
Un moniteur de référence sûr d'une base de données

Julien Blond* — **Charles Morisset****

* *Bertin Technologies*
10 bis avenue Ampère, F-78180 Montigny-le-bretonneux
blond@bertin.fr

** *Laboratoire d'Informatique de Paris 6*
8 rue du Capitaine Scott F-75015 Paris
charles.morisset@lip6.fr

RÉSUMÉ. La conception d'un logiciel chargé du contrôle d'accès d'un système doit s'effectuer selon des règles strictes, afin de prévenir la présence de failles dans le code qui sera élaboré. L'atelier Focal est un environnement permettant d'écrire, au sein d'un même cadre de travail, des spécifications formelles et du code en démontrant qu'il respecte ces spécifications. Nous avons utilisé cet outil pour formaliser une politique de contrôle d'accès, celle de Bell et La Padula, et l'implanter dans MySQL, un système de gestion de base de données. Cette implantation, fondé sur l'interprétation des requêtes SQL en termes d'accès, prend la forme d'un moniteur de référence qui filtre les requêtes, et rejette celles qui violent la politique de sécurité.

ABSTRACT. Designing a software in charge of the access control of a system must be carried out according to strict rules in order to avoid faults within this system. The Focal workshop is a programming environment making it possible to write, within the same working context, formal specifications and code with proof that it respects these specifications. We used this tool to formalise a policy of access control, the one of Bell and La Padula, and to implement it in MySQL, a database management system. This implementation, based on the translation of SQL requests in terms of Bell et La Padula requests, is done using an external reference monitor which filters the requests and rejects those which violate the security policy.

MOTS-CLÉS : Contrôle d'accès, Moniteur de référence, Bell et La Padula, Base de données

KEYWORDS: Access control, reference monitor, Bell and La Padula, Database

1. Introduction

L'utilisation de plus en plus importante de bases de données requiert la mise en place de systèmes de protection automatisés et fiables. La conception et le développement des programmes en charge de la sécurité d'une base de données, et plus particulièrement du contrôle des accès à cette base doivent par conséquent être réalisés avec un haut niveau de sûreté, c'est-à-dire sans aucune faille exploitable par un attaquant.

Les Critères Communs (recueil de normes définies par des agences gouvernementales) fournissent une méthodologie permettant d'atteindre des hauts niveaux de sécurité. Ils définissent à la fois un cadre de travail pour la conception et la réalisation de logiciels et une référence pour les utilisateurs de ces logiciels. Les hauts niveaux de sûreté des Critères Communs (EAL 5 à 7) requièrent l'utilisation de méthodes formelles dans les étapes de spécification et de conception du logiciel. C'est pour cette raison que nous avons utilisé l'atelier Focal (Rioboo *et al.*, 2006) pour implanter un moniteur de référence. Focal permet d'écrire une spécification formelle du logiciel à développer en énonçant des propriétés en logique du premier ordre. Puis, par des étapes de conception successives, il est possible d'arriver à du code dans un style fonctionnel à la ML et faire des preuves avec l'outil de démonstration automatique Zenon, ces preuves étant ultimement vérifiées par le système Coq. On peut ainsi obtenir progressivement une implantation correcte à partir d'une spécification formelle.

D'après les Critères Communs, un système est vu comme une installation donnée de technologies de l'information, avec un objectif et un environnement opérationnel particuliers. Une politique de sécurité est un ensemble de règles qui précisent comment gérer, protéger ou distribuer les informations ou ressources du système. Un moniteur de référence est une machine abstraite qui applique les politiques de contrôle d'accès d'un système, ces politiques étant un sous-ensemble des politiques de sécurité.

Toujours selon les Critères Communs, un moniteur de référence doit posséder les trois caractéristiques suivantes :

- des sujets non sûrs ne peuvent pas interférer avec son fonctionnement, *i.e.* il est à l'épreuve d'une intrusion physique,
- des sujets non sûrs ne peuvent pas court-circuiter les contrôles qu'il effectue, *i.e.* il est systématiquement appelé,
- il est suffisamment simple pour être analysé et pour que son comportement soit compris (*i.e.* sa conception est simple).

Ces trois caractéristiques, introduites dans (Anderson, 1972), sont connues sous l'acronyme *NEAT*, pour *non-bypassable* (il est n'est pas possible d'éviter les fonctions de sécurité), *evaluatable* (les fonctions de sécurité sont suffisamment simples pour être mathématiquement vérifiées et évaluées), *always invoked* (les fonctions de sécurité sont tout le temps appelées) et *tamperproof* (les fonctions de sécurité ne peuvent pas être altérées). Cette acronyme est défini dans le cadre de *MILS* (*Multiple Independent Levels of Security*), une approche de développement de systèmes sécurisés (<http://www.ois.com/MILS/>). On peut noter que l'utilisation de méthodes formelles

pour la conception d'un moniteur de référence facilite son évaluation et sa vérification, et il n'est donc pas forcément indispensable que ce dernier soit « simple ».

Dans cet article, nous montrons comment utiliser l'atelier Focal pour mettre en œuvre la politique de sécurité de Bell et La Padula au sein du gestionnaire de base de données MySQL.

Tout d'abord, nous présentons le modèle de Bell et La Padula (nous utilisons dans la suite du papier l'abréviation BLP pour désigner Bell et La Padula) ainsi que sa généralisation dans l'algèbre de sécurité de McLean (section 2). Pour utiliser ce modèle, il faut pouvoir déterminer les accès engendrés par chaque requête SQL. Nous introduisons alors une sémantique formelle du langage SQL en termes d'accès engendrés. (section 3). Puis nous présentons Focal ainsi que l'architecture générale de l'implantation, sous la forme d'un module externe qui filtre les requêtes SQL fournies par l'utilisateur (section 4).

2. Le modèle de sécurité

2.1. Bell et La Padula

2.1.1. Modèle

Le modèle de Bell et La Padula (Bell *et al.*, 1973) est l'un des premiers modèles de sécurité développés dans le domaine militaire. Il spécifie à la fois les éléments du système (les entités, les niveaux de sécurité, les accès) et la politique de sécurité (sous quelles conditions un accès est autorisé). On distingue deux sortes d'entités. Les *entités actives*, également appelées sujets, correspondent aux utilisateurs, aux processus ou d'une manière générale, à toute entité pouvant effectuer une action dans le système. Les *entités passives*, également appelées objets, correspondent aux fichiers, aux ressources ou plus généralement à toute entité pouvant subir une action. Notons que ces deux familles d'entités ne sont pas forcément disjointes. Un accès est un triplet (s, o, x) , où s est un sujet, o un objet et x un ensemble de modes d'accès inclus dans l'ensemble $\mathcal{A} = \{\text{read, write, execute, append, control}\}$.

Chaque entité est associée à un niveau de sécurité par la donnée de fonctions f_s (pour les sujets) et f_o (pour les objets) à valeur dans un treillis de niveaux de sécurité \mathcal{L}^S . Ce dernier est défini comme le treillis produit $T_c \times T_k$ où T_c est le treillis des classifications (par exemple *Top Secret, Secret, Confidentiel*, etc) et T_k est l'ensemble des parties finies de l'ensemble \mathcal{K} des besoins-d'en-connaître (par exemple *Nucléaire, Marine, OTAN*, etc). Un niveau de sécurité $l_1 = (c_1, e_1)$ est inférieur à un niveau $l_2 = (c_2, e_2)$ si et seulement si $c_1 < c_2$ et $e_1 \subseteq e_2$. Par exemple le niveau (*Secret, {Nucléaire}*) est inférieur à (*Top Secret, {Nucléaire, Marine}*).

Un état σ est un quadruplet (m, d, f_s, f_o) , où m est l'ensemble des accès courants, c'est-à-dire tous les accès considérés comme en cours à un instant dans le système, d est l'ensemble des droits discrétionnaires et f_s (resp. f_o) la fonction qui retourne le niveau de sécurité de chaque sujet (resp. objet). Les droits discrétionnaires sont

les droits qui sont définis par les sujets propriétaires des objets. Ils sont ainsi appelés car ce sont les propriétaires qui choisissent les utilisateurs autorisés à accéder à leur objets. Tout droit sur un objet peut être donné à n'importe quel sujet, à l'exception du droit dit de « contrôle », qui identifie le propriétaire. Ce type de contrôle d'accès, DAC (*Discretionary Access Control*), est utilisé par différents modèles, comme celui introduit par (Lampson, 1971), le modèle HRU (Harrison *et al.*, 1976) ou encore la matrice d'accès typée (Sandhu, 1992). Le principal inconvénient du DAC est qu'il est propice aux attaques de type « cheval de troie », conduisant un sujet à exécuter des opérations à son insu. Il peut alors involontairement donner les droits de ses objets à un utilisateur malveillant. Dans certains systèmes critiques, comme cela est fréquent dans le domaine militaire, une notion de contrôle d'accès obligatoire, également appelée MAC (*Mandatory Access Control*), est introduite, pour lutter plus efficacement contre ce type d'attaque. Ce type de contrôle d'accès ne repose pas sur les droits définis par les sujets, mais sur de l'information de sécurité prédéterminée sur les sujets et les objets, comme la notion de niveau de sécurité introduite précédemment. Un sujet ne pourra pas toujours autoriser n'importe quel utilisateur à accéder à ses objets, et il pourrait même se voir refuser l'accès à certains de ses fichiers.

Le modèle BLP met en œuvre à la fois une sécurité obligatoire et une sécurité discrétionnaire. Un état contient des informations de sécurité sur les sujets et les objets (f_s et f_o), ainsi que des droits discrétionnaires. Dans notre formalisation ceux-ci prennent la forme d'un ensemble d'accès d , tel que $(s, o, x) \in d$ implique que s a les droits x sur l'objet o . A partir de la notion d'état, nous pouvons à présent définir la politique de sécurité, qui prend la forme d'un prédicat sur les états caractérisant les états sûrs.

2.1.2. Politique de sécurité

Puisque le modèle de BLP met en œuvre du contrôle d'accès discrétionnaire et obligatoire, un état $\sigma = (m, d, f_s, f_o)$ est sûr si et seulement s'il vérifie les trois propriétés suivantes :

$$\begin{aligned} & \forall s \in \mathcal{S} \forall o \in \mathcal{O} \forall x \subseteq \mathcal{A} \quad (s, o, x) \in m \Rightarrow (s, o, x) \in d \\ & \forall s \in \mathcal{S} \forall o \in \mathcal{O} \forall x \subseteq \mathcal{A} \quad (s, o, x \cup \{\text{read}\}) \in m \Rightarrow f_s(s) \succeq f_o(o) \\ & \forall s \in \mathcal{S} \forall o_1, o_2 \in \mathcal{O} \forall x_1, x_2 \subseteq \mathcal{A} \\ & \left(\begin{array}{l} (s, o_1, x_1 \cup \{\text{read}\}) \in m \\ \wedge (s, o_2, x_2 \cup \{\text{write}\}) \in m \end{array} \right) \Rightarrow f_o(o_1) \preceq f_o(o_2) \end{aligned}$$

La première propriété exprime directement le contrôle d'accès discrétionnaire, car elle impose que tout accès courant appartienne à l'ensemble des accès discrétionnaires. La deuxième concerne le contrôle d'accès obligatoire, car elle impose que pour qu'un sujet puisse lire un objet, il faut que son niveau de sécurité soit supérieur à celui de l'objet. Enfin, la dernière propriété, connue sous le nom de sécurité-*, permet d'éviter de la fuite d'informations d'un objet vers un objet dont le niveau de sécurité est inférieur.

2.1.3. Fonction de transition de BLP

Outre la spécification des entités, la définition des états et celle de la politique de sécurité, Bell et La Padula définissent également une fonction de transition, qui permet à partir d'un état et d'une requête, de retourner une réponse (oui ou non) ainsi qu'un nouvel état (éventuellement identique au précédent). De plus, cette fonction de transition est prouvée vérifier le *Basic Security Theorem*, c'est-à-dire que pour tout état sûr et pour toute requête, l'état renvoyé par cette fonction est également sûr. Il est également prouvé que tout état initial, c'est-à-dire tout état dont l'ensemble des accès courants est vide, est sûr, ce qui garantit par induction que tout état accessible est sûr.

Pour des raisons de place, nous ne donnons pas ici la définition de la fonction de transition de BLP, cette dernière pouvant être trouvée dans (Gureghian *et al.*, 2003), où elle a été décrite en Coq. Néanmoins, comme nous le verrons dans la section 3, chaque requête SQL étant traduite vers une ou plusieurs requêtes BLP, nous décrivons celles-ci, en prenant comme exemple l'ensemble de sujets $\mathcal{S} = \{Jean, Anne\}$ et l'ensemble d'objets $\mathcal{O} = \{F12.tex, F56.ps\}$.

Une requête est un 5-uplet (s_1, t, s_2, o, p) , où s_1 est le sujet effectuant la requête, t est le type de la requête (G, R, C ou D), s_2 est le sujet concerné par la requête, o est l'objet concerné par la requête et p est le paramètre de la requête, qui peut être de la forme χ_ϵ lorsqu'il est vide, $\chi_A(x)$ où x est un mode d'accès, ou $\chi_{\mathcal{F}}(o, c, E)$, où o est un objet, c une classification et E un ensemble de besoins-d'en-connaître. Le sujet s_1 effectuant la requête n'est précisé que si nécessaire, lorsqu'il est réellement important de connaître ce sujet. C'est le cas dans BLP, où un sujet peut donner ses droits sur un objet à un autre sujet. Dans ce cas, s_1 est le premier sujet et s_2 le second. C'est le cas aussi lorsqu'un sujet veut changer le niveau de sécurité d'une entité et dans ce cas, s_1 sera le sujet et s_2 sera le sujet vide (ce que l'on note par σ_ϵ). Comme nous l'avons dit, il existe quatre types de requêtes : G pour *Get access*, R pour *Release access*, C pour *Create object* et D pour *Delete object*. Chaque type de requête a une structure particulière :

- les requêtes de type G concernent les demandes d'accès. Si *Jean* veut lire *F56.ps*, il créera la requête suivante : $(\sigma_\epsilon, G, Jean, F56.ps, \chi_A(\text{read}))$,
- les requêtes de type R concernent le relâchement d'accès. Quand *Jean* a fini de lire *F56.ps*, il soumet la requête suivante : $(\sigma_\epsilon, R, Jean, F56.ps, \chi_A(\text{read}))$,
- les requêtes de type C concernent la création d'objets et demande en fait deux requêtes. En effet, nous considérons que l'ensemble \mathcal{O} des objets contient tous les objets possibles, chaque objet pouvant être soit actif, c'est-à-dire utilisé dans le système, soit inactif. Etant donné qu'on interdit le changement de niveau de sécurité d'un objet actif, la création d'un objet nécessite l'affectation d'un niveau de sécurité à un objet inactif, puis l'activation de cet objet. Par exemple, si *Anne* veut créer le fichier *F12.tex*, elle envoie la requête : $(\sigma_\epsilon, C, \sigma_\epsilon, F12.tex, \chi_{\mathcal{F}}(F12.tex, Secret, \{OTAN, Marine\}))$. pour lui donner la classification *Secret* et l'ensemble de besoins-d'en-connaître $\{OTAN, Marine\}$. L'activation de l'objet est réalisée par la requête : $(\sigma_\epsilon, C, Anne, F12.tex, \chi_\epsilon)$,

– les requêtes de type D concernent la destruction d’objets. Si *Anne* veut détruire *F56.ps* elle soumet la requête : $(\sigma_\epsilon, D, Anne, F56.ps, \chi_\epsilon)$.

Chaque type de requête est exécuté par une fonction différente. La fonction t_G exécute les requêtes de type G , t_R celles de type R , t_C celles de type C et t_D celles de type D .

2.2. L’algèbre de sécurité

L’algèbre de sécurité définie par (McLean, 1988) est une formalisation du modèle de BLP qui apporte essentiellement deux nouveaux aspects par rapport à ce dernier. Le premier est la notion d’accès conjoints, c’est-à-dire des accès effectués non pas par un seul sujet comme c’est le cas usuellement, mais par un groupe de sujets. Cette notion est justifiée par la présence dans certains domaines de protocoles nécessitant une action conjointe et simultanée, comme le lancement d’un missile qui nécessite que deux personnes habilitées appuient sur un bouton en même temps. Il est toutefois possible d’imposer la contrainte que tout groupe de sujets doit être un singleton, et de retrouver ainsi la définition classique des accès. Le deuxième aspect introduit dans cette formalisation est la définition pour chaque sujet et objet des groupes de sujets autorisés à demander leur changement de niveau de sécurité. Cette association permet d’éviter la création de systèmes modifiant arbitrairement le niveau de sécurité des entités sans requête explicite d’un sujet. Par exemple, le système Z introduit par (McLean, 1990) est un système implantant la politique de sécurité de BLP mais qui descend automatiquement les niveaux de sécurité de toutes les entités au plus bas niveau. Bien qu’intuitivement non sûr, ce système vérifie néanmoins le *Basic Security Theorem*, décrit précédemment.

L’introduction des deux notions précédentes permet de définir à partir d’un modèle comme celui de BLP un ensemble de modèles considérés comme des variations du modèle original. On a ainsi les modèles de BLP avec accès conjoints ou sans accès conjoints, ceux où tous les sujets peuvent demander à changer tous les niveaux de sécurité (sans que cela soit toutefois tout le temps autorisé, cela dépend également de la politique de sécurité) ou au contraire ceux où aucun sujet ne peut modifier aucun niveau de sécurité, etc. Le modèle original de BLP correspond d’ailleurs à l’instance de l’algèbre de sécurité où les accès ne sont effectués que par des singletons de sujets et où tous les sujets peuvent demander à changer le niveau de sécurité d’une entité.

Cette algèbre de sécurité ayant fait l’objet de récentes études (Jaume *et al.*, 2006a) ainsi que d’une implantation au sein de l’atelier Focal, dont nous parlons par la suite, nous avons choisi de réutiliser cette implantation pour mettre en application la politique de sécurité de BLP dans une base de données.

3. Sémantique de SQL

Afin d'implémenter la politique de sécurité de BLP au sein d'une base de données, nous devons au préalable définir une sémantique des constructions du langage SQL en termes d'accès qu'elles engendrent. Dans cette section nous définissons une telle sémantique. La sémantique complète se trouve en annexe A.

3.1. Syntaxe abstraite

Il existe deux familles de requêtes SQL. La première est celle des requêtes simples, qui ne dépendent pas d'autres requêtes, comme :

```
SELECT * FROM t1, t2 ;
INSERT INTO t1 VALUES (2, 3, 4) ;
```

La deuxième famille est celle des requêtes imbriquées, qui dépendent du résultat d'une ou plusieurs requêtes SELECT, comme :

```
CREATE TABLE t1 SELECT name FROM t2 ;
INSERT INTO t1 SELECT name FROM t3 ;
```

Afin de pouvoir définir la sémantique en termes d'accès de SQL, nous supposons que l'utilisateur soumettant cette requête est connu et que son identificateur est disponible dans l'environnement global. Comme nous l'expliquons dans la section 4.1.1, la granularité des objets est la table, c'est-à-dire qu'un objet représente une table complète, nous n'avons pas besoin de la liste des colonnes. La syntaxe abstraite d'une requête SQL doit donc contenir uniquement l'objet correspondant à la table accédée, ainsi que la syntaxe abstraite des autres requêtes dans le cas d'une requête imbriquée. On définit dans un premier temps le *select-statement*, qui sera \perp pour les requêtes simples ou le résultat d'un SELECT pour celles imbriquées.

$$sel ::= \perp \mid SEL(o, sel)$$

On peut alors définir la syntaxe abstraite des requêtes SQL CREATE, DELETE, INSERT, UPDATE, SELECT et DROP :

$$ast ::= \begin{array}{l} sel \\ | CRE(o, sel) \\ | DEL(o, sel) \\ | INS(o, sel) \\ | UPD(o) \\ | SEL(o, sel) \\ | DRO(o) \end{array}$$

3.2. Demande et relâchement des accès

Lorsque l'on veut définir une sémantique pour SQL en termes d'accès dans le modèle de BLP, il faut prendre en compte la notion de persistance des accès. En effet, dans le modèle de BLP, chaque accès se décompose en deux phases : premièrement, l'accès est demandé, deuxièmement, l'accès est relâché. Tant qu'un accès n'est pas relâché, il reste dans l'ensemble des accès courants. Cette caractéristique permet de modéliser le fait que dans certains systèmes, comme par exemple un gestionnaire de fichiers, les accès sont persistants.

Dans une base de données SQL, du fait de la structure client/serveur, une requête n'est pas persistante. En effet, un client soumet une requête, le serveur analyse cette requête, ouvre les objets correspondants, lit ou écrit dans ces objets, ferme l'objet et renvoie la réponse au client. Nous définissons ainsi deux relations $\langle \rangle_g$ et $\langle \rangle_r$: $\langle R \rangle_g$ effectue la demande d'accès pour la requête R et $\langle R \rangle_r$ effectue le relâchement d'accès pour la requête R .

Nous allons voir maintenant comment fonctionnent ces deux relations avec l'exemple de la requête SQL INSERT, qui permet d'ajouter des entrées dans une table.

3.3. Une sémantique de SQL en termes d'accès

La syntaxe SQL de la requête INSERT est la suivante :

```
INSERT INTO table-name [(column-list)] VALUES (value-list)
| INSERT INTO table-name [(column-list)] select-statement
```

Dans le premier cas, elle permet d'insérer à la fin de la table *table-name* les valeurs saisies après le mot-clé VALUES. Dans le deuxième cas, les valeurs insérées sont celles résultant de l'évaluation d'une requête SELECT. Par exemple, supposons que nous ayons une table t_1 contenant une colonne nom de famille et une colonne prénom, ainsi qu'une table t_2 où chaque tuple est un triplet (*nom*, *prenom*, *ville*). Pour insérer dans t_1 les noms et prénoms des personnes habitant à Pauillac, il suffit d'exécuter la requête suivante :

```
INSERT INTO  $t_1$  (SELECT nom, prenom FROM  $t_2$  WHERE ville = "PAUILLAC")
```

3.3.1. Demande d'accès : $\langle \rangle_g$

Etudions maintenant l'évaluation de la requête INSERT par $\langle \rangle_g$. Cette relation associe un quadruplet formé d'une requête abstraite, un sujet, le niveau de sécurité de cet objet et un état BLP à un couple composé d'une réponse et d'un nouvel état BLP. Si l'évaluation par $\langle \rangle_g$ de *select-statement* est négative, c'est-à-dire que la requête SELECT correspondante viole la politique de sécurité, alors l'évaluation du INSERT par $\langle \rangle_g$ doit également être négative :

$$(INS_g^1) \frac{\langle sel, s, c, \rho \rangle_g \rightarrow (\text{no}, \rho)}{\langle INS(o, sel), s, c, \rho \rangle_g \rightarrow (\text{no}, \rho)}$$

où s est le sujet soumettant la requête, c son niveau de sécurité et ρ l'état BLP courant.

En revanche, si l'évaluation de *select-statement* par $\langle \rangle_g$ est positive et renvoie un nouvel état ρ' , alors le résultat est la réponse retournée par la fonction t_G avec une demande d'accès en ajout et l'état ρ' , ce qui donne la règle d'inférence suivante :

$$(INS_g^2) \frac{\langle sel, s, c, \rho \rangle_g \rightarrow (\text{yes}, \rho')}{\langle INS(o, sel), s, c, \rho \rangle_g \rightarrow t_G((\sigma_\epsilon, G, \sigma(s), o, \chi_A(\mathbf{a})), \rho')}$$

La requête INSERT peut être vue comme un accès en ajout et non pas comme un accès en écriture. En effet, un sujet peut effectuer un INSERT sur une table même sans y avoir accès en lecture, à condition toutefois qu'il ne soit pas en train de faire un SELECT sur une table de niveau de sécurité supérieur.

3.3.2. Relâchement d'accès : $\langle \rangle_r$

Le relâchement des requêtes peut sembler relativement plus simple que la demande d'accès, étant donné que si un sujet demande à relâcher un accès qui lui a été autorisé, alors la réponse est toujours positive. Il est alors intéressant de constater que si l'évaluation d'une requête par la relation $\langle \rangle_r$ est négative, cela peut laisser supposer qu'il y a eu une modification de l'état BLP par un moyen autre que le moniteur de référence, ou bien que ce dernier a effectué une mauvaise opération, ce qui dans les deux cas doit être considéré comme une attaque potentielle.

De la même manière que pour la relation $\langle \rangle_g$, si l'évaluation par $\langle \rangle_r$ de *select-statement* du INSERT est négative, alors l'évaluation par $\langle \rangle_r$ doit être négative.

$$(INS_r^1) \frac{\langle sel, s, c, \rho \rangle_r \rightarrow (\text{no}, \rho)}{\langle INS(o, sel), s, c, \rho \rangle_r \rightarrow (\text{no}, \rho)}$$

En revanche, si l'évaluation par $\langle \rangle_r$ de *select-statement* est positive, alors l'évaluation par $\langle \rangle_r$ de la requête INSERT est le résultat renvoyé par la fonction t_R de BLP demandant de relâcher l'accès en ajout de l'utilisateur sur la table concernée.

$$(INS_r^2) \frac{\langle sel, s, c, \rho \rangle_r \rightarrow (\text{yes}, \rho')}{\langle INS(o, sel), s, c, \rho \rangle_r \rightarrow t_R((\sigma_\epsilon, R, \sigma(s), o, \chi_A(\mathbf{a})), \rho')}$$

Lors de l'évaluation d'une requête R , il est clair qu'il faut d'abord appeler $\langle R \rangle_g$, puis $\langle R \rangle_r$. En revanche, la question est plus complexe lors de l'évaluation de plusieurs requêtes en séquence. A première vue, pour évaluer une séquence

de requêtes il suffit d'évaluer chaque requête une par une, indépendamment des autres. Mais comme nous allons le voir, cela peut créer des canaux cachés. Nous allons maintenant étudier trois modes d'évaluation différents.

3.4. Modes d'évaluation

Pour ne pas violer la propriété de sécurité- \star définie en 2.1.2, une requête comme :

```
INSERT INTO table1 SELECT * FROM table2
```

doit être refusée si le niveau de sécurité de *table1* est strictement inférieur à celui de *table2*. En effet, dans le cas contraire, de l'information de haut-niveau (*table2*) se retrouverait dans un objet de bas-niveau (*table1*).

Avec la définition de la relation $\langle \rangle_g$, cette requête est effectivement refusée. En effet, lors de l'évaluation de cette requête, le SELECT est tout d'abord évalué par $\langle \rangle_g$, et sans relâcher cet accès, le INSERT est ensuite évalué, ce qui est logiquement refusé.

Mais imaginons maintenant que l'utilisateur soumette successivement les requêtes suivantes :

```
R1   :   SELECT * FROM table2 ;
R2   :   INSERT INTO table1 VALUES val1, . . . , valn
```

et que les val_i soient les valeurs que l'utilisateur lit sur l'écran et proviennent du SELECT. Il est clair que cette séquence de requêtes a le même effet que la requête imbriquée donnée ci-avant et cette séquence devrait être donc refusée. Nous introduisons ci-après trois stratégies différentes pour évaluer une suite de requêtes, en prenant comme exemple la suite de $R_1; R_2$

3.4.1. Mode requête

Le premier mode est l'évaluation requête par requête :

$$\langle R_1 \rangle_g \langle R_1 \rangle_r \langle R_2 \rangle_g \langle R_2 \rangle_r$$

Le SELECT est évalué, et si il est correct, alors il est relâché. Ainsi, lorsque l'on évalue le INSERT, on ne se souvient pas du SELECT, et donc le INSERT est accepté. On constate que cette séquence est acceptée dans ce mode d'évaluation, alors qu'elle ne devrait pas l'être. Nous sommes dans le cas typique d'un canal caché, en l'occurrence l'écran.

3.4.2. Mode strict

Le second mode consiste à ne jamais relâcher les accès :

$$\langle R_1 \rangle_g \langle R_2 \rangle_g$$

Dans ce mode, la séquence est ainsi refusée. Cette option évite certes les canaux cachés, mais elle est excessivement restrictive. Par exemple, si un utilisateur exécute une requête SELECT sur un objet avec le plus haut niveau de sécurité, alors il ne pourra

plus jamais effectuer de INSERT sur une table de niveau inférieur. Ce mode doit donc être réservé aux bases de données extrêmement critiques.

3.4.3. *Mode session*

Etant donné que le mode requête crée des canaux cachés et que le mode strict est trop restrictif, nous introduisons le mode session où chaque requête est relâchée dans l'ordre inverse d'exécution :

$$\langle R_1 \rangle_g \langle R_2 \rangle_g \langle R_2 \rangle_r \langle R_1 \rangle_r$$

Supposons qu'un utilisateur exécute un SELECT puis un INSERT, le SELECT est évalué, et s'il est correct, il n'est pas relâché tout de suite. Le INSERT est ensuite évalué et s'il est correct, il est alors relâché, suivi du SELECT. Ce mode d'évaluation effectue les appels aux relations $\langle \rangle_g$ et $\langle \rangle_r$ dans le même ordre que si les deux requêtes étaient imbriquées dans une unique requête. De manière générale, lorsque l'utilisateur se connecte, toutes ses requêtes sont évaluées à l'aide de $\langle \rangle_g$ et les appels correspondants à $\langle \rangle_r$ sont empilés. Lorsque l'utilisateur quitte la session, la pile des $\langle \rangle_r$ est vidée. Il faut toutefois que l'environnement d'exécution soit « nettoyé » après la fin de session de l'utilisateur. En effet, si ce dernier exécute un SELECT et se déconnecte, mais que le résultat est toujours affiché à l'écran, il pourrait alors se reconnecter et ainsi exécuter un INSERT avec les valeurs affichées. Il pourrait ainsi violer la politique de sécurité sans que cela soit détecté. Il faudrait alors avoir un mécanisme permettant d'éviter de telles situations, et qui pourrait par exemple effacer l'écran ainsi que les fichiers temporaires créés.

4. L'atelier Focal

L'atelier Focal (Prevosto, 2003; Rioboo *et al.*, 2006) a pour but de construire un environnement de développement intégré (IDE) logiquement fondé, avec une sémantique claire et qui permet d'obtenir des implantations correctes et efficaces. Cet atelier possède des traits fonctionnels et orientés-objet et permet aux programmeurs d'écrire des preuves formelles de leur code. Ils sont aidés dans leur tâche par Zenon, un démonstrateur automatique. Les preuves obtenues sont vérifiées par Coq (Logical Project, 2002). Focal permet ainsi d'écrire au sein d'un même cadre de travail des spécifications, des programmes et des preuves par des étapes successives de conception. En effet, les mécanismes d'héritage et de raffinement sont particulièrement bien adaptés au développement d'applications, notamment celles liées à la sécurité, car ils permettent d'effectuer plusieurs raffinements depuis une spécification abstraite jusqu'à un code exécutable.

Un programme Focal est la donnée d'une hiérarchie d'espèces. Une espèce peut être vue comme un ensemble de méthodes, identifiées par leur nom. Chaque méthode peut être soit déclarée (constantes, opération et propriété) soit définie (implantations des opérations et preuves des théorèmes), et appartient à l'un des trois types suivants :

(i) le type support est la méthode qui représente l'ensemble sous-jacent à la structure définie par l'espèce. Chaque espèce doit avoir un seul type support, et il peut être soit déclaré soit défini. Un type support seulement déclaré correspond alors à un type générique (variable de type), et un type support défini correspond à un type concret,

(ii) les méthodes opérationnelles sont les constantes ou les fonctions de la structure. Les méthodes déclarées sont appelées des signatures. Le langage utilisé pour définir ces méthodes est similaire au cœur fonctionnel d'OCaml, avec une construction permettant d'appeler une méthode d'une espèce donnée,

(iii) les méthodes logiques décrivent les propriétés des méthodes opérationnelles. Dans ce contexte, la déclaration d'une méthode logique est simplement l'énoncé d'une propriété, tandis que la définition d'une méthode logique est la preuve que cette propriété est vraie.

Le langage utilisé pour les énoncés de propriétés est composé des connecteurs logiques de base et des quantificateurs existentiels et universels sur les types Focal. Ces propriétés sont donc des formules de logique du premier ordre, contenant les noms des méthodes liées par déclaration/définition, héritage ou paramétrisation. Une preuve en Focal est soit un script Coq interprété dans le contexte de l'espèce où la propriété est prouvée, soit une preuve contenant plus ou moins de détails écrite en Cutter, le langage déclaratif sur lequel est basé le démonstrateur automatique Zenon. Il faut noter que certaines caractéristiques de la programmation orientée-objet ont été limitées afin d'éviter des constructions non correctes, par exemple la récursion ouverte qui peut mener à des incohérences lorsqu'elle est mal utilisée.

Au bas de la hiérarchie Focal, on trouve les collections. Une collection est construite sur une espèce complètement définie (toutes les méthodes opérationnelles sont définies et toutes les méthodes logiques sont prouvées). De plus, une collection est « figée », on ne peut pas y définir de nouvelles méthodes ni en redéfinir. Enfin, toutes les définitions (type support et preuves compris) sont masquées à l'utilisateur afin que ce dernier n'ait accès qu'à l'interface de la collection (noms des méthodes et propriétés).

Des exemples précis et complets de développement avec l'atelier Focal sont présents dans le manuel de référence ainsi que dans le tutoriel, tous les deux disponibles sur le site internet de l'atelier Focal <http://focal.inria.fr>.

4.1. Implémentation

4.1.1. Définition du Framework

Le modèle de BLP étant déjà défini en Focal (environ 2 000 lignes de code, sans tenir compte de la librairie Focal), seules les espèces correspondants aux sujets, aux objets, aux classifications et aux besoins-d'en-connaître sont à définir. Dans l'exemple d'utilisation de notre prototype, nous avons utilisé l'ensemble ordonné $\{public \leq confidential \leq secret\}$ pour les classifications et l'ensemble $\{Air, Terre, Marine\}$

pour les besoins-d'en-connaître. Chaque sujet ainsi que chaque objet est représenté par un identifiant entier unique.

Nous définissons dans la base de données une table *sujets* contenant pour chaque sujet son identifiant unique, son login, son mot de passe, ainsi que son niveau de sécurité. De même, il existe une table *objets* contenant pour chaque objet son identifiant unique, le nom de la table à laquelle il correspond ainsi que son niveau de sécurité. Nous avons choisi de relier un objet à une table complète et non pas à un tuple pour plusieurs raisons. La première est qu'associer un objet à un tuple nécessite la définition d'un niveau de sécurité à chaque tuple, ce qui est relativement coûteux en termes d'occupation mémoire. La deuxième est que pour connaître les tuples concernés par une requête SQL, il faut au préalable exécuter cette requête, ce qui peut poser des problèmes de sécurité. Par exemple, dans le cas d'une requête DELETE, des informations pourraient être effacées de manière illicite. Il aurait alors fallu mettre en place un système de transaction afin d'exécuter la requête pour connaître les tuples concernés, puis de pouvoir revenir à l'état précédent si la requête viole la politique de sécurité. La dernière raison pour laquelle nous avons choisi de ne pas définir un objet comme un tuple est que cela conduit à un phénomène de polyinstanciation (Sandhu *et al.*, 1998), c'est-à-dire lorsque deux tuples ont la même clé primaire, mais des informations et des niveaux de sécurité différents. Des solutions existent pour pallier ce problème (Bertino *et al.*, 2005), mais elles sortent du cadre de la définition de ce prototype.

Puisque les tables *sujets*, *objets* ainsi que les tables utilisées pour stocker l'état BLP courant ne sont pas présentes dans la table des objets, elles sont donc invisibles aux utilisateurs. Cela garantit qu'aucune requête passant par le moniteur de référence ne pourra modifier ces informations, sauf si bien sûr elles proviennent de la fonction de transition de BLP ou du moniteur de référence lui-même.

4.1.2. Architecture

L'architecture de l'implémentation se décompose globalement en trois parties : la politique de sécurité, le gestionnaire de base de données et le moniteur de référence. Comme nous l'avons dit précédemment, la partie politique de sécurité, entièrement définie en Focal, repose principalement sur l'implémentation déjà réalisée du modèle de BLP. Elle fournit au moniteur de référence la fonction de transition de BLP, ainsi que des fonctions permettant de créer un état BLP. L'accès au gestionnaire de base de données s'effectue grâce à une interface définie en Focal proposant des fonctions de connexion/déconnexion à la base de données, ainsi que des fonctions permettant l'exécution d'une requête et la récupération du résultat. Ces fonctions reposent sur la librairie *ocaml-mysql*, et sont spécifiques au gestionnaire de base de données MySQL. N'étant pas entièrement définies dans Focal, il est relativement difficile d'énoncer des propriétés sur ces fonctions, et donc de faire des preuves. De plus, le serveur MySQL est configuré pour n'accepter que les connexions provenant du moniteur de référence. Enfin, le moniteur de référence introduit la syntaxe abstraite des requêtes SQL, les relations $\langle \rangle_g$ et $\langle \rangle_r$, ainsi qu'une fonction de top-level dont nous décrivons le fonctionnement ci-dessous permettant l'exécution d'une requête SQL.

4.1.3. *Le moniteur de référence*

Nous donnons ici le pseudo-code du moniteur de référence pour l'évaluation des requêtes en mode session.

```

Début
  Connexion à la base de données ;
  Saisir login/mot de passe de l'utilisateur ;
   $\rho \leftarrow$  Création état BLP ;
  Faire
     $R \leftarrow$  Saisir requête SQL ;
    Si  $\langle R, \rho \rangle_g = (\text{yes}, \rho')$  Alors
      exécuter  $R$  ;
       $\rho \leftarrow \rho'$  ;
      empiler  $\langle R \rangle_r$  ;
    Sinon Afficher Message erreur ; Fin Si ;
  Tant Que  $R \neq \text{Exit}$  ;
  Dépiler tous les  $\langle R \rangle_r$ , les évaluer et modifier  $\rho$  ;
  Stocker  $\rho$  dans la base de données ;
Fin

```

5. Conclusion

Ce développement montre que les méthodes formelles peuvent être utilisées dans la pratique pour accroître la sécurité d'un système. En effet, la formalisation en Focal de la fonction de transition de BLP ainsi que de sa preuve et la définition d'un moniteur de référence relativement simple permet de vérifier « mathématiquement » ce dernier. De plus, le fait de configurer le serveur MySQL afin de n'accepter que les connexions provenant du moniteur de référence garantit que ce dernier sera systématiquement appelé. Enfin, la sémantique forte de Focal permet de s'assurer de l'absence d'erreurs à l'exécution, et de prévenir ainsi une attaque reposant sur la défaillance du moniteur.

Bien qu'ayant montré son intérêt, l'implantation décrite ici n'a pas dépassé le statut de prototype (environ 400 lignes de codes pour la sémantique des requêtes, 100 lignes pour le moniteur et 600 lignes pour les fonctions d'accès à MySQL). Elle a toutefois servi de guide à une implantation en cours de réalisation en milieu industriel et des optimisations en termes de temps d'exécution sont envisageables. Il serait par exemple intéressant de limiter le nombre de requêtes BLP exécutées. En effet, avec notre sémantique, plusieurs requêtes SQL différentes peuvent correspondre aux mêmes requêtes BLP. Par exemple, si un utilisateur effectue plusieurs SELECT de suite sur la même table, il demande à chaque fois le même accès en lecture. S'il n'y a pas d'accès en parallèle, il suffit alors de faire une seule demande d'accès en lecture, puis un seul relâchement d'accès.

Cette formalisation nous a permis de distinguer trois modes d'accès possible en fonction du degré de sécurité voulu : le mode requête, facile à intégrer mais pouvant violer la propriété de sécurité- \star de BLP par canal caché (l'écran par exemple),

le mode strict où les accès ne sont jamais relâchés, et enfin le mode session où les accès sont empilés/dépilés. Le choix du mode d'évaluation des requêtes dépend du niveau de sécurité voulu, ainsi que du niveau de confiance portée aux utilisateurs. Dans le cas d'une base de données d'un haut niveau de sécurité dans un système ouvert, c'est-à-dire lorsque les utilisateurs se connectent au serveur grâce à un réseau non fiable (comme internet), il est préférable d'utiliser le mode strict. En revanche, dans un système fermé où il est possible de contrôler les autres opérations effectuées par un utilisateur, et donc de prévenir un certain nombre de canaux cachés, on peut utiliser le mode session.

L'architecture en trois parties distinctes de l'implémentation permet d'envisager une extension à d'autres politiques de sécurité ainsi qu'à d'autres gestionnaires de base de données. En effet, en intercalant entre le moniteur de référence et la politique de sécurité une interface proposant une définition générique de politiques de sécurité, il est possible d'intégrer un autre contrôle d'accès, tel que celui basé sur les rôles (Ferraiolo *et al.*, 1992). Il serait alors intéressant de comparer cette approche en termes de performances et de pouvoir d'expression à une implantation intégrée comme (Ramaswamy *et al.*, 1998). De récents travaux (Jaume *et al.*, 2006b) introduisent un cadre formel dans lequel il est possible de définir de nombreuses politiques de contrôle d'accès, notamment celle de Bell et La Padula. Ce cadre formel, possédant un pouvoir d'expression plus riche que l'algèbre de sécurité de McLean conviendrait à une telle approche. Il faudrait alors définir une sémantique du langage SQL plus générale, ne reposant pas uniquement sur le modèle de BLP. De plus, cette sémantique ne dépendant pas du gestionnaire de base de données, il est possible d'implémenter ce moniteur de référence dans un autre gestionnaire, à condition toutefois de définir dans Focal les fonctions de bas niveau permettant d'accéder à la base de données.

Pour conclure, ce travail montre que les méthodes formelles peuvent être utilisées avec succès dans la pratique lors du développement de logiciels en charge de la sécurité d'un système. S'il est acceptable que tout le code ne soit pas entièrement prouvé, le cœur de sécurité, en l'occurrence la fonction de transition de la politique de sécurité, est défini dans un environnement permettant d'énoncer des propriétés et de les prouver. De plus, l'emploi de l'atelier Focal facilite la réutilisation, permettant ainsi la définition d'un moniteur de référence générique pouvant implanter *a priori* n'importe quelle politique de contrôle d'accès dans n'importe quel gestionnaire de base de données relationnelle, en minimisant à chaque fois la quantité de code à réécrire.

Remerciements

Nous remercions vivement les relecteurs pour leurs remarques constructives, ainsi que Damien Doligez, Emmanuel Guréghian, Thérèse Hardin, Eric Jaeger, Mathieu Jaume, Virgile Prevosto et Renaud Rioboo pour toutes les discussions fructueuses que nous avons eues à propos de ce travail.

6. Bibliographie

- Anderson J. P., Computer Security Technology Planning Study, Technical Report n° ESD-TR-73-51, USAF Electronic Systems Division, Hanscom Air Force Base, Bedford, Massachusetts, October, 1972.
- Bell D., LaPadula L., Secure Computer Systems : a Mathematical Model, Technical Report n° MTR-2547 (Vol. II), MITRE Corp., Bedford, MA, May, 1973.
- Bertino E., Sandhu R. S., “Database Security-Concepts, Approaches, and Challenges”, *IEEE Trans. Dependable Sec. Comput.*, vol. 2, n° 1, 2005, p. 2-19.
- Ferraiolo D. F., Kuhn D., “Role-Based Access Control”, *Proceedings of the 15th National Computer Security Conference*, 1992.
- Gureghian E., Hardin T., Jaume M., A full formalisation of the Bell and Lapadula security model, Technical Report n° 2003-007, Univ. Paris 6, LIP6, 2003.
- Harrison M. A., Ruzzo W. L., Ullman J. D., “Protection in operating systems”, *Commun. ACM*, vol. 19, n° 8, 1976, p. 461-471.
- McLean J., “The Algebra of Security”, *Proc. IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, 1988, p. 2-7.
- Jaume M., Morisset C., “A formal approach to implement access control”, *Journal of Information Assurance and Security*, vol. 2, June, 2006, p. 137-148.
- Jaume M., Morisset C., “Towards a formal specification of access control”, *Proceedings of the LICS-Affiliated Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis*, 2006.
- Lampson B., “Protection”, *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*, Princeton University, 1971, p. 437-443.
- Logical Project, *The Coq Proof Assistant Reference Manual Version 7*, INRIA-Rocquencourt. 2002.
- McLean J., “The Specifications and Modeling of Computer Security”, *IEEE Computer*, vol. 23, n° 1, 1990, p. 9-16.
- Prevosto V., Conception et Implantation du langage FoC pour le développement de logiciels certifiés, PhD thesis, Université Paris 6, sep, 2003.
- Ramaswamy C., Sandhu R., “Role-Based Access Control Features in Commercial Database Management Systems”, *Proc. 21st NIST-NCSC National Information Systems Security Conference*, 1998, p. 503-511.
- Rioboo R., Doligez D., Prevosto V., Jaume M., Maarek M., Dubois C., Fechter S., Ménissier-Morain V., Pons O., Delahaye D., Viguié V., Hardin T., *Focal, version 3.1 Tutorial and reference manual*, LIP6/INRIA/CNAM. 2006, <http://focal.inria.fr>.
- Sandhu R., Chen F., “The multilevel relational (MLR) data model”, *ACM Trans. Inf. Syst. Secur.*, vol. 1, n° 1, 1998, p. 93-132.
- Sandhu R. S., “The Typed Access Matrix Model”, *SP '92 : Proceedings of the 1992 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1992, p. 122.

A. Sémantique de SQL

A.1. Règles d'inférence pour $\langle \rangle_g$

On définit dans un premier temps les règles d'inférence pour $\langle \rangle_g$, qui correspondent aux demandes d'accès.

A.1.1. \perp

Dans n'importe quel état BLP, \perp est toujours accepté :

$$(\perp) \frac{}{\langle \perp, s, c, \rho \rangle_g \rightarrow (\text{yes}, \rho)}$$

A.1.2. CREATE

La création d'une nouvelle table est refusée si l'évaluation du *select-statement* est refusée :

$$(CRE_g^1) \frac{\langle sel, s, c, \rho \rangle_g \rightarrow (\text{no}, \rho)}{\langle CRE(o, sel), s, c, \rho \rangle_g \rightarrow (\text{no}, \rho)}$$

De même, la création d'une nouvelle table est refusée si l'affectation du niveau de sécurité du sujet à un objet inactif est refusé :

$$(CRE_g^2) \frac{\langle sel, s, c, \rho \rangle_g \rightarrow (\text{yes}, \rho') \quad t_C((\sigma_\epsilon, C, \sigma_\epsilon, o, \chi_f(o, c)), \rho') = (\text{no}, \rho')}{\langle CRE(o, sel), s, c, \rho \rangle_g \rightarrow (\text{no}, \rho)}$$

En revanche, si l'affectation du niveau de sécurité est accepté, alors l'évaluation du CREATE est acceptée si et seulement si l'activation de l'objet par le sujet est acceptée par la fonction t_C de BLP (cf section 2.1.3).

$$(CRE_g^3) \frac{\langle sel, s, c, \rho \rangle_g \rightarrow (\text{yes}, \rho') \quad t_C((\sigma_\epsilon, C, \sigma_\epsilon, o, \chi_f(o, c, E)), \rho') = (\text{yes}, \rho'')}{\langle CRE(o, sel), s, c, \rho \rangle_g \rightarrow t_C((\sigma_\epsilon, C, \sigma(s), o, \chi_\epsilon), \rho'')}$$

A.1.3. INSERT

L'insertion de tuples comme le résultat d'un SELECT est refusée si l'évaluation du SELECT est refusée :

$$(INS_g^1) \frac{\langle sel, s, c, \rho \rangle_g \rightarrow (\text{no}, \rho)}{\langle INS(o, sel), s, c, \rho \rangle_g \rightarrow (\text{no}, \rho)}$$

Si le *select-statement* est correct, alors l'évaluation du INSERT est égal au résultat renvoyé par la fonction t_G BLP lors d'une demande d'accès en ajout.

$$(INS_g^2) \frac{\langle sel, s, c, \rho \rangle_g \rightarrow (\text{yes}, \rho')}{\langle INS(o, sel), s, c, \rho \rangle_g \rightarrow t_G((\sigma_\epsilon, G, \sigma(s), o, \chi_A(a)), \rho')}$$

A.1.4. *DELETE*

L'évaluation du DELETE est la même que celle du INSERT, excepté que l'on appelle la fonction de transition sur une demande d'accès en écriture.

$$(DEL_g^1) \frac{\langle sel, s, c, \rho \rangle_g \rightarrow (\text{no}, \rho)}{\langle DEL(o, sel), s, c, \rho \rangle_g \rightarrow (\text{no}, \rho)}$$

$$(DEL_g^2) \frac{\langle sel, s, c, \rho \rangle_g \rightarrow (\text{yes}, \rho')}{\langle DEL(o, sel), s, c, \rho \rangle_g \rightarrow t_G((\sigma_\epsilon, G, \sigma(s), o, \chi_A(w)), \rho')}$$

A.1.5. *UPDATE*

Etant donné que la requête UPDATE ne peut pas dépendre du résultat d'un SELECT, on peut directement appeler la fonction t_G de BLP sur une demande d'accès en écriture.

$$(UPD_g) \frac{}{\langle UPD(o), s, c, \rho \rangle_g \rightarrow t_G((\sigma_\epsilon, G, \sigma(s), o, \chi_A(w)), \rho)}$$

A.1.6. *SELECT*

L'évaluation du SELECT est identique à celle du INSERT, excepté que l'on appelle la fonction t_G de BLP sur une demande d'accès en lecture.

$$(SEL_g^1) \frac{\langle sel, s, c, \rho \rangle_g \rightarrow (\text{no}, \rho)}{\langle SEL(o, sel), s, c, \rho \rangle_g \rightarrow (\text{no}, \rho)}$$

$$(SEL_g^2) \frac{\langle sel, s, c, \rho \rangle_g \rightarrow (\text{yes}, \rho')}{\langle SEL(o, sel), s, c, \rho \rangle_g \rightarrow t_G((\sigma_\epsilon, G, \sigma(s), o, \chi_A(r)), \rho')}$$

A.1.7. *DROP*

La requête DROP ne dépendant pas du résultat d'une requête SELECT, son évaluation est celle de la fonction t_D de BLP appliquée à une demande de destruction d'objet.

$$(DRO_g) \frac{}{\langle DRO(o), s, c, \rho \rangle_g \rightarrow t_D((\sigma_\epsilon, D, \sigma(s), o, \chi_\epsilon), \rho)}$$

A.2. Règles d'inférence pour $\langle \rangle_r$

Dans un deuxième temps, on peut définir les règles d'inférence pour $\langle \rangle_r$ qui sont quasiment identiques à celles de $\langle \rangle_g$, sauf pour les requêtes INSERT, DELETE, UPDATE et SELECT. En effet, au lieu d'appeler la fonction t_G pour effectuer une demande d'accès, on appelle la fonction t_R pour une demande de relâchement.

A.2.1. INSERT

$$(INS_r^1) \frac{\langle sel, s, c, \rho \rangle_r \rightarrow (\text{no}, \rho)}{\langle INS(o, sel), s, c, \rho \rangle_r \rightarrow (\text{no}, \rho)}$$

$$(INS_r^2) \frac{\langle sel, s, c, \rho \rangle_r \rightarrow (\text{yes}, \rho')}{\langle INS(o, sel), s, c, \rho \rangle_r \rightarrow t_R((\sigma_\epsilon, R, \sigma(s), o, \chi_{\mathcal{A}}(\mathbf{a})), \rho')}$$

A.2.2. DELETE

$$(DEL_r^1) \frac{\langle sel, s, c, \rho \rangle_r \rightarrow (\text{no}, \rho)}{\langle DEL(o, sel), s, c, \rho \rangle_r \rightarrow (\text{no}, \rho)}$$

$$(DEL_r^2) \frac{\langle sel, s, c, \rho \rangle_r \rightarrow (\text{yes}, \rho')}{\langle DEL(o, sel), s, c, \rho \rangle_r \rightarrow t_R((\sigma_\epsilon, R, \sigma(s), o, \chi_{\mathcal{A}}(\mathbf{w})), \rho')}$$

A.2.3. UPDATE

$$(UPD_r) \frac{}{\langle UPD(o), s, c, \rho \rangle_r \rightarrow t_R((\sigma_\epsilon, R, \sigma(s), o, \chi_{\mathcal{A}}(\mathbf{w})), s, c, \rho)}$$

A.2.4. SELECT

$$(SEL_r^1) \frac{\langle sel, s, c, \rho \rangle_r \rightarrow (\text{no}, \rho)}{\langle SEL(o, sel), s, c, \rho \rangle_r \rightarrow (\text{no}, \rho)}$$

$$(SEL_r^2) \frac{\langle sel, s, c, \rho \rangle_r \rightarrow (\text{yes}, \rho')}{\langle SEL(o, sel), s, c, \rho \rangle_r \rightarrow t_R((\sigma_\epsilon, R, \sigma(s), o, \chi_{\mathcal{A}}(\mathbf{r})), s, c, \rho')}$$

A.2.5. *Autres requêtes*

Pour toutes les autres requêtes Q (y compris \perp), l'évaluation par $\langle \rangle_R$ est toujours acceptée.

$$(OTH_r) \frac{}{\langle Q, s, c, \rho \rangle_r \rightarrow (\text{yes}, \rho)}$$

Article reçu le 9 mai 2006

Accepté après révisions le 15 février 2007

***Julien Blond** effectue un doctorat en informatique à l'Université Pierre et Marie Curie en partenariat avec l'entreprise Bertin Technologies. Ses travaux portent sur la formalisation d'un compilateur afin de produire du code certifiable à un haut niveau normatif (notamment pour les Critères Communs). Il s'intéresse plus particulièrement à l'application de ses travaux à des politiques de sécurité dans les applications systèmes certifiés.*

***Charles Morisset** termine un doctorat en informatique à l'Université Pierre et Marie Curie. Ses travaux portent sur la conception et la définition d'un cadre mathématique permettant de formaliser des politiques de contrôle d'accès. Il s'intéresse notamment à la définition d'outils génériques permettant de comparer des politiques de contrôle d'accès.*