Access Control Mechanisms

Thomas Chabrier : thomas.chabrier@etu.u-bordeaux1.fr
Léo Letouzey : leo.letouzey@etu.u-bordeaux1.fr

6 décembre 2007

Table des matières

Chapitre 1

Introduction

Le besoin de politiques de sécurité pour les systèmes d'information est apparu dès les premières études sur les problématiques de confidentialité et d'intégrité. Elles font partie intégrante des critères d'évaluation de la sécurité, tant au niveau international dans le TCSEC, qu'au niveau européen dans ITSEC.

Afin de comprendre ce que sont les contrôles d'accès et l'administration de politiques de sécurité, nous commencerons par une présentation des différents modèles théoriques qui ont été développés pour y répondre, puis les implantations existant aujourd'hui.

Chapitre 2

Modèles théoriques de contrôle d'accès

Avec le contrôle d'accès, on s'intéresse à garantir deux propriétés fondamentales : la confidentialité et l'intégrité des informations contenues dans un SI.
La **confidentialité** étant la prévention des accès non autorisés à une information, et l'**intégrité**, la prévention des modifications non autorisées d'une information.

Afin de permettre l'implantation de politiques de confidentialité et d'intégrité en leur sein, les systèmes d'exploitation disposent de mécanismes de contrôle d'accès. Typiquement, ceux-ci fonctionnent sur le modèle suivant :

- Un sujet est une entité active du système (essentiellement les processus) ;
- Un objet estune entité passive, un conteneur d'information, sur lequel un sujet peut effectuer une action (les fichiers, sockets de communication, périphériques matériels. . .) ;
- Une permission est une certaine action sur un objet. Une permission peut être accordée ou refusée (par exemple, lecture, écriture, exécution. . .).

Le contrôle d'accès est configuré par un ensemble de règles spécifiant un sujet, un objet et des droits d'accès. Un cas particulier de règles est celui spécifiant une action d'un sujet vers un autre sujet (envoi de signal ou de message inter-processus).

La manière naturelle de représenter les règles d'accès est sous forme matricielle, par une matrice d'accès. L'ensemble des sujets correspond aux lignes, et l'ensemble des objets aux colonnes. Dans chaque case de la matrice, on trouve un sous-ensemble des permissions.

Ainsi, une contrainte sur la confidentialité se traduira par le refus de la permission de lecture : par exemple, pour empêcher que le serveur Web **apache** n'accède au fichier contenant les mots de passe des utilisateurs, `/etc/shadow`, on ne lui autorise pas la lecture de celui-ci.

La notion de matrice d'accès a été introduite par Lampson en 1971. Il propose de placer suivant les lignes l'ensemble D des domaines de protection (qui représentent des contextes d'exécution pour les programmes, i.e. les sujets), et en colonnes l'ensemble X des objets (qui inclut les domaines), dans la matrice A. Il pose ensuite deux définitions :

Capabilities Lists
Étant donné un domaine $d \in D$, la liste des capacités (capabilities) pour le domaine d est l'ensemble des couples $(o, A[d, o]), \forall o \in X$.

Access Control Lists
Étant donné un objet $o \in X$, la liste de contrôle d'accès (ACL) pour l'objet o est l'ensemble des couples $(d, A[d, o]), \forall d \in D$.

La première définition lie un domaine d avec les permissions $A[d, o]$, de la matrice d'accès A, qu'il possède sur chaque objet o, et la seconde définition lie un objet o avec les permissions $A[d, o]$ que la matrice A accorde à chaque domaine d.

Ensuite, il est nécessaire de pouvoir faire évoluer ces politiques de sécurité, afin par exemple de prendre en compte l'intégration de nouveaux utilisateurs et applications dans le système d'exploitation. Cette évolution doit être contrôlée pour prévenir les abus, ainsi que les configurations indésirables pour la politique. C'est pourquoi chaque mécanisme de contrôle d'accès définit des procédures de modification des droits d'accès sur les objets du système. Elles sont plus ou moins restrictives suivant le but recherché, et requièrent éventuellement un accès de niveau administrateur.

Typiquement, sous Linux, ces procédures sont accomplies par les commandes `chown` et `chmod`. La première modifie le propriétaire d'un fichier ou dossier, et ne peut-être utilisées que par root. La seconde modifie les droits d'accès sur un fichier ou dossier, et requiert que l'utilisateur en soit propriétaire. Par conséquent, pour modifier les droits d'accès d'un fichier appartenant à root, il est nécessaire d'utiliser le compte root.

Enfin, la croissance des vulnérabilités liées aux interactions entre des systèmes d'informations distants a poussé de nouvelles études sur l'administration de politiques à grande échelle. Ainsi, la prise en compte de multiples sites logiques d'application (différents réseaux par exemple) ou de différentes organisations amenées à coopérer sont les apports des recherches en matière de politiques multi-domaines. Également, le support des équipements relevant de langages de politiques de sécurité différents est au cœur des études sur les modèles multi-politiques.

Les parties suivantes présentent les différentes familles de modèles théoriques. On en distingue principalement trois :

Contrôle d'accès discrétionnaire

La caractéristique principale du DAC ou *Discretionary Access Control* est le fait que ce sont les utilisateurs qui attribuent les permissions sur les ressources qu'ils possèdent. C'est le type de mécanisme utilisé principalement dans les systèmes d'exploitation modernes. En effet, il est très léger en termes d'administration, étant donné que l'attribution des droits est faite par les utilisateurs et non pas par les administrateurs.

Contrôle d'accès obligatoire

Contrairement au DAC, le MAC ou *Mandatory Access Control* délègue l'attribution des permissions à une entité tierce, typiquement un administrateur externe de la politique de sécurité. Ainsi, les utilisateurs du système ne peuvent pas intervenir dans l'attribution des permissions d'accès, même s'ils disposent de droits d'administration dans le système d'exploitation.

Modèle de contrôle d'accès

Il existe plusieurs de modèle de contrôle d'accès, qui sont basés soient sur les rôles, les tches, ou sur le concept d'organisation. Ces modèles ont pour but de simplifier l'administration des droits d'accès des utilisateurs individuels en fournissant un niveau d'indirection supplémentaire.

2.1 DAC

Le contrôle d'accès discrétionnaire (DAC) est le modèle implanté dans la majorité des systèmes d'exploitation actuels (Microsoft Windows, Solaris, Linux, FreeBSD). Il doit son nom au fait que l'attribution des permissions d'accès se fait à la discrétion du propriétaire d'une ressource. Typiquement, les droits d'accès sur un fichier sont positionnés par l'utilisateur déclaré comme propriétaire de ce fichier.

Les modèles théoriques de DAC que nous allons étudier maintenant sont HRU et TAM.

2.1.1 Modèle HRU

Une matrice P représente l'ensemble des droits d'accès des sujets vers les objets. De plus, les sujets peuvent créer de nouveaux sujets ou objets, et ajouter des droits.

HRU propose de modéliser la protection dans les systèmes d'exploitation comme suit : l'ensemble des sujets S et l'ensemble des objets O sont tous deux l'ensemble des entiers de 1 à k. R est l'ensemble des

droits génériques (tels que possession, lecture, écriture, exécution). Le système d'exploitation contient un ensemble fini C de commandes $\alpha 1, ..., \alpha n$, qui représente toutes les opérations fournies par le système d'exploitation (création de fichier, modification des droits...). Du point de vue de la protection, les commandes de l'ensemble C ont toutes la même forme : elles prennent en paramètre des sujets et objets, et suivant la présence de certains droits dans la matrice P , elles effectuent un ensemble d'actions élémentaires sur le système. Ces actions élémentaires sont : `enter` et `delete` pour l'ajout et la suppression de droits, `create subject` et `create object` pour la création de nouveaux sujets et objets et enfin `destroy subject` et `destroy object` pour la destruction de sujets et objets. La configuration du système de protection est le triplet (S, O, P).

Pour étudier le problème de la sûreté d'un système de protection, HRU s'intéresse au transfert de droit, qui se produit lorsqu'une commande insère un droit particulier r dans la matrice P , dans une case ou il était précédemment absent. Par conséquent, étant donné une configuration initiale de la politique de sécurité, un système est considéré sûr (*safe*) pour un droit r si aucune des commandes ne provoque le transfert du droit r.

Les auteurs du modèle HRU ont prouvé que dans le cas d'un système de protection mono-opérationnel, i.e. dans lequel toutes les commandes ne contiennent qu'une seule action élémentaire, le problème de savoir si ce système est sûr est décidable, toutefois l'algorithme de vérification est NP-complet. Or la seule commande de création de fichier d'un système d'exploitation de type UNIX se compose déjà de deux actions : création d'un nouvel objet, et positionnement des droits sur celui- ci. Les auteurs s'intéressent donc au problème dans le cas général, et prouvent que le problème de la sûreté d'un système de protection est indécidable dans le cas général.

En outre, les auteurs mentionnent le fait que la taille du problème est réduite à une taille polynomiale dès lors que les actions de création de sujet ou d'objet sont retirées du système de protection.

2.1.2 Modèle TAM

Le modèle *Typed Access Matrix* (TAM), introduit par Sandhu en 1992, propose une extension du modèle HRU, en intégrant la notion de typage fort. Cette notion dérive de travaux plus anciens sur SPM (*Sandhu's Schematic Protection Model*) et se traduit par l'attachement de types de sécurité immuables à tous les sujets et objets du système d'exploitation.

Par rapport à la modélisation de HRU vue précédemment, il existe maintenant l'ensemble T des types de sécurité. Le point important est que cet ensemble est fini : la création de nouveaux types n'est pas possible. La gestion des types est prise en compte dans les opérations élémentaires décrites plus haut pour HRU.

Ensuite, Sandhu s'intéresse à la version monotone de TAM, soit MTAM (*Monotonic Typed Access Matrix*), obtenue en ôtant les opérations de suppression (droits, sujets ou objets). Il démontre que le problème de la sûreté est décidable dans le cas d'un modèle de protection MTAM où le graphe de création des sujets et objets est acyclique. Toutefois la complexité de ce problème reste NP. C'est pourquoi Sandhu définit le modèle MTAM ternaire, dans lequel toutes les commandes ont au maximum trois arguments. Au prix d'une perte d'expressivité, le problème de sureté voit sa complexité ramenée à un degré polynomial.

2.2 MAC

L'un des grands avantages du DAC est que l'attribution des permissions d'accès est gérée directement par les utilisateurs. Malheureusement, c'est aussi l'une de ses défaillances majeures. En effet, s'il est possible de se protéger contre des attaques externes (provenant de réseaux informatiques distants) en mettant en place des barrières de protection supplémentaires, il est par contre difficile de se prémunir contre les actions des utilisateurs malicieux. En particulier, les failles logicielles menant à l'obtention d'un accès de niveau administrateur fournissent à un pirate le moyen d'outrepasser complètement le DAC.

Pour intégrer un mécanisme de validation des accès entre sujets et objets, soit dans le système d'exploitation, soit directement dans les composants matériels, Anderson propose le modèle du Moniteur de Référence (*Reference Monitor*). Celui-ci est configuré à partir d'une matrice d'accès fixe, définie par une entité tierce externe au système, et spécifiant exhaustivement quels accès sont autorisés ou refusés entre les sujets et objets du système d'exploitation. Ainsi il devient possible de garantir que des droits d'accès considérés dangereux ne seront jamais donnés par erreur, puisque la matrice d'accès est fixe.

Le concept de Moniteur de Référence est au cœur de la définition du Contrôle d'accès obligatoire (Mandatory Access Control ou MAC). Ce terme, qui désignait initialement le modèle défini par Bell et LaPadula, a vu sa signification évoluer et représente aujourd'hui tout mécanisme qui place la gestion de l'attribution des permissions d'accès hors d'atteinte des utilisateurs concernés par ces permissions.

Les différents modèles présentés par la suite se positionnent à l'intérieur du Moniteur de Référence, comme mécanisme de validation. Ils ont tous pour objectif de spécifier une certaine politique de sécurité en matière de contrôle d'accès.

2.2.1 Modèle Bell-LaPadula

Le modèle Bell-LaPadula, établi par Bell et La Padula en 1973, plus couramment appelé BLP, est issu des besoins en confidentialité des données du monde militaire. En plus des notions de sujet, d'objet et de matrice d'accès, le modèle BLP introduit la notion de **label**. Un label est associé à chaque sujet et objet du système, et contient un niveau de sécurité. Ceux-ci sont composés de deux types d'identifiants de sécurité :

1. un identifiant hiérarchique : le **niveau de classification** pour les objets, ou **niveau de sensibilité**, et le niveau d'habilitation pour les sujets, qui sont typiquement non classifié, confidentiel, secret, top secret ;
2. des identifiants de catégories : indépendamment de la hiérarchie de confidentialité, différentes catégories d'informations existent, et correspondent aux différentes organisations manipulant ces données.

Un niveau de sécurité s'écrit donc : (classification, ensemble de catégories). De plus, il existe une relation de dominance entre les niveaux de classification. Sur cette base, il est maintenant possible de définir de nouvelles règles qui s'appliqueront en plus de la matrice d'accès classique. Ces nouvelles lois sont :

ss-property
Appelée *simple security property* ou *No Read Up*, elle garantit que lorsqu'un sujet demande un accès en lecture sur un objet, son niveau est supérieur ou égal à celui de l'objet (cf. flèche de gauche sur la figure 1).

***-property**
Appelée *star-property* ou *No Write Down*, cette loi garantit que lorsqu'un sujet demande un accès en écriture sur un objet, son niveau est inférieur ou égal à celui de l'objet (cf. flèche de droite sur la figure 1).

Le système est donc modélisé de la façon suivante : un ensemble de sujets S, un ensemble d'objets O, une matrice d'accès M et une fonction donnant le niveau f . On dispose également d'un ensemble de permissions d'accès $A = e, r, a, w$. Elles sont classées suivant leur capacité d'observation (lecture) et d'altération (écriture) de l'information :
`e` Ni observation ni altération (*execute*) ;
`r` Observation sans altération (*read*) ;
`a` Altération sans observation (*append*) ;
`w` Observation et altération (*write*).

Les deux lois sur les niveaux, illustrées par la figure 1, s'écrivent ainsi :

$$r \in M[s, o] \Rightarrow f(s) \geq f(o) \tag{2.1}$$
$$a \in M[s, o] \Rightarrow f(s) \leq f(o) \tag{2.2}$$
$$w \in M[s, o] \Rightarrow f(s) = f(o) \tag{2.3}$$

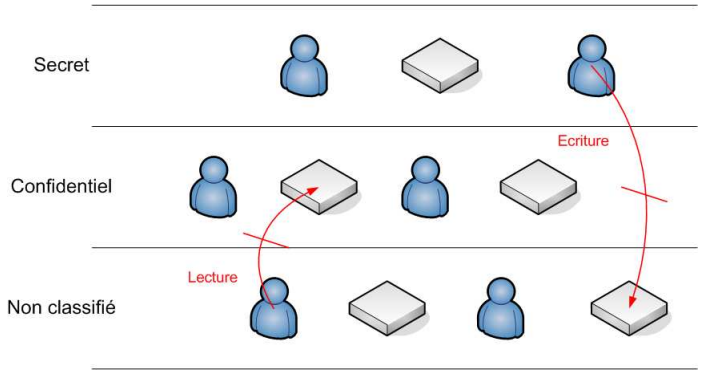

Fig 1 No Read Up et No Write Down

Ces lois signifient que 1) pour qu'un sujet s ait accès en lecture à un objet o, son niveau d'habilitation $f(s)$ doit être supérieur ou égal au niveau de classification $f(o)$ de l'objet, 2) pour qu'un sujet s ait accès en écriture à un objet o, son niveau d'habilitation doit être inférieur ou égal au niveau de classification de l'objet, et 3) pour qu'un sujet ait accès en lecture/écriture à un objet o, son niveau d'habilitation doit être égal au niveau de classification de l'objet.

Ce modèle a historiquement été implanté dans de nombreux systèmes d'exploitation. L'article de Bell et LaPadula décrit son intégration dans MULTICS, on le trouve également dans certaines versions de Solaris, HPUX ou autres systèmes UNIX. Il n'est pas désigné sous le nom BLP, mais plutôt MLS (*Multi-Level Security*) ou modèle de sécurité multi-niveaux. Par exemple, le système d'exploitation Unix System V/MLS intègre la gestion des niveaux de sécurité décrits dans le modèle BLP.

Toutefois ce modèle est difficile à appliquer tel quel à l'ensemble d'un système d'exploitation. Il est difficile d'attribuer des labels à certains sujets ou objets, par exemple le cas du dossier `/tmp` où tous les processus doivent pouvoir créer des fichiers. Des aménagements ont été prévus, le fait de migrer un objet d'un niveau de classification à un niveau supérieur lorsqu'il est accédé en écriture par un sujet de niveau supérieur (car normalement il est interdit de modifier un fichier de niveau inférieur). L'objet se trouve après l'accès au même niveau que le sujet. Mais l'effet néfaste lié à cet aménagement est que les objets ont tendance à être tirés vers le haut, et donc à tous se trouver sur le niveau le plus haut après un certain temps.

2.2.2 Modèle DTE

Le modèle *Domain and Type Enforcement* (DTE) est assez différent des précédents dans le sens où il ne cherche pas à garantir la sûreté de la protection qu'il fournit. Il s'agit plutôt d'un mécanisme générique à partir duquel on peut implanter diverses politiques de sécurité dans un système d'exploitation. A partir d'implémentations disponibles pour les systèmes d'exploitation commerciaux, il est particulièrement efficace pour les problématiques de confinement de processus. Le confinement diffère des propriétés de confidentialité ou d'intégrité. En effet, son objectif est de spécifier, pour chaque sujet du système, un ensemble d'actions autorisées. Il se rapproche du modèle TAM sur l'utilisation du typage fort.

Concrètement, dans un système d'exploitation, les politiques de sécurité définies via le modèle DTE visent à :

- Restreindre les ressources accessibles par un programme, notamment les programmes privilégiés (s'exécutant sous le compte root), suivant le principe de moindre privilège ;
- Contrôler quels programmes ont accès aux ressources sensibles, et empêcher l'accès par tout autre programme.

Par exemple, DTE peut être utilisé pour contrôler un programme tel que le serveur Web apache, afin de garantir par exemple que même s'il s'exécute sous le compte root, il n'a accès qu'à ses fichiers de configuration, ses librairies et aux pages web. Il pourra également garantir que seul apache ait accès à son fichier de configuration.

Le modèle reprend les principes de sujet, d'objet et de matrice d'accès. Toutefois, les lois d'accès n'opèrent pas sur les sujets, mais les **domaines**, et de même les objets sont englobés dans des **types**. Ainsi chaque objet du système possède un type, et chaque processus s'exécute dans un domaine précis, dont découlent des droits d'accès.

DTE définit trois tables :

1. La **table de typage**, qui assigne les types aux objets du système ;
2. La **table de définition des domaines** (DDT) qui spécifie les droits d'accès (lecture, écriture, exécution, ajout, suppression) de chaque domaine sur les différents types ;
3. La **table d'interaction des domaines** (DIT) qui définit les droits d'accès entre domaines (création, destruction, envoi de signal).

La DDT définit également les points d'entrée des domaines, c'est à dire quels sont les fichiers binaires qui, une fois exécutés, vont créer un processus dans tel ou tel domaine. La définition des trois tables est effectué par un administrateur, et les utilisateurs ne peuvent pas les modifier. C'est pourquoi DTE rentre dans la catégorie MAC .

2.3 Contrôle d'accès

I-BAC (*Identity Based Access Control*) a été le premier modèle de contrôle d'accès proposé dans la littérature. Ce modèle introduit les concepts fondamentaux de sujet, d'action et d'objet. L'objectif du modèle I-BAC est de contrôler tout accès direct des sujets aux objets via l'utilisation des actions. Ce contrôle est basé sur l'identité du sujet et l'identificateur de l'objet. Ce modèle introduit ensuite le concept de politique d'autorisation, représentée par une matrice de contrôle d'accès (les lignes et colonnes de la matrice correspondent respectivement à l'ensemble des sujets et des objets du SI).

A l'usage, une limite importante du modèle I-BAC est apparue : la politique d'autorisation devient rapidement complexe à exprimer et administrer. Il est en effet nécessaire d'énumérer les autorisations pour chaque sujet, action ou objet. En particulier, lorsqu'un nouveau sujet ou objet est créé, il est nécessaire de mettre à jour la politique d'autorisation pour définir les nouvelles permissions associées à ce sujet ou objet.

Pour pallier ce problème, d'autres modèles ont été définis. Tous ces modèles ont en commun de proposer une expression plus structurée de la politique d'autorisation. Nous présentons dans les paragraphes suivants, les modèles proposant respectivement une structuration des sujets, des actions et des objets.

2.3.1 Modèle R-BAC

Le modèle R-BAC (*Role Based Access Control*) propose de structurer l'expression de la politique d'autorisation autour du concept de rôle. Un rôle est un concept organisationnel : des rôles sont affectés aux utilisateurs conformément à la fonction attribuée à ces utilisateurs dans l'organisation.

Le principe de base du modèle R-BAC est de considérer que les autorisations sont directement associées aux rôles. Dans R-BAC, les rôles reoivent donc des autorisations de réaliser des actions sur des objets. Comme I-BAC, le modèle R-BAC ne considère que des autorisations positives (permissions) et fait donc l'hypothèse que la politique est fermée.

Un autre concept introduit par le modèle R-BAC est celui de session. Pour pouvoir réaliser une action sur un objet, un utilisateur doit d'abord créer une session et, dans cette session, activer un rôle qui a reu l'autorisation de réaliser cette action sur cet objet. Si un tel rôle existe et si cet utilisateur a été affecté à ce rôle, alors cet utilisateur aura la permission de réaliser cette action sur cet objet une fois ce rôle activé.

Lorsqu'un nouveau sujet est créé dans le SI, il suffit d'affecter des rôles au sujet pour que ce sujet puisse accéder au SI conformément aux permissions accordées à cet ensemble de rôles. Comparé au modèle I-BAC, la gestion de la politique d'autorisation s'en trouve simplifiée puisqu'il n'est plus nécessaire de mettre à jour cette politique chaque fois qu'un nouveau sujet est créé.

Dans R-BAC, il est également possible d'organiser les rôles de faon hiérarchique. Les rôles héritent des autorisations des autres rôles qui leur sont hiérarchiquement inférieurs.

De nombreux SI mettent en œuvre ce standard, par exemple Unix Solaris à partir de la version 8 ou l'API Authorization Manager RBAC de Windows Server 2003.

2.3.2 Modèle T-BAC

Le modèle T-BAC (*Task Based Access Control*) fut le premier modèle à introduire le concept de tche. D'autres modèles ont ensuite été développés pour contrôler l'exécution des activités dans un workflow. En particulier, l'utilisateur ne doit obtenir une permission que lorsque c'est nécessaire pour poursuivre l'exécution de l'activité considérée.

Il est parfaitement possible de combiner les concepts de rôle et de tche pour spécifier une politique d'autorisation et obtenir ainsi un modèle appelé TR-BAC (*Task and rôle BAC*). Dans ce cas, les permissions affectées aux rôles portent sur la réalisation des tches. Diverses contraintes peuvent être spécifiées pour, par exemple, spécifier qu'un même sujet doit intervenir dans certaines actions nécessaires à la réalisation de la tche (éventuellement avec des rôles différents).

2.3.3 Modèle Or-BAC

Dans Or-BAC, l'expression d'une politique d'autorisation est centrée sur le concept d'organisation (contrairement à R-BAC où la politique d'autorisation est centrée sur le concept de rôle). Les concepts de rôles, de vues et d'activités sont des concepts organisationnels. Chaque organisation définit ainsi les rôles, les activités et les vues dont elle souhaite réglementer l'accès en appliquant une politique d'autorisation.

Concept de vues

Les vues servent à faciliter l'expression et la gestion d'une politique d'autorisation, grce à un concept qui structure les objets. L'expression d'une politique de sécurité en SQL repose sur le concept de vue. Nous désignerons donc par V-BAC (*View Based Access Control*) ce type de modèle de contrôle d'accès. Intuitivement, dans une base de données relationnelle, une vue correspond au résultat d'une requête SQL auquel on a donné un nom. Ce concept de vue est ensuite utilisé pour structurer l'expression d'une politique d'autorisation à l'aide des instructions `GRANT` (qui permet d'accorder une nouvelle permission à un utilisateur) et `REVOKE` (qui permet de supprimer une permission que possédait un utilisateur). Une vue constitue donc un moyen efficace pour accorder un accès à l'ensemble des objets contenus dans la vue.

Le modèle Or-BAC introduit donc trois relations *relevant-role*, *relevant-activity* et *relevant-view* pour spécifier respectivement les rôles, les activités et les vues gérés par l'organisation. Ensuite, chaque organisation spécifie les affectations des sujets aux rôles en utilisant la relation ternaire *empower*. On peut remarquer que cette modélisation permet, par exemple, de considérer qu'un même sujet est affecté à des rôles différents suivant l'organisation considérée. De faon similaire, deux autres relations ternaires consider et use permettent de respectivement spécifier, pour chaque organisation, les relations entre action et activité d'une part, et entre objet et vue d'autre part.

Le modèle Or-BAC offre également la possibilité de spécifier des *role-definition*, *view-definition* et *activity-definition*. Une *role-definition* est une condition logique qui, si elle est satisfaite, permet de conclure qu'un sujet se trouve automatiquement affecté au rôle correspondant à la *role-definition*. De faon similaire, une *view-definition* et une *activity-definition* correspondent à des conditions logiques pour gérer respectivement les vues et les activités. Une politique d'autorisation s'exprime en spécifiant, pour chaque organisation considérée, les activités que les rôles ont la permission de réaliser sur les vues.

La politique d'autorisation s'exprime donc de façon complètement indépendante des ensembles de sujets, actions et objets gérés par le système d'information. On parle ainsi de politique d'autorisation **organisationnelle**. Le modèle propose une règle de passage pour dériver automatiquement, d'une politique d'autorisation organisationnelle, les permissions concrètes s'appliquant aux sujets, actions et objets. Cette règle spécifie que, dans une organisation donnée, si un sujet est affecté à un certain rôle, un objet est utilisé dans une certaine vue, une action implante une certaine activité et si la politique d'autorisation de cette organisation spécifie que ce rôle a la permission de réaliser cette activité sur cette vue, alors on peut dériver que ce sujet a la permission de réaliser cette action sur cet objet.

2.3.4 Modèle d'autorisation dynamiques et contextuelles

Dans la pratique, de nombreuses autorisations ne sont pas statiques mais dépendent de conditions qui, si elles sont satisfaites, permettent d'activer dynamiquement les autorisations. Dans ce cas, on parle souvent d'autorisations contextuelles.

Ainsi, les autorisations peuvent dépendre de contextes temporels (par exemple permission pendant les heures de travail), contextes géographiques (par exemple permission à l'intèrieur de l'enceinte sécurisée de l'entreprise), de contextes provisionnels (permission si d'autres actions ont au préalablement été réalisées comme dans le cas d'un workflow), ...

Ces autorisations contextuelles sont représentées par des modèles de sécurité de type Rule-BAC. Dans ces modèles, une politique d'autorisation correspond à un ensemble de règles de la forme **condition** → **permission** qui spécifient qu'une permission peut-être dérivée lorsqu'une certaine condition est satisfaite.

Comparés aux modèles présentés précédemment, les modèles de Rule-BAC présentent une expressivité plus grande, utile pour spécifier des autorisations contextuelles. En contrepartie, on perd malheureusement la structuration de la politique d'autorisation que permettaient l'introduction des concepts de rôle, d'activité, de vue et d'organisation.

2.3.5 Modèle d'autorisation positives et négatives

Initialement, les modèles de contrôle d'accès ne permettaient que d'exprimer des autorisations positives (permissions). Mais des modèles offrant également la possibilité d'exprimer des autorisations négatives (interdictions) ont été proposés.

Combiner des autorisations positives et négatives dans une politique d'autorisation est intéressant pour plusieurs raisons :

- certaines politiques d'autorisation sont plus faciles à décrire en termes d'interdictions que de permissions. Par exemple, considérer une règle spécifiant que l'accès à une certaine vidéo est interdite aux personnes de moins de 12 ans ;
- lorsque la politique d'autorisation doit être mise à jour, il est parfois plus simple d'insérer une interdiction que de supprimer une permission existante ;
- la combinaison des permissions et interdictions constitue un moyen simple pour exprimer des règles présentant des **exceptions**.

Un nouveau problème apparaît lorsque l'on considère des politiques d'autorisation mixtes (elle inclut des autorisations positives et négatives) : celui de la gestion des conflits entre les autorisations positives et négatives. En effet, pour un sujet, une action et un objet donnés, il est possible que la politique d'autorisation permette de dériver que l'accès est à la fois permis et interdit.

Pour résoudre ces situations de conflits, certaines approches ne considèrent que des stratégies simples, telles que les interdictions l'emportent toujours sur les permissions, ou les permissions l'emportent toujours sur les interdictions. Ces stratégies ne conviennent pas pour traiter le problème des exceptions. En effet, si une interdiction agit comme exception à une permission, alors cette interdiction doit l'emporter sur cette permission. Mais dans le cas contraire, c'est-à-dire si une permission agit comme exception à une interdiction, alors c'est la permission qui doit l'emporter sur l'interdiction.

La plupart des pare-feu donnent des exemples représentatifs de mise en oeuvre de ce type de stratégies. En effet, on peut interpréter les règles de filtrage d'un pare-feu comme un ensemble d'autorisations positives (dans le cas où la décision de la règle de filtrage est d'accepter le paquet) ou négatives (dans le cas où la décision est de rejeter le paquet). La priorité entre règles de filtrage correspond en général à l'ordre dans lequel les règles ont été écrites. C'est la stratégie du **first matching**.

L'ordonnancement des règles constitue donc un moyen simple, mais efficace pour résoudre les conflits dans une politique d'autorisation mixte. Cependant, cette ordonnancement rend la politique d'autorisation plus complexe à exprimer et à mettre à jour. En effet, d'autres problèmes peuvent apparaître : le masquage (*shadowing*) et la redondance. L'anomalie de masquage apparaît lorsqu'une règle devient inapplicable car des règles plus prioritaires seront systématiquement appliquées avant elle. L'anomalie de redondance existe lorsqu'une règle peut être supprimée sans que cela change la politique d'autorisation.

Chapitre 3

Implantations système du contrôle d'accès

Dans la partie précédente nous avons abordé les modèles théoriques de contrôle d'accès. Au-delà de la définition de ces modèles, il est essentiel de disposer d'implantations fiables de ceux-ci sur les systèmes d'exploitation courants pour pouvoir réaliser pleinement leurs objectifs. Nous nous intéresserons ici spécialement aux implantations disponibles pour le système Linux. Pour commencer, nous établirons une liste des implantations relevant des modèles théoriques présentés précédemment. Ensuite, nous étudierons plus spécifiquement deux implantations : SELinux et grsecurity.

3.1 Les différentes implantations pour Linux

Le noyau Linux, au cœur des systèmes d'exploitations dits GNU/Linux, est relativement jeune comparé aux modèles théoriques que nous avons étudiés dans la section précédente. Toutefois, les implantations de modèles de contrôle d'accès type MAC et RBAC n'ont commencé que tard dans l'évolution de celui-ci. En effet, le développement de Linux a débuté en 1991, et les premières implantations de mécanismes avancés de contrôlé d'accès (c'est-à-dire autres que le DAC déjà intégré au noyau) ne sont apparues que vers 1997. Les différentes implantations présentées par la suite sont :

- Medusa DS9
- RSBAC
- LIDS
- SELinux
- grsecurity

3.1.1 Medusa DS9

Le projet Medusa DS9 (2006) est le premier à avoir implémenté une forme de MAC pour le noyau Linux, dès 1997. En fait, il reprend le modèle DTE, avec la **notion de Virtual Space (VS)** qui s'apparente aux types.

Dans Medusa, chaque objet est associé à un ou plusieurs VS. Ensuite les processus ont un ensemble de capacités, qui sont des associations entre un ensemble d'opération et un ensemble de VS. Ainsi, on associe des permissions d'accès entre sujets (processus) et objets via les VS.

3.1.2 RSBAC

Issu du travail de Ott (1997), RSBAC (Rule Set-Based Access Control pour Linux) est une architecture générique d'implantation de modèles de sécurité s'appuyant sur **GFAC**.

GFAC est un modèle d'implantation de mécanismes de contrôle d'accès dans les systèmes d'exploitation, qui à pour objectif de supporter la plupart des modèles de contrôle d'accès existants. Pour cela, les auteurs ont, dans un premier temps, distingué les composants d'une politique de contrôle d'accès : l'**autorité**, qui écrit la politique de sécurité, identifie les informations pertinentes pour la sécurité, et

assigne les valeur aux attributs des ressources contrôlées ; les **attributs** décrivent les caractéristiques des sujets et objets, par exemple le type des objets, le domaine des processus, la classification, le propriétaire ; enfin les règles sont des expressions formelles décrivant les autorisations d'accès entre les attributs, en accord avec la politique de sécurité. Dans un second temps, les auteurs ont identifié les deux composants nécessaire à la réalisation des mécanismes de contrôle d'accès :
Access Control Decision Facility (ADF), la partie responsable des décisions de contrôle d'accès, qui connait la politique de sécurité ; et *Access Control Enforcement Facility* (AEF), la partie qui applique les décisions prises par ADF.

Grâce à la généricité de l'architecture, héritée du modèle GFAC, de nombreux modules sont disponibles, et chacun d'eux implante une politique particulière. Dans la suite, on trouvera une liste des modules les plus intéressants par rapport aux modèles vus précédemment.

AUTH : Authenticated User

Le module AUTH est le module principal de RSBAC. Il gère les permissions de changement d'UID. En fait, chaque processus a une liste d'UID possibles (capabilities).

UM : User Management

Le module UM effectue une gestion des utilisateurs et groupes standard de Linux directement dans l'espace noyau. Ainsi les procédures d'authentification et de changement de mot de passe sont implantées via des appels système.

RC : Role Compatibility

Le module RC effectue la gestion des rôles, suivant le modèle RBAC. Chaque rôle a des listes de rôles et types avec lesquels il peut interagir.

ACL : Access Control Lists

Le module ACL implémente les listes de contrôle d'accès. Ce sont des listes de triplets :
- sujet (utilisateur, rôle RC ou groupe spécial ACL)
- objets (par type : FD, DEV, PROCESS, IPC, SCD)
- types de requêtes

MAC : Mandatory Access Control

Le module MAC est une implantation directe du modèle MLS. Il autorise donc l'usage de niveaux de classification et de sensibilité comme décrit dans le modèle BLP.

3.1.3 LIDS

Le projet LIDS, pour Linux Intrusion Detection System, lancé en 1999, avait pour objectif initial l'implantation d'un système de détection d'intrusion dans le noyau Linux. Finalement le travail a débouché sur un mécanisme de contrôle d'accès mandataire, dont un des intérêts est la commande de configuration, qui a une syntaxe proche de celle de iptables (la commande de configuration du pare-feu de Linux).

Les possibilités en matière de contrôle d'accès sont les suivantes :
- les sujets sont les fichiers binaires exécutables
- les objets sont tous les fichiers du système
- les ACL définissent quels types d'accès sont permis entre sujets et objets
- l'ajout ou le retrait de capabilities POSIX est également contrôlable

Par exemple, pour n'autoriser l'accès en lecture à `/etc/shadow` qu'au serveur SSH, on exécutera :

```
lfs# lidsadm -A -o /etc/shadow -j DENY
lfs# lidsadm -A -s /usr/sbin/sshd -o /etc/shadow -j READ
```

Autre exemple, pour empêcher que le serveur apache ne tente d'utiliser un port TCP autre que le port standard 80 :

```
lfs# lidsadm -A -s /usr/sbin/httpd  -o CAP_BIND_NET_SERVICE 80-80 -j GRANT
```

3.2 SELinux

Le projet SELinux, commencé dans les année 2000, est issu des travaux sur la sécurité du système d'exploitation DTOS (à partir de 1997). Un des intérêts majeurs est que ce projet est intégré au développement du noyau Linux, et est à l'origine de Linux Security Modules.

3.2.1 Linux Security Modules

Le projet Linux Security Modules ou LSM est un support pour l'écriture de modules de sécurité pour le noyau Linux. Il intègre dans le noyau des points d'ancrage au niveau des appels système, et via un système de callbacks, il rend possible l'implantation de nouveaux modèles de contrôle d'accès de faon modulaire.

LSM dérive de l'architecture **Flask** (*FLux Advanced Security Kernel*) qui est à l'origine de SELinux.

Dans le but de fournir des mécanismes de contrôle d'accès génériques pour le noyau Linux, le projet LSM relève de deux objectifs précis : l'intégration des nouveaux points d'ancrage dans le code source du noyau (*Security Hooks*), et le support de l'empilement des modules de sécurité.

Security Hooks

L'objectif premier de LSM est l'intégration de points de contrôle dans les parties critiques du noyau, c'est-à-dire les fonctions qui manipulent les entités nommées du système (processus, fichiers, sockets, mémoire partagée, IPC). Dans les structures noyau associées à ces entités, LSM ajoute un champ dédié à la sécurité, utilisé pour l'enregistrement des informations et identifiants spécifiques, notamment les labels de sécurité. Ceux-ci sont essentiels dans le procédé de déploiement d'une politique de sécurité.

Des points de contrôle (*hooks*) sont ajoutés dans les fonctions associées aux appels système, qui constituent les points d'entrée du noyau pour la manipulation des entités nommées du système. Certains sont utilisés pour créer et modifier les champs concernant la sécurité, et le reste (la majorité) sont responsables du contrôle d'accès. Normalement, ces points de contrôle interviennent après les vérifications liées au contrôle d'accès standard du noyau Linux, ainsi les permissions normales s'appliquent toujours. Une des conséquences est que les modules de sécurité implantés via LSM sont des modules restrictifs, ils ne peuvent qu'interdire des actions qui seraient autorisées par Linux.

Enfin, certaines fonctions de LSM sont utilisées lors de l'initialisation du noyau. Ainsi un module de sécurité peut être chargé au démarrage du système d'exploitation, et gérer proprement les activités de démarrage du noyau, par exemple pour créer les structures de sécurité associées aux processus, vérifier si le module de sécurité doit être activé ou ne pas être activé.

LSM fournit également des moyens de communication avec l'espace utilisateur, via un pseudo système de fichiers. En particulier, celui-ci contient des statistiques sur le fonctionnement du module, et l'accès aux différents paramètres de configuration existant pour chaque module de sécurité.

Empilement des modules

Les modules LSM peuvent être empilés, i.e. il est possible de coordonner les fonctions de sécurité de plusieurs modules. Cependant, ce mécanisme comporte une restriction majeure. Le premier module chargé sera considéré comme primaire, et responsable du chargement d'un module secondaire, et ainsi de suite, jusqu'à former une chaîne de modules. Hors chacun des modules est responsable de la prise en compte des décisions d'accès produites par le suivant. Du coup, aucune garantie n'est fournie quant à l'application de la politique de sécurité décrite par un module qui ne serait pas le premier.

3.2.2 Security Enhanced Linux

SELinux est un projet lancé par la NSA, et aujourd'hui totalement intégré au noyau Linux. L'objectif est l'implantation d'un contrôle d'accès de type MAC, robuste et finement configurable, au moyen de l'architecture LSM.

Architecture Flask

SELinux provient de l'architecture Flask dont la caractéristique principale est la séparation des parties de prise de décision (*decision-making*) et d'application (*policy enforcement*). La partie décision est contenue dans un composant appelé **Security Server**, ou serveur de sécurité, qui contient la politique

de sécurité du système. La partie d'application est constituée de l'ensemble des points de contrôle fournis par LSM. A l'origine, la conception de Flask est issue du modèle GFAC , mais par la suite elle a donné lieu à l'implémentation de Linux Security Modules.

SELinux fournit également un système de cache entre ces deux parties, appelé *Access Vector Cache* (AVC). Ainsi un vecteur de permissions d'accès entre un sujet et un objet donnés n'est calculé par le Security Server qu'une seule fois, et ensuite fourni directement par le cache. De plus, dans le cas d'un changement de la politique de sécurité, le cache est bien évidemment invalidé. L'AVC a pour objectif d'améliorer les performances de SELinux.

Security Server

Le Security Server contient la configuration de SELinux, soit la politique de sécurité. Il est responsable des décisions d'accès.

Les informations utilisées pour calculer les permissions d'accès sont les contextes de sécurité du sujet et de l'objet, et la classe de l'objet. Le résultat est un vecteur de permissions, contenant des booléens pour tous les types d'accès associés à la classe de l'objet.

Les décisions sont de deux types : décisions d'accès et décisions de transition. Les décisions d'accès concernent les tentatives d'accès d'un sujet vers un objet, pour accorder ou refuser l'accès. Les décisions de transition ont lieu lorsqu'une nouvelle entité est créée dans le système, pour calculer son contexte de sécurité. Par exemple, ces calculs ont lieu lorsqu'un nouveau processus est exécuté, ou qu'un nouveau fichier est créé.

Contextes de sécurité

Afin de garantir la neutralité du mécanisme de sécurité vis-à-vis de la politique de sécurité à appliquer, Flask utilise des contextes de sécurité (aussi appelés **labels** dans le modèle BLP). Ces labels ne sont compréhensibles que par le *Security Server*. Du point de vue de la partie enforcement, les tentatives d'accès d'un sujet vers un objet ne sont que des interactions entre deux contextes de sécurité.

Le format des contextes est le suivant : **<ID> :<role> :<type>**

La partie ID est liée aux UID standards de Linux, mais diffère dans le sens où il s'agit d'identifiants spécifiques à SELinux. L'intérêt est le suivant : l'UID peut changer au cours des actions de l'utilisateur sur le système (notamment via les programmes ayant le bit setuid activé), mais l'ID SELinux ne change pas, et pointe toujours vers l'utilisateur d'origine (qui a ouvert une session sur le système d'exploitation).

La partie role est liée au modèle RBAC. Dans la définition de la politique de sécurité SELinux, les utilisateurs ont accès à un ensemble de rôles, et chaque rôle représente un ensemble de types manipulables. Lorsqu'un utilisateur se connecte au système d'exploitation, après s'être authentifié, il choisit un rôle parmi l'ensemble de ceux qui lui sont autorisés.

La partie type prend son sens dans le modèle DTE, qui est au cœur de la définition de la politique de sécurité SELinux.

3.3 grsecurity

grsecurity est un projet démarré en 2001, qui a pour objectif global de renforcer la sécurité du noyau Linux. Son développement n'est pas intégré au noyau, et grsecurity se présente donc sous forme d'un patch pour le code source de Linux. Il fournit trois types de protection :

- confinement : pour appliquer le principe de moindre privilège aux programmes du système
- prévention : pour prémunir le système contre les techniques génériques d'exploitation de vulnérabilités (comme par exemple les buffer overflow)
- détection : pour surveiller l'activité du système, et généralement auditer les activités suspectes pour lesquelles il n'existe pas de moyen de prévention

3.3.1 Confinement

grsecurity fournit trois mécanismes de confinement :

- Trusted Path Execution (TPE)
- RBAC
- un système d'ACL

Le but de TPE est d'éviter que les utilisateurs exécutent des programmes auxquels l'administrateur ne fait pas confiance. C'est pourquoi le principe de fonctionnement de TPE est de n'autoriser l'exécution que pour des fichiers se trouvant dans des dossiers appartenant à root, et non modifiables par les utilisateurs (c'est par exemple le cas des dossiers /bin et /usr/bin).

Le système d'ACL de grsecurity ressemble à celui de LIDS. Les sujets sont identifiés par le chemin d'un fichier exécutable, et pour chaque sujet on peut définir des règles. Celles-ci spécifient les droits d'accès sur le système de fichiers, sur les connexions réseau (TCP et UDP), sur les capabilities POSIX et sur la quantité de ressources système utilisable (mémoire, nombre de fichiers ouverts. . .). Ainsi il est possible de définir des ACL soit sur des fichiers précis, soit sur des dossiers.

Le comportement par défaut est qu'un processus fils hérite des droits d'accès de son père. Par exemple, en mettant une ACL sur un dossier, celle-ci sera héritée par tous les fichiers qu'il contient, et par les sous-dossiers. Néanmoins, il existe une directive pour briser l'héritage. Si elle est spécifiée, seuls les droits définis pour ce sujet particulier seront appliquées.

Les fonctionnalités RBAC de grsecurity ont pour objectif d'autoriser la définition de différents ensembles de règles, et de les associer aux utilisateurs suivant leur rôle. Sans cela, tous les utilisateurs ont les mêmes droits, ce qui est difficilement exploitable en environnement multi-utilisateur. De plus, un utilisateur peut, s'il en a le droit, changer de rôle pendant une même session, moyennant une authentification.

Voici un exemple de règles pour le serveur apache, qui restreint l'accès en lecture aux pages web et à la configuration, l'accès en écriture au fichier de log, et qui ne donne que la capability **CAP_SETUID** pour qu'Apache puisse changer d'utilisateur lors de son exécution :

```
/usr/sbin/apache
/var/www r
/var/log/apache w
/etc/apache r

-CAP_ALL
+CAP_SETUID
```

3.3.2 Prévention

grsecurity introduit dans le noyau Linux plusieurs mécanismes de prévention, dont le but est très différent de celui du confinement vu dans le paragraphe précédent. En effet, le confinement réduit l'impact d'une attaque réussie, alors que la prévention rend plus difficile l'exploitation des vulnérabilités connues, habituellement utilisées par les pirates informatiques pour attaquer un système.

- PaX a pour objectif de rendre certaines pages mémoire non exécutables, notamment la pile et le tas des processus, empêchant ainsi les attaques par dépassement de tampon (buffer overflow)
- ASLR (Address Space Layout Randomization) introduit de l'aléa dans le choix des adresses mémoire des processus, notamment au niveau de la pile et des librairies
- Enfin, grsecurity peut rendre aléatoire le choix de certains identifiants du système, notamment les identifiants de paquets IP, de processus,. . .

En outre, grsecurity apporte des renforcements à l'appel système chroot(), normalement utilisé pour changer la racine du système de fichiers pour un processus particulier. Il existe plusieurs possibilités pour s'évader de la nouvelle racine, et grsecurity essaie de prévenir celles qui sont connues.

3.3.3 Détection

grsecurity a de nombreuses possibilités d'audit d'évènements du système, autorisant la surveillance des actions effectuées sur le système d'exploitation, et la détection d'activité suspectes.

D'une part, le système d'ACL peut définir, en parallèle des règles de contrôle d'accès, des règles d'audit, avec la même précision. Ainsi, pour tout droit d'accès pris en considération par grsecurity (lecture, écriture. . .), il est possible de demander l'audit de ce type d'accès, sur l'objet de l'ACL, et donc d'obtenir le résultat des demandes d'accès qui ont lieu.

D'autre part, il est possible d'auditer :

- les appels système effectués par les processus

- les évènements liés à PaX, notamment les tentatives d'exécution dans la pile d'un processus, souvent symptomatiques d'une attaque par buffer overflow
- et beaucoup d'autres évènements spécifiques à la sécurité dand le noyau Linux

Chapitre 4

Discussion des modèles existants

Après avoir décrit les différents modèles de contrôle d'accès et les implantations existantes, dans un second temps les modèles d'expression de politiques de sécurité, on va dégager les caractéristiques essentielles des modèles de contrôle d'accès, des implantations système, et des modèles d'administration de politique, pour déterminer leur intérêt.

Nous allons tout d'abord effectuer une critique des modèles de contrôle d'accès. Nous verrons que la faiblesse du DAC est communément établie. Puis nous verrons que les modèles MAC sont bien moins intéressants que le modèle DTE. Enfin, nous reviendrons sur le modèle RBAC, qui est également indispensable pour notre architecture.

Enfin, nous discuterons les implantations systèmes. Nous verrons que les plus appropriées sont RSBAC, grsecurity, et plus particulièrement SEL INUX.

4.1 Discussion des modèles de contrôle d'accès

Comme précédemment, nous allons discuter des modèles de contrôle d'accès suivant les trois grandes familles existantes. D'abord nous reviendrons sur le contrôle discrétionnaire (DAC), et donc les modèles HRU et TAM. Ensuite nous aborderons la discussion sur le contrôle d'accès obligatoire, en particulier les modèles BLP et DTE. Enfin nous terminerons par la famille de modèle à base de rôles RBAC.

4.1.1 DAC

Le modèle de contrôle d'accès couramment utilisé sur les systèmes d'exploitation actuels est le Discretionary Access Control. Le DAC délègue l'accord des permissions d'accès à la discrétion des propriétaires des ressources du système. Dans le cas des fichiers, il s'agit des utilisateur et groupe propriétaire, pour toutes les autres ressources, c'est un super-utilisateur (par exemple, root) qui est propriétaire.

En pratique, ce modèle de contrôle d'accès a clairement montré ses limites. En effet, les attaques possibles contre les systèmes d'exploitation visent à obtenir un accès de niveau super-utilisateur. Lorsqu'une telle attaque est réussie, l'attaquant obtient des pouvoirs qui outrepassent le DAC et donnent un accès complet à l'ensemble des ressources du système d'information. De fait, la faiblesse de ce contrôle est que la politique de sécurité peut être à tout moment modifiée par le super-utilisateur du système d'exploitation.

De plus, diverses études ont établi la faiblesse des modèles DAC. En effet, le contrôle d'accès discrétionnaire repose sur la capacité des utilisateurs à définir correctement les permissions sur les fichiers dont ils sont propriétaires. Toute erreur peut mener à une défaillance de sécurité, comme par exemple si le fichier `/etc/shadow` venait à être autorisé en lecture pour tous les utilisateurs.

Le modèle HRU établit que le problème de déterminer si le modèle de protection est sûr, c'est-à-dire si on a l'assurance qu'un certain droit d'accès ne sera jamais accordé à un utilisateur donné, est un problème indécidable.

Il ressort de l'analyse des modèles DAC que ceux-ci n'offrent pas de réelle garantie quant à la protection de la confidentialité et de l'intégrité des informations manipulées. Pour que la protection puisse être prouvée, il est nécessaire d'utiliser un modèle MAC. Grâce à ceux-ci, on pourra notamment restreindre

les droits accordés au super-utilisateur (root), et éviter que les utilisateurs modifient les permissions de leurs fichiers de faon erronée.

4.1.2 MAC

Les modèles Mandatory Access Control (MAC) ont donc été envisagés pour répondre à la faiblesse des modèles DAC. Ils reposent sur la délégation du contrôle d'accès à une entité indépendante, et évitent donc que les permissions soient définies de faon incorrecte comme cela peut arriver avec le DAC. L'existence de cette entité indépendante garantit que la politique de sécurité ne soit pas modifiable directement par les utilisateurs du système d'information. On distingue deux orientations dans les modèles de type MAC.

Une première orientation est constituée par les modèles visant à assurer la confidentialité et l'intégrité des données dans les environnement militaires et commerciaux. Ces modèles [Bell et La Padula 1973, Biba 1975, Clark et Wilson 1987, Brewer et Nash 1989] répondent à des problématiques très précises, par exemple la nécessité de disposer d'une habilitation adéquate pour la lecture de documents classifiés dans BLP. Cependant, l'usage courant des systèmes d'exploitation moderne ne peut souvent pas être décrit par une seule de ces problématiques, mais relèvent plutôt de problématiques plus complexes comme le principe de moindre privilège, la confidentialité des données entre utilisateurs, la nécessité de prendre en compte les activités des utilisateurs sur le système.

Une seconde orientation est le modèle DTE. Plutôt que de considérer une problématique particulière, celui-ci fournit un mécanisme générique, et autorise la spécification de politiques adaptées à tout environnement. S'il est possible de l'utiliser pour implanter une politique de type BLP ou BIBA, une autre utilisation envisageable est celle du confinement qui répond notamment aux problèmes posés par les vulnérabilités logicielles, souvent présentes dans les logiciels utilisés couramment. En revanche, si le langage fourni par DTE est simple, il est difficile d'écrire une politique accordée à ses besoins. Par exemple, dans le cas où l'on souhaite confiner une application, il faudra déterminer un ensemble de droits à accorder à celle-ci ni trop restreint (pour ne pas empêcher le fonctionnement), ni trop peu restreint (car le confinement ne serait plus assuré).

En outre, les modèles MAC fournissent une sécurité prouvable, car ils reposent sur des modèles mathématiques, souvent appuyés sur des théorèmes et des preuves. Par exemple, BLP repose sur deux lois qui empêchent la diffusion non autorisée d'information [Bell et La Padula 1973]. Enfin, ces modèles ont une politique de sécurité fixée, et non modifiable par les utilisateurs, ce qui exclut les problèmes typiques d'indécidabilité du DAC.

En revanche, l'utilisation des modèles MAC est complexe. Soit ils fournissent une politique trop restreinte, trop peu générale et sont alors difficile à déployer en pratique. Soit ils fournissent des mécanismes génériques (DTE), mais alors le travail à fournir pour définir la politique de sécurité est bien plus exigeant, et n'inclut pas de garantie contre les erreurs de l'administrateur qui la définit. En définitive, le juste milieu est une combinaison des différents modèles.

4.1.3 RBAC

La notion de **rôle** simplifie grandement la définition des politiques de sécurité. En effet, plutôt que d'attribuer directement des droits à chaque utilisateur du système (un système peut en compter plusieurs milliers), on crée un petit nombre de rôles, et chaque utilisateur est assigné à un ensemble de rôles. Ensuite les droits d'accès sont associés aux rôles plutôt qu'aux utilisateurs.

Cependant, en pratique, RBAC ne se suffit pas à lui-même pour le contrôle d'accès dans un système d'exploitation. Il faut le conjuguer avec un mécanisme de contrôle d'accès de type MAC ou DAC, pour pouvoir l'utiliser. En effet, définir les attributions de droits d'accès entre sujets et objets ne fait pas partie des objectifs des modèles RBAC. RBAC est un composant indépendant du contrôle d'accès, complémentaire à MAC et DAC.

Prenons par exemple le choix fait dans SEL INUX : le modèle de contrôle d'accès est en fait une combinaison de Type Enforcement (TE, un allègement de DTE), RBAC et MLS. Ainsi, les utilisateurs du système d'exploitation sur lequel SEL INUX est déployé ont accès à un ensemble de rôles (modèle RBAC). Ensuite chaque rôle est associé à un ensemble de types qu'il a le droit de manipuler (lien entre RBAC et TE). La politique de contrôle d'accès comprend un ensemble de règles d'interaction entre types

(modèle TE). Enfin MLS est utilisé pour avoir des niveaux de classification des utilisateurs et des données dans le système d'exploitation.

C'est bien la combinaison des modèles RBAC et TE qui est fondamentale dans SEL INUX. Le modèle MLS est aussi intégré, pour le support des niveaux de confidentialité, mais ne constitue pas l'intérêt premier de SE LINUX. En effet, la combinaison TE / RBAC autorise la définition de politiques de protection de niveau système, par une configuration très fine de l'accès aux différents appels système du noyau Linux.

4.2 Implantations système

Nous allons ici étudier les caractéristiques de ces implantations, soit Medusa DS9, LIDS et plus particulièrement RSBAC , SEL INUX et grsecurity.

4.2.1 Medusa

Medusa est le plus ancien projet de MAC pour Linux. Il a démontré qu'on pouvait utiliser un modèle dérivé de DTE pour renforcer la sécurité du contrôle d'accès du système Linux. Cependant, son développement semble être arrêté aujourd'hui.

4.2.2 LIDS

LIDS est une implantation très intéressante du point de vue du confort d'utilisation. Sa commande d'administration a une syntaxe très intuitive et très puissante. LIDS s'appuie sur un modèle dérivé de DTE pour la définition des droits d'accès par processus. De plus il intègre deux fonctionnalités pour améliorer la sécurité :

- La première, Trusted Path Execution, pose deux contraintes sur les fichiers binaires que les utilisateurs normaux peuvent exécuter : ils doivent se trouver dans des dossiers dont le propriétaire est root, et ne doivent pouvoir être modifiés que par leur propriétaire (seul le propriétaire.de l'exécutable a le droit d'écriture sur le fichier binaire).
- La seconde, Trusted Domain Execution, peut placer dans un environnement d'exécution restreint (une sandbox) une application qui a acquis des données par des chemins non sûrs (clavier, fichier pouvant être modifiés par tout les utilisateurs). En effet, à partir du moment où une application reoit des données par un chemin non sûr, il est possible que ces données aient été formatées de faon à exploiter une vulnérabilité présente dans celle-ci. Dès lors, afin de prévenir tout risque de contamination globale du système par une attaque réussie, il est intéressant de placer l'application dans un environnement restreint.

Le fichier de configuration de LIDS est difficilement exploitable directement. En effet, il s'agit d'un format binaire non documenté, et il n'existe pas de librairie de fonctions de manipulations. La configuration se fait plutôt via des scripts invoquant la commande lidsadm (une fois pour chaque droit d'accès à accorder). D'un point de vue système réparti, il serait possible de distribuer des scripts sur l'ensemble du réseau.

4.2.3 RSBAC

RSBAC constitue une architecture très flexible. Il autorise le déploiement de modèles de contrôle d'accès très variés. Son architecture reprend les principes de GFAC, comme SEL INUX, et on retrouve donc d'une part, le composant AEF dans les appels système de Linux, et d'autre part le composant ADF qui contient une partie fixe, et les différents modèles de contrôle d'accès choisis par l'administrateur sous forme modulaire. Les modules fournis par RSBAC implantent notamment les modèles MLS et RBAC. De plus, le module ACL implante un modèle proche de DTE.

Par contre, la configuration de RSBAC se fait principalement via des outils en ligne de commande. On ne trouve pas de documentation sur les formats des fichiers de configuration. De fait, il semble difficile de configurer RSBAC simplement par déploiement d'un fichier de règles. Il est nécessaire de disposer des commandes d'administration sur tout ordinateur où l'on souhaite l'utiliser. Ceci complique un déploiement

à grande échelle, car il est plus simple de répliquer un fichier sur un ensemble de noeuds, que d'exécuter un ensemble de commandes.

4.2.4 SE Linux

SE LINUX dispose d'une architecture claire et flexible, héritée de GFAC comme pour RSBAC. Mais plutôt que de fournir différents modules, SE LINUX dispose d'un langage de configuration à la fois simple et puissant. Simple car le nombre de mots-clés est réduit, mais puissant car il implante les modèles RBAC et DTE. Ainsi il est possible de déployer différents types de politique de sécurité, même si SE LINUX est utilisé avant tout pour faire du confinement d'application.

Il faut noter que la configuration de SEL INUX repose sur l'attribution de labels, ou contextes de sécurité. L'ensemble des ressources ayant reu le même contexte de sécurité forme une classe d'équivalence. Par rapport à des implantations comme LIDS ou grsecurity, cela signifie que la définition d'une loi d'accès se fait en deux temps : 1) attribution des labels au sujet et à l'objet correspondants, puis 2) écriture d'une règle d'accès entre les labels. Ceci implique que l'utilisation de SEL INUX est moins intuitive, par rapport à la spécification directe de règles d'accès entre fichiers du système d'exploitation.

Enfin, cette simplicité du langage implique un effort plus grand pour l'administrateur. En effet, l'écriture d'une configuration de SEL INUX correspondant à une politique de sécurité correcte, implique une très bonne connaissance de l'ensemble des fichiers et programmes présents sur le système d'exploitation. De fait, pour réaliser le confinement d'une application, il faut savoir précisément quels accès celle-ci va demander sur les ressources du système, et parmi ces accès, savoir dire lesquels sont légitimes, interdits ou superflus.

4.2.5 grsecurity

grsecurity offre des mécanismes de protection variés, certains n'ayant rien à voir avec le contrôle d'accès. Cependant la partie responsable du confinement est aussi intéressante que SEL INUX, car elle implante une combinaison de RBAC et DTE. Les règles sont définies par des fichiers de configuration, dont la syntaxe est documentée. Celle-ci utilise directement les chemins des fichiers pour définir les sujets et objets, simplifiant ainsi l'écriture d'une configuration, comparativement à l'étape d'attribution des contextes de sécurité dans SEL INUX.

Par contre, il est difficile de comparer l'expressivité des langages de SEL INUX et grsecurity. Le premier distingue davantage de types d'accès, quasiment chaque appel système peut être accordé ou refusé, alors que le second distingue seulement une dizaine de droits d'accès, dont certains n'ont pas d'équivalent dans SEL INUX. Ensuite, les contextes de sécurité de SEL INUX forment des classes d'équivalence sur des objets qui peuvent être très variés, alors que, concernant les fichiers, grsecurity fonctionne uniquement par héritage des permissions des dossiers parents vers les sous-dossiers. Enfin la désignation des sujets dans les règles d'accès se fait à l'aide de contextes de sécurité dans SEL INUX, et par des chemins de fichiers dans grsecurity. Du coup, SEL INUX peut définir des règles pour plusieurs programmes d'un coup, tandis que dans grsecurity il faut écrire une configuration pour chaque programme.

Chapitre 5

Conclusion : Bilan sur les mécanismes de contrôle d'accès

Après avoir examiné les différents modèles de contrôle d'accès existants, il est clair qu'il est nécessaire de disposer d'une combinaison DAC / MAC / RBAC pour obtenir à la fois un contrôle d'accès robuste, et un langage de configuration clair. D'abord, DAC doit être conservé car certaines parties du système ne nécessitent pas de contrôle d'accès obligatoire (les fichiers des utilisateurs par exemple). Ensuite, MAC est nécessaire pour assurer que certains droits d'accès ne seront jamais accordés, notamment sur les parties cruciales pour le fonctionnement du système d'exploitation comme les fichiers d'authentification. Plus particulièrement, c'est le modèle DTE qui sera utilisée pour cette partie. Enfin, RBAC fournit un intermédiaire entre utilisateurs et droits d'accès, via les rôles, et il serait difficile de s'en priver.

La coopération des différents modèles est illustrée par la figure 2. Les utilisateurs sont associés à un ensemble de rôles. Par exemple, l'utilisateur root sera associé au rôle **admin_r**. Ensuite, les rôles ont le droit de manipuler un ensemble de types, ainsi le lien est fait entre les rôles et les permissions d'accès. Par exemple le rôle **admin_r** aura l'autorisation de manipuler le type **admin_t**, qui est associé aux applications d'administration. Enfin, au sein des classes d'équivalence que définissent les types, le DAC est utilisé pour toutes les parties du système ou la protection du MAC n'est par requise. Par exemple, c'est le DAC qui s'applique lorsque l'utilisateur root accède aux fichiers qui se trouvent dans le dossier `/root`.

Les implantations système disponibles sont variées. Suivant les objectifs définis en 1, nous souhaitons réutiliser, au sein de notre architecture de contrôle d'accès, les mécanismes existants.

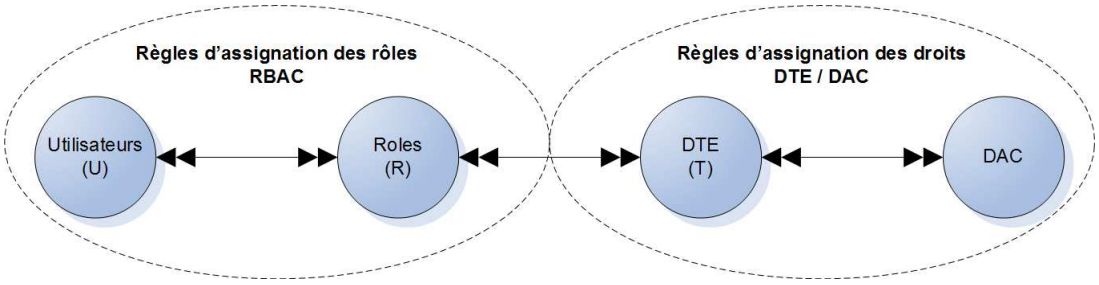

Fig 2 Coopération des modèles RBAC, MAC et DAC.

Bibliographie

[1] L. Mé, Y. Desswarte, Sécurit des systèmes d'information, Lavoisier, 2006.

[2] D. E. Bell et L. J. La Padula, Secure computer systems : Mathematical foundations and model, 1973.

[3] Orbac.org, http ://www.orbac.org/.

[4] Medusa DS9, http ://medusa.fornax.sk/, Medusa ds9 security system.

[5] LIDS Project, LIDS Secure Linux System. http ://www.lids.org/, LIDS Secure Linux System, 2006.

[6] Linux Security Module, Linux Magazine HS 17.

[7] C. Wright, C. Cowan, S. Smalley, J. Morris et G. Kroah-Hartman, Linux security modules : General security support for the linux kernel, 2002.

[8] B. Spengler, Detection, prevention and containment : A study of grsecurity, http ://www.grsecurity.net/papers.php, 2002.

[9] B. Splenger, Increasing performance and granularity in role-based access control system, http ://www.grsecurity.net/researchpaper.pdf, 2005.

[10] A. Ott, RSBAC benchmarks, http ://rsbac.org/documentation/benchmarks, 2006.

[11] J. Morris, Recent developments in selinux kernel performance, 2004.

[12] M. Blanc, Trusted linux systems and application to cluster architecture, 2004.