Certification of Programs for Secure Information Flow

Dorothy E Denning & Peter J. Denning

CS 711

Sept 24, 2003

Siggi Cherem

Introduction

- Security guarantees
- Runtime mechanisms
- Compile time mechanism
  - Comprehension
  - Correctness
  - No speed impact
  - Run-time support

Goal and agenda

A compile-time mechanism to check for violations of an information flow policy

- Information flow model
- Security definition
- Certification algorithm
- Applications and limitations

Information flow model

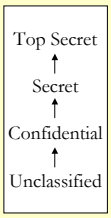
- Policies are described in a Lattice — $(S,\rightarrow)$
- Assign security classes (static labels) — x
- Permissible flow (flow is allowed) — $x \rightarrow y$
- Meet (LUB) and Join (GLB) — $\oplus, \otimes$
- Top ($\top$) and Bottom ($\bot$) — H, L

Characteristics:

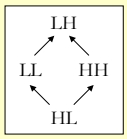
- Reflexive, transitive
- Finite number of labels, finite lattice

Lattices examples

Priority lattices



Property Lattices



Flow of information

- Explicit (assignments, I/O, procedure calls) — $x \Rightarrow y$

  `y := x`
- Implicit (conditional tests)

  `if (x = 0) then y := 1`
- Direct

  `y := f(… x …)`
- Indirect (by transitivity)

  `z := x; y:= z`

Security requirement

A necessary, sufficient and undecidable condition:

Program p is secure iff
Any execution of p has x ⇒ y only if x → y

A more conservative, but decidable condition:

Program p is secure iff
Any execution specified by p (any control flow)
has x ⇒ y only if x → y

Certification mechanism

- A compiler phase (as a type checker, optimizer)
- Efficient, advantages of model
  - Transitivity
  - Meet/Join operators
- Run-time support (external objects comply with specification)

```
if (e) then
    x:= y
    z:= x
else
    w:=0
```

Basic algorithm (I)

Inference of labels

3

x : **int security class** L

$$\frac{\Gamma \vdash a1 : \underline{a1} \quad \Gamma \vdash a2 : \underline{a2}}{\Gamma \vdash a1 \text{ op } a2 : \underline{a1} \oplus \underline{a2}}$$

e := a1 op a2 — $\underline{e} := \underline{a1} \oplus \underline{a2}$

Assignment

$$\frac{\Gamma \vdash e : \underline{e} \qquad \underline{e} \rightarrow \underline{x}}{\Gamma \vdash (x := e) : \text{certified}}$$

x := e — if not ($\underline{e} \rightarrow \underline{x}$) Cert = **false**

Basic algorithm (II)

Input/Output

input a,b,c,d **from** f — $\underline{f} \rightarrow \underline{a} \otimes \underline{b} \otimes \underline{c} \otimes \underline{d}$

output a,b,c,d **to** f — $\underline{a} \oplus \underline{b} \oplus \underline{c} \oplus \underline{d} \rightarrow \underline{f}$

Simple Control Structures

if (e) **then** {s1; s2; s3} [**else** s4] — $\underline{e} \rightarrow \underline{s1} \otimes \underline{s2} \otimes \underline{s3}$ [$\otimes \underline{s4}$]

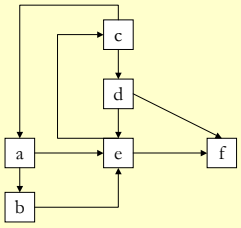
while (e) **do** s — $\underline{e} \rightarrow \underline{s}$

Theorem:
A program is **certified** only if it is **secure**

Advanced certification (I): goto's

Unstructured control flows

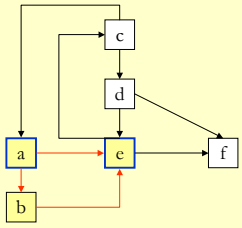
```
a: A
   if (a1) goto e
b: B
   goto e
c: C
   if (c1) goto a
d: D
   if (d1) goto f
e: E
   if (e1) goto c
f: …
```



Advanced certification (I): goto's

Unstructured control flows

```
a: A
   if (a1) goto e
b: B
   goto e
c: C
   if (c1) goto a
d: D
   if (d1) goto f
e: E
   if (e1) goto c
f: …
```

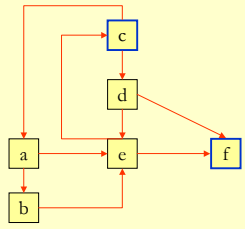


$\underline{a1} \rightarrow \underline{B}$

Advanced certification (I): goto's

Unstructured control flows

```
a: A
   if (a1) goto e
b: B
   goto e
c: C
   if (c1) goto a
d: D
   if (d1) goto f
e: E
   if (e1) goto c
f: …
```



$\underline{c1} \rightarrow \underline{A} \otimes \underline{B} \otimes \underline{C} \otimes \underline{D} \otimes \underline{E}$

Advanced certification (II): data structures

- Array rules

x := a[i]	$\underline{a[i]} \rightarrow \underline{x}$ $\quad \underline{a[i]} := \underline{a} \oplus \underline{i}$
a[i] := 1	$\underline{i} \rightarrow \underline{a}$

- Need for runtime array bound check
- Record rules

x := r.xi	$\underline{r.xi} \rightarrow \underline{x}$
output r **to** f	$\oplus_i \underline{r} \rightarrow \underline{f}$
s := r	$\underline{r.xi} \rightarrow \underline{s.xi}$ $\oplus_i \underline{r} \rightarrow \otimes_i \underline{s}$

Advanced certification (III): procedure calls

- q must be secure
- Secure assignment of
  - parameters $\underline{a_i} \rightarrow \underline{x_i}$
  - returned values $\underline{y_j} \rightarrow \underline{b_j}$
- Check for implicit flows
- Modular context insensitive analysis:
  - identify in p all calling context to q: $\underline{c} = \underline{c_1} \oplus \underline{c_2} \oplus \ldots \oplus \underline{c_k}$
  - identify in q all object receiving flow: $\underline{c} = \underline{c_1} \otimes \underline{c_2} \otimes \ldots \otimes \underline{c_l}$
- Check $\underline{c} \rightarrow \underline{c}$ at linking time

procedure q($x_1,\ldots,x_m,y_1,\ldots,y_n$)
…
procedure p(…)
call q($a_1,\ldots,a_m,b_1,\ldots,b_n$)

Advanced certification (III): procedure calls (cont)

procedure write(x,f)
output x **to** f
procedure p(…)
if (L) **then**
write(L,L)
if (H) **then**
write(H,H)

- Weak polymorphism $\underline{x_i}$ = H, therefore $\underline{b_j}$ = H
- Context sensitive approach (generally not tractable)
- Restrict procedures behavior: $\underline{c} \rightarrow \underline{b_1} \otimes \underline{b_2} \otimes \ldots \otimes \underline{b_n}$,
  $\underline{a_1} \oplus \underline{a_2} \oplus \ldots \oplus \underline{a_m} \rightarrow \underline{b_1} \otimes \underline{b_2} \otimes \ldots \otimes \underline{b_n}$

Advanced certification (IV): Traps

- Need to declare traps and handlers

on overflow sum
do e:= false;

- Mechanism to detect all possible traps
- Run-time support for exception handling

```
i: int … L
e:bool … L
f: file … L
x,sum: int … H

e := true
while (e) do
   sum += x;
   i++;
   output i to f
```

Applications

- Confinement problem: customer confidential information
  - A procedure p is secure so it can't encode H in L local vars
  - Inter-procedural restrictions forbids writing H to files
  - A procedure p can't call unconfined procedures
- State variables: storage channels, locks
- DB Confidentiality: labels for users
- Doesn't detect information leaks to covert channels

Related work & conclusions

- Use of lattices in model
- Language extension and static analysis
- Extensions
  - Interference
  - Synchronization
  - Cryptography