

Certification of Programs for Secure Information Flow

Dorothy E Denning & Peter J. Denning

CS 711
Sept 24, 2003
Siggi Chereem

Introduction

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

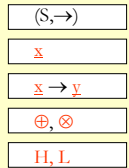
Goal and agenda

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
- Assign security classes (static labels)
- Permissible flow (flow is allowed)
- Meet (LUB) and Join (GLB)
- Top (\top) and Bottom (\perp)

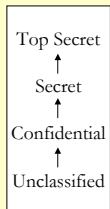


Characteristics:

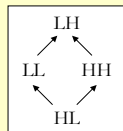
- Reflexive, transitive
- Finite number of labels, finite lattice

Lattices examples

Priority lattices



Property Lattices



Flow of information

- Explicit (assignments, I/O, procedure calls)

$x \Rightarrow y$

$y := x$

- Implicit (conditional tests)

$\text{if } (x = 0) \text{ then } y := 1$

- Direct

$y := f(\dots x \dots)$

- 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 $x \Rightarrow y$ only if $\underline{x} \rightarrow \underline{y}$

A more conservative, but decidable condition:

Program p is secure iff
Any execution specified by p (any control flow)
has $x \Rightarrow y$ only if $\underline{x} \rightarrow \underline{y}$

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

Basic algorithm (I)

Inference of labels

3

$\Gamma \vdash a1 : \underline{a1} \quad \Gamma \vdash a2 : \underline{a2}$

$x : \text{int security class } L$

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

$e := a1 \text{ op } a2$

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

Assignment

$x := e$

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

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

Basic algorithm (II)

Input/Output

input a,b,c,d from f

$f \rightarrow a \otimes b \otimes c \otimes d$

output a,b,c,d to f

$a \oplus b \oplus c \oplus d \rightarrow f$

Simple Control Structures

if (c) **then** {s1; s2; s3}
[**else** s4]

$\underline{c} \rightarrow \underline{s1} \otimes \underline{s2} \otimes \underline{s3}$
[$\otimes \underline{s4}$]

while (c) **do** s

$\underline{c} \rightarrow \underline{s}$

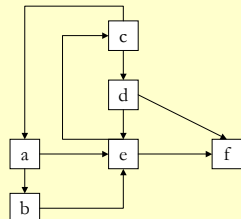
Theorem:

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

Advanced certification (I): goto's

Unstructured control flows

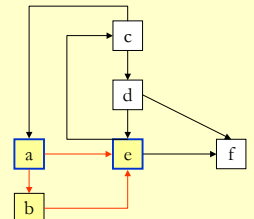
- a: A
- if (a1) goto e
- b: B
- goto e
- c: C
- if (c1) goto a
- d: D
- if (d1) goto f
- e: E
- if (e1) goto c
- f: ...



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: ...

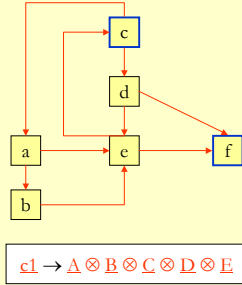


$\underline{a1} \rightarrow \underline{B}$

Advanced certification (I): goto's

Unstructured control flows

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



Advanced certification (II): data structures

- Array rules

$x := a[i] \quad a[i] \rightarrow x \quad a[i] := a \oplus i$

$a[i] := 1 \quad i \rightarrow a$

- Need for runtime array bound check

- Record rules

$x := r.xi \quad r.xi \rightarrow x$

output r to f $\oplus_i r \rightarrow f$

$s := r \quad r.xi \rightarrow s.xi$

$\oplus_i r \rightarrow \oplus_i s$

Advanced certification (III): procedure calls

- q must be secure
- Secure assignment of
 - parameters $a_i \rightarrow \underline{s}_i$
 - returned values $y_j \rightarrow \underline{b}_j$
- Check for implicit flows
- Modular context insensitive analysis:
 - identify in p all calling context to q: $\underline{c} = \underline{c}_1 \oplus \underline{c}_2 \oplus \dots \oplus \underline{c}_k$
 - identify in q all object receiving flow: $\underline{c} = \underline{c}_1 \otimes \underline{c}_2 \otimes \dots \otimes \underline{c}_l$
- Check $\underline{c} \rightarrow \underline{c}$ at linking time

```

procedure q( $x_1, \dots, x_m, y_1, \dots, y_n$ )
  ...
procedure p(...)
  call q( $a_1, \dots, a_m, b_1, \dots, b_n$ )
    
```

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 $\underline{s}_i = H$, therefore $\underline{b}_i = H$
- Context sensitive approach (generally not tractable)
- Restrict procedures behavior: $\underline{c} \rightarrow \underline{b}_1 \otimes \underline{b}_2 \otimes \dots \otimes \underline{b}_n$,
 $a_1 \oplus a_2 \oplus \dots \oplus a_m \rightarrow \underline{b}_1 \otimes \underline{b}_2 \otimes \dots \otimes \underline{b}_n$

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