Contrôle de Flux d'Information et Déclassification

Gérard Boudol
INRIA Sophia Antipolis

ACI Sécurité Informatique, Projet CRISS

- Travail effectué en collaboration avec Ana Matos (Thèse à venir).
- Publié dans les actes du 18ème IEEE Computer Security Foundations Workshop, Aix-en-Provence, juin 2005.
- Amélioré dans « *On typing information flow* », Intern. Coll. on Theoretical Aspects of Computing, Hanoi, octobre 2005.
- Sélectionné pour un numéro spécial du Journal of Computer Security (article disponible sur ma page web, et via la page du projet CRISS).

Le Projet Criss

Contrôle de
Ressources = *analyse statique de la complexité*
et d'
Interférences *et du flux d'information*
pour
Systèmes
Synchrones. *pour des programmes concurrents et réactifs.*

Equipes : PPS (Paris, Roberto Amadio), MOVE (LIF Marseille, Solange Coupet), LIPN (Villetaneuse, Patrick Baillot), CALLIGRAMME (LORIA Nancy, Jean-Yves Marion), MIMOSA (INRIA Sophia, Gérard Boudol).

site web : `http ://www.pps.jussieu.fr/~amadio/Criss/criss.html`

Motivation

Confidentialité = un utilisateur (« sujet ») ne doit pouvoir accéder à une information protégée par des droits d'accès que s'il en a l'autorisation.

Le contrôle d'accès n'est pas suffisant : un « sujet » autorisé peut révéler publiquement une information confidentielle.

➥ contrôle du flux d'information.

Approche langage : vérification de programmes (= « sujets »).

Objectif : éviter des erreurs de programmation amenant à des violations de la confidentialité.

Un Exemple

Un logiciel de diffusion d'articles d'un journal électronique doit :

- avoir accès aux articles de la base de données – une information privée.
- être publiquement accessible à tout utilisateur – mal intentionné ou non (moteurs de recherche).

Mais ce logiciel peut contenir des erreurs de programmation permettant à un client d'obtenir un article gratuitement.

↪ des moyens de vérifier que les programmes ne contiennent pas de telles erreurs, sources potentielles de fuites d'information.

ÉTAT de l'ART (1/2)

Les ingrédients de l'analyse de flux d'information :

- un langage de programmation, avec une sémantique.
- un treillis $(\mathcal{S}, \preceq)$ de niveaux de confidentialité ℓ [Bell & La Padula 76, Denning 76], p. ex.

$$institution \prec group \prec user \prec root$$

  permettant de classifier les « objets » (= addresses mémoire, fichiers, entrées des bases de données, ports de communication...).
- une propriété de sécurité : un programme P est **sûr** s'il n'implante que des flux d'information autorisés, i.e. d'objets classifiés ℓ vers des objets classifiés ℓ' avec $\ell \preceq \ell'$ [Cohen 77 « strong dependency », Goguen & Meseguer 82 « non-interference »].

ÉTAT de l'ART (2/2)

- une analyse statique du flux d'information : systèmes de types.
- un résultat d'adéquation : tout programme typable est sûr [Volpano, Smith & Irvine 96].

Une théorie bien établie :

- de nombreuses études, p. ex. JAVA [Myers 99], CAML [Pottier & Simonet 03].
- un article de survol « Language-based Information-flow Security » [Sabelfeld & Myers 03].

Le Problème : Déclassification

La propriété de « non-interference » est inutilisable en pratique : nécessité de déclassifier les informations. Exemples :

- le logiciel de diffusion d'articles doit délivrer un texte (normalement gardé secret) pour lequel l'utilisateur a payé.
- procédure de vérification de mot de passe [Jones & Lipton 75].
- diffusion de messages cryptés, etc.

➥ **Question** : est-il possible d'adapter la théorie du flux d'information pour admettre des programmes déclassifiants ? Nécessite une **nouvelle « politique de sécurité »** ($\neq$ non-interference).

Notre Approche

(1) **Quoi ?** Quelle information, ou combien d'information est-il acceptable de révéler ?

Le programmeur a la responsabilité de répondre à cette question (une activité de « preuve de programme »).

(2) **Comment ?** Comment accepter des programmes intentionnellement déclassifiants dans une approche langage de la confidentialité ?

Le (concepteur du) langage de programmation doit fournir une réponse à celle-ci.

➥ On s'intéresse à la question « **comment ?** » – conception de langage. Objectif : aider le programmeur à éviter des erreurs concernant la confidentialité.

Notre Solution

Garder les bonnes choses de l'approche langage : analyse de flux d'information en utilisant un treillis de sécurité, la « non-interference », les systèmes de types adéquats...

... mais : en rendant cela **local** à des sous-programmes,

en introduisant une construction de **déclaration de flux local :**

$$(\text{flow } F \text{ in } P)$$

où F est une relation binaire sur les niveaux de confidentialité. Par exemple

$$(\text{if } paid \text{ then } (\text{flow } A \prec B \text{ in } y_B := !\, x_A) \text{ else } \cdots)$$

déclassifiant si $A \prec B$ n'est pas autorisé dans la politique (= treillis) de flux globale.

Le Langage

- Un langage fonctionnel (= λ-calcul en appel par valeur) et impératif, à la ML, étendu avec des « threads » et les déclarations de flux local :

$$\begin{array}{rcl} M,\ N\ldots & ::= & V \mid x \mid (\text{if } M \text{ then } N \text{ else } N') \mid (MN) \\ & \mid & M\ ;\ N \mid (\text{ref}_{\ell,\theta}\ N) \mid (!\ N) \mid (M := N) \\ & \mid & \varrho x V \mid (\text{thread } M) \mid (\text{flow } F \text{ in } M) \\ V,\ W\ldots & ::= & u_{\ell,\theta} \mid \lambda x M \mid tt \mid ff \mid () \end{array}$$

- Sémantique opérationnelle :

$$\frac{(M,\mu) \underset{F'}{\longrightarrow} (M',\mu')}{((\text{flow } F \text{ in } M),\mu) \underset{F\cup F'}{\longrightarrow} ((\text{flow } F \text{ in } M'),\mu')}$$

$$((\text{flow } F \text{ in } V),\mu) \underset{\emptyset}{\rightarrow} (V,\mu)$$

Les Treillis de Sécurité

- un ensemble $\mathcal{P}$ de « principaux » est donné (p. ex. $Alice$, Bob...).
- niveau de confidentialité $=$ ensemble $\ell \subseteq \mathcal{P}$ de principaux, assigné aux adresses mémoire ($\sim$ « access control list »).
- $\ell \supseteq \ell'$ signifie « ℓ' est plus restrictif que ℓ » : l'information peut aller du niveau ℓ vers ℓ'.
- toute relation de flux, i.e. $F \subseteq \mathcal{P} \times \mathcal{P}$, determine un préordre

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

avec des inf $\ell \cup \ell'$ et des sup

$$\ell \sqcup \ell' = \{ \, q \mid \exists p \in \ell \, \exists p' \in \ell' . \; p F^* q \; \& \; p' F^* q \, \}$$

La Propriété de Sécurité (1/2)

- « Non-interference » généralisée = « non-interference » à chaque pas de calcul, relativement à la politique de flux courante. Si G est la relation de flux globale, et

$$(P, \mu) \underset{F}{\rightarrow} (P', \mu')$$

alors P est autorisé à lire la mémoire μ selon la politique $G \cup F$.

- mémoires égales jusqu'à un certain niveau ℓ :

$\mu =^{F,\ell} \nu$ si $\mu(u) = \nu(u)$ pour toute adresse u classifiée $\ell' \preceq_F \ell$.

La Propriété de Sécurité (2/2)

Une formulation basée sur la notion de bisimulation :

P est **sûr**, relativement à une politique de flux globale G, ssi $P \boxtimes^{G,\ell} P$ pour tout niveau de confidentialité ℓ, où

$\boxtimes^{G,\ell}$ est la plus large relation satisfaisant : si $P \boxtimes^{G,\ell} Q$ alors

$$si \quad (P,\mu) \underset{F}{\rightarrow} (P',\mu') \;\&\; \mu =^{F \cup G,\ell} \nu$$

$$alors \quad \exists Q', \nu'.\ (Q,\nu) \stackrel{*}{\rightarrow} (Q',\nu') \;\&\; P' \boxtimes^{G,\ell} Q' \;\&\; \mu' =^{G,\ell} \nu'$$

Le Système de Types et d'Effets

- Types

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

- Séquents

$$G; \Gamma \vdash M : s, \tau$$

où

- G est la politique de flux qui vaut pour l'évaluation de M, utilisée pour vérifier le flux d'information,
- Γ est le contexte de typage,
- τ est le type,
- s est l'effet de sécurité de M.

Les Effets de Sécurité

sont des triples $s = (s.\mathsf{r}, s.\mathsf{w}, s.\mathsf{t})$ où

- $s.\mathsf{r}$, effet de lecture = borne supérieure des niveaux de confidentialité des adresses mémoire que l'expression peut lire,
- $s.\mathsf{w}$, effet d'écriture = borne inférieure des niveaux des adresses mémoire que l'expression peut mettre à jour,
- $s.\mathsf{t}$, effet de terminaison = borne supérieure des niveaux de confidentialité des adresses mémoire que l'expression peut lire, et dont la terminaison ou non du calcul peut dépendre.

Typiquement, si M est du type $\theta\,\mathsf{ref}_\ell$ alors $(!\,M)$ a un effet de lecture au moins ℓ, et $(M := N)$ a un effet d'écriture au plus ℓ.

Quelques Règles de Typage

Affectation

$$\frac{G;\Gamma \vdash M : s, \theta\ \mathsf{ref}_\ell \quad G;\Gamma \vdash N : s', \theta \quad s.\mathsf{t} \preceq_G s'.\mathsf{w},\ \ s.\mathsf{r} \sqcup s'.\mathsf{r} \preceq_G \ell}{G;\Gamma \vdash (M := N) : s \sqcup s' \sqcup (\bot, \ell, \bot), \mathsf{unit}}$$

Branchement conditionnel

$$\frac{G;\Gamma \vdash M : s, \mathsf{bool} \quad G;\Gamma \vdash N_i : s_i, \tau \quad s.\mathsf{r} \preceq_G s_0.\mathsf{w} \cup s_1.\mathsf{w}}{G;\Gamma \vdash (\mathsf{if}\ M\ \mathsf{then}\ N_0\ \mathsf{else}\ N_1) : s \sqcup s_0 \sqcup s_1 \sqcup (\bot, \top, s.\mathsf{r}), \tau}$$

Déclaration de flux local

$$\frac{F, G;\Gamma \vdash M : s, \tau \quad s.\mathsf{r} \preceq_{G \cup F} r \quad s.\mathsf{t} \preceq_{G \cup F} t \preceq_G r}{G;\Gamma \vdash (\mathsf{flow}\ F\ \mathsf{in}\ M) : (r, s.\mathsf{w}, t), \tau}$$

Résultat et Conclusion

Theorem. Toute expression typable dans le contexte d'une politique de flux G est sûre pour cette politique, i.e.

$$G; \Gamma \vdash M : s, \tau \;\Rightarrow\; \forall \ell \subseteq \mathcal{P}.\ M \bowtie^{G,\ell} M$$

- La construction de déclaration de flux local semble facile à comprendre et donc à utiliser, et facile à adapter à d'autres langages.
- Une expérimentation pratique est nécessaire.
- Implantation : généraliser l'inférence de type de Flow Caml ?