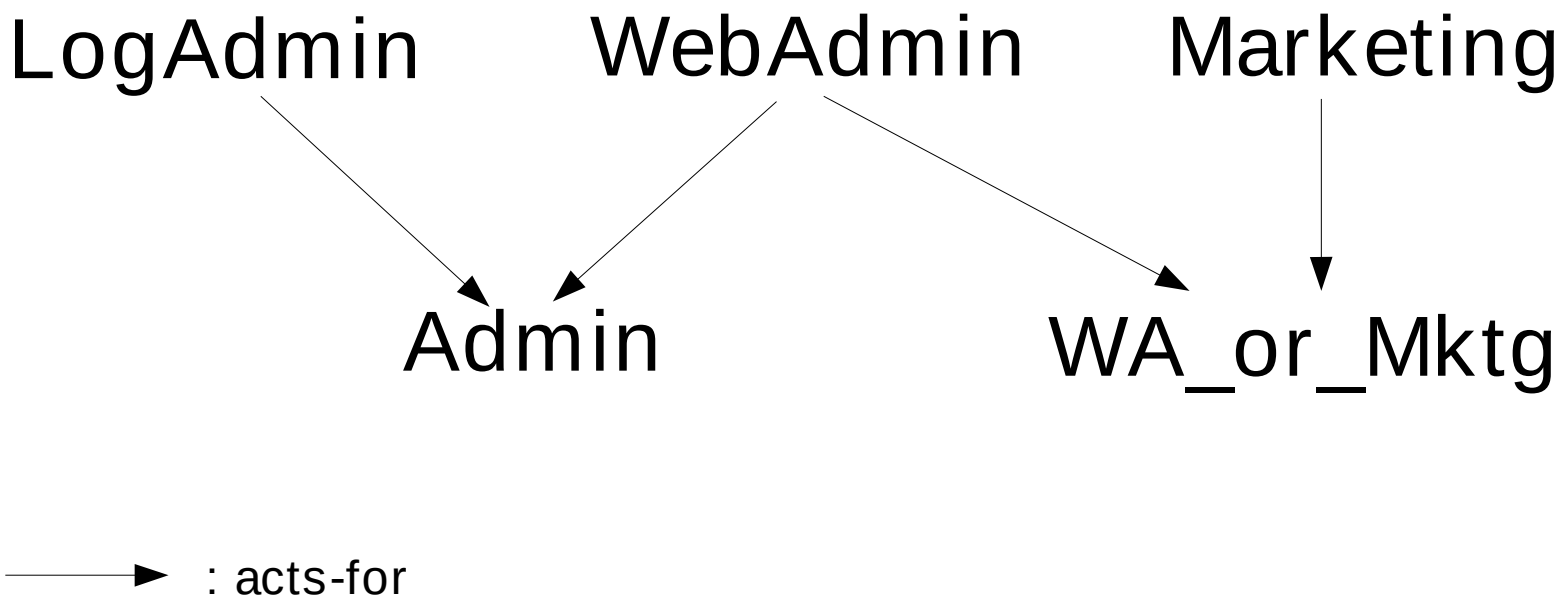
An example with purposes

```
class LogProcessor {

  public int{WebAdmin: } total_hits(...) {
    ...
  }
}
...
int{Marketing: } hits =
  (new LogProcessor(...)).total_hits(...);
```

ERROR: Incompatible labels

Multiple purposes and subpurposes via acts-for



Retention

- P3P retentions:
  - no-retention
  - stated-purpose
  - legal-requirement
  - business-practices
  - indefinitely
- We can view retention enforcement partly as an information-flow problem
  - Data marked as “no-retention” should not flow into data marked “indefinitely”

Retentions in an extension of Jif

- We extend the labels with retentions, and the ordering on labels accordingly

```
int{Marketing: ;; noretention} a = 1;
int{Marketing: ;; indefinitely}  b = a+1;
```

ERROR:
ephemeral data flows into permanent data

```
int{Marketing: ;; indefinitely} c = 0;
int{Marketing: ;; business}    d = c+1;
```

this is ok

Assurance, the downside

- If a program typechecks, it could still contain:
  - labeling errors, e.g.:
    - a principal called “Statistics” may perform non-statistical functions
    - a cookie which is only supposed to be retained temporarily might be annotated as “legal-requirement”
  - inappropriate declassifications

Assurance, the upside

- Annotations help focus auditing:
  - declassifications are important and easy to track
  - checking that “Statistics” performs statistical functions is a local problem
- Assurance could combine:
  - the formal reasoning of the type system
  - a statement asserting that the code does what it is supposed to do
  - perhaps formal proofs

Conclusion

- Information-flow control can help in supporting privacy policies
- Basic control of leaks, purposes and retentions can be encoded using Jif or a mild extension
- An annotated program is a better basis for assurance than a plain program

Open problems

- Suitability for large-scale software engineering projects?
- Stronger assurance
- Additional privacy properties
  - Anonymous use of data
- Relating P3P policies with language-level interfaces