Types for Information Flow

Armando Solar-Lezama
Computer Science and Artificial Intelligence Laboratory
MIT

Based on the paper by Myers, A. C. “JFlow: practical mostly-static information flow control”. In POPL '99

October 14, 2015
Recap

Functional World:
- evaluation proceeds through reduction rules
- types impose constraints on the shape of the program
- a program with a legal shape (according to the type system)
  - always has an available reduction rule (unless it has terminated)
  - the reduction rule will produce a new program with a legal shape
Recap

Imperative World:
- evaluation involves updating a store
- types place restrictions on the program store
  - this allows static reasoning about legal operations on the objects in the store
Recap

Imperative World:
- evaluation involves updating a store
- types place restrictions on the program store
  - this allows static reasoning about legal operations on the objects in the store
Enforcing Security Properties

Rx myrx = getMyRx();
Wikipedia w = getWPEntry(“Armando”);

w.addEntry(myrx.toString());

w.write(“YES”);
Enforcing Security Properties

```
Rx myrx = getMyRx();
Wikipedia w = getWPEntry(“Armando”);
w.write(“Hemorrhoids :”);
p.val = myrx.contains(“Preparation H”);
if(q.val){
    w.write(“YES”);
}
```

Even if p!=q, information can still leak if p!=q was caused by some information about myrx.

If p==q information clearly leaks
Enforcing Security Properties

class Doctor{
    Rx cureFlu() {
        Rx myrx = new Rx();
        Wikipedia w = getWPEntry("Flu");
        myrx.set(w.getSubEntry("Treatment"));
        return myrx;
    }
}
What is information flow?

If there is no information flow from private to public, then a change in a private input can not affect a public output – you can’t determine this from a single execution

For all $L_i, H_i, H'_i$
Solution Strategy

We proceed through the following two steps

- Define a dynamic labeling scheme so that at any given time, the labels in a piece of data tell us whether it’s OK to leak it or not.
  - Labels turn a global property about all executions into a local property in a conservative way
  - This will be the dynamic semantics against which we can prove type safety.

- Define a type system that allows us to approximate the set of labels that the data pointed at by a variable can have.
  - If an action is ok according to the conservative approximation, we know it would be ok according to the dynamic scheme.
Labeling Data With Security Policies

Policies for information flow

Owner: reader1, reader2, reader3

- “according to owner, this data can only be read by reader1, reader2, or reader3”

Label

{ policy1, policy2, policy3 }

- If an owner is not mentioned, it is assumed she has no privacy concerns

Why do we need an owner?
Principals

Owners and readers are principals
  - user, group or role

act_for relationship
  - allows principals to act for other principals

    Armando act_for Faculty
Labels form a lattice

\[ L_1 \leq L_2 \]

L1 can be relabeled to L2
- means that L2 is more restrictive (fewer readers)
- Warning: this is counterintuitive
  - L2 actually has fewer readers.

Partial Order defines a lattice
- Least upper bound \( \sqcup \)
- Least fixed point
- bottom

If a variable is certified to handle data with L2 labels correctly, we can trust that variable to hold a value with label L1
- Just like subtyping!
Labels form a lattice

Question

\{Joe: Ann, Jill\} \leq \{Joe:Ann\}

\{Joe: (Ann, Jill), Tim:Ann\} \leq \{Joe:(Ann), Tim:Ann\}

\{Joe: (Ann), Tim:Ann\} \ ??? \ \{Joe:(Ann)\}
Assignment

\[
x_{L2} := v_{L1};
\]

\[
L1 \leq L2
\]

Can only assign to a variable to a more restrictive label
Binary Operations

a\{L1\} + b\{L2\};

Trick question:
- What should be the label for a+b?

```
int\{Joe:everyone\} a, b, c;
...
int\{Joe:Joe\} p;
c = 0;
if(p){
    c = a + b;
}
```

- What information would be leaked if this code were to execute?
Information flow through control

Information flow through the PC

- We need to keep track of the information that is leaked just from knowing that the computation reached a particular point.

```
int{Joe:everyone} a, b, c;
...
int{Joe:Joe} p;
c = 0;
if(p){
c = a + b;
}
```

Simple scheme except for non-structured control

- return, continue, throw, break
Formalizing the type system

Basic judgments

\[ A \vdash E : X \]

Set of relevant labels.
X is a map with several values
- \( X[nv] \) = label of the expression if it terminates normally
- \( X[n] \) = label that would be leaked if execution terminated after evaluating this expression
- ...

Type Environment
Expression
If evaluating a literal somehow caused the program to terminate, I would leak the pc label.

The value of the literal also carries information about the PC label.

if(p){
x = literal
}

This is what prevents the code above from leaking information; the assignment only type checks if x is compatible with the PC label.
Rules

\[ A[v] = \langle \text{var} \ [\text{final}] \ T\{L\} \ \text{uid} \rangle \]

\[ X = X_0 \ [n := A[pc], \ nv := L \sqcup A[pc]] \]

\[ A \vdash v : X \]

Least upper bound. The return value must carry the labels of both the variable and the pc.
This is the label of expression E. It has to be less restrictive than L.
This computes the join of $XE$, $X_1$, $X_2$, except we don’t care about $XE[n]$ so we set it to $\{\}$.
Rules

extend the environment to add any new variable declarations

$A \vdash S_1 : X_1$

$\text{extend}(A, S_1)[pc := X_1[n]] \vdash S_2 : X_2$

$X = X_1[n := \emptyset] \oplus X_2$

$A \vdash S_1; S_2 : X$

update PC in the new environment
Example

\[ x \{Joe: Erika\} = \{Joe: Erika, Peter\} \]
\[ \text{if}(x)\{ \]
\[ \quad p\{Tim:Erika, Joe:Erika\} = \{Tim: Everyone\} \]
\[ \} \]