4.6 Corrections

Solution de l'exercice 39

1. — `enter` a `into` $M(s, o)$:
   $(S', O', M') = (S, O, M)$ si $(s, o) \notin S \times O$
   $S' = S, O' = O, \forall (s', o') \neq (s, o).M'(s', o') = M(s', o'), M'(s, o) = M(s, o) \cup \{a\}$ sinon.
   — `delete` a `into` $M(s, o)$:
   $(S', O', M') = (S, O, M)$ si $(s, o) \notin S \times O$
   $S' = S, O' = O, \forall (s', o') \neq (s, o).M'(s', o') = M(s', o'), M'(s, o) = M(s, o) \backslash \{a\}$ sinon.
   — `create subject` s :
   $(S', O', M') = (S, O, M)$ si $s \in S$
   $(S' \cup \{s\}, O, M)$ sinon
   — `delete subject` s
   $(S', O', M') = (S, O, M)$ si $s \notin S$
   $(S' \backslash \{s\}, O, M)$ sinon
2. Attention aux cas limites pour l'exécution de commande, comme la vérification $(s, o) \in S \times O$, pour `enter` a `into` $M(s, o)$. On obtient la matrice suivante :

	Fichier1	*Fichier3*	*Fichier4*	*Fichier5*
Alice (a)	owns	rw		
Bob (b)	r	rw	owns	
Charly (c)	r	owns	rw	
Denise (d)		r	rw	

3. Elle exprime la délégation du droit de lecture du propriétaire de o vers s_2. La commande permet de garantir qu'un sujet n'a pas accès en lecture a un fichier sauf si ce droit a été explicitement accordé par le propriétaire.
4. **command** $confer.write(s_1, s_2, o)$
   if $owns \in M(s_1, o)$ **then**
   `enter` w `into` $M(s_2, o)$
   end if
   command $revoke.read(s_1, s_2, o)$
   if $owns \in M(s_1, o)$ **then**
   `delete` r `into` $M(s_2, o)$
   end if
5. **command** $create.file(s, o)$
   if $o \notin O \wedge s \in S$ **then**
   `create object` o
   `enter` $owns$ `into` $M(s, o)$
   for all $s' \in S \backslash \{s\}$ **do**
   `enter` r `into` $M(s', o)$
   end for
   end if
6. Les SGF privilégient la forme ACL car les objets du domaines sont les fichiers, il s'agit donc d'une vision naturelle vis-à-vis de l'application. De plus, comme d'une part il y a beaucoup de fichiers et peu d'utilisateurs, et d'autre part car les primitives des SGF vont porter sur la modification des propriétés du fichiers, il vaut mieux indexer selon les fichiers pour des raisons de performances. La vision CL est adaptées sur des systèmes centrés sur l'utilisateur,

comme l'authentification à un service web, où la chaîne associée à l'utilisateur représente la collection des droits dont il dispose.

Solution de l'exercice 40

1. *cl* signifie *clearance*[1].

	1	2	3	4
a	w	rw	w	rw
b	w	r	rw	r
c	w	r	w	r

2. Deux lois définissent la transmission des droits dans un modèle à niveaux :
   — $(s, r, o) \in M$ si seulement si $cl(s) \geq la(o)$
   — $(s, w, o) \in M$ si seulement si $cl(s) \leq la(o)$
   Si $(s, r, o) \in M \wedge (s, w, o) \in M$ alors $cl(s) \leq la(o) \wedge cl(s) \geq la(o)$, par antisymétrie on a $la(o) = cl(s)$. C'est immédiat dans l'autre sens, si $la(o) = cl(s)$ alors $cl(s) \geq la(o) \wedge cl(s) \leq la(o)$ et donc $(s, r, o) \in M \wedge (s, w, o) \in M$.
3. Si on peut toujours dériver une matrice à partir d'une politique à niveaux, l'inverse n'est pas toujours possible. C'est par exemple impossible si $\exists (s, o) \in S \times O$ tel que $M(s, o) = \emptyset$ quand les niveaux ont un *ordre total*. En effet, par totalité soit $cl(s) \geq la(o)$ donc $(s, r, o) \in M$; soit $cl(s) \leq la(o)$ et alors $(s, w, o) \in M$.
   Le contre-exemple suivant est valable dans le cas général :

	1	2
a	rw	rw
b	rw	r

   D'après la propriété précédente on a $cl(a) = ca(1) = ca(2)$ sur la première ligne et $cl(b) = ca(1) \neq ca(2)$ sur la deuxième, ce qui est impossible. On a donc plus de matrices que d'images matricielles de politiques à niveaux : ces dernières sont donc moins expressives en un sens.
4. Pour avoir le diagramme de Hasse de l'ordre produit il faut dessiner le cube obtenu en « faisant glisser » l'ordre $(\{\emptyset, \{compta\}, \{inge\}, \{compta, inge\}\}, \subseteq)$ le long de l'axe perpendiculaire à celui $public \leq prive$.
5. Cela permet de disposer de niveaux de sécurité expressifs, obtenus par la combinaison de niveaux de confidentialités d'une part ($public \leq prive$) et de classification des objets par des annotations d'autre par ($\{compta, inge\}$). Cela permet de combiner deux politiques définies sur des niveaux indépendants et de faire prendre les décisions d'accès sur l'ordre produit. C'est ainsi que le modèle est d'ailleurs présenté dans l'article fondateur de Bell et LaPadula.

Solution de l'exercice 41

1. Pour se faire, on commencer par calculer le produit de *URA* et *PRA* comme matrices booléennes. Ensuite, on regroupe les colonnes `ri` et `wi` qui concernent un même objet `i`. On obtient la matrice vue en cours.
2. C'est l'exemple de hiérarchie dans le support de cours.
3. Ce sont deux cas limites :
   — les rôles sans utilisateurs directs sont des *rôles abstraits* (c.f., parallèle avec classes *abstraites en objets*), qui permettent de factoriser des privilèges communs aux descendants du rôle. On peut les supprimer en dupliquant les affectations de permissions aux rôles. Ce peut être intéressant car ces rôles peuvent avoir un sens en tant que groupes de permissions.

1. http://en.wikipedia.org/wiki/Security_clearance

— les rôles sans privilèges directs apparaissent en cas d'héritage multiple. On peut les supprimer en dupliquant les affectations de rôles aux utilisateurs. C'est à privilégier plutôt que multiplier des rôles pour représenter des affectations multiples.

Solution de l'exercice 42

1. On prend plutôt une arête dirigée pour l'héritage et non-dirigée pour l'exclusion (car elle est symétrique). Pour les affectations, on prendra une couleur différente et un symbole différents pour les nœuds utilisateurs et permissions. On ne représentera que les utilisateurs et permissions affectés, pas ceux hérités.
2. On a plusieurs erreurs :
   1. une erreur statique, $D \succeq A$ (par transitivité) et $D \succeq G$, or $(A, G) \in Ssd$: il s'agit d'une erreur de configuration, soit on enlève l'héritage multiple soit l'exclusion ;
   2. une erreur statique, $E \succeq C$ (par transitivité) et $E \succeq B$, or $(B, C) \in Ssd$;
   3. une erreur statique, $B \succeq F$ et $B \succeq A$, or $(F, A) \in Ssd$; de plus, il y a une utilisateur explicitement affecté à ce rôle, ce qui devrait être impossible.
   4. une erreur dynamique, $role(s_1) = \{B, D\}$ or $\{B, D\} \not\subseteq \{r \mid URA(user(s_1), r)\}$: une telle session ne devrait pas pouvoir être ouverte.

Solution de l'exercice 43

C'est un exercice sur la modélisation de l'exclusion mutuelle, et en un sens, sur l'intégrité des politiques basées sur les rôles en présence de contraintes et de hiérarchie. En effet, si on assimile le modèle RBAC à un schéma de relation (Datalog éventuellement si besoin de la récursion pour les fermetures transitives) et une politique à une instance sur le schéma (une base de fait en extension pour Datalog), alors les règles de l'exercice sont essentiellement des contraintes qui garantissent l'intégrité des instances.

1. On reformulera la règle *l'exclusion est transmise par la relation d'héritage* par *si un rôle s hérite d'un rôle a et que a est en exclusion avec un rôle b, alors le rôle s est aussi en exclusion avec b*. Plutôt qu'en logique, on pourrait représenter (tout aussi formellement) ces règles graphiquement, sous forme de règles de graphes.

σ_1	*aucun rôle n'est en exclusion avec lui même* $Ssd(r, r) \Rightarrow \bot$
σ_2	*la relation d'exclusion est symétrique* $Ssd(r_1, r_2) \Rightarrow Ssd(r_2, r_1)$
σ_3	*deux rôles en exclusion mutuelle n'héritent pas l'un de l'autre* $r_1 \succeq r_2 \Rightarrow \neg Ssd(r_1, r_2) \wedge \neg Ssd(r_2, r_1)$
σ_4	*aucun rôle n'hérite de deux rôles en exclusion mutuelle* $Ssd(r_1, r_2) \wedge s \succeq r_1 \wedge s \succeq r_2 \Rightarrow \bot$
σ_5	*si un rôle s hérite d'un rôle r_1 et que r_1 est en exclusion avec un rôle r_2, alors le rôle s est aussi en exclusion avec b* $s \succeq r_1 \wedge Ssd(r_1, r_2) \Rightarrow Ssd(s, r_2)$.

2. Soit $\Sigma = \{\sigma_1, \sigma_2, \sigma_5\}$, on doit montrer que $\Sigma \models \sigma_3$ et $\Sigma \models \sigma_4$. On donne une preuve rédigée, mais une preuve formelle inspirée par des règles de déduction (e.g., déduction naturelle ou règle de typage pour un λ calcul) serait tout aussi claire.
   1. On peut remarquer que $\sigma_2 \wedge (r_1 \succeq r_2 \wedge Ssd(r_1, r_2) \Rightarrow \bot) \Rightarrow \sigma_3$. On se contentera donc de prouver cette nouvelle formule. Supposons $r_1 \succeq r_2$ et $Ssd(r_1, r_2)$, Ssd étant symétrique (σ_2) on a $Ssd(r_2, r_1)$. En appliquant σ_5 avec $r_1 \succeq r_2$ et $Ssd(r_2, r_1)$ on obtient $Ssd(r_1, r_1)$ ce

qui conduit à une contradiction car la relation Ssd est irréflexive (σ_1). Ainsi, on a prouvé $\Sigma \models r_1 \succeq r_2 \wedge Ssd(r_1, r_2) \Rightarrow \bot$.

2. Supposons $Ssd(r_1, r_2)$, $s \succeq r_1$ et $s \succeq r_2$. Ssd étant symétrique (σ_2) on a $Ssd(r_2, r_1)$. On applique σ_5 avec $s \succeq r_1$ et $Ssd(r_1, r_2)$, on obtient $Ssd(s, r_2)$. Par symétrie on a $Ssd(r_2, s)$. On applique σ_5 à nouveau avec $s \succeq r_2$ et $Ssd(r_2, s)$ pour obtenir $Ssd(r_2, r_2)$, ce qui est une contradiction car la relation Ssd est irréflexive. Ainsi, on a prouvé $\Sigma \models Ssd(r_1, r_2) \wedge s \succeq r_1 \wedge s \succeq r_2 \Rightarrow \bot$. Notons qu'on peut obtenir une preuve plus courte en utilisant σ_3 démontrée juste avant.

3. La première motivation est scientifique : on énonce une vérité et on donne les moyens de la vérifier la preuve. On peut trouver une motivation technique lorsqu'on doit implémenter un système RBAC : il faut s'assurer que on ne va pas mettre la politique dans un état incohérent en vérifiant les règles données lorsque des opérations administratives sont effectuées. Les propriétés de la question 2 garantissent que les règles σ_3 et σ_4 sont en fait implicitement vérifiées si on teste seulement les 3 autres. Notons que l'on a pas démontré que ces règles suffisent à s'assurer de la cohérence du système, mais qu'elles sont juste nécessaires.

Solution de l'exercice 44

1. On a $U_iC \succeq Ens$ et $U_iR \succeq U_iC$ pour $i = 1 \dots 3$. PhD est hors de la hiérarchie. On peut aussi créer deux rôles supplémentaires C et R dont héritent respectivement les C_i et les R_i.
2. Les droits RU_i sont affecté à Ens, les droits WU_i aux U_iC et SU_i aux U_iR correspondants.
3. On peut traduire la contrainte « aucun étudiant en thèse ne peut être responsable d'une UE » en imposant $(PhD, U_iR) \in Ssd$ pour chaque UE. Si on a introduit un rôle R comme indiqué, alors il suffit de spécifier $(PhD, R) \in Ssd$.
4. Pour la première, une exclusion entre les U_iR suffit. Pour la seconde, il faudrait disposer d'une notion d'exclusion plus fine. Pour la troisième, il faudrait introduire des contraintes de cardinalité que l'on ne peut pas exprimer avec de l'exclusion.

Solution de l'exercice 45

TODO

Solution de l'exercice 46

1. Voir slide 28 du support de cours.
2. Voir slide 29 du support de cours. On choisira éventuellement un identificateur pour les permissions et on garantira la contrainte d'unicité de session (exactement un utilisateur par session) avec un attribut de la relation session.
3. Il faut vérifier l'inclusion $SU \bowtie SR \subseteq URA$ lors de l'insertion dans SR. Un `CHECK` sera sans doutes insuffisant, on prendra un trigger `BEFORE INSERT`.
4. On évalue la requête booléenne $\pi_\emptyset(\sigma_{U=\%u \wedge A=\%a \wedge O=\%o}(SU \bowtie SR \bowtie PRA))$ et on regarde si son résultat n'est pas vide pour autoriser l'accès.
5. Pour n niveaux, on va ajouter une famille de relations $RH_1, RH_2 \subseteq R \times R$ et étendre le terme $SU \bowtie SR \bowtie PRA$ de la réponse précédente à $(SU \bowtie SR) \bowtie \pi_{\{R,R'\}}(\rho_{B/X}(RH_1) \bowtie \rho_{A/X}(RH_2)) \bowtie \rho_{R/R'}PRA$
6. Il n'y a pas de calcul de couverture de transitive natif en algèbre relationnelle mais :
   — les SGBD-R peuvent l'implémenter ;
   — on peut la calculer une bonne fois pour toute tant que les hiérarchies sont stables ;

Solution de l'exercice 47

TODO

Solution de l'exercice 48

Note : L'évaluation *security constraints* étudiées dans cet exercice est implémentée dans Tomcat par la méthode `findSecurityConstraints(...)` dont le code est accessible dans le fichier source `/apache/catalina/realm/RealmBase.java`.

1. La décision est booléenne, les actions sont les méthodes HTTP, les objets les urls du serveur. Quant aux sujets, ils sont en fait implicites : ce sont les utilisateurs authentifiés qui accèdent au serveur.
2. Pour les cas où plusieurs règles sont applicables, l'explication du $\emptyset$ tient dans la citation :

   > *The special case of an authorization constraint that names no roles shall combine with any other constraints to override their affects and cause access to be precluded.*

url-pattern	http-method	roles
/*	DELETE	$\emptyset$
/*	PUT	$\emptyset$
/acme/wholesale/*	DELETE	$\emptyset$
/acme/wholesale/*	GET	{CONTRACTOR, SALESCLERK}
/acme/wholesale/*	POST	{CONTRACTOR}
/acme/wholesale/*	PUT	$\emptyset$
/acme/retail/*	DELETE	$\emptyset$
/acme/retail/*	GET	{CONTRACTOR, HOMEOWNER}
/acme/retail/*	POST	{CONTRACTOR, HOMEOWNER}
/acme/retail/*	PUT	$\emptyset$

3. L'idée est d'arriver à apprécier le caractère assez surprenant de la sémantique donnée, en particulier les choix qui sont faits pour savoir si une règle est applicable :

   > *Otherwise the container shall determine if the HTTP method of the request is constrained at the selected pattern. If it is not, the request shall be accepted.*

   On désigne par $\top$ l'ensemble des utilisateurs, y compris ceux qui ne sont pas identifiés.

url-pattern	http-method	roles
/acme/	PUT	$\top$
/acme/	DELETE	$\top$
/acme/	GET	{HOMEOWNER}

4. $$A \otimes B = \begin{cases} \emptyset & \text{si } A = \emptyset \vee B = \emptyset \\ A \bigsqcup B & \text{sinon} \end{cases}$$

   Avec $A \bigsqcup B = min_{\leq}\{x \mid A \leq x \wedge B \leq x\}$, la borne supérieure de A et B dans le treillis $\mathcal{L}$, *id est*, le plus petit des éléments qui soit à la fois plus grand que A et plus grand que B.
   On apprécie ainsi le caractère assez curieux de la définition de l'opération. En effet, du point de vue algébrique, on s'attendrait à ce que $\emptyset$ soit un élément neutre plutôt qu'un élément asborbant pour $\otimes$!
5. On pourrait proposer :
   — de modifier la sémantique (*id est*, de changer la spécification) pour que le cas par défaut, quand aucune règle ne s'applique, soit de refuser l'accès au lieu de l'autoriser ;

— de modifier la sémantique de l'applicabilité d'une règle : on teste simultanément l'url-pattern *et* la méthode HTTP, et non pas alternativement comme spécifié actuellement ;
— un jeu d'essai plus complet et qui permet d'apprécier les comportements « aux limites » ;
— un ensemble de bonnes pratiques pour éviter les comportements contre-intuitifs, typiquement, pour chaque chemin spécifié dans un `<wrcl>` doit être associée une règle par défaut ;
— de formaliser la spécification.

Solution de l'exercice 49

1. Les *targets* de `PS_patient`, `P_patient_record` et `P_medical_record` sont toutes satisfaites. Aucune règle `RP1`, `RP2` ou `RP3` ne matche. La décision de `P_patient_record` est donc $\bot$ car la combinaison est `d-o`. Pour `P_medical_record`, on ne matche que `RM2` est on obtient $\top_d$. La décision finale est $\top_d$ car la combinaison de $\bot$ et $\top_d$ par `d-o` est $\top_d$.
2. On peut construire une requête qui matches deux targets en concaténant les attributs demandés par exemple par `RP2` et `RM1`.
3. On donne par exemple le code C++ suivant.

```
1  //g++ -Wall -Wextra -Weffc++ -std=c++14
2  #include <iostream>
3  #include <vector>
4
5  using namespace std;
6
7  enum dec_t {PERMIT=1, DENY=-1, NA=0}; //NA = Not Applicable
8
9  dec_t deny_overrides(const vector<dec_t> &v){
10   dec_t r = NA;
11   for (auto const &x : v){
12     if (x == DENY)
13       return DENY;
14     else if (x == PERMIT)
15       r = PERMIT;
16   }
17
18   return r;
19 }
20
21 int main (void){
22   vector<vector<dec_t>> test_set;
23   test_set = {
24     {},
25     {PERMIT},
26     {DENY},
27     {NA},
28     {PERMIT, NA},
29     {PERMIT, DENY},
30     {NA, DENY},
31     {NA, PERMIT, DENY},
32   };
33
34   for (size_t i = 0; i < test_set.size(); ++i)
35     cout << "\tdeny_overrides(" << i << ") = " << deny_overrides(test_set[i
         ]) << endl;
36
37   return EXIT_SUCCESS;
38 }
```

Solution de l'exercice 50

1. 1. moindre privilège : les utilisateurs ne doivent disposer que des droits minimaux pour effectuer leurs tâches ;
   2. interdiction par défaut : principe de la *black-list* le paquet est interdit *a priori* : seul le traffic explicitment autorisé peut traverser le pare-feux (contrairement au code pénal, où tout ce qui n'est pas explicitement interdit est permis) ;
   3. défense en profondeur : on érige des barrières successives (e.g., métaphore des périmètres d'enceinte concentriques dans un chateau fort) en plusieurs endroits ; comme l'antivirus dans le serveur de mail et sur le client, idem pour le filtrage de paquet ;
   4. goulet d'étranglement : on fait en sorte que *toutes* les communications passent par le pare-feu : il ne doit pas exister d'autre point d'entrée dans le réseau (c.f., critère *non-bypassable* des moniteurs de contrôle d'accès). L'introduction de nouveaux vecteurs de communications non contrôlés est une menace (e.g., modem, wifi) ;
   5. simplicité : *Keep It Simple and Stupid* pour éviter les (déjà trop fréquentes) erreurs d'administration ;
   6. participation des utilisateurs : comme la qualité, la sécurité est l'affaire de tous, même pour le filtrage de paquet : les utilisateurs doivent comprendre les raisons et les objectifs de sécurité du pare-feu pour en accepter la contrainte ;

Solution de l'exercice 51

1. Sous l'hypothèse que ces spécifications sont exhaustives, en appliquant le principe du moindre privilège on ne doit autoriser *que* ces services. Il faut alors pour chaque service extérieur `ssh` ou `www` : n'autoriser que les paquets provenant du port consacré vers la machine, et symétriquement, n'autoriser la machine à utiliser que ces ports-ci pour l'extérieur. Pour le service interne `ssh`, le comportement est symétrique. Par application du principe d'interdiction par défaut, on rejete tous les autres paquets.

source	*port*	*destination*	*port*	*protocole*	*action*	SYN
*	*	203.167.75.1	22	tcp	accept	ok
203.167.75.1	22	*	*	tcp	accept	nok
*	22	203.167.75.1	*	tcp	accept	nok
203.167.75.1	*	*	22	tcp	accept	ok
*	80	203.167.75.1	*	tcp	accept	nok
203.167.75.1	*	*	80	tcp	accept	ok
*	*	*	*	tcp	drop	ok

2. On doit accepter les SYN uniquement pour la connection de l'extérieur sur le port 22 de la machine, pour la connection vers l'extérieur sur les ports 80 et 22. Pour les autres lignes, il ne faut pas accepter les SYN car la connection est supposée déjà être établie.