

Information Flow Security in Imperative Languages

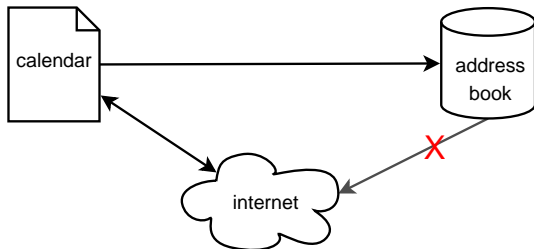
Robert Grabowski

Institut für Informatik
Ludwig-Maximilians-Universität München

14. Kolloquium Programmiersprachen und
Grundlagen der Programmierung
Timmendorfer Strand, Oktober 2007

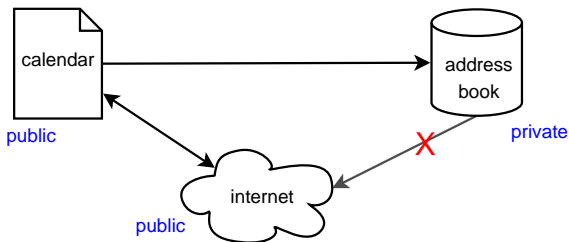
Gefördert durch DFG-Projekt InfoZert (Be 3712/2-1)

What are secure information flows?



- control flow of information between objects of a computing system (like variables, files, channels)
- restricting access to objects is not flexible enough

What are secure information flows?



- assign confidentiality domains to objects
- specify secure information flows using a *flow policy*
 - e.g. allow flows from public to private domain, but not vice versa

Secure flows in imperative languages

A simple WHILE language

$x \in \mathcal{X}$

$v \in \mathcal{V} = \mathbb{N}$

$e ::= x \mid v \mid e_1 + e_2 \mid \dots$

$C ::= x := e \mid C_1 ; C_2 \mid \mathbf{skip} \mid$

$\mathbf{if } e \mathbf{ then } C_1 \mathbf{ else } C_2 \mid \mathbf{while } e \mathbf{ do } C$

program states: $\sigma, \tau : \mathcal{X} \rightarrow \mathcal{V}$

given semantics: $\llbracket e \rrbracket_\sigma$ and $\sigma \xrightarrow{C} \tau$

Secure flows in imperative languages

Identifying secure flows

- security objects: variables
- classified into *low* and *high* domain: $\mathcal{X}_{low} \uplus \mathcal{X}_{high} = \mathcal{X}$
- flow policy: *low* \rightsquigarrow *high*, but *high* $\not\rightsquigarrow$ *low*

Data flow examples

assume $l \in \mathcal{X}_{low}, h \in \mathcal{X}_{high}$

$h := l$ (ok)

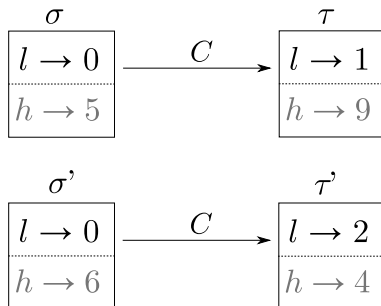
$l := h$ (insecure)

$l := h \bmod 2$ (insecure)

if $h > 0$ **then** $l := 1$ **else** $l := 3$ (insecure)

Non-interference

- initial values of *high* variables may not affect final values of *low* variables
- equivalently: when observing only *low* variables, the behaviour of the program must be deterministic



Non-interference

Definition

Two states σ, σ' are *indistinguishable*, written $\sigma \sim \sigma'$, if for all $x \in \mathcal{X}_{low}$: $\sigma(x) = \sigma'(x)$.

A program C is *secure* if whenever $\sigma \sim \sigma'$ and $\sigma \xrightarrow{C} \tau$ and $\sigma' \xrightarrow{C} \tau'$, then $\tau \sim \tau'$.

Other types of information flows

flows not covered by non-interference definition:

- termination behaviour
- timing aspects
- resource usage
- ...

here: only flows between variables (direct and implicit)

Static flow analysis

- type system by Volpano, Smith, Irvine (1996)
- check assignments for invalid direct flows
- implicit flows: do not allow assignment to *low* variables under *high* guard

if $h > 0$ then $l := 1$ else $l := 3$

Typing expressions

$k ::= low \mid high$

$\boxed{\vdash e : k}$

$$\frac{}{\vdash e : high} \quad \frac{Vars(e) \cap \mathcal{X}_{high} = \emptyset}{\vdash e : low}$$

- if $\vdash e : low$, then e does not depend on *high* variables

Typing programs

$k ::= low \mid high$ $h \in \mathcal{X}_{high}, l \in \mathcal{X}_{low}$

$[k] \vdash C$

$$\frac{}{[high] \vdash h := e} \quad \frac{\vdash e : low}{[low] \vdash l := e}$$

Typing programs

$k ::= low \mid high$ $h \in \mathcal{X}_{high}, l \in \mathcal{X}_{low}$

$[k] \vdash C$

$$\frac{}{[high] \vdash h := e} \quad \frac{\vdash e : low}{[low] \vdash l := e}$$

- k is a lower bound of the types of variables assigned in C

Typing programs

$k ::= low \mid high$ $h \in \mathcal{X}_{high}, l \in \mathcal{X}_{low}$

$[k] \vdash C$

$$\frac{}{[high] \vdash h := e} \quad \frac{\vdash e : low}{[low] \vdash l := e}$$
$$\frac{\vdash e : k \quad [k] \vdash C_1 \quad [k] \vdash C_2}{[k] \vdash \mathbf{if} \ e \ \mathbf{then} \ C_1 \ \mathbf{else} \ C_2} \quad \frac{\vdash e : k \quad [k] \vdash C}{[k] \vdash \mathbf{while} \ e \ \mathbf{do} \ C}$$

- k is a lower bound of the types of variables assigned in C

Typing programs

$k ::= low \mid high$ $h \in \mathcal{X}_{high}, l \in \mathcal{X}_{low}$

$[k] \vdash C$

$$\frac{}{[high] \vdash h := e} \quad \frac{\vdash e : low}{[low] \vdash l := e}$$
$$\frac{\vdash e : k \quad [k] \vdash C_1 \quad [k] \vdash C_2}{[k] \vdash \mathbf{if} \ e \ \mathbf{then} \ C_1 \ \mathbf{else} \ C_2} \quad \frac{\vdash e : k \quad [k] \vdash C}{[k] \vdash \mathbf{while} \ e \ \mathbf{do} \ C}$$
$$\frac{}{[high] \vdash \mathbf{skip}} \quad \frac{[k] \vdash C_1 \quad [k] \vdash C_2}{[k] \vdash C_1 ; C_2} \quad \frac{[high] \vdash C}{[low] \vdash C}$$

- k is a lower bound of the types of variables assigned in C

Typing programs

$k ::= low \mid high$ $h \in \mathcal{X}_{high}, l \in \mathcal{X}_{low}$

$[k] \vdash C$

$$\frac{}{[high] \vdash h := e} \quad \frac{\vdash e : low}{[low] \vdash l := e}$$
$$\frac{\vdash e : k \quad [k] \vdash C_1 \quad [k] \vdash C_2}{[k] \vdash \mathbf{if} \ e \ \mathbf{then} \ C_1 \ \mathbf{else} \ C_2} \quad \frac{\vdash e : k \quad [k] \vdash C}{[k] \vdash \mathbf{while} \ e \ \mathbf{do} \ C}$$
$$\frac{}{[high] \vdash \mathbf{skip}} \quad \frac{[k] \vdash C_1 \quad [k] \vdash C_2}{[k] \vdash C_1 ; C_2} \quad \frac{[high] \vdash C}{[low] \vdash C}$$

- k is a lower bound of the types of variables assigned in C
- Soundness: if $[low] \vdash C$, then C is secure, i.e. non-interferent

Further extensions

Richer language

- recursive functions
- references, objects, inheritance
- exceptions

Improve analysis

- more complete type system
- program logics

Our goal: secure flow framework

secure software design

- extend UML with specifications for security domains and flow policies
- translate requirements to language level

proof-carrying code architecture

- static flow analysis during compilation
- distribute proof as certificate for compiled code
- client (e.g. mobile device) can easily check whether certificate is correct and satisfies system policy

Limitations of static analysis

Problem: data objects only known at run-time

- example: program with file I/O
- to ensure non-interference for file contents, need to know which files are manipulated
- requires model of file names
- but: file names are interpreted strings

Dynamic labels

Solution

- associate data objects with labels (type variables)
- actual type available at runtime, may be verified against policy
- prove that program is secure for every type instantiation

Extended syntax

$C ::= \dots \mid \mathbf{with} (x_\alpha \mapsto f(e)) \mathbf{do} C \mid \mathbf{if} \alpha \rightsquigarrow \beta \mathbf{then} C_1 \mathbf{else} C_2$

Example

with ($f_\alpha \mapsto \mathit{openFile}(fn)$) **do**
 if $\alpha \rightsquigarrow \mathit{low}$ **then** $l := f$ **else skip**

Other practical issues

- non-interference is often too strict
 - certain situations require a controlled way to *declassify* and release sensitive information
 - need intuitive formalisation
- concurrent and distributed systems
- portability: programmer's intended policy vs client policy

some issues covered by *Jif* (Java with information flows)

Summary

- restrict flow of information, but not access to it
- non-interference formalises confidential data confinement
- static program analysis using type system
- embed in framework for flow-secure software engineering
- challenge: adopt security notion to real-world software
- contribution: dynamic labels for run-time objects