02244 Logic for Security
Information Flow
Week 10: Volpano's Approach

Sebastian Mödersheim

April 15, 2024

Challenges for Denning's

We add arrays and procedures to our language

- D ::= ... | **integer array** C A[n] | ...
  ... | **proc** p(**in** T_1 C_1 $\mathrm{var}_{\mathrm{in}}$, **out** T_2 C_2 $\mathrm{var}_{\mathrm{out}}$) is S_{body}
- E ::= ... | A[E_1]
- S ::= ... | A[E_1] := E_2 | **call** p(E,var)

Define the new constraints that we need to impose.

Challenges for Denning's

We add arrays and procedures to our language

- D ::= ... | **integer array** C A[n] | ...
  ... | **proc** p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}
- E ::= ... | A[E_1]
- S ::= ... | A[E_1] := E_2 | **call** p(E,var)

Rules

D	**Security Class**
integer C A[n]	$\underline{A} =$
proc p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}	$\underline{p} =$ $\underline{var_{in}} =$ $\underline{var_{out}} =$

E	**Security Class**
A[E_1]	$\underline{E} =$

Challenges for Denning's

We add arrays and procedures to our language

- D ::= ... | **integer array** C A[n] | ...
  ... | **proc** p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}
- E ::= ... | A[E_1]
- S ::= ... | A[E_1] := E_2 | **call** p(E,var)

Rules

D	**Security Class**
integer C A[n]	$\underline{A} = C$
proc p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}	$\underline{p} =$ $\underline{var_{in}} =$ $\underline{var_{out}} =$

E	**Security Class**
A[E_1]	$\underline{E} =$

Challenges for Denning's

We add arrays and procedures to our language

- D ::= ... | **integer array** C A[n] | ...
  ... | **proc** p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}
- E ::= ... | A[E_1]
- S ::= ... | A[E_1] := E_2 | **call** p(E,var)

Rules

D	**Security Class**
integer C A[n]	$\underline{A} = C$
proc p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}	$\underline{p} = \underline{S_{body}}$ $\underline{var_{in}} =$ $\underline{var_{out}} =$

E	**Security Class**
A[E_1]	$\underline{E} =$

We add arrays and procedures to our language

- D ::= ... | **integer array** C A[n] | ...
  ... | **proc** p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}
- E ::= ... | A[E_1]
- S ::= ... | A[E_1] := E_2 | **call** p(E,var)

Rules

D	**Security Class**
integer C A[n]	$\underline{A} = C$
proc p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}	$\underline{p} = \underline{S_{body}}$ $\underline{var_{in}} = C_1$ $\underline{var_{out}} = C_2$

E	**Security Class**
A[E_1]	$\underline{E} =$

Challenges for Denning's

We add arrays and procedures to our language

- D ::= ... | **integer array** C A[n] | ...
  ... | **proc** p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}
- E ::= ... | A[E_1]
- S ::= ... | A[E_1] := E_2 | **call** p(E,var)

Rules

D	**Security Class**
integer C A[n]	$\underline{A} = C$
proc p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}	$\underline{p} = \underline{S_{body}}$ $\underline{var_{in}} = C_1$ $\underline{var_{out}} = C_2$

E	**Security Class**
A[E_1]	$\underline{E} = \underline{A} \sqcup \underline{E_1}$

Challenges for Denning's

We add arrays and procedures to our language

- D ::= ... | **integer array** C A[n] | ...
  ... | **proc** p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}
- E ::= ... | A[E_1]
- S ::= ... | A[E_1] := E_2 | **call** p(E,var)

Rules

S	**security class** of S	**constraint**
A[E_1] := E_2	$\underline{S}$ =	
call p(E,var)	$\underline{S}$ =	

Challenges for Denning's

We add arrays and procedures to our language

- D ::= ... | **integer array** C A[n] | ...
  ... | **proc** p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}
- E ::= ... | A[E_1]
- S ::= ... | A[E_1] := E_2 | **call** p(E,var)

Rules

S	**security class** of S	**constraint**
A[E_1] := E_2	$\underline{S} =$	$\underline{E_1} \sqcup \underline{E_2} \sqsubseteq \underline{A}$
call p(E,var)	$\underline{S} =$	

We add arrays and procedures to our language

- D ::= ... | **integer array** C A[n] | ...
  ... | **proc** p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}
- E ::= ... | A[E_1]
- S ::= ... | A[E_1] := E_2 | **call** p(E,var)

Rules

S	**security class** of S	**constraint**
A[E_1] := E_2	$\underline{S} = \underline{A}$	$\underline{E_1} \sqcup \underline{E_2} \sqsubseteq \underline{A}$
call p(E,var)	$\underline{S} =$	

Challenges for Denning's

We add arrays and procedures to our language

- D ::= ... | **integer array** C A[n] | ...
  ... | **proc** p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}
- E ::= ... | A[E_1]
- S ::= ... | A[E_1] := E_2 | **call** p(E,var)

Rules

S	**security class** of S	**constraint**
A[E_1] := E_2	$\underline{S} = \underline{A}$	$\underline{E_1} \sqcup \underline{E_2} \sqsubseteq \underline{A}$
call p(E,var)	$\underline{S} = \underline{p}$	

Challenges for Denning's

We add arrays and procedures to our language

- D ::= ... | **integer array** C A[n] | ...
  ... | **proc** p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}
- E ::= ... | A[E_1]
- S ::= ... | A[E_1] := E_2 | **call** p(E,var)

Rules

S	**security class** of S	**constraint**
A[E_1] := E_2	$\underline{S} = \underline{A}$	$\underline{E_1} \sqcup \underline{E_2} \sqsubseteq \underline{A}$
call p(E,var)	$\underline{S} = \underline{p}$	$\underline{E} \sqsubseteq \underline{var_{in}}$

Challenges for Denning's

We add arrays and procedures to our language

- D ::= ... | **integer array** C A[n] | ...
  ... | **proc** p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}
- E ::= ... | A[E_1]
- S ::= ... | A[E_1] := E_2 | **call** p(E,var)

Rules

S	**security class** of S	**constraint**
A[E_1] := E_2	$\underline{S} = \underline{A}$	$\underline{E_1} \sqcup \underline{E_2} \sqsubseteq \underline{A}$
call p(E,var)	$\underline{S} = \underline{p}$	$\underline{E} \sqsubseteq \underline{var_{in}}$
		$\underline{var_{out}} \sqsubseteq \underline{var}$

Challenge for Myers'

```
coronatest = record [ subject : cpr  {shs:shs},
                      testdate: date {shs:shs},
                      positive: bool {shs:shs} ]
publish_stat(db:array[coronatest:{⊥}]{⊥},
             day:date{shs:shs})
returns infections:int{⊥}

lookup_result(db:array[coronatest:{⊥}]{⊥},
               today:date{⊥},
               client:cpr{⊥})
returns result:bool{client:client}
```

`shs` is for sundhedsstyrelsen

Implement the functions `publish_stat` and `lookup_result`

- What authority/declassifications do the functions need?
- Prove that this is indeed safe.

So What?

Up to this point, we (and Denning and Denning) have only...

- defined a set of security classes
- decorated variables with these security classes
- decorated programs with constraints on security classes

So?

- If a program satisfies the constraints, then what guarantees do we really obtain from that?
- Given a set of rules for information flow (like the ones we just gave for arrays and procedures):
  how can one judge if they are "correct" or "wrong"?

Volpano's Approach to IFC

Dennis M. Volpano and Geoffrey Smith and Cynthia E. Irvine: *A Sound Type System for Secure Flow Analysis*, Journal of Computer Security, 4(2/3), pp. 167–188, 1996. [2]

- Essentially the same approach as Denning and Denning [1]
- ... but represented as a type system
  - ★ Security classes like Low and High are considered as types
  - ★ The ordering of security classes $\sqsubseteq$ is considered as a subtype relation
  - ★ Information Flow Analysis is described as a set of type-inference rules
- They give a natural semantics for the programming language.
- Type system and semantics allows for proving a precise statement about the security of programs that fulfill the information flow policy: a Non-Interference Result.

Volpano's Approach to IFC

Dennis M. Volpano and Geoffrey Smith and Cynthia E. Irvine: *A Sound Type System for Secure Flow Analysis*, Journal of Computer Security, 4(2/3), pp. 167–188, 1996. [2]

We simplify the paper a bit:

- Volpano et al. distinguish *variables* and *memory locations*, we treat them here all as variables (dropping the concept of introducing local variables).
- The corresponding symbol tables γ and λ are merged to just γ.
- We directly work with the syntax-directed form and do not need to distinguish the syntactic roles of variables, expressions and commands in the type system (i.e., we do not have *var* τ etc.)

Syntax

Syntax

$$\begin{array}{llll} \text{Expressions} & e & ::= & x \mid n \mid e + e' \mid \ldots \mid e = e' \mid \ldots \\ \text{Commands} & c & ::= & x := e \mid c; c' \mid \mathsf{if}\ e\ \mathsf{then}\ c\ \mathsf{else}\ c' \mid \mathsf{while}\ e\ \mathsf{do}\ c \end{array}$$

where x is for identifiers (for variables) and n is for constants.

Typing Rules

We define type inference rules where:

- γ is a type environment: it gives for every variable its type.
- The types are the security classes SC (e.g. High and Low) and $\sqsubseteq$ is the comparison relation on the security classes.
- Type judgements for expressions are of the form

$$\gamma \vdash e : \tau$$

Read: under γ, the expression e can be typed as τ.

Typing Rules (Expressions)

$$\text{(CONST)}\ \frac{}{\gamma \vdash n : \tau}$$

$$\text{(VAR)}\ \frac{}{\gamma \vdash x : \tau}\ \gamma(x) \sqsubseteq \tau$$

$$\text{(ARITH)}\ \frac{\gamma \vdash e : \tau \quad \gamma \vdash e' : \tau}{\gamma \vdash e\ op\ e' : \tau}$$

Typing Rules

- Type judgements for commands are of the form

$$\gamma \vdash c : \tau$$

Read: under γ, the command c can be typed as τ.

Typing Rules (Commands)

$$\text{(ASSIGN)} \quad \frac{\gamma \vdash e : \tau}{\gamma \vdash x := e : \tau'} \; \gamma(x) = \tau, \tau' \sqsubseteq \tau$$

$$\text{(COMPOSE)} \quad \frac{\gamma \vdash c : \tau \quad \gamma \vdash c' : \tau}{\gamma \vdash c; c' : \tau}$$

$$\text{(IF)} \quad \frac{\gamma \vdash e : \tau \quad \gamma \vdash c : \tau \quad \gamma \vdash c' : \tau}{\gamma \vdash \text{if } e \text{ then } c \text{ else } c' : \tau'} \; \tau' \sqsubseteq \tau$$

$$\text{(WHILE)} \quad \frac{\gamma \vdash e : \tau \quad \gamma \vdash c : \tau}{\gamma \vdash \text{while } e \text{ do } c : \tau'} \; \tau' \sqsubseteq \tau$$

Example

- Let $\gamma(x) = H$ and $\gamma(y) = L$

$$\frac{TODO}{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : \tau_1}$$

Constraints:

Fitting rule

$$\text{(IF)} \quad \frac{\gamma \vdash e : \tau \quad \gamma \vdash c : \tau \quad \gamma \vdash c' : \tau}{\gamma \vdash \text{if } e \text{ then } c \text{ else } c' : \tau'} \; \tau' \sqsubseteq \tau$$

Example

- Let $\gamma(x) = H$ and $\gamma(y) = L$

$$\dfrac{\dfrac{TODO}{\gamma \vdash x > 1000 : \tau_2} \quad \dfrac{TODO}{\gamma \vdash x := 5 : \tau_2} \quad \dfrac{TODO}{\gamma \vdash y := y + x * 3 : \tau_2}}{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : \tau_1}$$

Constraints: $\tau_1 \sqsubseteq \tau_2$

Fitting rule

$$\text{(ARITH)} \quad \dfrac{\gamma \vdash e : \tau \quad \gamma \vdash e' : \tau}{\gamma \vdash e \ op\ e' : \tau}$$

Example

- Let $\gamma(x) = H$ and $\gamma(y) = L$

$$\dfrac{\dfrac{\dfrac{TODO}{\gamma \vdash x : \tau_2} \quad \dfrac{TODO}{\gamma \vdash 1000 : \tau_2}}{\gamma \vdash x > 1000 : \tau_2} \quad \dfrac{TODO}{\gamma \vdash x := 5 : \tau_2} \quad \dfrac{TODO}{\gamma \vdash y := y + x * 3 : \tau_2}}{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : \tau_1}$$

Constraints: $\tau_1 \sqsubseteq \tau_2$

Fitting rule

(VAR) $\dfrac{}{\gamma \vdash x : \tau} \; \gamma(x) \sqsubseteq \tau$

Example

- Let $\gamma(x) = H$ and $\gamma(y) = L$

$$\dfrac{\dfrac{}{\gamma \vdash x : H} \quad \dfrac{TODO}{\gamma \vdash 1000 : H}}{\gamma \vdash x > 1000 : H} \quad \dfrac{TODO}{\gamma \vdash x := 5 : H} \quad \dfrac{TODO}{\gamma \vdash y := y + x * 3 : H}$$
$$\overline{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : \tau_1}$$

Constraints: $\tau_1 \sqsubseteq \tau_2 \wedge \gamma(x) = H \sqsubseteq \tau_2$ Thus: $\tau_2 = H$.

Fitting rule

(CONST) $\dfrac{}{\gamma \vdash n : \tau}$

Example

- Let $\gamma(x) = H$ and $\gamma(y) = L$

$$\dfrac{\dfrac{\overline{\gamma \vdash x : H} \quad \overline{\gamma \vdash 1000 : H}}{\gamma \vdash x > 1000 : H} \quad \dfrac{TODO}{\gamma \vdash x := 5 : H} \quad \dfrac{TODO}{\gamma \vdash y := y + x * 3 : H}}{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : \tau_1}$$

Constraints:

Fitting rule

$$\text{(ASSIGN)} \quad \frac{\gamma \vdash e : \tau}{\gamma \vdash x := e : \tau'} \; \gamma(x) = \tau, \tau' \sqsubseteq \tau$$

Example

- Let $\gamma(x) = H$ and $\gamma(y) = L$

$$\dfrac{\dfrac{\dfrac{}{\gamma \vdash x : H} \quad \dfrac{}{\gamma \vdash 1000 : H}}{\gamma \vdash x > 1000 : H} \quad \dfrac{\dfrac{}{\gamma \vdash 5 : H}}{\gamma \vdash x := 5 : H} \quad \dfrac{TODO}{\gamma \vdash y := y + x * 3 : H}}{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : \tau_1}$$

Constraints:

Fitting rule

$$(\text{ASSIGN}) \quad \dfrac{\gamma \vdash e : \tau}{\gamma \vdash x := e : \tau'} \; \gamma(x) = \tau, \tau' \sqsubseteq \tau$$

Example

- Let $\gamma(x) = H$ and $\gamma(y) = L$

$$\dfrac{\dfrac{\dfrac{}{\gamma \vdash x : H} \quad \dfrac{}{\gamma \vdash 1000 : H}}{\gamma \vdash x > 1000 : H} \quad \dfrac{\dfrac{}{\gamma \vdash 5 : H}}{\gamma \vdash x := 5 : H} \quad \dfrac{\color{red}{\text{requires } \gamma(y) \sqsupseteq H!}}{\gamma \vdash y := y + x * 3 : H}}{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : \tau_1}$$

Constraints:

Fitting rule

$$\text{(ASSIGN)} \quad \frac{\gamma \vdash e : \tau}{\gamma \vdash x := e : \tau'} \; \gamma(x) = \tau, \tau' \sqsubseteq \tau$$

There is no way to finish the red part of the proof. It does not type!

Example

- Let $\gamma(x) = L$ and $\gamma(y) = H$

$$\frac{TODO}{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : \tau_1}$$

Constraints:

Fitting Rule

$$(\text{IF}) \quad \frac{\gamma \vdash e : \tau \quad \gamma \vdash c : \tau \quad \gamma \vdash c' : \tau}{\gamma \vdash \text{if } e \text{ then } c \text{ else } c' : \tau'} \; \tau' \sqsubseteq \tau$$

Example

- Let $\gamma(x) = L$ and $\gamma(y) = H$

$$\dfrac{\dfrac{TODO}{\gamma \vdash x > 1000 : \tau_2} \quad \dfrac{TODO}{\gamma \vdash x := 5 : \tau_2} \quad \dfrac{TODO}{\gamma \vdash y := y + x * 3 : \tau_2}}{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : \tau_1}$$

Constraints: $\tau_1 \sqsubseteq \tau_2$

Fitting Rule

$$(\text{ARITH}) \quad \dfrac{\gamma \vdash e : \tau \quad \gamma \vdash e' : \tau}{\gamma \vdash e \; op \; e' : \tau}$$

Example

- Let $\gamma(x) = L$ and $\gamma(y) = H$

$$\dfrac{\dfrac{\dfrac{TODO}{\gamma \vdash x : \tau_2} \quad \dfrac{}{\gamma \vdash 1000 : \tau_2}}{\gamma \vdash x > 1000 : \tau_2} \quad \dfrac{TODO}{\gamma \vdash x := 5 : \tau_2} \quad \dfrac{TODO}{\gamma \vdash y := y + x * 3 : \tau_2}}{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : \tau_1}$$

Constraints: $\tau_1 \sqsubseteq \tau_2$

Fitting Rule

(VAR) $\dfrac{}{\gamma \vdash x : \tau} \; \gamma(x) \sqsubseteq \tau$

Example

- Let $\gamma(x) = L$ and $\gamma(y) = H$

$$\dfrac{\dfrac{\dfrac{}{\gamma \vdash x : \tau_2} \quad \dfrac{}{\gamma \vdash 1000 : \tau_2}}{\gamma \vdash x > 1000 : \tau_2} \quad \dfrac{TODO}{\gamma \vdash x := 5 : \tau_2} \quad \dfrac{TODO}{\gamma \vdash y := y + x * 3 : \tau_2}}{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : \tau_1}$$

Constraints: $\tau_1 \sqsubseteq \tau_2 \qquad L \sqsubseteq \tau_2$ (trivially true)

Fitting Rule

(ASSIGN) $\dfrac{\gamma \vdash e : \tau}{\gamma \vdash x := e : \tau'} \; \gamma(x) = \tau, \tau' \sqsubseteq \tau$

Example

- Let $\gamma(x) = L$ and $\gamma(y) = H$

$$\dfrac{\dfrac{\dfrac{}{\gamma \vdash x : L} \quad \dfrac{}{\gamma \vdash 1000 : L}}{\gamma \vdash x > 1000 : L} \qquad \dfrac{\dfrac{}{\gamma \vdash 5 : L}}{\gamma \vdash x := 5 : L} \quad \dfrac{TODO}{\gamma \vdash y := y + x * 3 : L}}{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : L}$$

Constraints: $\tau_1 \sqsubseteq \tau_2 \wedge \tau_2 \sqsubseteq L = \gamma(x)$. Thus $\tau_1 = \tau_2 = L$.

Fitting Rule

$$(\text{ASSIGN}) \ \dfrac{\gamma \vdash e : \tau}{\gamma \vdash x := e : \tau'} \ \gamma(x) = \tau, \tau' \sqsubseteq \tau$$

Example

- Let $\gamma(x) = L$ and $\gamma(y) = H$

$$\frac{\dfrac{\ldots}{\gamma \vdash x > 1000 : L} \quad \dfrac{\ldots}{\gamma \vdash x := 5 : L} \quad \dfrac{TODO}{\gamma \vdash y := y + x * 3 : L}}{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : L}$$

Constraints:

Fitting Rule

$$(\text{ASSIGN}) \ \frac{\gamma \vdash e : \tau}{\gamma \vdash x := e : \tau'} \ \gamma(x) = \tau, \tau' \sqsubseteq \tau$$

Example

- Let $\gamma(x) = L$ and $\gamma(y) = H$

$$
\dfrac{\dfrac{\dots}{\gamma \vdash x > 1000 : L} \quad \dfrac{\dots}{\gamma \vdash x := 5 : L} \quad \dfrac{\dfrac{\dfrac{}{\gamma \vdash y : H} \quad \dfrac{\dfrac{}{\gamma \vdash x : H} \quad \dfrac{}{\gamma \vdash 3 : H}}{\gamma \vdash x * 3 : H}}{\gamma \vdash y + x * 3 : H}}{\gamma \vdash y := y + x * 3 : L}}{\gamma \vdash \text{if } x > 1000 \text{ then } x := 5 \text{ else } y := y + x * 3 : L}
$$

Constraints: $\gamma(y) \sqsupseteq L$

Fitting Rule

(VAR) $\dfrac{}{\gamma \vdash x : \tau} \; \gamma(x) \sqsubseteq \tau$

It types!

Semantics

- To make a meaningful statement about the language, we must define what programs mean.
- Since we are reading and writing variables in our program, we need a concept of memory μ:
  - ★ $\mu(x)$ means reading the memory location for variable x (yields an integer if x is defined)
  - ★ $\mu[x := n]$ means writing the integer n into the memory location of variable x
- We define the semantics as two relations:
  - ★ For expressions e we define

    $$\mu \vdash e \Rightarrow n$$

    to mean: at memory state μ, the expression e gets evaluated to value n.
  - ★ For commands c we define

    $$\mu \vdash c \Rightarrow \mu'$$

    to mean: starting at memory state μ, the execution of command c changes the memory state to μ'.

Semantics

Definition (Semantics of Expressions)

$$\text{(BASE)} \quad \frac{}{\mu \vdash n \Rightarrow n}$$

$$\text{(CONTENT)} \quad \frac{}{\mu \vdash x \Rightarrow \mu(x)}$$

$$\text{(ADD)} \quad \frac{\mu \vdash e \Rightarrow n \quad \mu \vdash e' \Rightarrow n'}{\mu \vdash e + e' \Rightarrow n + n'}$$

Similar for other operators

Example

$$\underbrace{\boxed{\begin{array}{l} x := 3 \\ y := -1 \end{array}}}_{\mu} \vdash \underbrace{3 * x + y}_{e} \Rightarrow 8$$

Semantics

Definition (Semantics of Commands)

$$\text{(UPDATE)}\quad \frac{\mu \vdash e \Rightarrow n}{\mu \vdash x := e \Rightarrow \mu[x := n]}$$

$$\text{(SEQUENCE)}\quad \frac{\mu \vdash c_1 \Rightarrow \mu' \quad \mu' \vdash c_2 \Rightarrow \mu''}{\mu \vdash c_1; c_2 \Rightarrow \mu''}$$

$$\text{(IFTRUE)}\quad \frac{\mu \vdash e \Rightarrow 1 \quad \mu \vdash c_1 \Rightarrow \mu'}{\mu \vdash \text{if } e \text{ then } c_1 \text{ else } c_2 \Rightarrow \mu'}$$

$$\text{(IFFALSE)}\quad \frac{\mu \vdash e \Rightarrow 0 \quad \mu \vdash c_2 \Rightarrow \mu'}{\mu \vdash \text{if } e \text{ then } c_1 \text{ else } c_2 \Rightarrow \mu'}$$

Definition (Semantics of Commands)

$$\text{(WHILETRUE)} \quad \frac{\mu \vdash e \Rightarrow 1 \quad \mu \vdash c; \text{while } e \text{ do } c \Rightarrow \mu'}{\mu \vdash \text{while } e \text{ do } c \Rightarrow \mu'}$$

$$\text{(WHILEFALSE)} \quad \frac{\mu \vdash e \Rightarrow 0}{\mu \vdash \text{while } e \text{ do } c \Rightarrow \mu}$$

Example

$$\underbrace{\boxed{\begin{array}{l} x := 2 \\ y := 3 \end{array}}}_{\mu} \vdash \underbrace{\begin{array}{l} z := 1; \\ \text{while } y > 0 \text{ do} \\ \quad z := z * x; \quad y := y - 1 \end{array}}_{c} \Rightarrow \underbrace{\boxed{\begin{array}{l} x := 2 \\ y := 0 \\ z := 8 \end{array}}}_{\mu'}$$

Result

Theorem (Non-Interference instantiated for $L \sqsubseteq H$-security)

- *Suppose a program c satisfies information flow policy γ:*

  $$\gamma \vdash c : \tau \qquad \textit{(for any type } \tau\textit{, does not matter)}$$

- *Suppose μ_1 and μ_2 are memories that are equal on all low variables:*

  $$\mu_1(x) = \mu_2(x) \textit{ for every } x \textit{ with } \gamma(x) = L$$

- *If we run the program on these memories:*
  - $\mu_1 \vdash c \Rightarrow \mu_1'$
  - $\mu_2 \vdash c \Rightarrow \mu_2'$

  (and they both terminate)

- *...then the result will be the same on all low variables:*

  $$\mu_1'(x) = \mu_2'(x) \textit{ for every } x \textit{ with } \gamma(x) = L$$

Illustration

$$\begin{array}{l} \mu_1 \\ x := 42 \\ y := 5 \\ z := 3 \end{array} \xRightarrow[c]{} \begin{array}{l} \mu_1' \\ x := 12 \\ y := 0 \\ z := 7 \end{array}$$

$$\begin{array}{l} \mu_2 \\ x := 24 \\ y := 2 \\ z := 3 \end{array} \xRightarrow[c]{} \begin{array}{l} \mu_2' \\ x := 7 \\ y := 3 \\ z := 7 \end{array}$$

- If the low-variables (in the example: z) of μ_1 and μ_2 agree,
- then they still agree after running any command c that satisfies information flow.

Illustration

$$\begin{array}{l}\mu_1\\ \boxed{\begin{array}{l}x := 42\\ y := 5\end{array}}\\ z := 3\end{array} \xRightarrow[c]{} \begin{array}{l}\mu_1'\\ \boxed{\begin{array}{l}x := 12\\ y := 0\end{array}}\\ z := 7\end{array}$$

$$\begin{array}{l}\mu_2\\ \boxed{\begin{array}{l}x := 24\\ y := 2\end{array}}\\ z := 3\end{array} \xRightarrow[c]{} \begin{array}{l}\mu_2'\\ \boxed{\begin{array}{l}x := 7\\ y := 3\end{array}}\\ z := 7\end{array}$$

- If the low-variables (in the example: z) of μ_1 and μ_2 agree,
- then they still agree after running any command c that satisfies information flow.
- Thus, from observing the low variables we get no information about the high variables.

Relation to Secrecy and Privacy

- Note that we do not assume that the values in the high variables are strong secrets. They may contain...
  - personal information like name, age, CPR, medical data...
  - a guessable password
  - a credit card number and expiration information
  - a vote

Relation to Secrecy and Privacy

- Note that we do not assume that the values in the high variables are strong secrets. They may contain...
  - ★ personal information like name, age, CPR, medical data...
  - ★ a guessable password
  - ★ a credit card number and expiration information
  - ★ a vote
- Information Flow/Noninterference guarantees that the intruder does not learn anything about it as long as he can only read low variables.

Relation to Secrecy and Privacy

- Note that we do not assume that the values in the high variables are strong secrets. They may contain...
  - personal information like name, age, CPR, medical data...
  - a guessable password
  - a credit card number and expiration information
  - a vote
- Information Flow/Noninterference guarantees that the intruder does not learn anything about it as long as he can only read low variables.
- It is thus...
  - ... not like secrecy in OFMC
  - ... a bit like guessable secrecy in OFMC
  - ... close to α-β-privacy!

Noninterference vs. α-β-privacy

$$\begin{array}{c} \mu_1 \\ \boxed{\begin{array}{l} x := 42 \\ y := 5 \end{array}} \\ z := 3 \end{array} \xrightarrow[c]{\quad} \begin{array}{c} \mu_1' \\ \boxed{\begin{array}{l} x := 12 \\ y := 0 \end{array}} \\ z := 7 \end{array}$$

- Idea for encoding:
  - ★ α says that initially x and y are *some* integer values
  - ★ β contains the values of z and how the algorithm manipulates x, y, and z.[1]

[1] This does not directly work on while-loops with conditions of type High.

Noninterference vs. α-β-privacy

$$
\begin{array}{l}
\mu_1 \\
x := 42 \\
y := 5 \\
z := 3
\end{array}
\quad \overset{}{\underset{c}{\Longrightarrow}} \quad
\begin{array}{l}
\mu_1' \\
x := 12 \\
y := 0 \\
z := 7
\end{array}
$$

- Idea for encoding:
  - ★ α says that initially x and y are *some* integer values
  - ★ β contains the values of z and how the algorithm manipulates x, y, and z.[1]
- It is a violation of privacy if the intruder can tell anything about the variables x and y.

[1] This does not directly work on while-loops with conditions of type High.

Noninterference vs. α-β-privacy

$$\begin{array}{lcl} \mu_1 & & \mu_1' \\ x := 42 & & x := 12 \\ y := 5 & \xRightarrow[c]{} & y := 0 \\ z := 3 & & z := 7 \end{array}$$

- Idea for encoding:
  - ★ α says that initially x and y are *some* integer values
  - ★ β contains the values of z and how the algorithm manipulates x, y, and z.[1]
- It is a violation of privacy if the intruder can tell anything about the variables x and y.
- α-β privacy allows to release some information:
  - ★ E.g. the intruder may learn an encrypted message containing a High value (as long as he cannot decrypt)
  - ★ E.g. we may release the total result of an election

[1] This does not directly work on while-loops with conditions of type High.

Noninteference for Authentication / Integrity

μ_1		μ'_1
$x := 42$		$x := 12$
$y := 5$	$\xRightarrow{c}$	$y := 0$
$z := 3$		$z := 7$

μ_2		μ'_2
$x := 24$		$x := 7$
$y := 2$	$\xRightarrow{c}$	$y := 3$
$z := 3$		$z := 7$

- Suppose we define the variables x and y as type Untrusted and z as type Trusted.
- Suppose *Trusted* $\sqsubseteq$ *Untrusted*.

Noninteference for Authentication / Integrity

$$\begin{array}{c} \mu_1 \\ \begin{array}{|l|} \hline x := 42 \\ y := 5 \\ \hline z := 3 \\ \hline \end{array} \end{array} \xRightarrow[c]{} \begin{array}{c} \mu_1' \\ \begin{array}{|l|} \hline x := 12 \\ y := 0 \\ \hline z := 7 \\ \hline \end{array} \end{array}$$

$$\begin{array}{c} \mu_2 \\ \begin{array}{|l|} \hline x := 24 \\ y := 2 \\ \hline z := 3 \\ \hline \end{array} \end{array} \xRightarrow[c]{} \begin{array}{c} \mu_2' \\ \begin{array}{|l|} \hline x := 7 \\ y := 3 \\ \hline z := 7 \\ \hline \end{array} \end{array}$$

- Suppose we define the variables x and y as type Untrusted and z as type Trusted.
- Suppose *Trusted* $\sqsubseteq$ *Untrusted*.
- Suppose further the intruder can manipulate the untrusted variables, but not the trusted ones.

Noninteference for Authentication / Integrity

$$\begin{array}{c} \mu_1 \\ \begin{array}{l} x := 42 \\ y := 5 \\ z := 3 \end{array} \end{array} \underset{c}{\Longrightarrow} \begin{array}{c} \mu_1' \\ \begin{array}{l} x := 12 \\ y := 0 \\ z := 7 \end{array} \end{array}$$

$$\begin{array}{c} \mu_2 \\ \begin{array}{l} x := 24 \\ y := 2 \\ z := 3 \end{array} \end{array} \underset{c}{\Longrightarrow} \begin{array}{c} \mu_2' \\ \begin{array}{l} x := 7 \\ y := 3 \\ z := 7 \end{array} \end{array}$$

- Suppose we define the variables x and y as type Untrusted and z as type Trusted.
- Suppose *Trusted* $\sqsubseteq$ *Untrusted*.
- Suppose further the intruder can manipulate the untrusted variables, but not the trusted ones.
- Then $\gamma \vdash c : \tau$ guarantees us that the intruder cannot do anything that has any influence on the trusted variables.

Result

Theorem (Non-Interference instantiated for T$\sqsubseteq$U)

- *Suppose a program c satisfies information flow policy γ:*

$$\gamma \vdash c : \tau \qquad \textit{(for any type } \tau\textit{, does not matter)}$$

- *Suppose μ_1 and μ_2 are memories that are equal on all trusted variables:*

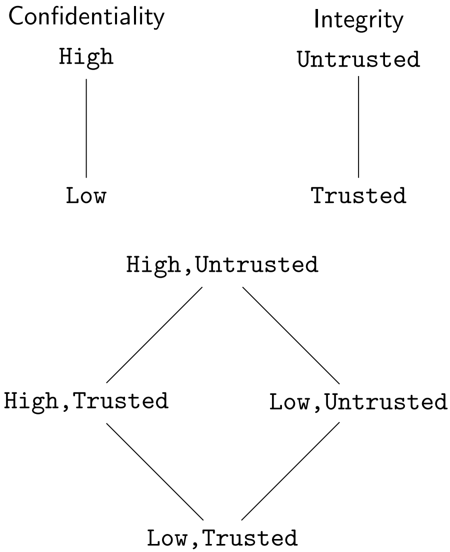
$$\mu_1(x) = \mu_2(x) \text{ for every } x \text{ with } \gamma(x) = T$$

- *If we run the program on these memories:*
  - ★ $\mu_1 \vdash c \Rightarrow \mu_1'$
  - ★ $\mu_2 \vdash c \Rightarrow \mu_2'$

  (and they both terminate)
- *…then the result will be the same on all trusted variables:*

$$\mu_1'(x) = \mu_2'(x) \text{ for every } x \text{ with } \gamma(x) = T$$

Confidentiality and Integrity in One Go?



Question

How can we become more general?

Question

How can we become more general?

Answer

Security Policy Frameworks
for Mandatory Access Control.

Security Policy Framework

A **security policy framework** is a 4-tuple $(S, \sqsubseteq, \sqcup, \sqcap)$ where

- S is a **finite** and **non-empty** set of **security labels**.

Security Policy Framework

A **security policy framework** is a 4-tuple $(S,\sqsubseteq,\sqcup,\sqcap)$ where

- S is a **finite** and **non-empty** set of **security labels**.
- $\sqsubseteq$: $S \times S$ is a **binary relation**
  - (a) $\sqsubseteq$ is **reflexive** : for all $s \in S : s \sqsubseteq s$
  - (b) $\sqsubseteq$ is **transitive**: for all $s_1,\ s_2,\ s_3 \in S$:

    $$s_1 \sqsubseteq s_2 \wedge s_2 \sqsubseteq s_3 \Rightarrow s_1 \sqsubseteq s_3$$
  - (c) $\sqsubseteq$ is **anti-symmetric**: for all $s_1,\ s_2 \in S$:

    $$s_1 \sqsubseteq s_2 \wedge s_1 \sqsupseteq s_2 \implies s_1 = s_2$$

Security Policy Framework

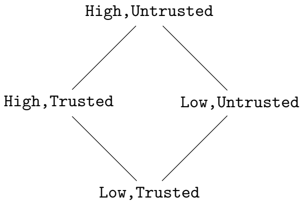
A **security policy framework** is a 4-tuple $(S, \sqsubseteq, \sqcup, \sqcap)$ where

- S is a **finite** and **non-empty** set of **security labels**.
- $\sqsubseteq$: $S \times S$ is a **binary relation**
  - (a) $\sqsubseteq$ is **reflexive** : for all $s \in S : s \sqsubseteq s$
  - (b) $\sqsubseteq$ is **transitive**: for all $s_1, s_2, s_3 \in S$:
    $$s_1 \sqsubseteq s_2 \wedge s_2 \sqsubseteq s_3 \Rightarrow s_1 \sqsubseteq s_3$$
  - (c) $\sqsubseteq$ is **anti-symmetric**: for all $s_1, s_2 \in S$:
    $$s_1 \sqsubseteq s_2 \wedge s_1 \sqsupseteq s_2 \implies s_1 = s_2$$
- $\sqcup : S \times S \to S$ and $\sqcap : S \times S \to S$ are two operations for **combining labels** such that

  for all $s_1, s_2 \in S$:
  $$s_1 \sqsubseteq s_1 \sqcup s_2 \text{ and } s_2 \sqsubseteq s_1 \sqcup s_2$$
  $$s_1 \sqcap s_2 \sqsubseteq s_1 \text{ and } s_1 \sqcap s_2 \sqsubseteq s_2$$

Security Lattice

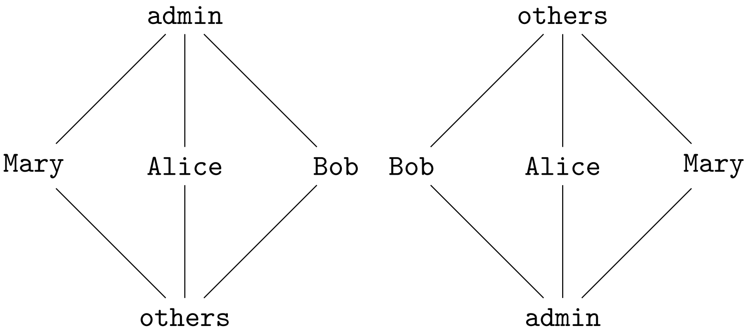
$(S, \sqsubseteq)$ is a *lattice*. This allows to depict it nicely by just denoting the direct successors of security classes:



In a security framework $(S,\sqsubseteq,\sqcup,\sqcap)$

- there is the bottom security label $\perp$ that can be obtained as $s_1 \sqcap\ s_2 \ldots \sqcap s_n$ (if $S = \{s_1, \ldots, s_n\}$).
- there is the top security label $\top$ that can be obtained as $s_1 \sqcup\ s_2 \ldots \sqcup s_n$ (if $S = \{s_1, \ldots, s_n\}$).

Example



The set of security labels

$$S = \{\texttt{others}, \texttt{Mary}, \texttt{Alice}, \texttt{Bob}, \texttt{admin}\}$$

What about $\sqsubseteq$, $\sqcup$ and $\sqcap$?

Typical Security Policy Frameworks
Components

Start with

- A finite and non-empty set C of **security categories**.

Typical Security Policy Frameworks
Components

Start with

- A finite and non-empty set C of **security categories**.

Security labels can be all subsets of C:

- $(\mathrm{PowerSet}(C),\supseteq,\cap,\cup)$ for **confidentiality policies**.
- $(\mathrm{PowerSet}(C),\subseteq,\cup,\cap)$ for **integrity policies**.

A security label $s \in \mathrm{Powerset}(C)$ is called a **component**.

Recall $\mathrm{Powerset}(C) = \{C_0 \mid C_0 \subseteq C\}$, i.e., the set of subsets of C.

Example (Confidentiality): (PowerSet(C),$\supseteq$,$\cap$,$\cup$)

$$
\begin{tikzcd}[column sep=small]
 & \{\} \arrow[dl, dash] \arrow[dr, dash] & \\
\{\texttt{medical}\} \arrow[dr, dash] & & \{\texttt{finance}\} \arrow[dl, dash] \\
 & \{\texttt{finance}, \texttt{medical}\} &
\end{tikzcd}
$$

Example (Confidentiality): (PowerSet(C),$\supseteq$,$\cap$,$\cup$)

$$
\begin{tikzcd}[column sep=small]
& \{\} \arrow[dl, dash] \arrow[dr, dash] & \\
\{\texttt{medical}\} \arrow[dr, dash] & & \{\texttt{finance}\} \arrow[dl, dash] \\
& \{\texttt{finance}, \texttt{medical}\} &
\end{tikzcd}
$$

- $C = \{\texttt{finance}, \texttt{medical}\}$
- The **more departments** can **read** to the data, the **less sensitive** it is.

Example (Confidentiality): $(\mathsf{PowerSet}(C), \supseteq, \cap, \cup)$

$$\begin{tikzcd}
 & \{\} \arrow[dl, dash] \arrow[dr, dash] & \\
\{\texttt{medical}\} \arrow[dr, dash] & & \{\texttt{finance}\} \arrow[dl, dash] \\
 & \{\texttt{finance}, \texttt{medical}\} &
\end{tikzcd}$$

- $C = \{\texttt{finance}, \texttt{medical}\}$
- The **more departments** can **read** to the data, the **less sensitive** it is.
- Ordering labels examples ($\sqsubseteq\, = \,\supseteq$):
  $\{\texttt{finance}, \texttt{medical}\} \sqsubseteq \{\texttt{medical}\}$, $\{\texttt{medical}\} \not\sqsubseteq \{\texttt{finance}\}$,
  $\{\texttt{finance}\} \not\sqsubseteq \{\texttt{medical}\}$

Example (Confidentiality): (PowerSet(C),$\supseteq$,$\cap$,$\cup$)

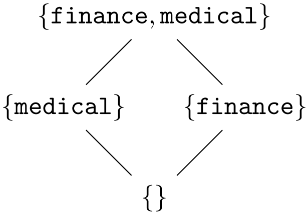
$$\begin{tikzcd}
 & \{\} \arrow[dl, no head] \arrow[dr, no head] & \\
\{\mathtt{medical}\} \arrow[dr, no head] & & \{\mathtt{finance}\} \arrow[dl, no head] \\
 & \{\mathtt{finance}, \mathtt{medical}\} &
\end{tikzcd}$$

- $C = \{\mathtt{finance}, \mathtt{medical}\}$
- The **more departments** can **read** to the data, the **less sensitive** it is.
- Ordering labels examples ($\sqsubseteq\; = \;\supseteq$):
  $\{\mathtt{finance}, \mathtt{medical}\} \sqsubseteq \{\mathtt{medical}\}$, $\{\mathtt{medical}\} \not\sqsubseteq \{\mathtt{finance}\}$,
  $\{\mathtt{finance}\} \not\sqsubseteq \{\mathtt{medical}\}$
- Combining labels examples ($\sqcup = \cap$, $\sqcap = \cup$):
  $\{\mathtt{finance}, \mathtt{medical}\} \sqcup \{\mathtt{medical}\} = \{\mathtt{medical}\}$
  $\{\mathtt{medical}\} \sqcap \{\mathtt{finance}\} = \{\mathtt{finance}, \mathtt{medical}\}$

Example (Integrity): (PowerSet(C),$\subseteq$,$\cup$,$\cap$)

$$\begin{tikzcd}[column sep=small]
 & \{\texttt{finance}, \texttt{medical}\} \arrow[dl, dash] \arrow[dr, dash] & \\
\{\texttt{medical}\} \arrow[dr, dash] & & \{\texttt{finance}\} \arrow[dl, dash] \\
 & \{\} &
\end{tikzcd}$$

Example (Integrity): (PowerSet(C),$\subseteq$,$\cup$,$\cap$)



- $C = \{\texttt{finance}, \texttt{medical}\}$
- The **more departments** can **write** to the data, the **less trustworthy** it is.

Example (Integrity): (PowerSet(C),$\subseteq$,$\cup$,$\cap$)

$$\begin{tikzcd}[column sep=small]
 & \{\mathtt{finance}, \mathtt{medical}\} \arrow[dl, no head] \arrow[dr, no head] & \\
\{\mathtt{medical}\} \arrow[dr, no head] & & \{\mathtt{finance}\} \arrow[dl, no head] \\
 & \{\} &
\end{tikzcd}$$

- $C = \{\mathtt{finance}, \mathtt{medical}\}$
- The **more departments** can **write** to the data, the **less trustworthy** it is.
- Ordering labels examples ($\sqsubseteq \, = \, \subseteq$):
  $\{\mathtt{medical}\} \sqsubseteq \{\mathtt{finance}, \mathtt{medical}\}$, $\{\mathtt{medical}\} \not\sqsubseteq \{\mathtt{finance}\}$,
  $\{\mathtt{finance}\} \not\sqsubseteq \{\mathtt{medical}\}$

Example (Integrity): (PowerSet(C),$\subseteq$,$\cup$,$\cap$)

$$\begin{tikzcd}[column sep=small]
 & \{\mathtt{finance}, \mathtt{medical}\} \arrow[dl, dash] \arrow[dr, dash] & \\
\{\mathtt{medical}\} \arrow[dr, dash] & & \{\mathtt{finance}\} \arrow[dl, dash] \\
 & \{\} &
\end{tikzcd}$$

- $C = \{\mathtt{finance}, \mathtt{medical}\}$
- The **more departments** can **write** to the data, the **less trustworthy** it is.
- Ordering labels examples ($\sqsubseteq\, = \,\subseteq$):
  $\{\mathtt{medical}\} \sqsubseteq \{\mathtt{finance}, \mathtt{medical}\}$, $\{\mathtt{medical}\} \not\sqsubseteq \{\mathtt{finance}\}$,
  $\{\mathtt{finance}\} \not\sqsubseteq \{\mathtt{medical}\}$
- Combining labels examples ($\sqcup = \cup$, $\sqcap = \cap$):
  $\{\mathtt{finance}\} \sqcup \{\mathtt{medical}\} = \{\mathtt{finance}, \mathtt{medical}\}$
  $\{\mathtt{medical}\} \sqcap \{\mathtt{finance}\} = \{\}$

Combining Security Policy Frameworks
Product Construction

Whenever $(S_1,\sqsubseteq_1,\sqcup_1,\sqcap_1)$ and $(S_2,\sqsubseteq_2,\sqcup_2,\sqcap_2)$ are two security frameworks, we can construct the framework $(S,\sqsubseteq,\sqcup,\sqcap)$ where

- $S = S_1 \times S_2$
- for $(s_{11}, s_{12}), (s_{21}, s_{22}) \in S$:

$$(s_{11}, s_{12}) \sqsubseteq (s_{21}, s_{22}) \text{ iff } s_{11} \sqsubseteq_1 s_{21} \wedge s_{12} \sqsubseteq_2 s_{22}$$

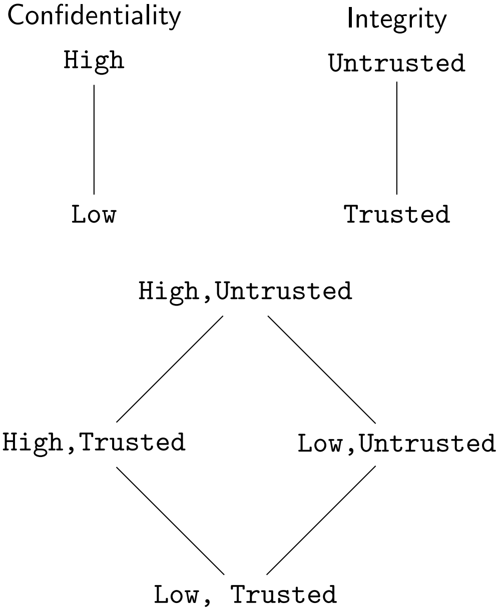
- for $(s_{11}, s_{12}), (s_{21}, s_{22}) \in S$:

$$(s_{11}, s_{12}) \sqcup (s_{21}, s_{22}) = (s_{11} \sqcup_1 s_{21}, s_{12} \sqcup_2 s_{22})$$

  and

$$(s_{11}, s_{12}) \sqcap (s_{21}, s_{22}) = (s_{11} \sqcap_1 s_{21}, s_{12} \sqcap_2 s_{22})$$

Example (Combining Integrity and Confidentiality):



Result

Theorem (Non-Interference for an arbitrary Security Policy Framework)

- *Suppose a program c satisfies information flow policy γ:*

$$\gamma \vdash c : \tau \qquad \textit{(for any type } \tau\textit{, does not matter)}$$

- *Suppose μ_1 and μ_2 are memories that are equal on all variables up to level τ_0:*

$$\mu_1(x) = \mu_2(x) \text{ for every } x \text{ with } \gamma(x) \sqsubseteq \tau_0$$

- *If we run the program on these memories:*
  - ★ $\mu_1 \vdash c \Rightarrow \mu_1'$
  - ★ $\mu_2 \vdash c \Rightarrow \mu_2'$

  (and they both terminate)

- *…then the result will be the same on all variables up to level τ_0:*

$$\mu_1'(x) = \mu_2'(x) \text{ for every } x \text{ with } \gamma(x) \sqsubseteq \tau_0$$

Review: Myers' Security Labels

We have the security framework ($P \hookrightarrow \text{PowerSet}(P)$, $\sqsubseteq$, $\sqcup$, $\sqcap$) where

- $P \hookrightarrow \text{PowerSet}(P)$ is the set of all **partial mappings** from P to PowerSet(P).
- For a label s we define $\text{Owners}(s) = \text{Domain}(s)$
- For a security label s and principal p define
  $\text{Readers}(s, p) = \begin{cases} s(p) & \text{if } p \in \text{Owners}(s) \\ P & \text{if } p \notin \text{Owners}(s) \end{cases}$
- Alternative notation (bit easier to read):

$$\{(A : A, C), (B : B, C)\} \text{ for } \{A : \{A, C\}, B : \{B, C\}\}$$

Review: Myers' Security Labels/Ordering

- for two security labels s_1, s_2 we have
  - ★ $s_1 \sqsubseteq s_2$ iff
    $\mathrm{Owners}(s_1) \subseteq \mathrm{Owners}(s_2)$ and
    $\mathrm{Readers}(s_1, o) \supseteq \mathrm{Readers}(s_2, o)$ for every $o \in \mathrm{Owners}(s_1)$
  - ★ $s_1 \sqcup s_2$ such that
    $$\mathrm{Owners}(s_1 \sqcup s_2) = \mathrm{Owners}(s_1) \cup \mathrm{Owners}(s_2)$$
    $$\mathrm{Readers}(s_1 \sqcup s_2, o) = \mathrm{Readers}(s_1, o) \cap \mathrm{Readers}(s_2, o) \quad \text{for every } o \in \mathrm{Owners}(s_1 \sqcup s_2)$$
  - ★ $s_1 \sqcap s_2$ such that
    $$\mathrm{Owners}(s_1 \sqcap s_2) = \mathrm{Owners}(s_1) \cap \mathrm{Owners}(s_2)$$
    $$\mathrm{Readers}(s_1 \sqcap s_2, o) = \mathrm{Readers}(s_1, o) \cup \mathrm{Readers}(s_2, o) \quad \text{for every owner } o \in \mathrm{Owners}(s_1 \sqcap s_2)$$

Integrity

The Myers-Liskov approach also allows for integrity –
For integrity one can use the dual definitions:

- Instead of Readers we have Writers
- $s_1 \sqsubseteq s_2$ iff $\quad$ $\mathrm{Owners}(s_1) \supseteq \mathrm{Owners}(s_2)$ and
  $\forall o \in \mathrm{Owners}(s_2) : \mathrm{Writers}(s_1, o) \subseteq \mathrm{Writers}(s_2, o)$
- $s_1 \sqcup s_2$ s.t. $\quad$ $\mathrm{Owners}(s_1 \sqcup s_2) = \mathrm{Owners}(s_1) \cap \mathrm{Owners}(s_2)$
  $\mathrm{Writers}(s_1 \sqcup s_2) = \mathrm{Writers}(s_1, o) \cup \mathrm{Writers}(s_2, o)$
- $s_1 \sqcap s_2$ s.t. $\quad$ $\mathrm{Owners}(s_1 \sqcap s_2) = \mathrm{Owners}(s_1) \cup \mathrm{Owners}(s_2)$
  $\mathrm{Writers}(s_1 \sqcap s_2) = \mathrm{Writers}(s_1, o) \cap \mathrm{Writers}(s_2, o)$

Declassification for Integrity?

- A core feature of decentralized label model is declassification for confidentiality.
- For integrity, declassification does not really make sense.
- Since everything in integrity "mirrors" confidentiality, is there something like declassification for integrity?

Declassification for Integrity?

- A core feature of decentralized label model is declassification for confidentiality.
- For integrity, declassification does not really make sense.
- Since everything in integrity “mirrors” confidentiality, is there something like declassification for integrity?

Yes:

Endorsement

- Declassification: we make the exception that something confidential is allowed to become public.
- Endorsement: we make the exception that something untrusted is now considered trusted.

Example: two untrusted but independent sources tell the same story.

Security Policy for the Assignment

Remember that for the assignment you are asked to use Myers-Liskov decentralized label model:

- You can use decentralized confidentiality labels (owners and readers)
- You can use decentralized integrity labels (owners and writers)
- You can use the product lattice of the two
- You can use declassification and endorsement, but only in the way defined by Myers-Liskov – an owner can relax or remove their own constraint.

Challenge

Let $\gamma = [x \mapsto \mathtt{Low}, y \mapsto \mathtt{High}]$ and the context $\mathsf{ctx} = \mathtt{Low}$

- Is the program $\mathsf{S} = (\textbf{while } y > 0 \textbf{ do } y := y + 1)\ ;\ x := 2$ secure?

Use Volpano's typing rules to determine if S is secure

- Do you agree with the result of the type system?

References I

D. E. Denning and P. J. Denning.
Certification of programs for secure information flow.
Commun. ACM, 20(7):504–513, 1977.

D. M. Volpano, G. Smith, and C. E. Irvine.
A sound type system for secure flow analysis.
Journal of Computer Security, 4(2/3):167–188, 1996.