RenPar'20 / SympA'14 / **CFSE 8**
Saint-Malo, France, du 10 au 13 mai 2011

# PIGA-OS: retour sur le système d'exploitation vainqueur du défi sécurité

Jérémy Briffaut, Martin Peres, Jonathan Rouzaud-Cornabas, Jigar Solanki*, Christian Toinard, Benjamin Venelle
Laboratoire d'Informatique Fondamentale d'Orléans
ENSI de Bourges – LIFO, 88 bd Lahitolle, 18020 Bourges cedex, France
*Laboratoire Bordelais de Recherche en Informatique
Unité Mixte de Recherche CNRS (UMR 5800) 351, cours de la Libération F-33405 Talence cedex
{jeremy.briffaut,jonathan.rouzaud-cornabas,christian.toinard}@ensi-bourges.fr, jigar.solanki@labri.fr

**Résumé**

Le partage d'un système d'exploitation par différents domaines pose des problèmes de sécurité difficiles à résoudre tout particulièrement pour le poste de travail d'un Internaute. Par exemple, le système d'exploitation doit empêcher les flux d'information entre des données ou processus relevants du domaine `ecommerce` et des données ou processus d'un domaine `mail` à risques. Pour résoudre ces problèmes, il est nécessaire de fournir un système d'exploitation garantissant une sécurité en profondeur en contrôlant dynamiquement les différents domaines requis par l'utilisateur. L'approche proposée dans `PIGA-OS` est une protection obligatoire à tous les niveaux du système avec une méthode dynamique qui adapte les protections en fonction des domaines rencontrés. L'article décrit comment `PIGA-OS` garantit des propriétés de confidentialité et d'intégrité très avancées c'est à dire pouvant nécessiter le contrôle de combinaisons complexes de flux d'information indirects. `PIGA-OS` constitue à notre connaissance la première solution opérationnelle offrant une protection obligatoire en profondeur multi-domaines et elle a remporté les trois phases du défi sécurité de l'ANR.

## 1. Introduction

C'est pour améliorer la sécurité des systèmes d'exploitation que l'Agence Nationale de la Recherche a lancé le programme défi sécurité Sec&Si et a retenu trois équipes compétitrices `OS4` impliquant EADS et Supélec Rennes, `SafeOS` avec le LRI et LIP6, et `SPACLik` développé par le LIFO. Le but du défi sécurité est de proposer un système d'exploitation GNU/Linux garantissant une sécurité en profondeur, c'est à dire une protection à tous les niveaux du système (processus, interface graphique, réseau). La protection d'un système d'exploitation consiste à minimiser les privilèges des processus, c'est à dire à restreindre les droits des processus sur les ressources du système d'exploitation. Une propriété de sécurité revient à définir une minimisation des privilèges pour un ensemble donné de processus et de ressources. Un système d'exploitation doit garantir des objectifs de sécurité (ensemble de propriétés de confidentialité, d'intégrité et de disponibilité) pour contrôler les flux entre différents domaines (un domaine correspond à un usage particulier du système tel que la navigation web, le commerce électronique, le courrier électronique, les impôts et les services banquaires en ligne). Les différentes méthodes possibles consistent en des techniques de minimisation telles que des chroots, c'est à dire des bacs à sable restreignants les flux d'information entrants et sortants, des méthodes de virtualisation par domaine pour restreindre les flux entre ces domaines ou des protections obligatoires, correspondant à un ensemble de propriétés de sécurité, fournies par le noyau du système d'exploitation. Seule une protection obligatoire garantit réellement des propriétés de sécurité. Cependant, les protections obligatoires actuelles sont complexes, elles ne facilitent pas l'expression des objectifs de sécurité, traitent mal les flux d'informations indirectes ou nécessitent le développement d'un nouveau système avec des problèmes de pérénité et de maintenabilité. La réalisation d'une protection obligatoire en profondeur, permettant de garantir un contrôle efficace des flux entre les domaines, reste donc un problème de recherche complètement ouvert.

PIGA-OS propose de nouvelles méthodes de protection obligatoire. Au niveau du noyau du système d'exploitation, PIGA-MAC permet de faciliter l'expression d'objectifs de sécurité et garantit un large ensemble de propriétés d'intégrité et de confidentialité en traitant efficacement les flux indirects. Au niveau des applications et des domaines requis, PIGA-SYSTRANS calcule dynamiquement en fonction des

interactions des utilisateurs l'ensemble des politiques obligatoires nécessaires aux différents niveaux du système. L'avantage est de pouvoir réutiliser des politiques obligatoires contrôlant uniquement les flux directs, tels que SELinux et XSELinux, pour les compléter avec les politiques obligatoires PIGA-MAC et PIGA-FIREWALL pour contrôler les flux indirects et réseau.
L'avantage de l'approche PIGA-OS est de réutiliser et renforcer les protections obligatoires de systèmes patrimoniaux tels que SELinux en offrant un contrôle dynamique des domaines. Elle permet d'offrir un système d'exploitation Linux pérenne et bien maintenu.
L'article présente l'architecture générale en motivant par des scénarios d'attaques et détaille les solutions PIGA-MAC et PIGA-SYSTRANS. Ensuite, nous donnons d'une part les performances en comparant avec les solutions concurrentes du défi et d'autre part les expérimentations correspondant aux vulnérabilités trouvées avec les scores permettant d'établir l'efficacité des protections. Enfin, nous concluons en donnant les perspectives pour la protection des systèmes répartis et des systèmes embarqués.

## 2. Protection obligatoire en profondeur dynamique

### 2.1. Motivations

**Protection obligatoire en profondeur**

Le contrôle obligatoire d'un processus revient à lui associer un contexte de sécurité (identifiant) sujet et des droits (appels système autorisés) sur un ensemble de ressources (fichier, réseau, mémoire, ...) associées chacune à un contexte de sécurité objet particulier. Les processus peuvent changer de sujet lors de certains appels système. Par exemple, les transitions SELinux, lors des exécutions de fichiers, permettent de minimiser les privilèges en changeant le sujet du processus. En principe, un objet ne change pas de contexte. Par exemple, avec SELinux les contextes des fichiers ne peuvent pas être modifiés (sauf en relabelisant le système). Cependant, la protection obligatoire SELinux contrôle uniquement les flux d'information directs entre un sujet et un objet.
Le premier objectif est donc de contrôler les flux indirects qui transitent par des ressources intermédiaires (canaux cachés). Cela permet d'empêcher des attaques non couvertes par SELinux du type écriture puis exécution d'un fichier en séparant les privilèges pour différents processus.
Un second objectif est de faciliter l'expression d'un large ensemble d'objectifs de sécurité, comme la séparation de privilèges ou les modifications indirectes. Ce langage doit supporter des propriétés avancées, associées au contrôle de plusieurs flux d'information directs ou indirects. Un administrateur système doit pouvoir réutiliser des propriétés de sécurité prédéfinies au moyen de notre langage afin de simplifier son travail.
Notre troisième objectif est de fournir une protection obligatoire efficace qui puisse garantir toutes les propriétés de sécurité supportées par notre langage.
Un quatrième objectif est que des protections obligatoires doivent protéger tous les composants du système d'exploitation (processus, ressource système, interface graphique, réseau). Elle doit par exemple empêcher des flux par copier-coller où une l'information copiée du domaine ecommerce est alors accessible au domaine mail.

**Protection obligatoire en profondeur multi-domaines**

Dans un système présentant plusieurs domaines, la protection obligatoire en profondeur est difficile. Elle nécessite d'appliquer dynamiquement différentes protections obligatoires sur les différents niveaux du système en fonction des domaines demandés. Par exemple, lorsque l'utilisateur accède au site web http ://www.ebay.fr correspondant au domaine `ecommerce`, il doit pouvoir sauvegarder des données et consulter les emails dans le domaine ecommerce et provenant d'ebay. Cependant, la lecture des emails à partir du domaine `mail` ne doit être pas permettre l'accès aux données ni aux emails du domaine `ecommerce`. Cela évite par exemple les attaques par ameçonnage (fishing) permettant à partir du lecteur de courrier électronique d'accéder aux informations de `ecommerce`. La difficulté consiste à permettre au même lecteur de courrier de lire des emails sûrs, ici provenant de sites d'ecommerce, et non sûrs.
Un cinquième objectif est d'avoir une protection qui permet de contrôler les flux entre domaines.
Un sixième objectif est que la méthode de protection dynamique retenue doit être aisément configurable afin par exemple d'autoriser certains flux entre domaines tout en interdisant d'autres flux entre ces mêmes domaines.

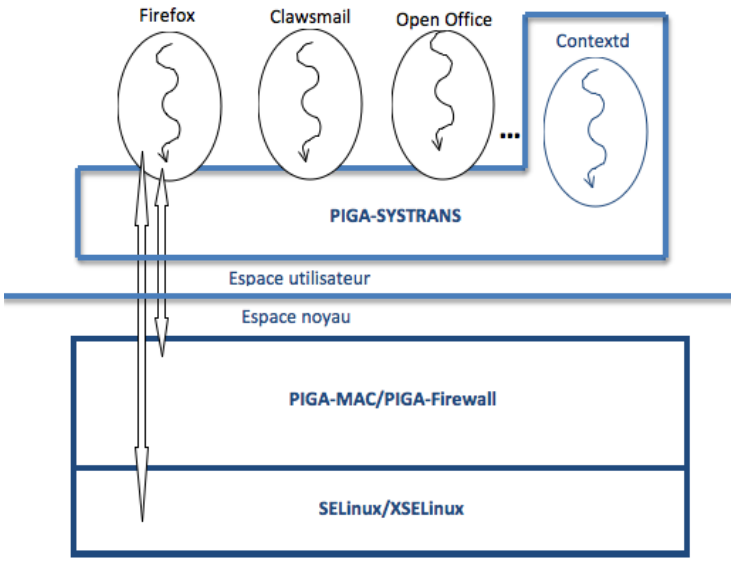


FIGURE 1 – Protection obligatoire en profondeur multi-domaines

### 2.2. Architecture générale de PIGA-OS

L'architecte de PIGA-OS répond aux six objectifs définis précédemment. Le système d'exploitation proposent les fonctionnalités suivantes :
– Un navigateur web (Firefox)
– Un lecteur de mails (Clawsmail)
– Une suite bureautique (OpenOffice)
– Un lecteur PDF (xpdf)
D'autres applications peuvent facilement être ajoutées et contrôlées mais ces applications étaient celles requises par le règlement du défi sécurité.

**Protection obligatoire en profondeur**

PIGA-OS est un système d'exploitation basé sur GNU/Linux dérivé de la distribution Gentoo Hardened à laquelle s'ajoute différentes protections obligatoires :
– SELinux et XSELinux pour contrôler les flux directs des applications et de l'interface graphique XWindow ;
– PIGA-MAC pour contrôler les flux indirects au moyen d'un langage permettant d'exprimer des propriétés avancées disponibles sous forme de canevas réutilisables
– PIGA-FIREWALL pour la protection obligatoire du réseau ;
PIGA-OS comporte également un système de mise à jour automatique et de stockage sécurisé des documents puisque ceux-ci sont labelisés avec un contexte spécifique à l'application, à l'utilisateur et au domaine.

**Protection obligatoire en profondeur multi-domaines**

PIGA-SYSTRANS contrôle les applications et les actions des utilisateurs pour définir et déployer dynamiquement les politiques obligatoires en fonction des domaines demandés. Un mécanisme d'administration facilite la configuration de la dynamique autorisée entre ces domaines.
Comme le montre la figure 1, les processus utilisateurs intéragissent avec PIGA-SYSTRANS pour que celui-ci configure correctement les politiques obligatoires sous-jacentes. Ils interagissent aussi avec les protections obligatoires pour que celles-ci calculent une décision pour l'appel système ou la requête vers le serveur X.
Considérons par exemple une session qui commence avec l'utilisateur demandant à accéder au site web `http://www.ebay.fr`, firefox envoie alors une requête à Contextd qui vérifie s'il est possible de pas-

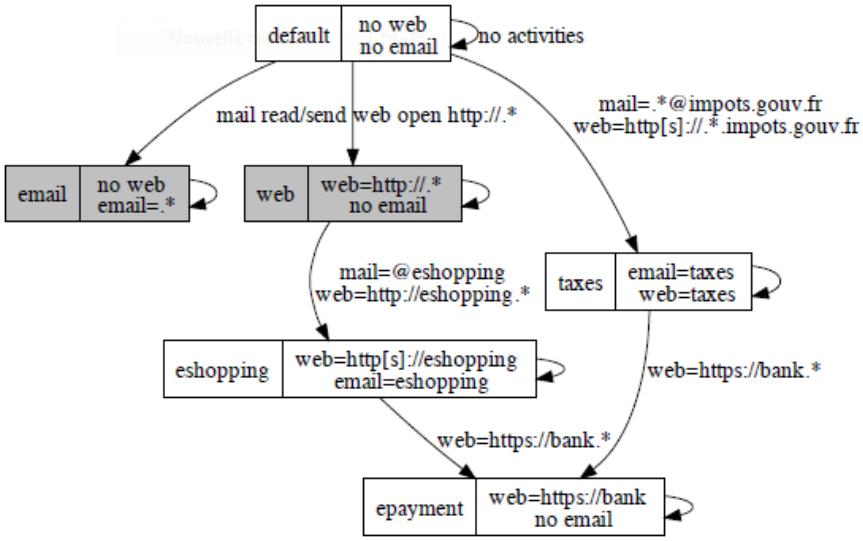


FIGURE 2 – Spécification des domaines

ser dans le domaine `ecommerce` et configure les politiques obligatoires SELinux, XSELinux, PIGA-MAC et PIGA-FIREWALL pour contrôler le domaine `ecommerce` et labeliser correctement toutes les ressources créées dans ce domaine. Ensuite, l'utilisateur demande, via Clawsmail, à passer dans le domaine `mail` pour lire les courriers standards c'est à dire provenant d'expéditeurs non sûrs. Dans ce cas, Contextd demande préalablement à toutes les applications de changer leur état afin par exemple pour Firefox de redémarrer en fermant les onglets issus du domaine `ecommerce`, pour Clawsmail de fermer tous les courriers et pour OpenOffice tous les documents du domaine `ecommerce`. Ensuite, Contextd met en place les nouvelles politiques obligatoires pour le domaine `ecommerce`.

### 2.3. Protection obligatoire avancée

Le composant PIGA-MAC traite les trois premiers objectifs de sécurité définis dans la partie motivation. Tout d'abord, il permet de contrôler les flux indirects passant par les différentes ressources (fichiers, sockets réseau, mémoires, périphériques, ...) du système d'exploitation.

Ensuite, il repose sur un langage qui facilite l'expression des objectifs de sécurité en permettant de manipuler simplement les flux entre les contextes de sécurité. Ainsi, nous proposons des canevas de sécurité prédéfinis qu'un administrateur se contente de paramétrer pour garantir des propriétés de sécurités avancées pouvant contrôler plusieurs flux.

Enfin, nous proposons une méthode de protection obligatoire efficace. Elle repose sur un compilateur du langage qui prend en entrée des politiques SELinux pour calculer de façon exhaustive toutes les activités qui peuvent violer les propriétés requises. Cette méthode est performante parce que grâce à SELinux nous pouvons réduire l'ensemble des activités illégales et pouvons précalculer exhaustivement toutes les activités illicites résiduelles.

Dans cette partie nous montrons d'abord comment grâce à ce langage nous pouvons définir des canevas de protection obligatoire et ensuite comment un administrateur peut réutiliser ces canevas pour protéger un système.

#### 2.3.1. Canevas de protection obligatoire

Bien que notre langage a permis de modéliser toutes les propriétés de confidentialité et d'intégrité disponible dans la littérature concernant la protection et à en définir de nouvelles, nous donnons ici uniquement quelques exemples représentatifs des canevas de protection utilisés dans PIGA-OS.

La propriété suivante a été définie pour garantir l'intégrité d'un ensemble d'objets sc2 vis à vis d'un ensemble de sujets sc1. Cette propriété empêche les flux en écriture qu'ils soient directs $\triangleright$ ou indirects $\triangleright\!\!\!\triangleright$ . Nous verrons un usage dans le cas d'une application exécutée depuis une session utilisateur.

```
1  define integrity( sc1 in SCS, sc2 in SCO ) [
2          ¬(sc1 ▷ sc2)
3          ¬(sc1 ▷▷ sc2)
4   ];
```

La propriété suivante a été introduite spécifiquement pour combler une vulnérabilité trouvée durant la première phase du défi. Cette propriété permet d'éviter qu'un processus crée un fichier, exécute un interpréteur (par exemple bash) qui ensuite essaierait d'exécuter le fichier. Nous en verrons un exemple d'usage pour protéger Firefox des exploits javascript.

```
1  define dutiesseparationbash( sc1 IN SC ) [
2          Foreach sc2 IN SCO, Foreach sc3 IN SC,
3                  ¬ ( ( sc1 -write-▷ sc2) ——then→ (sc1 -execute-▷ sc3 ) ——then→ (sc3 -read-▷ sc2))
4   ];
```

### 2.3.2. Utilisation des canevas dans le cadre de PIGA-OS

En pratique, le travail d'un administrateur système est simplifié. Celui-ci doit définir un certain nombre de règles de protection en fournissant des paramètres aux canevas dont il dispose. Dans le cas de PIGA-OS, les paramètres sont des contextes SELinux. En effet, il est plus simple de réutiliser les contextes SELinux existants que de devoir définir un format spécifique. Cependant, un contexte est un identifiant c'est à dire une chaine de caractères et les canevas proposés sont applicables à d'autres contextes que ceux de SELinux. Dans le cas de SELinux, un contexte est composé d'un triplet utilisateur :rôle :type.

Dans le cadre de PIGA-OS, une dizaine de règles suffit à fournir une protection obligatoire avancée empêchant ainsi des scénarios d'attaques permis par SELinux. Ici nous ne détaillons que quatre règles particulières en montrant les scénarios d'attaques qu'elles préviennent.

La règle suivante empêche des attaques reposant sur des exécutions interprétées des fichiers téléchargés. Cette propriété empêche tous les processus utilisateur, associés à l'expression régulière user_u : user_r : user. $*$ _t, de télécharger (par exemple via firefox) un script pour ensuite lancer un programme, par exemple bash, qui viendrait lire ce script. Cette propriété a notamment été utilisée pour contrer une vulnérabilité où un javascript soumis à firefox permettait d'ouvrir un interpréteur appelé busybox pour lire et donc exécuter les scripts bash que l'exploit javascript a préalablement téléchargé. Cette règle permet non seulement de contrer cette attaque mais aussi toutes celles du même type exécutées depuis les processus utilisateur.

```
1  dutiesseparationbash( $sc1:="user_u:user_r:user.*_t" );
```

Les deux règles suivantes garantissent l'intégrité des exécutables et des fichiers de configuration du système vis à vis des processus utilisateur. Par exemple, cela empêche un exploit javascript dans un programme comme Firefox ayant obtenu des privilèges administrateur de pouvoir modifier des fichiers systèmes (binaires ou fichiers de configuration) comme par exemple les mots de passe /etc/shadow.

```
1  integrity( $sc1:="user_u:user_r:user.*_t", $sc2:=".*:.*:.*_exec_t" );
2  integrity( $sc1:="user_u:user_r:user.*_t", $sc2:=".*:.*:.*etc_t" );
```

La règle suivante autorise le contexte firefox à lire ses fichiers de configuration lors de son lancement, puis empêche ensuite un exploit, par exemple un javascript, d'accéder à la configuration pour compromettre sa confidentialité ou son intégrité. Le premier paramètre est le contexte du processus firefox, le second paramètre correspond aux contextes des objets de configuration de firefox et le troisième paramètre au contexte des extensions chargées par firefox après l'initialisation avec les fichiers de configuration. Cela évite, par exemple, le téléchargement par firefox d'un javascript qui viendrait lire les mots de passe et les clés de chiffrement contenus dans les fichiers de configuration signons.txt et key3.db de firefox pour les transmettre à un attaquant distant via HTTP. Nous avons la même propriété pour clawsmail ou OpenOffice.

```
1  notrereadconfigfile( $sc1:=user_u:user_r:user_mozilla_t, $sco1:=$NOREADFIREFOX, $scoafter:=system_u:object_r:context_lib_t);
```

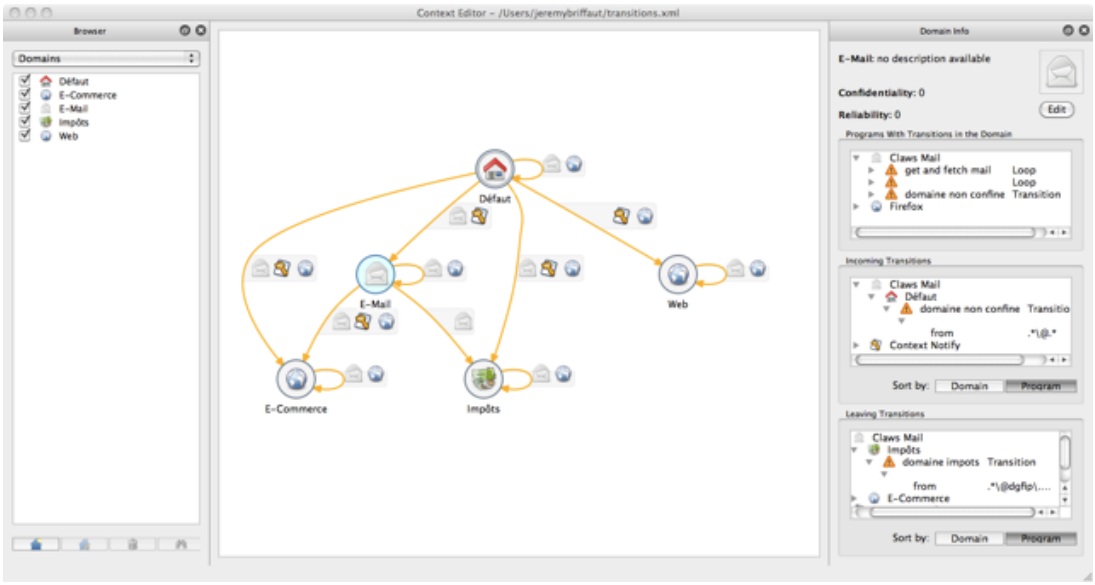


FIGURE 3 – Outil graphique de configuration des domaines

### 2.3.3. Principe de PIGA-MAC

PIGA-MAC prend en entrée les politiques obligatoires existantes, telles que les politiques SELinux ou GRSecurity, ainsi que les instances des propriétés requises telles que définies dans la section précèdante. PIGA-MAC convertit les politiques SELinux en un graphe décrivant les flux d'information autorisés. Les noeuds de ce graphe sont les contextes de sécurité et les arcs les flux permis par les différents appels système. Ce graphe contient par exemple un arc allant de l'objet $user\_u : object\_r : user\_cache\_file\_t$ au sujet $user\_u : user\_r : user\_mozilla\_t$ puisque firefox peut écrire dans son cache. Un arc décrit un flux direct $\triangleright$. PIGA-MAC analyse chaque instance de propriété requise qui peut porter à la fois sur des flux directs $\triangleright$ mais aussi des flux indirects $\triangleright\!\!\triangleright$ comme par exemple pour $integrity(sc1 := "user\_u : user\_r : user.* \_t", sc2 := ".* : .* : .* \_exec\_t");$. PIGA-MAC recherche dans le graphe tous les flux directs concernés ainsi que tous les flux indirects et énumère ainsi toutes les activités du système qui permettent de violer la propriété à garantir. Cette énumération des activités illicites sert ensuite pour le contrôle de chaque appel système. En effet, PIGA-MAC vérifie pour chaque appel système que celui-ci ne correspond pas aux activités énumérées.

L'approche est efficace car d'une part les politiques SELinux permettent de réduire la taille du graphe et d'autre part parce que nous précalculons de façon exhaustive toutes les activités illicites. En pratique, on observe un surcoût de l'ordre de 10% de temps processeur.

PIGA-MAC améliore grandement la protection SELinux puisqu'une seule règle telle que $integrity(\$sc1 := "user\_u : user\_r : user.* \_t", \$sc2 := ".* : .* : .* \_exec\_t")$ peut correspondre à plusieurs milliers d'activités malveillantes qui pourraient violer de façon directe mais surtout de façon indirecte la propriété requise.

## 2.4. Protection obligatoire en profondeur multi-domaines

### 2.4.1. Configuration de la protection multi-domaines

La configuration de PIGA-SYSTRANS correspond à définir un automate comme présenté sur l'image 2. Cet automate présente 6 états (default, email, web, eshopping, epayment, taxes) et définit les transitions possibles entre ces états. Chacun des ces états est associé à un ensemble de politiques obligatoires (selinux, xselinux, piga-mac et piga-firewall) qui sont déployées automatiquement par le processus Contexd. Bien que l'automate soit traduit sous forme d'un ensemble de fichiers XML, nous disposons d'un outil grahique 4 permettant de configurer les différents domaines et les transitions entre ces domaines. La figure illustre la configuration du domaine email et des différentes applications autorisées dans ce do-

maine. L'outil permet de décrire les programmes qui permettent de transiter vers le domaine email ainsi que les transitions entrantes et sortantes.
Une transition provient d'une application demandeuse d'un domaine cible. Une règle de transition est composée 1) des conditions de transitions (`mrule`) et 2) des domaines sources autorisés. La transition une fois définie graphiquement est transposée automatiquement par l'outil graphique en un ensemble de fichiers XML.
Un fichier XML permet de décrire les règles de transitions. Ainsi, la règle suivante concerne la transition vers le domaine impôts pour l'application firefox. Si la requête envoyée par firefox à Contexd correspond à l'établissement d'une connexion en https vers un site dont l'URL valide l'expression `.*\.impots\.gouv\.fr`, alors la transition sera autorisée si le domaine actuel est `impots` ou `default`.

```
<!--Impots-->
<rule app_name="firefox" to_domain="impots" notify="false" prompt="true" display_name="domaine impots">
        <matching>
                <mrule>
                        <host>.*\.impots\.gouv\.fr</host><path>.*</path><protocol>https</protocol>
                </mrule>
        </matching>
        <transitions>
                <allow>
                        <transit from_domain="impots" notify="false" prompt="false"/><transit from_domain="default" />
                </allow>
        </transitions>
</rule>
```

D'autres fichiers permettent de configurer les politiques obligatoires nécessaires pour le domaine cible. Par exemple pour la protection SELinux, la politique `partage-impots` est chargée automatique par Contexd pour l'entrée dans le domaine `impots` :

```
<PIGA_SELinux_Plugin>
        <module full_path="/etc/context.d/selinux/partage-impots.pp" app_name="*" domain="impots" />
</PIGA_SELinux_Plugin>
```

#### 2.4.2. Exemple de contrôle de flux entre domaines

Les figures 4 et 5 illustrent un scénario d'attaque que contrôle notre approche dynamique entre les deux domaines e-commerce et mail. Il s'agit du scénario en deux étapes décrit dans la partie motivation. La première étape est l'accès au domaine e-commerce lorsque l'utilisateur entre l'URL du site ebay. Dans ce domaine, l'utilisateur peut lire les emails provenant d'ebay et peut créer un fichier ebay.odt dans le répertoire /user/download/mail. La seconde étape est l'accès au domaine mail permettant à l'utilisateur de lire des emails non sûrs. On voit que depuis ce domaine alors qu'il utilise la même application Clawsmail et qu'il n'est pas sorti de sa session, il n'a pour autant accès ni aux emails d'ebay (d'ailleurs il ne voit plus l'existence de ces emails) ni aux données issues du domaine e-commerce (il ne voit pas non plus l'existence du fichier ebay.odt dans /user/download/mail).

## 3. Performances et expérimentations

### 3.1. Performances comparatives des solutions

Le tableau 1 donne les performances relatives des trois solutions du défi. Il compare les temps 1) de démarrage et d'arrêt des 3 systèmes en concurrence, 2) d'ouverture de session, 3) de lancement des applications, 4) de chargement de pages Web et 5) d'accès aux emails.
Globalement les performances de PIGA-OS sont meilleures que les deux autres. Malgré le temps nécessaire pour la protection en profondeur multi-domaines, ces résultats établissent que PIGA-OS est plus performant que les approches de virtualisation légères (Vserver pour OS4) ou classiques (Xen pour SAFE-OS). En effet dans le cas d'OS4 et de SAFE-OS, il est nécessaire d'avoir une image virtuelle par domaine.

### 3.2. Scores comparatifs des solutions

Le tableau 2 résume les scores liés aux vulnérabilités trouvées dans les trois solutions. PIGA-OS arrive en tête des trois phases d'attaques que comprenaient le défi. Sans entrer dans le détail de toutes les failles trouvées et des correctifs proposés, on voit que PIGA-OS protège mieux contre les vulnérabilités.

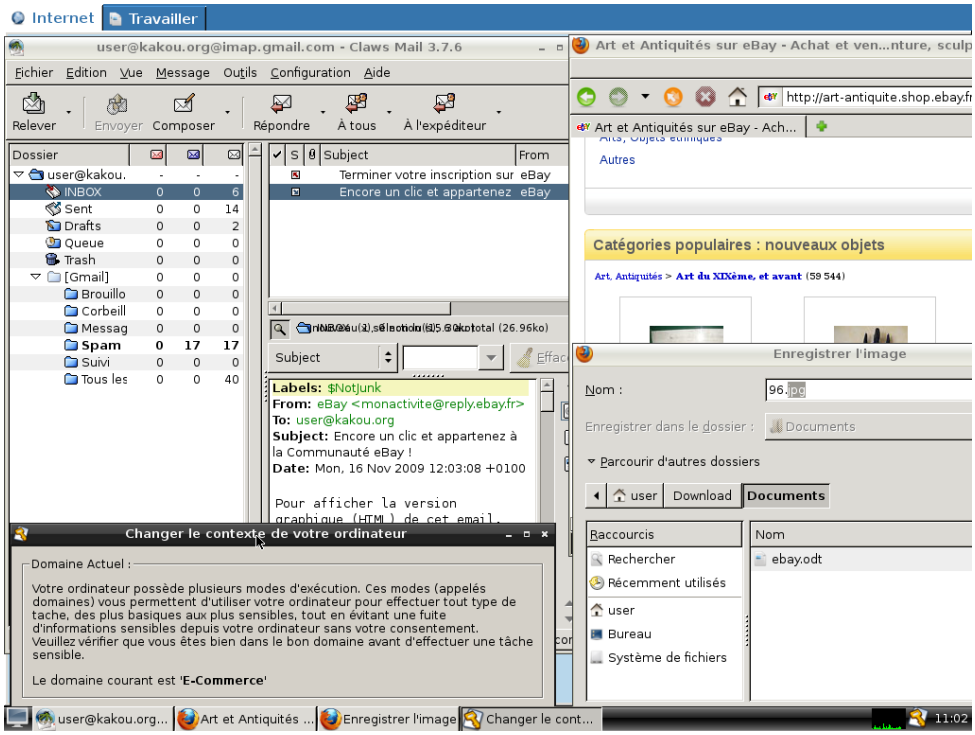


FIGURE 4 – Etape 1 : accès au domaine e-commerce

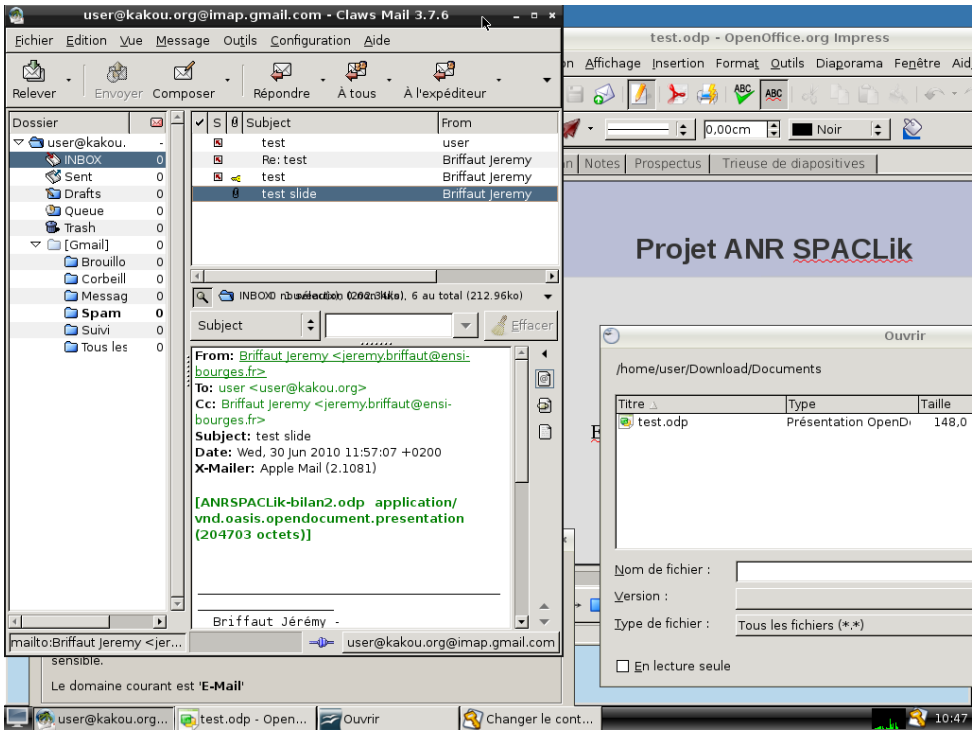


FIGURE 5 – Etape 2 : accès au domaine mail

TABLE 1 – Performances comparatives des 3 solutions

| Temps | | PIGA-OS | SAFE-OS | OS4 |
|---|---|---|---|---|
| Démarrage | premier | 01 :03 | 04 :32 | 00 :40 |
| | suivants | 01 :00 | 01 :10 | 00 :40 |
| Arret | | 00 :35 | 00 :23 | 00 :48 |
| ouverture session | | 00 :02 | 00 :04 | 00 :14 |
| Lancement | Web | 00 :03 | 00 :09 | 00 :11 |
| | Mail | 00 :02 | 00 :10 | - |
| | Office | 00 :03 | - | - |
| | PDF | 00 :01 | - | - |
| Chargement Page | ecommerce | 00 :04 | 00 :06 | 00 :05 |
| | Youtube | 00 :04 | 00 :02 | - |
| | impots | 00 :04 | 00 :05 | - |
| Mail | chargement | 00 :12 | 00 :15 | - |
| | rechargement | 00 :04 | 00 :06 | - |

Les objectifs fixés par le défi étaient difficiles à atteindre puisqu'il s'agit d'empêcher des scénarios d'attaques sophistiqués tel que l'exploitation d'une vulnérabilité dans une application permettant des flux d'informations compromettant la confidentialité ou l'intégrité. PIGA-OS offre un moyen très efficace de corriger les vulnérabilités puisqu'il permet d'utiliser le scénario d'attaque pour définir la propriété qui empêche la vulnérabilité et toutes ses variantes. Par exemple, la propriété `dutiesseparationbash` a permis de contrer une attaque reposant sur le téléchargement par firefox d'un script puis l'exécution interprétée de ce script.
Ces scores confirment qu'une virtualisation à elle seule, telle que proposée dans les solutions SAFE-OS et OS4, ne peut pas garantir de propriétés de sécurité. Seule une approche de protection obligatoire en profondeur, comme celle de PIGA-OS, permet de garantir les objectifs de sécurité visés par le défi.

TABLE 2 – Expérimentation dans le cadre du défi sécurité

| | PIGA-OS | SAFE-OS | OS4 |
|---|---|---|---|
| 1ere phase | 210 | 31 | 59 |
| 2eme phase | 124 | 96 | 80 |
| 3eme phase | 103 | 100 | 97 |

## 4. État de l'art

Il existe deux types de protection, discrétionnaire et obligatoire, utilisés par les systèmes d'exploitation. Le contrôle d'accès basé sur les rôles (RBAC) [5] ne change rien en terme de garantie, il facilite uniquement l'écriture de politiques de sécurité. Avec le modèle DAC, les privilèges sur un objet sont définis par l'utilisateur propriétaire de l'objet. Plusieurs études [10, 5, 13] ont montré la faiblesse des modèles DAC. Ce modèle ne permet pas de garantir des propriétés de sécurité.
Seule une protection obligatoire MAC permet à un système d'exploitation de garantir des propriétés de sécurité. Pour contrôler les accès entre les sujets et les objets du système, Anderson [1] a introduit le moniteur de référence pour décrire tout mécanisme qui place la gestion de la politique de sécurité hors d'atteinte des utilisateurs. Les approches Type Enforcement [7] associent des types précis aux ressources

mais nécessitent des politiques constituées de plusieurs milliers voir millions de règles de protection. Par exemple, [2] analyse des politiques SELinux et montre que ces politiques autorisent plusieurs millions d'activités permettant de violer une des propriétés de sécurité requises.
D'autres approches comme [12, 6, 14] essayent de combiner la facilité d'utilisation du DAC avec la qualité de protection du MAC. Cependant, elles ne permettent pas de formaliser des propriétés de sécurité et la qualité de leur protection est discutable car les flux d'information indirects ne sont pas pris en compte.
Des approches comme [17, 18] sont orientées vers le contrôle de flux d'information. Il s'agit d'approches obligatoires où la politique est écrite par les développeurs d'application comme dans le système Flume [4] basé sur GNU/Linux. Cependant, ces approches ne permettent pas la définition d'objectifs de sécurité mellant différents flux et nécessitent de réécrire une partie des applications.
Il existe également des solutions de protection dédiées au navigateur Web comme NetTop [15] ou Tahoma [3] ou à la protection de JavaScript [9, 16]. [11]. Ces approches prennent en compte des objectifs de sécurité limités. Dans [8], les auteurs proposent une protection en profondeur pour le navigateur Web nécessitant une virtualisation. Cependant, ils ne traitent pas d'autres objectifs de sécurité que le modèle de confidentialité Bell&LaPadula.
En conclusion, les protections obligatoires existantes ne traitent pas ou partiellement les activités indirectes, elles sont trop complexes, ne couvrent pas tous les objectifs de sécurité nécessaires pour une protection en profondeur. De plus, il manque une coordination dynamique des différents niveaux de protection obligatoire en fonction des activités des processus et des utilisateurs.

## 5. Conclusion

PIGA-OS a montré son efficacité dans le cadre du défi sécurité. C'est une approche légère qui fournit une adaptation dynamique de politiques obligatoires en fonction des domaines d'usage (impôts, ecommerce, web, email, ...). Pour avoir une sécurité en profondeur, nous proposons des protections obligatoires (SELinux, PIGA-MAC, XSELinux, PIGA-FIREWALL) pour tous les niveaux (processus et ressources, interface graphique, réseau) du système. Par rapport à SELinux, PIGA-MAC offre une protection obligatoire qui contrôle efficacement les flux d'informations indirects correspondants à des canaux cachés. PIGA-MAC permet la définition d'objectifs de sécurité avancés et autorise des canevas de protection qui facilitent le travail d'un administrateur système. L'approche développée dans PIGA-OS complète la protection obligatoire de PIGA-MAC par une gestion dynamique des domaines d'usage grâce à PIGA-SYSTRANS. Ce composant capture les interactions au niveau des applications et permet de modifier à la volée les politiques obligatoires pour contrôler les flux entre les différents domaines.
L'approche développée dans le défi présente de nombreuses extensions, notamment pour la protection des clusters, des grilles de calcul et des clouds mais aussi des systèmes embarqués ou virtualisés. En effet, la notion de contexte de sécurité est très générale et extensible. Un contexte peut représenter un processus sur un noeud de calcul, une méthode d'un service web voir un objet mémoire. Différents travaux sont en cours pour la protection des clusters, des grilles, des systèmes virtualisés et des clouds. Nous sommes en train d'adapter la solution à des systèmes Windows 7, ce qui montre l'extensibilité de l'approche. Enfin, nous travaillons au contrôle des flux mémoire au sein d'un processus et du noyau, ce qui permettra de couvrir d'autres types de canaux cachés.

## Remerciements

Nous tenons à remercier Steve Dodier pour sa contribution au développement de l'outil de configuration pour PIGA-SYSTRANS.

## Support

Ce travail a été supporté par le programme ANR "Défi Sécurité Système d'Exploitation Cloisonné et Sécurisé pour l'Internaute" (Sec&Si) de 2008 à 2010.

**Bibliographie**

1. J.P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, April 1980.
2. Jérémy Briffaut, Jean-François Lalande, and Christian Toinard. Formalization of security properties : enforcement for mac operating systems and verification of dynamic mac policies. *International journal on advances in security*, 2 :325–343, 2009.
3. Richard S. Cox, Steven D. Gribble, Henry M. Levy, and Jacob Gorm Hansen. A safety-oriented platform for web applications. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 350–364, Washington, DC, USA, 2006. IEEE Computer Society.
4. Petros Efstathopoulos and Eddie Kohler. Manageable fine-grained information flow. *SIGOPS Oper. Syst. Rev.*, 42(4) :301–313, 2008.
5. D. F. Ferraiolo and D. R. Kuhn. Role-based access controls. In *15th National Computer Security Conference*, pages 554–563, Baltimore, MD, USA, October 1992.
6. T. Fraser. LOMAC : Low Water-Mark integrity protection for COTS environments. *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pages 230–245, 2000.
7. T. Fraser and L. Badger. Ensuring continuity during dynamic security policy reconfiguration in dte. pages 15 –26, may. 1998.
8. Boniface Hicks, Sandra Rueda, Dave King, Thomas Moyer, Joshua Schiffman, Yogesh Sreenivasan, Patrick McDaniel, and Trent Jaeger. An architecture for enforcing end-to-end access control over web applications. In *Proceeding of the 15th ACM symposium on Access control models and technologies*, SACMAT '10, pages 163–172, New York, NY, USA, 2010. ACM.
9. O. Ismail, M. Etoh, Y. Kadobayashi, and S. Yamaguchi. A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability. In *Advanced Information Networking and Applications, 2004. AINA 2004. 18th International Conference on*, volume 1, pages 145 – 151 Vol.1, 2004.
10. ITSEC. Information Technology Security Evaluation Criteria (ITSEC) v1.2. Technical report, June 1991.
11. Collin Jackson, Andrew Bortz, Dan Boneh, and John C. Mitchell. Protecting browser state from web privacy attacks. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 737–744, New York, NY, USA, 2006. ACM.
12. Ninghui Li, Ziqing Mao, and Hong Chen. Usable mandatory integrity protection for operating systems. In *SP '07 : Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 164–178, Washington, DC, USA, 2007. IEEE Computer Society.
13. Peter A. Loscocco, Stephen D. Smalley, Patrick A. Muckelbauer, Ruth C. Taylor, S. Jeff Turner, and John F. Farrell. The Inevitability of Failure : The Flawed Assumption of Security in Modern Computing Environments. In *Proceedings of the 21st National Information Systems Security Conference*, pages 303–314, Arlington, Virginia, USA, October 1998.
14. Ziqing Mao, Ninghui Li, Hong Chen, and Xuxian Jiang. Trojan horse resistant discretionary access control. In *SACMAT '09 : Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 237–246, New York, NY, USA, 2009. ACM.
15. R. Meushaw and D. Simard. Nettop-commercial technology in high assurance applications. *National Security Agency, Tech Trend Notes*, 2000.
16. Charles Reis, John Dunagan, Helen J. Wang, Opher Dubrovsky, and Saher Esmeir. Browsershield : Vulnerability-driven filtering of dynamic html. *ACM Trans. Web*, 1, September 2007.
17. Steve Vandebogart, Petros Efstathopoulos, Eddie Kohler, Maxwell Krohn, Cliff Frey, David Ziegler, Frans Kaashoek, Robert Morris, and David Mazières. Labels and event processes in the asbestos operating system. *ACM Trans. Comput. Syst.*, 25(4) :11, 2007.
18. Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. Making information flow explicit in histar. In *OSDI '06 : Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, pages 19–19, Berkeley, CA, USA, 2006. USENIX Association.