Protection des systèmes d'exploitation

Christian Toinard

Introduction [1]

Un système d'exploitation est fait pour ''cloisonner'' les processus s'exécutant en espace utilisateur =

- un espace d'adressage spécifique contenant le code et les données (on parle de binaire)

- une ''activité'' disposant des droits nécessaires pour accéder à cet espace et maintenant un contexte d'avancement (plus ou moins spécifique à chaque OS)
  - registres processeur (compteur programme c'est à dire la position dans le code, pointeur de pile, etc…)
  - information sur l'état notamment les descripteurs d'E/S ouverts et les segments mémoire partagés
  - etc…

Introduction [2]

Un système d'exploitation permet de contrôler les privilèges des processus utilisateurs vis à vis des ressources

- en associant les processus à un contexte de sécurité (par exemple l'utilisateur qui a lancé le processus). On peut parler de contextes sujets.
- en associant aux ressources (essentiellement les fichiers) un contexte de sécurité (par exemple celui du processus, c'est à dire de l'utilisateur pour lequel il s'exécute, qui a créé le sujet). On peut parler de contextes objet (la distinction entre sujet et objet n'est pas fondamentale).
- en permettant de définir les privilèges entre les processus et les ressources (par exemple le droit pour un sujet de lire un ensemble de contextes de sécurité objet).

Objectifs [1]

- La protection ne traite pas de la façon dont les sujets ou objets sont authentifiés ni chiffrés. C'est un problème qui est considéré comme résolu (même s'il est en soi difficile).

- La protection traite uniquement de la façon dont on définit et garantit les privilèges entre contextes de sécurité qui sont considérés comme correctement authentifiés.

- La protection est donc complémentaire de la cryptographie. Les deux sont indispensables pour garantir la sécurité.

Exemples :

- un fichier correctement chiffré mais qui est déchiffré et écrit dans une ressource en libre accès.
- un accès légitime à un fichier qui est ensuite transmis légitimement par réseau mais en clair

Objectifs [2]

La protection vise à garantir :

- la confidentialité, non par du chiffrement ou des techniques cryptographiques, mais en limitant l'accès en ''lecture'' à l'information (les accès en lecture sont fermés pour ceux qui n'ont pas besoin de l'information).
- l'intégrité, non par des méthodes de signature ou d'autres techniques cryptographiques, mais en limitant l'accès en ''écriture'' à l'information (les écritures sont interdites pour ceux qui n'ont pas à modifier de l'information).

Confidentialité et intégrité contribuent à la disponibilité. Exemples :

- la confidentialité évite que l'on découvre des mots de passe pour casser le système.
- l'intégrité évite que l'on modifie les données du ''système'' (exemple les binaires) pour le rendre indisponible.

Objectifs [3]

La protection vise à garantir :

- le confinement des vulnérabilités, c'est en fait par le biais de la confidentialité et de l'intégrité que l'on arrive à cela.
  Exemples :
  - un exploit ne pourra pas accéder à l'information des mots de passe (/etc/shadow) et interdira donc des usurpations (login) ou escalades de privilèges (sudo).
  - un exploit téléchargé ne pourra pas être installé sur le système de fichier (pas les droits nécessaires pour compromettre l'intégrité)

  - un exploit ne pourra pas lire la pile et donc ne pourra pas récupérer de l'information issue des appels de fonctions (valeurs de certains paramètres contenant par exemple un compte bancaire)
  - un exploit ne pourra pas exécuter la pile et donc ne pourra pas faire fonctionner un débordement de tampon (buffer overflow) qui a mis du code dans la pile.

Objectifs [4]

La protection vise à contrôler les flux entre les contextes :

- c'est par ce moyen que l'on peut garantir la confidentialité et l'intégrité
- il est possible de contrôler les flux d'information :
  - entre les processus et les ressources
  - entre des processus communiquant par une ressource partagée
  - entre une activité du processus et certaines zones de son espace d'adressage (exemple : la pile)
- il est difficile de :
  - contrôler les flux entre les mêmes zones d'un espace d'adressage (par exemple : entre des variables allouées sur le tas)
  - contrôler les flux à l'intérieur du noyau (par exemple : entre deux variables du noyau)
  - contrôler les flux entre deux segments de mémoire d'un espace d'adressage (exemple : les flux entre deux segments mémoire de la zone de code)

Concepts [1]

La minimisation des privilèges est le concept clef de la protection car c'est en pratique ce mécanisme qui permet de garantir la confidentialité et l'intégrité.

Chaque processus doit donc avoir uniquement l'ensemble des privilèges nécessaires, ni plus ni moins.

- s'il a plus de privilèges il pourra compromettre la sécurité
- s'il en a moins il ne fonctionnera pas

Le dilemme est dans ce qui est ''nécessaire'' :

- ce qui est nécessaire au bon fonctionnement peut permettre de compromettre la sécurité
- il faut alors privilégier soit le bon fonctionnement (la disponibilité) soit la ''sécurité'' (la confidentialité et l'intégrité)

Un système parfaitement sécurisé n'est pas disponible (il n'est pas en route) !
Il y a une contradiction inhérente entre d'une part la disponibilité et d'autre part la confidentialité et l'intégrité

Concepts [2]

Le contrôle d'accès aux ressources permet de minimiser les privilèges, c'est par ce mécanisme de contrôle des flux que l'on minimise les privilèges.

En pratique, il faut définir une politique d'accès :

- soit faite par les utilisateurs finaux soit par une autorité tierce

- au moyen d'une **formalisation des privilèges** autorisés (positifs) et interdits (négatifs)

Il faut aussi un **moyen de garantir cette politique d'accès** c'est à dire les privilèges nécessaires (qu'ils soient positifs ou négatifs).

Concepts [3]

Politique d'accès direct c'est dire définissant les **flux directs** (S-P->O) entre les processus et les ressources :

- c'est la **formalisation la plus simple**
- la **garantie est facile à développer**

Les solutions courantes (Unix, SELinux, GRsecurity, etc…) reposent là dessus.

Exemples :

La formalisation des privilèges sous Unix :

```
Chris$ ls -l
total 6610960
drwxr-xr-x  10 Chris  staff        340 10 nov 17:41 Virtual Machines.localized
-rw-------   1 Chris  staff  1125539840 30 nov 18:38 gentoo2.ova
-rw-------   1 Chris  staff  1129331200 30 nov 18:57 gentoo2Final.ova
-rw-------   1 Chris  staff  1129331712  1 déc 10:23 gentoo3.ova
Chris$
```

La formalisation des privilèges sous GrSecurity :

```
role root uG
role_transitions admin
role_allow_ip   0.0.0.0/32
subject /  {
        /
        /dev/initctl
        /sbin/gradm
        /var/spool/mail
        -CAP_ALL
        bind    disabled
        connect disabled
}
```

Concepts [4]

Politique d'accès indirect c'est dire définissant les **flux indirects** (exemple : S1-W->O1, S2-R->O1 : S1>>S2) entre les processus et les ressources :

- la **formalisation est possible** mais peut être compliquée
- la **garantie est présente dans différentes solutions**

Les solutions classiques (Windows MIC ! surprenant non, permet le cloisonnement des niveaux à la ''mode'' BIBA) et pas mal de solutions ''recherche'' (Asbestos, Histar) avec souvent une intervention des programmeurs d'application pour spécifier les politiques (difficile en pratique).

Concepts [5]

Politique de propriétés de sécurité :

- prise en compte explicite ou implicite des flux directs et indirects et expression de **propriétés avancées**.
- la **garantie est parfois problématique** soit parce que présentant des effets dissuasifs ou peu connue donc peu utilisée à part des experts.

Exemples :

- Modèles Bell et Lapadula (machines Unix BLP) ou BIBA
- Propriétés PIGA

```
define confidentiality( $sc1 IN SCS, $sc2 IN SCO
[ ST { $sc2 > $sc1 }, { not(exist()) };
  ST { $sc2 >> $sc1 }, { not(exist()) }; ];
```

```
dutiesseparationinterpreter( sc1 IN SC ) [
 Foreach sc2 IN SCO, Foreach sc3 IN SC,
        ¬ ( ( sc1 >write sc2) -then-> (sc1 >execute sc3 ) -then-> (sc3 <read sc2))
```

Concepts [6]

Protection discrétionnaire (Discretionary Access Control) ce sont les utilisateurs finaux qui définissent les politiques d'accès.

Exemples DAC : Unix, MS-Windows XP

Protection obligatoire (Mandatory Access Control) c'est l'administrateur de la sécurité (différent de l'administrateur de la machine) qui définit les politiques d'accès.

Dans ce cas, même l'administrateur de la machine (root) a des privilèges restreints.

Exemples MAC : SELinux, GRSecurity, MS-Windows Vista, MS-Windows 7, PIGA-OS, etc …

Résultat fondamental

Impossibilité

Il est prouvé qu'un système discrétionnaire ne peut pas garantir de propriétés de sécurité.

[Harrison et al. 1976] Harrison, M. A., Ruzzo, W. L. et Ullman, J. D. (1976). Protection in operating systems. Communications of the ACM, 19(8):461–471.

L'idée en pratique est simple dès que ce sont les utilisateurs finaux qui définissent les droits :

- les sources d'erreurs sont nombreuses
- les vulnérabilités ne sont pas confinées correctement (par exemple : une escalade de privilèges d'un processus qui parvient à obtenir les droits root obtient tous les droits sur le système).

Difficulté des approches obligatoires [1]

Elles proviennent de l'expérience pratique et sont donc discutables. Cependant, on peut tirer les enseignements suivants du défi sécurité de l'ANR piloté par l'ANSSI.

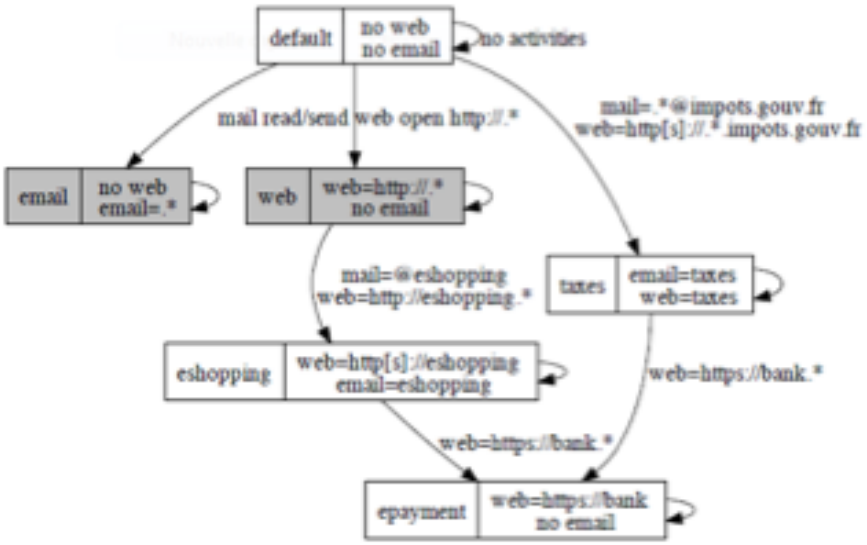
Il est nécessaire d'avoir une protection dite en profondeur, c'est à dire protégeant tous les niveaux du système, puisque la sécurité est celle du maillon le plus faible.

Donc rien ne sert de faire des politiques obligatoires fines si par exemple les interfaces graphiques ne sont pas protégées.

Difficulté des approches obligatoires [2]

Contrôler différents domaines qui partagent le même système

Contrôler les flux entre différents domaines (web, mail, e-commerce) pour que chaque domaine applique des protections à tous les niveaux. En pratique, super-niveau d'administration et de coordination des politiques de plus bas niveaux (interface utilisateur, processus, programme, noyau, réseau, etc…)
Exemple : PIGA-SYSTRANS



Difficulté des approches obligatoires [3]

Contrôle des interfaces graphiques

Le but est de contrôler les flux entre les composants graphiques (par exemple empêcher certaines widgets d'accéder au composant partagé contenant les copiés).

Exemple : XSELinux
neverallow domain clipboard_xselection_t : x_selection
{ write read };

Difficulté des approches obligatoires [4]

Contrôle des processus exécutant les applications

Le but est de contrôler les flux issus des processus utilisateurs.

Exemple : SELinux + PIGA

dutiesseparation(
$sc1:="user_u:user_r:user.*_t");

Difficulté des approches obligatoires [5]

Contrôle des flux à l'intérieur des programmes

Le but est de contrôler les flux entre les données (variables statiques et dynamiques) et entre les composants du programme (fonction, classe, composant, etc…)

- analyse statique des programmes
- suivi dynamique des exécutions (par exemple à l'intérieur d'une machine virtuelle Java)

Très souvent le programmeur définit des contextes de sécurité associés aux données ou aux composants.

Complexe en pratique : on peut s'en dispenser si le reste est bien fait (en effet, une vulnérabilité du programme ne conduira pas à une compromission du système).

Difficulté des approches obligatoires [6]

Contrôle des flux du noyau

Le but est de contrôler les flux au sein du noyau (variables statiques et dynamiques) et entre les composants du noyau (fonctions, classes, composants, etc…).

Même problème que précédemment sauf qu'il s'applique à tout le code noyau, qui est parfois bourré de ''hacks'' donc difficile à certifier.

En pratique : on peut faire l'hypothèse d'un noyau sûr.
Les attaques sur le noyau sont plus complexes car le code s'exécute dans un mode privilégié qui n'est pas accessible facilement.

Difficulté des approches obligatoires [7]

Contrôle des matériels

Le but est de contrôler les flux à l'intérieur des composants matériels.

Ce problème est clairement important mais il est du ressort des fabricants de matériel.

Il revient à intégrer des mécanismes de **protection obligatoire au niveau des matériels**, c'est donc un problème que l'on peut traiter de façon identique en adaptant les méthodes de protection pour qu'elles ne soient pas pénalisantes.

Difficulté des approches obligatoires [8]

Contrôle des flux réseau

Le but est de contrôler les flux entre les processus passant par le réseau. Cela revient à transmettre les contextes de sécurité lors des échanges réseau.

Exemple : Iptables SELinux
On spécifie pour chaque application, les contextes des paquets autorisés en envoie/réception.

allow user_clawsmail_t {dns_client_packet_t imaps_client_packet_t smtps_client_packet_t}:packet { send recv };

Grâce à iptables les contextes sont envoyés en fonction du port de destination.
iptables –t mangle –A OUTPUT –p tcp ––dport 80 –m state ––state NEW –j SEL_WEBC 2 iptables –t mangle –A SEL_WEBC –j SECMARK ––selctx system_u:object_r:http_client_packet_t

Approches existantes [1]

Protection Unix

DAC

Associer à chaque processus un utilisateur et un groupe ainsi qu'à chaque objet.
On définie les droits en lecture, écriture, exécution pour le propriétaire, le groupe et les autres.
Ce modèle est inchangé bien que l'on trouve aussi des notions de liste de contrôle d'accès (ACL).

MAC

Pour Unix, il n'y a pas de standard mais des classiques SELinux, GRSecurity.

Mais le DAC est en général prioritaire.

Approches existantes [2]

Protection Windows

Il est difficile de décrire de façon unique la protection Windows car elle évolue au fil des versions.

On distingue cependant :

- Windows NT
  DAC : les objets sont associés à une liste de contrôle d'accès (ACL) pour des utilisateurs et groupes
- Windows Vista et 7
  MAC + DAC: un contrôle MAC prioritaire associé à un niveau d'intégrité comme dans BIBA

Approches existantes [3]

Protection obligatoire

Le moniteur de référence est le composant système qui applique les politiques et garantit que les appels systèmes sont conformes à cette politique.
Il est donc important de savoir ce qu'est un appel système (une trappe pour demander des services au noyau).
Exemple : man 2 open

En général, le moniteur détourne les appels à son profit pour vérifier le respect de la politique.

Approches existantes [4]

Protection obligatoire

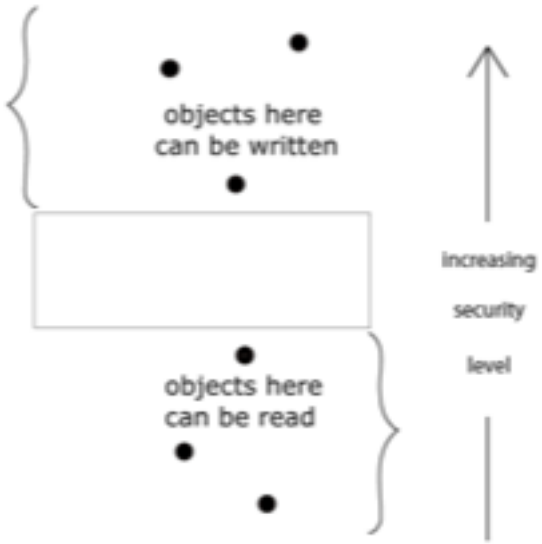
Il existe aussi des moniteurs de référence applicatifs notamment, dans tous les environnements de développement (.NET 3.5, .NET 4.0, Java, …).
Il faut à ce stade bien connaître les différences entre ces deux approches (Java = machine virtuelle offrant un bac à sable qui interdit les accès à l'hôte, .NET= pas de machine virtuelle donc les applications accèdent directement aux appels systèmes de l'hôte).

Les risques sont accrus pour .NET donc c'est là qu'il faut utiliser de préférence la protection obligatoire disponible avec le framework. Malheureusement, en pratique difficile à utiliser avec un simplification discutable à partir de la version 4.0.

Approches existantes [4]

Protection obligatoire

Bell et Lapadula ont proposé en 73 un modèle de confidentialité (no read-up, no write-down).



Cette approche est partiellement implantée dans certains Unix. Malheureusement, elle présente l'inconvénient de faire monter le niveau l'information et de la rendre inaccessible.

Approches existantes [5]

Protection obligatoire

Le modèle BIBA, défini en 75, est le dual du précédant, il permet de garantir l'intégrité (no read-down, no write-up).

L'inconvénient est que pour mettre à jour le système il faut autoriser des escalades de privilèges ou être administrateur ce qui est dangereux.

Ce modèle retrouve de la vigueur avec son utilisation par Microsoft pour empêcher les modifications malicieuses du système (ayant le niveau le plus élevé).

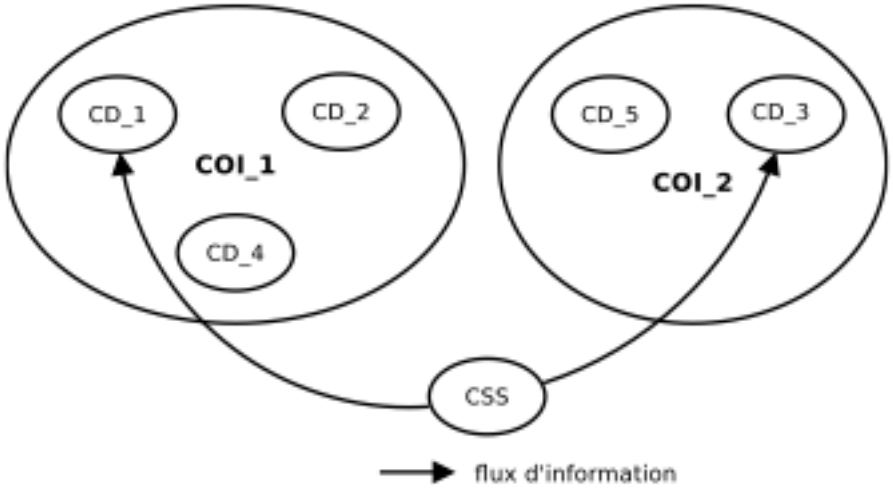
Il n'est pas possible d'étendre les modèles BLP et BIBA pour avoir d'autres propriétés plus adaptées d'intégrité ou de confidentialité.

Exemple : pour l'intégrité, autoriser dans certains cas des écritures d'un contexte vers des domaines plus forts tout en empêchant d'autres contextes.

Approches existantes [6]

Protection obligatoire

Le modèle de la Muraille de Chine permet d'éviter les actions malveillantes du type délit d'initiés (Inside Dealing).
Elle regroupe les données CD des concurrents dans une classe de conflit d'intérêts COI.



Ce modèle est difficile à appliquer car chacun a des conflits d'intérêts. Il faut dans certains cas autoriser des flux vers un CD conflictuel mais le modèle ne supporte pas des propriétés de ce type il interdit tout.

Approches existantes [6]

Contrôler des domaines et des types

Le modèle DTE correspond à associer des contextes à des processus (domaine du processus = notion différente de celle vue dans PIGA-SYSTRANS) et des contextes à des objets (type d'un objet).

Ce modèle DTE est celui qui est largement adopté (SELinux, GRSec, PIGA, …). Il permet de contrôler les flux directs, les flux indirects et des propriétés avancées définies sur mesure en fonction des besoins.

Classification du MAC [1]

Approches par automate

On modélise un système sous la forme d'un automate d'états.
L'analyse des propriétés de sécurité revient à rechercher des états satisfaisants des contraintes ou des critères. Le problème est le passage à l'échelle sur un système d'exploitation. Même avec des simplications, les automates restent très complexes et le contrôle consomme trop de temps. Il est donc difficile d'implanter une méthode de protection qui se base sur ce principe. Plutôt dédié à l'analyse hors ligne.

Classification du MAC [2]

Séparation par niveau

Approche plutôt d'intégrité. Un niveau de privilège élevé permet de modifier le système. Le noyau Windows, à partir de Vista, utilise des niveaux de privilèges qui peuvent servir, soit pour le contrôle d'intégrité, soit pour la confidentialité (?). Cependant, les failles trouvées sur ce type d'OS montre à l'évidence qu'il n'y a pas de contrôle avancé des flux d'information.

Base d'exécution

Permet de s'assurer que tous les binaires exécutés sur un système proviennent de répertoires reconnus comme sûr. Par exemple : autoriser l'exécution des binaires provenant uniquement des répertoires systèmes du type /usr/bin ou /bin.
Dans .NET, on a une base sûre Global Assembly Cache.

Classification du MAC [3]

Abus de privilèges

Spécification du comportement normal d'un programme : ensemble des appels système autorisés et les arguments typiques de ces appels système.

Appliquer ces méthodes à la protection de l'ensemble d'un système est très difficile et reviendrait en pratique à des dénis de service répétés en raison d'un apprentissage non optimal.

Séparation de privilèges

[Saltzer et Schroeder, 1975] définissent qu'une tâche critique doit être réalisée par k utilisateurs. Deux processus pour une tâche composée de deux opérations.

Classification du MAC [4]

Contrôle de concurrence

[McPhee, 1974] : ”il existe un ordonnancement imprévisible entre les accès de deux processus à une ressource partagée”. La détection est un problème NP-Complet [Netzer and Miller, 1990]. A part les systèmes transactionnels, aucune solution ne traite complètement les problèmes de concurrence. Les approches transactionnelles sont inadaptées pour être intégrées dans les systèmes d’exploitation.

Non interférence

[Goguen and Meseguer, 1982] : la non-interférence de X envers Y est respectée si X ne peut modifier Y. Cas spécifique de l’intégrité des sujets.

Contrôle des flux [1]

Coloration

Une étiquette, ou couleur, est affectée à une variable, c'est-à-dire un objet. Quand cet objet est lu par un sujet, ce dernier est coloré par l'étiquette de l'objet. Par transitivité, le système est coloré suivant la propagation de l'information.

Les seules approches qui permettent le passage à l'échelle d'un système sont des colorations au niveau matériel des registres et demandent une émulation et une instrumentation très lourdes. Elles provoquent un surcoût de 20 fois la normale. De plus, dans la plupart des cas, seuls les flux d'information directs (explicites) sont pris en compte et l'instrumentation du code n'est pas triviale.

Contrôle des flux [2]

Analyse de politique

Vérification de flux indirects [Guttman et al., 2005] ou d'un large ensemble de propriétés [Briffaut, 2007].

L'approche PIGA permet d'énumérer de façon exhaustive les activités illégales présentes dans une politique contrôlant les flux directs.

Contrôle des flux [3]

Formalisation des besoins de sécurité

Différentes approches orientées développeurs (exemple .NET 4.0) ou orientées administrateurs (SELinux, PIGA, …).

La difficulté est d'avoir un large ensemble de propriétés permettant une protection à l'échelle de tout un système (exemple : PIGA-MAC, PIGA-OS, PIGA-Virt, PIGA-Windows, …). En pratique, PIGA calcule soit statiquement soit dynamiquement les activités illégales. L'approche statique passe bien à l'échelle, elle peut être intégrée directement dans un noyau d'OS avec un surcoût acceptable. Dans tous les cas, l'intérêt est de quantifier le coût d'une propriété (exemple : 1 propriété conduit à un million de type d'activités différentes qu'il faut surveiller).

Conclusion

Très peu de solutions permettant un large spectre de protections.

Quasiment pas de solution offrant une protection en profondeur.

Nécessité de pouvoir adapter les protections aux besoins.

La virtualisation ne garantit rien en terme de sécurité. Plus les systèmes sont virtualisés plus il est difficile de les protéger (nombre de niveaux de protection). Mais des solutions MAC/virtualisation commencent à apparaître.

Le Cloud ne fera qu'adopter les solutions de protections en profondeur à venir.

Références

Jérémy BRIFFAUT ''Formalisation et garantie de propriétés de sécurité système : application à la détection d'intrusions'' THESE DE L'UNIVERSITE D'ORLEANS. 2007.

Jonathan ROUZAUD-CORNABAS. Formalisation de propriétés de sécurité pour la protection des systèmes d'exploitation. THESE DE L'UNIVERSITE D'ORLEANS. 2010.

Jérémy Briffaut, Martin Peres, Jonathan Rouzaud-Cornabas, Jigar Solanki, Christian Toinard, Benjamin Venelle ''PIGA-OS : retour sur le système d'exploitation vainqueur du défi sécurité'' Actes du Colloque Francophone sur les Systèmes d'Exploitation. Saint-Malo, France, du 10 au 13 mai 2011.

Jérémy Briffaut, Emilie Lefebvre, Jonathan Rouzaud-Cornabas, Christian Toinard ''PIGA-Virt: an Advanced Distributed MAC Protection of Virtual Systems.'' 6th Workshop on Virtualization and High-Performance Cloud Computing Euro-Par 2011, Bordeaux, France.