

Sanity-checking declassification via access control

Presented by Rob Simmons
November 14, 2007

Gérald Boudol and Marja Kolundžija's
“Access Control and Declassification”

Warning:

- This paper is presented in the right order!

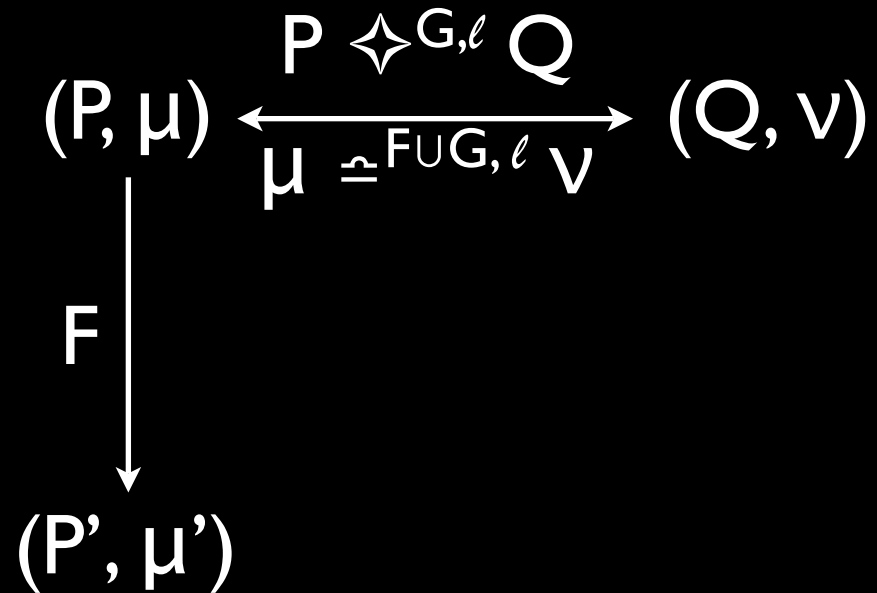
Warning:

- This paper is presented in the right order!
- This presentation: Gérald Boudol and Marja Kolundžija's 2007 paper "Access Control and Declassification"
- Follow-up to Boudol's 2005 paper "On Typing Information Flow" and Boudol and Kolundžija's 2007 paper "On declassification and the non-disclosure policy"

Typing Info Flow

- A process P satisfies non-disclosure with respect to a flow policy G if it satisfies $(P \star^{G,\ell} P)$, i.e. if it (G, ℓ) -bisimulates itself.

- $P \star^{G,\ell} Q$ is a symmetric relation on processes:

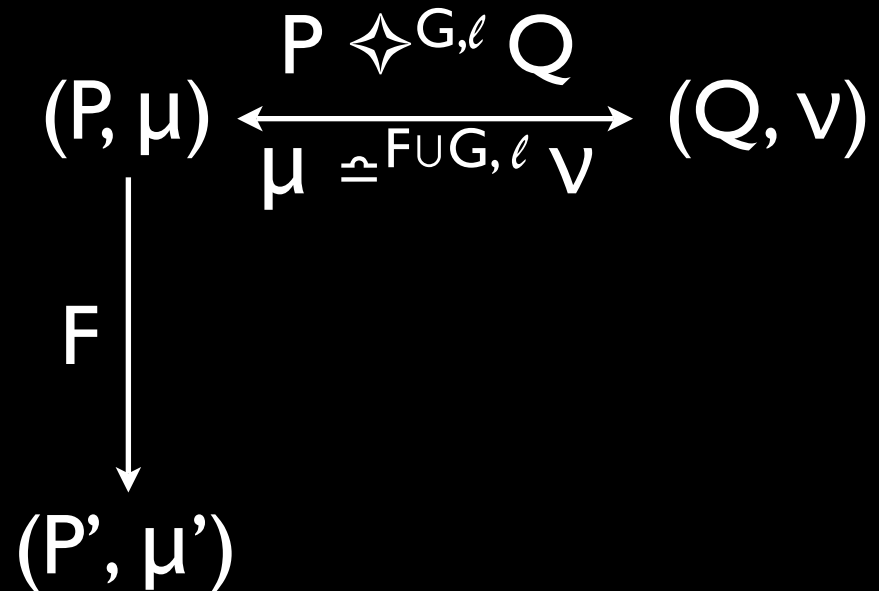


Typing Info Flow

- A process P satisfies non-disclosure with respect to a flow policy G if it satisfies $(P \star^{G,\ell} P)$, i.e. if it (G, ℓ) -bisimulates itself.

- $P \star^{G,\ell} Q$ is a symmetric relation on processes:

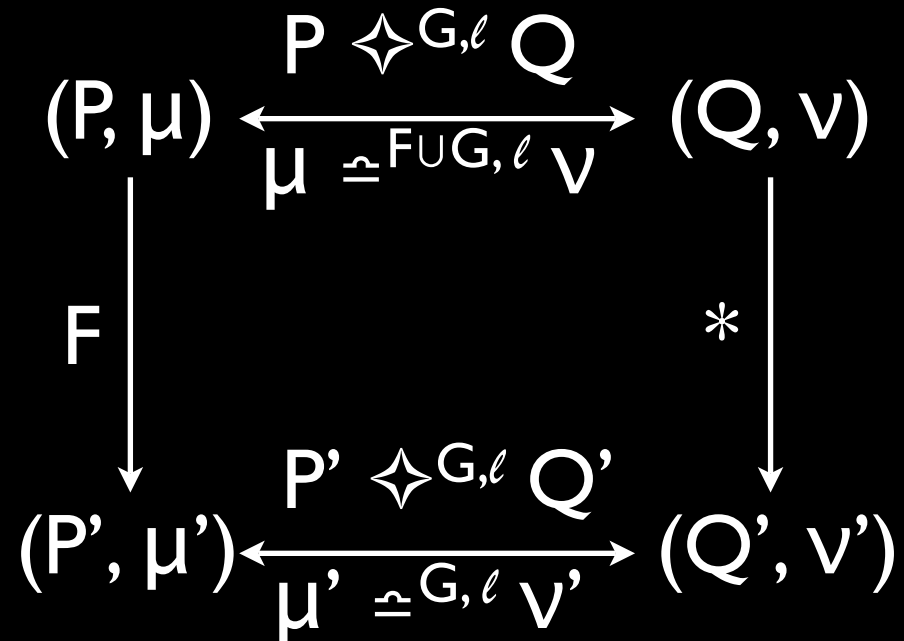
Side condition:
 $\text{dom}(\mu' - \mu)$ is
disjoint from v



Typing Info Flow

- A process P satisfies non-disclosure with respect to a flow policy G if it satisfies $(P \star^{G,\ell} P)$, i.e. if it (G, ℓ) -bisimulates itself.

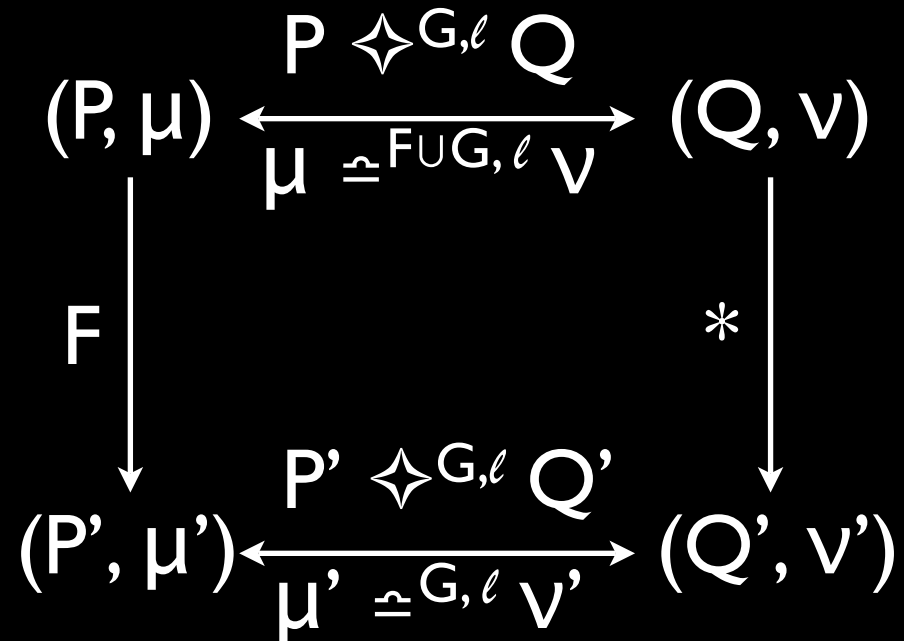
- $P \star^{G,\ell} Q$ is a symmetric relation on processes:



Typing Info Flow

- A process P satisfies non-disclosure with respect to a flow policy G if it satisfies $(P \diamond^{G, \ell} P)$, i.e. if it (G, ℓ) -bisimulates itself.

- $\mu \cong^{F, \ell} \nu$: μ and ν are identical when all cells with labels below ℓ in the lattice are equal



Typing Info Flow

- A process P satisfies non-disclosure with respect to a flow policy G if it satisfies $(P \star^{G, \ell} P)$, i.e. if it (G, ℓ) -bisimulates itself.
- Good if you want to eliminate the vast majority of your application that does not do declassification at all from suspicion
- Bad if you don't want my code to disclose your password. (NO restriction on use of the “flow” construct that permits declassification)

Access Control & Declassification

- A sensible next step?
- Running code has a set of permissions determined by “stack inspection” - i.e. evaluation contexts.
- A construct “enable” raises the execution read permissions of its subterm
 - Untrusted code can't be allowed “enable”
 - Untrusted code *can(?)* be allowed “flow”

Access Control & Declassification

- A process P satisfies non-disclosure with respect to a flow policy G if it satisfies $(P \star^{G, \ell} P)$, i.e. if it (G, ℓ) -bisimulates itself.

Access Control & Declassification

- A process P satisfies non-disclosure with respect to a default access level ℓ and a global flow policy G if it satisfies $(P \star^{\ell, G, \ell'} P)$, i.e. if it (ℓ, G, ℓ') -bisimulates itself.

Access Control & Declassification

- A process P satisfies non-disclosure with respect to a default access level ℓ and a global flow policy G if it satisfies $(P \star^{\ell, G, \ell'} P)$, i.e. if it (ℓ, G, ℓ') -bisimulates itself.
- I can declassify anything I can read
- Can't handle look-but-don't-touch requirements (remember, but don't send, my password)

Basics

- A label ℓ is a set of principles/permissions/privileges/(memberships?)

Basics

- A label ℓ is a set of principles/permissions/privileges/(memberships?)

```
$ pts member rjsimmon, twelf
```

```
Groups rjsimmon (id: 12071):
```

```
  rob_private
```

```
  decfive
```

```
  msri_yogi
```

```
  twelf_cvs
```

```
  twelf_wiki
```

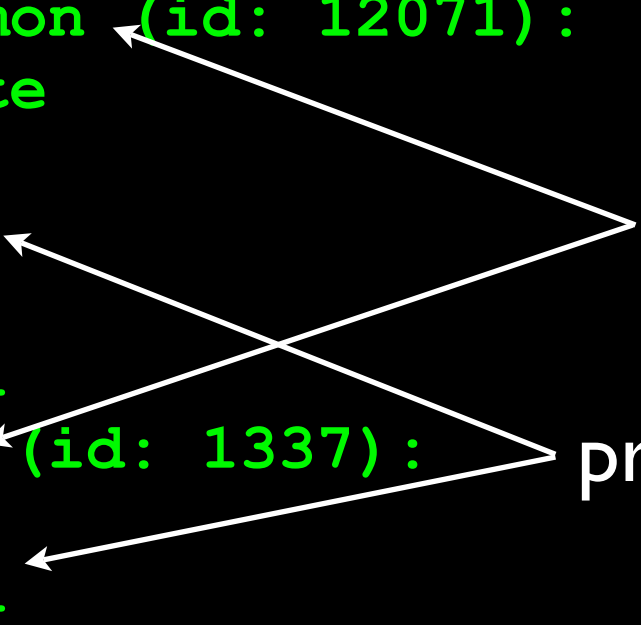
```
Groups twelf (id: 1337):
```

```
  twelf_cvs
```

```
  twelf_wiki
```

labels

principles




Basics

- A label ℓ is a set of principles/permissions/privileges/(memberships?)
- If $\ell \supseteq \ell'$, the value in u_ℓ can be transferred to the value in $v_{\ell'}$
 - The value in a cell marked with `rjsimmon` can be transferred to a cell marked with `twelf` - this is *classification*

Basics

- A label ℓ is a set of principles/permissions/privileges/(memberships?)
- If $\ell \supseteq \ell'$, the value in u_ℓ can be transferred to the value in $v_{\ell'}$
- A *flow policy* $(p_0 < q_0) \dots (p_n < q_n)$ establishes a hierarchy among principles



Info FROM
twelf_wiki
can go TO
twelf_cvs

– Say `twelf_wiki < twelf_cvs`

– But why not just make `twelf_cvs = {twelf_cvsonly, twelf_wiki}`?

Basics

- A label ℓ is a set of principles/permissions/privileges/(memberships?)
- If $\ell \supseteq \ell'$, the value in u_ℓ can be transferred to the value in $v_{\ell'}$
- A *flow policy* $(p_0 < q_0) \dots (p_n < q_n)$ establishes a hierarchy among principles
- F^* is the Kleene-closure of the flow policy
- $\ell \leq_F \ell'$ if $\forall p' \in \ell', \exists p \in \ell$ s.t. $p F^* p'$

Security pre-lattice

- $\ell \leqslant_F \ell'$ if $\forall p' \in \ell', \exists p \in \ell$ s.t. $p F^* p'$
 - In particular, $\ell \leqslant_F \ell'$ if $\ell \supseteq \ell'$,
and `rjsimmon` \leqslant_F `twelf`
 - Meet $\ell \wedge_F \ell'$ is $\ell \cup \ell'$
 - Join $\ell \vee_F \ell'$ is $\{q \mid \exists p \in \ell, p' \in \ell' \text{ s.t. } p F^* q \text{ and } p' F^* q\}$
 - `{twelf_wiki}` \vee_F `{twelf_wiki}` is, oddly,
`{twelf_wiki, twelf_cvs}`

Language

Expressions $M, N, \dots =$

$V \mid (\text{if } M \text{ then } N \text{ else } N') \mid (M \ N) \mid M; N$
 $\mid (\text{ref}_{\ell, \theta} M) \mid (! N) \mid (M := N) \mid (\ell \times M)$
 $\mid (\text{enable } \ell \text{ in } M) \mid (\text{test } \ell \text{ in } M \text{ else } N)$
 $\mid (\text{flow } F \text{ in } M)$

Values $V, \dots =$

$x \mid u_{\ell, \theta} \mid \text{rec } f(x). M \mid tt \mid ff \mid ()$

Operational Semantics I

- Standard operational semantics based on evaluation contexts (want to say they're doing *stack inspection*)
- Stuck expressions:
 - (if V then N else N') w/ $V \neq tt$ or ff
 - (VV') w/ $V \neq \text{rec } f(x). M$
 - $(! V)$ or $(V := V')$ where $V \neq u_{\ell, \theta}$
 - $(! u_{\ell', \theta})$ with a read at level ℓ' not permitted

Read permissions

- Assumes that programs execute in some global policy G and some default confidentiality level ℓ
 - Alternatively, could specify the default confidentiality level to be, say, \perp
 - Increase with (enable ℓ in M)

Read permissions

- Assumes that programs execute in some global policy G and some default confidentiality level ℓ
- $\{E\}_\ell$ computes the read permissions of the term's context:
 - $\{\square\}_\ell = \ell$
 - $\{E[\text{enable } \ell' \text{ in } \square]\}_\ell = \{E\}_\ell \vee_G \ell'$
 - $\{E[\ell \times \square]\}_\ell = \{E\}_\ell \wedge_G \ell'$

Operational Semantics 2

If I'm operating at

$\ell = \text{twelf}$

I can read from a cell labeled with

$\ell' = \text{rjsimmon}$

not the other way around

$\ell' \leq_G \ell$

$\ell \vdash_G (! u_{\ell', \theta}, \mu) \rightarrow (\mu(u_{\ell', \theta}))$

Operational Semantics 2

$$\frac{}{\ell \vdash_G (\text{flow } F \text{ in } V, \mu) \rightarrow (V, \mu)}$$

$$\frac{}{\ell \vdash_G (\text{enable } \ell' \text{ in } V, \mu) \rightarrow (V, \mu)}$$

$$\frac{}{\ell \vdash_G (\ell \times V, \mu) \rightarrow (V, \mu)}$$

$$\frac{}{\ell \vee_G \ell' \vdash_G (M, \mu) \rightarrow (M', \mu')}$$

$$\frac{}{\ell \vdash_G (\text{enable } \ell' \text{ in } M, \mu) \rightarrow (\text{enable } \ell' \text{ in } M', \mu')}$$

$$\frac{}{\ell \wedge_G \ell' \vdash_G (M, \mu) \rightarrow (M', \mu')}$$

$$\frac{}{\ell \vdash_G (\ell \times M, \mu) \rightarrow (\ell \times M', \mu')}$$

$$\ell' \leq_G \ell$$

$$\frac{}{\ell \vdash_G (\text{test } \ell' \text{ then } M \text{ else } N, \mu) \rightarrow M}$$

$$\ell' \not\leq_G \ell$$

$$\frac{}{\ell \vdash_G (\text{test } \ell' \text{ then } M \text{ else } N, \mu) \rightarrow N}$$

Operational Semantics 2

- Under the operational semantics I can run “flow (`rob_private` < `twelf_wiki`) in M,” but if run with $\ell = \{\text{twelf_wiki}\}$, if M does not include any “enable,” it won’t be able to read from any cell $u_{\{\text{rob_private}\}\theta}$ in order to successfully declassify.

Type System

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

Type System

$$l; F; \Gamma \vdash_G M : s, \tau$$
$$l; F; \Gamma \vdash_G M : (l_c, l_w, l_t), \tau$$

Type System

$$\ell; F; \Gamma \vdash_G M : s, \tau$$
$$\ell; F; \Gamma \vdash_G M : (l_c, l_w, l_t), \tau$$
$$\ell; F; \Gamma \vdash_G M : (s.c, s.w, s.t), \tau$$

Type System

$$\ell; F; \Gamma \vdash_G V : (\perp, \top, \perp), \dots$$

Type System

$$\ell; F; \Gamma \vdash_G V : (\perp, \top, \perp), \dots$$
$$\ell; F; \Gamma \vdash_G N : s_0, \tau$$
$$\ell; F; \Gamma \vdash_G M : s, \text{bool} \quad \ell; F; \Gamma \vdash_G N' : s_1, \tau \quad s.r \preceq_{FUG} s_0.w \cup s_1.w$$

$$\ell; F; \Gamma \vdash_G (\text{if } M \text{ then } N \text{ else } N') : (\perp, \top, t), \tau$$

Type System

$$\ell; F; \Gamma \vdash_G V : (\perp, \top, \perp), \dots$$
$$\ell; F; \Gamma \vdash_G N : s_0, \tau$$
$$\ell; F; \Gamma \vdash_G M : s, \text{bool} \quad \ell; F; \Gamma \vdash_G N' : s_1, \tau \quad s.r \preceq_{FUG} s_0.w \cup s_1.w$$

$$\ell; F; \Gamma \vdash_G (\text{if } M \text{ then } N \text{ else } N') : (\perp, \top, t), \tau$$

(t is either s.c or

\perp if you can prove that both N and N' terminate)

Type System

$$\ell; F; \Gamma \vdash_G V : (\perp, \top, \perp), \dots$$
$$\ell; F; \Gamma \vdash_G N : s_0, \tau$$
$$\ell; F; \Gamma \vdash_G M : s, \text{bool} \quad \ell; F; \Gamma \vdash_G N' : s_1, \tau \quad s.r \preceq_{\text{FUG}} s_0.w \cup s_1.w$$

$$\ell; F; \Gamma \vdash_G (\text{if } M \text{ then } N \text{ else } N') : (\perp, \top, t), \tau$$
$$\ell; F; \Gamma \vdash_G M : s, \tau \xrightarrow[\ell', F']{\ell, F} s', \tau' \quad \ell' \preceq_G \ell \quad s.t \preceq_{\text{FUG}} s''.w$$
$$\ell; F; \Gamma \vdash_G N : s'', \tau \quad s.r \vee s''.r \preceq_{\text{FUG}} s'.w$$

$$\ell; F; \Gamma \vdash_G (M N) : s \vee s' \vee s'', \tau'$$

Type System

$$\frac{\ell; F; \Gamma \vdash_G M : s, \tau \quad s.r \preceq_{\text{FUG}} \ell'}{\ell; F; \Gamma \vdash_G \text{ref}_{\ell', \tau} M : (\perp, s.w, s.t), \tau \text{ ref}_{\ell'}}$$

$$\frac{\ell; F; \Gamma \vdash_G M : s, \tau \text{ ref}_{\ell'} \quad \ell' \preceq_G \ell}{\ell; F; \Gamma \vdash_G (! M) : (\perp, \top, t), \tau}$$

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

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

(no join?) \nearrow

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

(etc...)

Type safety

- Types are preserved and effects are decreasing
- The expressions we want to be stuck aren't well-typed
- Can-read checks are never needed dynamically

Security

- A process P satisfies non-disclosure with respect to a default access level ℓ and a global flow policy G if it satisfies $(P \star^{\ell, G, \ell'} P)$, i.e. if it (ℓ, G, ℓ') -bisimulates itself.
- If M is type in the context of an access level ℓ , a global flow policy G and a local policy F , then M satisfies the non-disclosure policy with respect to FUG

Comments

- How much stronger is the final result?
- Are there situations you need to read information that you can't declassify?
 - Yes, but in what situations?
 - It's hard to do this without needing to make values
- Removes random concurrency bits that seemed unmotivated before.
- We have shifted the problem, does it help?