

Home Page

Print

Title Page

Contents

◀ ▶

◀ ▶

Page 1 of 24

Go Back

Full Screen

Close

Quit

Gérard Boudol & Marija Kolundžija

Access Control and Declassification

[Home Page](#)

[Print](#)

[Title Page](#)

[Contents](#)



[Page 2 of 24](#)

[Go Back](#)

[Full Screen](#)

[Close](#)

[Quit](#)

Contents

1	Access Control	4
2	Information Flow	5
3	Declassification	6
4	Integrated Approach	7
5	Our Integrated Approach	9
6	The Language	11
7	Operational Semantics	13
8	The Interdependence	15
9	The Non-Disclosure Policy	16
10	The Type and Effect System	17
11	Security Effects	18
12	Sample Rules	19
13	Type Soundness – Secure Information Flow	22
14	Future work	23

Home Page

Print

Title Page

Contents



Page 3 of 24

Go Back

Full Screen

Close

Quit

- **Language-based security** – static checking of access rights and information flow
- We address the issues of:
 - **Access control** – whether a user is allowed to access an object
 - **Information flow** – whether the secret information is passed to public outputs
 - **Declassification** – enabling *useful information leaks*

Home Page

Print

Title Page

Contents

◀ ▶

◀ ▶

Page 4 of 24

Go Back

Full Screen

Close

Quit

1. Access Control

- Involves sets of *principals* \mathcal{P} and a *binary relation* \succeq , and a notion of *trust*; the intuition for $p \succeq q$ is that *q trusts p*
- Controls which the user has right to read what information
- Usually not enough – no control over *propagation* of legally accessed information

Home Page

Print

Title Page

Contents



Page 5 of 24

Go Back

Full Screen

Close

Quit

2. Information Flow

- Information classified into *security levels* which form a lattice \mathcal{L} and a preorder relation \preceq ; intuition for $\ell \preceq \ell'$ is that *information can flow from ℓ to ℓ'*
- Changes in secret parts of memory should not be visible to public observers (**non-interference**)
- Non-interference considered too strict

Home Page

Print

Title Page

Contents

◀

▶

◀

▶

Page 6 of 24

Go Back

Full Screen

Close

Quit

3. Declassification

Intentional leaks can be harmless, even useful

```
Pwd_chk(usr_id, pwd) = if !(pwd_db.usr_id)Sec = pwd  
                        then outputPub := "welcome"  
                        then outputPub := "wrong!"
```

$\text{Sec} \not\leq \text{Pub}$, otherwise the entire password could be leaked.

Solution: Local flow declaration with limited scope
→ non-interference enforced at local level = *the non-disclosure policy* (Almeida Matos & Boudol [CSFW 2005])

▶ How to control (prevent) declassification?

4. Integrated Approach

Access control *and* information flow – independent or interdependent?

(\mathcal{P}, \succeq) vs. (\mathcal{L}, \preceq) – whether and how to relate them

Previous work:

- [Pottier, Skalka & Smith](#) [ACM TPLS 2005] and [Fournet & Gordon](#) [POPL 2002] – static access control checking for stack inspection algorithms
- [Almeida Matos & Boudol](#) [CSFW 2005] – introduction of the [non-disclosure](#) policy for handling declassification

Home Page

Print

Title Page

Contents

◀ ▶

◀ ▶

Page 8 of 24

Go Back

Full Screen

Close

Quit

- [Banerjee & Naumann](#) [JFP 2005] – static access control combined with information flow for a Java-like language, no declassification
- [Pottier & Conchon](#) [ICFP 2000] – information flow with access control mechanism in a functional setting – two mechanisms unrelated

Home Page

Print

Title Page

Contents



Page 9 of 24

Go Back

Full Screen

Close

Quit

5. Our Integrated Approach

- A set of *principals* \mathcal{P} (Ann, Bob, Charlie...)
- *Security levels* ($\ell \subseteq \mathcal{P}$) assigned to memory locations
– also represent who has access
- $\ell \subseteq \ell'$ means ℓ is *more restrictive* than ℓ' , i.e. $\ell' \preceq \ell$
- *Flow relation* $F \subseteq \mathcal{P} \times \mathcal{P}$ – generates *preorder*

$$\ell \preceq_F \ell' \Leftrightarrow_{\text{def}} \forall q \in \ell'. \exists p \in \ell. p F^* q$$

Home Page

Print

Title Page

Contents

◀ ▶

◀ ▶

Page 10 of 24

Go Back

Full Screen

Close

Quit

- Security levels are organized in **security lattices**
 - **meet**: $\ell \wedge \ell' = \ell \cup \ell'$
 - **join**: $\ell \vee \ell' = \{ q \mid \exists p \in \ell. \exists p' \in \ell'. p F^* q \ \& \ p' F^* q \}$

6. The Language

An ML-like language (functional + imperative), extended with [local flow declarations](#) (Almeida Matos & Boudol [CSFW 2005] without threads) and with [access control management constructs](#) (Pottier, Skalka & Smith [ACM TPLS 2005])

$$\begin{aligned}
 M, N \dots \in \mathcal{E}xpr & ::= V \mid (\text{if } M \text{ then } N \text{ else } N') \mid (MN) \\
 & \mid M ; N \mid (\text{ref}_{\ell, \theta} N) \mid (!N) \mid (M := N) \\
 & \mid (\text{flow } F \text{ in } M) \\
 & \mid (\ell \times M) \mid (\text{enable } \ell \text{ in } M) \\
 & \mid (\text{test } \ell \text{ then } M \text{ else } N) \\
 V \in \mathcal{V}al & ::= x \mid u_{\ell, \theta} \mid \text{rec } f(x).M \mid tt \mid ff \mid ()
 \end{aligned}$$

Home Page

Print

Title Page

Contents



Page 12 of 24

Go Back

Full Screen

Close

Quit

- global flow policy G
- default access right ℓ
- *scoping constructs*
 - $(\text{flow } F \text{ in } M)$ locally extends G with F
 - $(\ell' \times M)$ locally restricts access right of M to ℓ'
 - $(\text{enable } \ell' \text{ in } M)$ locally extends access right of M by ℓ'
- $(\text{test } \ell' \text{ then } M \text{ else } N)$ checks if ℓ' is locally enabled, and proceeds accordingly

7. Operational Semantics

Statements:

$$\ell \vdash_G (M, \mu) \rightarrow (M', \mu')$$

G – global flow policy, ℓ – default access right

Evaluation contexts:

$$\mathbf{E} ::= \square \mid \mathbf{E}[\mathbf{F}]$$

frames:

$$\begin{aligned} \mathbf{F} ::= & \text{(if } \square \text{ then } M \text{ else } N) \mid (\square N) \mid (V \square) \\ & \mid \square ; N \mid (\text{ref}_{\ell, \theta} \square) \mid (! \square) \mid (\square := N) \mid (V := \square) \\ & \mid (\ell \times \square) \mid (\text{enable } \ell \text{ in } \square) \mid (\text{flow } F \text{ in } \square) \end{aligned}$$

$\llbracket \mathbf{E} \rrbracket_\ell$ — access level granted by the context — a form of **stack inspection**:

$$\begin{aligned} \llbracket [] \rrbracket_\ell &= \ell \\ \llbracket \mathbf{E}[\mathbf{F}] \rrbracket_\ell &= \begin{cases} (\llbracket \mathbf{E} \rrbracket_\ell) \wedge_G \ell' & \text{if } \mathbf{F} = (\ell' \times []) \\ (\llbracket \mathbf{E} \rrbracket_\ell) \vee_G \ell' & \text{if } \mathbf{F} = (\text{enable } \ell' \text{ in } []) \\ \llbracket \mathbf{E} \rrbracket_\ell & \text{otherwise} \end{cases} \end{aligned}$$

Reduction rules dependent on the context access level:

$$\begin{aligned} \ell \vdash_G (\mathbf{E}[(! u_{\ell', \theta})], \mu) &\rightarrow (\mathbf{E}[V], \mu) \\ &\quad \mu(u_{\ell', \theta}) = V \quad \& \quad \ell' \preceq_G \llbracket \mathbf{E} \rrbracket_\ell \\ \ell \vdash_G (\mathbf{E}[(\text{test } \ell' \text{ then } M \text{ else } N)], \mu) &\rightarrow (\mathbf{E}[M], \mu) \\ &\quad \ell' \preceq_G \llbracket \mathbf{E} \rrbracket_\ell \\ \ell \vdash_G (\mathbf{E}[(\text{test } \ell' \text{ then } M \text{ else } N)], \mu) &\rightarrow (\mathbf{E}[N], \mu) \\ &\quad \ell' \not\preceq_G \llbracket \mathbf{E} \rrbracket_\ell \end{aligned}$$

Local flow policies do not affect read clearance!

Home Page

Print

Title Page

Contents



Page 15 of 24

Go Back

Full Screen

Close

Quit

8. The Interdependence

If the current read level is $\{L\}$, $\{H\} \not\leq \{L\}$

- ▶ (flow $H \prec L$ in $v_L := !u_H$) will be blocked – a program cannot declassify what it has no right to read
- ▶ expression $(v_L := !u_H)$ is *secure* from an information flow point of view

9. The Non-Disclosure Policy

- Access violations as runtime errors
- Non-interference at each small step w.r.t. [the current flow policy](#)
- Small-step reduction on configurations (*program, memory*) $\ell \vdash_G (P, \mu) \rightarrow (P', \mu')$
- Low-equality of memory $\mu \simeq^{F, \ell} \mu'$ relative to [the current flow policy](#) F is preserved with each reduction step
- [Bisimulation-based formulation](#)
 A process P is **secure** from the confidentiality point of view with respect to the default access level ℓ and the global flow policy G if it satisfies $P \sqsubset^{\ell, G, \ell'} P$ for all ℓ' , i.e. $P \in \mathcal{ND}(\ell, G)$.

10. The Type and Effect System

► Judgements

$$\ell; F; \Gamma \vdash_G M : s, \tau$$

ℓ – current read clearance

F – local flow policy

Γ – typing context, assigning types to variables

G – global flow policy

s – security effect – a triple (ℓ_0, ℓ_1, ℓ_2) of confidentiality levels

► Types

$$\tau, \sigma, \theta \dots ::= t \mid \text{bool} \mid \text{unit} \mid \theta \text{ref}_\ell \mid \left(\tau \xrightarrow[\ell, F]{s} \sigma \right)$$

11. Security Effects

$$s = (\ell_0, \ell_1, \ell_2)$$

- ℓ_0 or *s.c* is the *confidentiality level* of M – an upper bound (up to the current flow relation) of the confidentiality levels of the references the expression M reads that may influence its *resulting value*
- ℓ_1 or *s.w* is the *writing effect*, that is a lower bound (w.r.t. the relation \preceq) of the level of references that the expression M may *update*
- ℓ_2 or *s.t* is the *termination effect* – an upper bound (w.r.t. the current flow relation) of the levels of the references the expression M reads that may influence its *termination*

12. Sample Rules

$$\frac{\ell; F; \Gamma \vdash_G M : s, \theta \text{ ref}_{\ell'} \quad \ell' \preceq_G \ell}{\ell; F; \Gamma \vdash_G (!M) : s \Upsilon (\ell', \top, \perp), \theta} \text{ (DEREF)}$$

$$\frac{\ell \wedge_G \ell'; F; \Gamma \vdash_G M : s, \tau}{\ell; F; \Gamma \vdash_G (\ell' \times M) : s, \tau} \text{ (RESTRIC)}$$

$$\frac{\ell \Upsilon_G \ell'; F; \Gamma \vdash_G M : s, \tau}{\ell; F; \Gamma \vdash_G (\text{enable } \ell' \text{ in } M) : s, \tau} \text{ (ENABLE)}$$

$$\frac{\ell'; F; \Gamma \vdash_G M : s, \tau \quad \ell; F; \Gamma \vdash_G N : s', \tau}{\ell; F; \Gamma \vdash_G (\text{test } \ell' \text{ then } M \text{ else } N) : s \Upsilon s', \tau} \text{ (TEST)}$$

Home Page

Print

Title Page

Contents

◀ ▶

◀ ▶

Page 20 of 24

Go Back

Full Screen

Close

Quit

$$\frac{\begin{array}{l} \ell; F; \Gamma \vdash_G M : s, \tau \xrightarrow[\ell', F']{s'} \sigma \quad \ell' \preceq_G \ell \quad s.t \preceq_{F \cup G} s''.w \\ \ell; F; \Gamma \vdash_G N : s'', \tau \quad s.r \Upsilon s''.r \preceq_{F \cup G} s'.w \end{array}}{\ell; F \cup F'; \Gamma \vdash_G (MN) : s \Upsilon s' \Upsilon s'' \Upsilon (\perp, \top, s.c \Upsilon s''.c), \sigma} \text{ (APP)}$$

The context should grant at least the access level required by the body of the function.

Home Page

Print

Title Page

Contents



Page 21 of 24

Go Back

Full Screen

Close

Quit

- **Subject reduction** – typable expressions reduce to typable expressions with less critical effects
- **Type safety** – typable expressions do not get stuck
→ **no access violations** guaranteed at compile time

Home Page

Print

Title Page

Contents



Page 22 of 24

Go Back

Full Screen

Close

Quit

13. Type Soundness – Secure Information Flow

Theorem (Soundness). If an expression is *typable*, then it satisfies *the non-disclosure policy*.

$$\ell; F; \Gamma \vdash_G M : s, \tau \implies M \in \mathcal{ND}(\ell', F \cup G) \quad \forall \ell'$$

Proof: by constructing a bisimulation that contains pairs (M, M) of typable expressions, closed w.r.t. co-inductive property.

Home Page

Print

Title Page

Contents



Page 23 of 24

Go Back

Full Screen

Close

Quit

14. Future work

- finding ways of better control of declassification – à la robust declassification in an integrated approach?
- finding a more intuitive notion of a [security error](#)
- possible extension to [mobile code](#)

Home Page

Print

Title Page

Contents



Page 24 of 24

Go Back

Full Screen

Close

Quit

Thank you!