Protection et amélioration de la sécurité des systèmes d'exploitation

Jérémy Briffaut ENSI Bourges LIFO
Martin Perès Université de Bordeaux LaBRI
Jonathan Rouzaud-Cornabas INRIA Rhône Alpes
Jigar Solanki Université de Bordeaux LaBRI
Christian Toinard ENSI Bourges LIFO
Benjamin Venelle Alcatel Lucent Bells Labs

Objectifs (1)

Présenter l'approche développée durant le défi sécurité de l'Agence Nationale de la Recherche (ANR)

- Concurrents
  - EADS et Supélec Rennes
  - Université d'Orsay et Paris 6
- Projet SPACLik/PIGA OS : Vainqueur
- Présidé par l'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI) *https://adullact.net/projects/secsi/*
  http://www.agence-nationale-recherche.fr/magazine/actualites/detail/resultats-du-defi-sec-si-systeme-d-exploitation-cloisonne-securise-pour-l-internaute/
- Salué par la communauté française des systèmes d'exploitation
  Jérémy Briffaut, Martin Perès, Jonathan Rouzaud-Cornabas, Jigar Solanki, Christian Toinard et Benjamin Venelle, PIGA-OS : Retour sur le Système d'Exploitation Vainqueur du Défi Sécurité, 8ième Conférence Francophone sur les Systèmes d'Exploitation, Saint-Malo, France, 2011

Objectifs (2)

Introduire les retombées

- Protection des systèmes de calcul intensif
  D. Gros, M. Blanc, J. Briffaut, C. Toinard
  Advanced MAC in HPC systems : performance improvement.
  12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2012

- Protection des systèmes virtualisés
  J. Briffaut, E. Lefebvre, J. Rouzaud-Cornabas, C. Toinard
  PIGA-Virt: An Advanced Distributed MAC Protection of Virtual Systems,
  Euro-Par 2011: Parallel Processing Workshops, Springer, Lecture Notes in Computer Science, 7156, 416—425, 2012

- Protection des Clouds :
  Projet européen Seed4C
  Z. Afoulki, A. Bousquet, J. Briffaut, J. Rouzaud, C. Toinard
  MAC Protection of the OpenNebula Cloud Environment
  The 2012 International Conference on High Performance Computing & Simulation, HPCS, Madrid, Spain, July, 2012
  http://projects.celtic-initiative.org/seed4c/

- Protection des systèmes embarqués (téléphones, capteurs, cartes à puce, …)
  Brevet + projet européen

- …

Introduction [1]

Un système d'exploitation permet de **minimiser 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 sujet.
- en associant aux ressources (essentiellement les fichiers) un contexte de sécurité (par exemple celui du processus 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).

Introduction [2]

- 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.

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

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

Exemples :

- un fichier correctement chiffré mais qu'un processus déchiffre et écrit dans une ressource en libre accès.
- un accès légitime à un fichier que le processus transmet ensuite en clair par réseau

Introduction [3]

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.

Concepts [1]

Le contrôle d'accès permet de **minimiser les privilèges en contrôlant les flux** entre les contextes de sécurité.

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

- soit faite par les utilisateurs finaux (**Discretionary Access Control**) soit par une autorité tierce (**Mandatory Access Control**)
-
- au moyen d'une **formalisation des privilèges** autorisés (positifs) et interdits (négatifs)

Le MAC nécessite un **moniteur de référence pour garantir cette politique d'accès** c'est à dire les privilèges requis (positifs ou négatifs).

Concepts [2]

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 [3]

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 [4]

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** et nécessite de faciliter la définition des propriétés

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))
```

Approche PIGA OS [1]

Elle provient de l'expérience pratique.
On peut tirer les enseignements suivants du défi sécurité de l'ANR piloté par l'ANSSI.

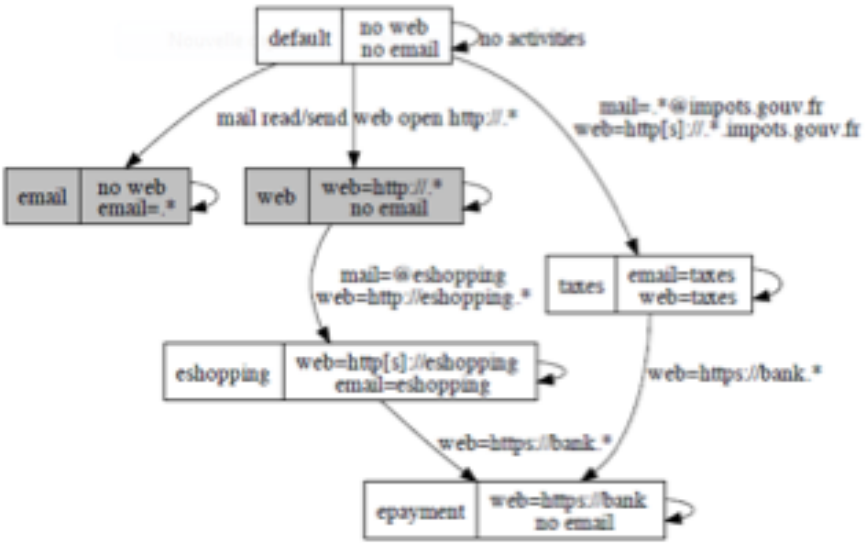
Il est nécessaire d'avoir une protection dite en profondeur, c'est à dire protégeant tous les niveaux du système, puisque la sécurité est celle du maillon le plus faible.

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

Approche PIGA OS [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



Approche PIGA OS [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 };

Approche PIGA OS [4]

Contrôle des processus exécutant les applications

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

Exemple : SELinux + PIGA MAC

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

Approche PIGA OS [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).

Travail en cours avec Martin Perès.

Approche PIGA OS [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.

Approche PIGA OS [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.

Travail en cours avec Jigar Solanki et Martin Perès.

Approche PIGA OS [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.

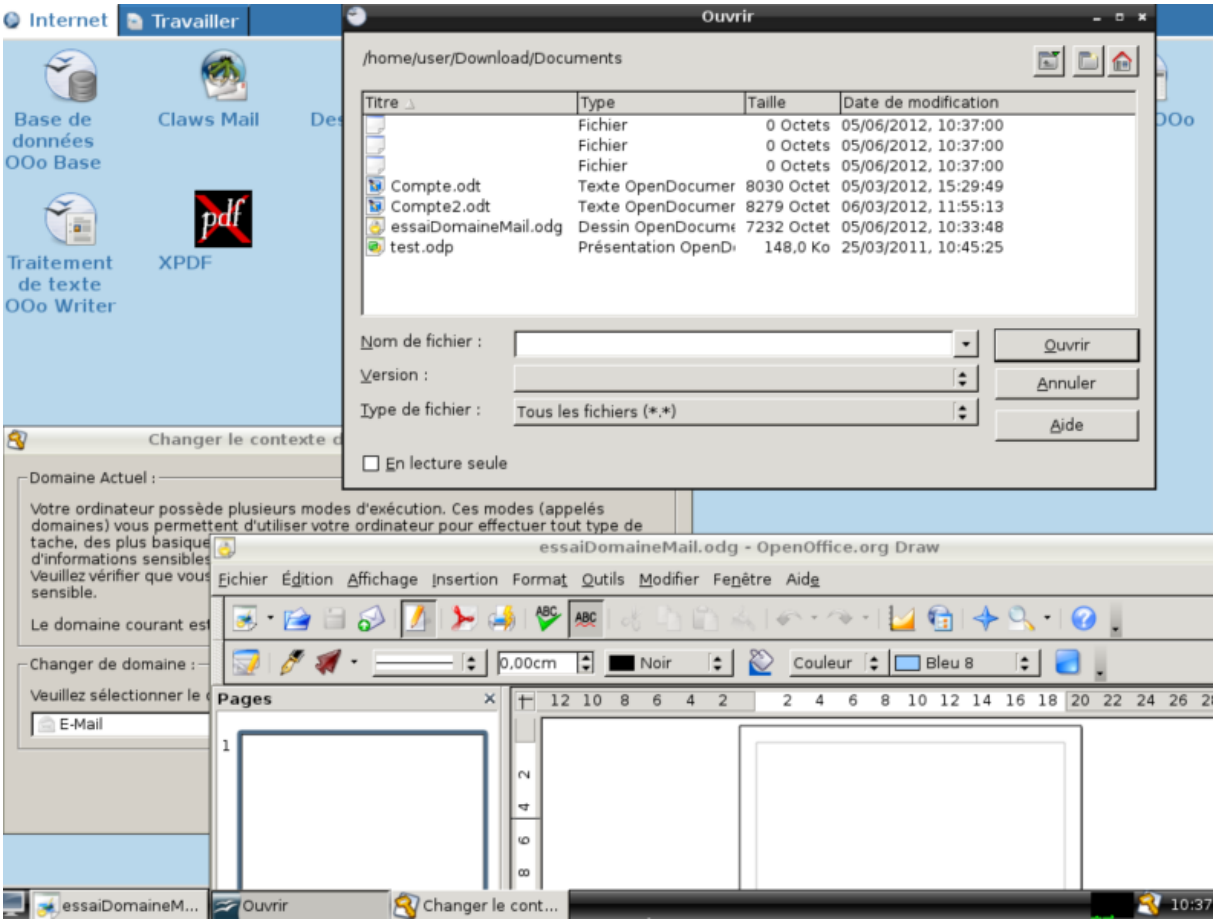
allow user_clawsmail_t {dns_client_packet_t imaps_client_packet_t smtps_client_packet_t}:packet { send recv };

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

Améliorations en cours avec Martin Perès et Benjamin Venelle.

Démonstration ‘’réduite’’ de PIGA OS [1]

Dans le domaine mail, on créée des documents. Les flux avec les données des autres domaines sont contrôlés et peuvent être interdits (la compromission du client mail/firefox ne permet pas d’accéder aux autres domaines… idem pour firefox… **Regardez bien les fichiers visibles**).



Démonstration ‘’réduite’’ de PIGA OS [2]

Dans le domaine ecommerce on accède aux documents. Les flux avec les données de mail sont bien contrôlés (les informations restent dans ce domaine… donc réciproquement mail ne peut transmettre des informations de ecommerce…

D’autres fichiers apparaissent ‘’magiquement’’).