energie atomique • energies alternatives

Contrôle d’accès mandataire pour Windows 7

De la théorie à la pratique...

Damien Gros

Doctorant au CEA/DAM/DIF
F-91297 Arpajon, France

Commissariat à l’Energie Atomique et aux Energies Alternatives

Directeur : Christian Toinard
Encadrants : Mathieu Blanc, Jérémy Briffaut

damien.gros@cea.fr

7 Juin 2012

energie atomique • energies alternatives

Sommaire

Introduction 1/2

- Le contrôle d'accès discrétionnaire (DAC) est devenu insuffisant :
  - L'utilisateur choisit les permissions (quand il veut bien le faire) ;
  - Pas de consistance de la politique DAC ;

Définition : Politique consistante

A partir d'un état sûr du système, il est difficile de démontrer qu'il n'existe pas au moins un chemin pour atteindre un état non sûr de la machine.

  - Très limité au niveau contrôle d'accès (lecture, écriture, exécution) ;
- Pas de possibilité de faire de la *confidentialité* ou de l'*intégrité*.

Introduction 2/2

- Mise en place d'un contrôle d'accès mandataire :
  - Implémente un moniteur de référence : applique la politique de sécurité définie par l'administrateur ;
  - Sépare le stockage de la politique et la prise de décision ;
  - Gère les flux d'information directs (un sujet agit directement sur un objet ou un autre sujet) ;
  - Refuse toute action non présente dans la politique.

- Apporte la possibilité de vérifier la politique vis-à-vis d'objectifs de sécurité.

Pour aller plus loin...

- Gestion des flux d’information indirects (combinaison de flux directs) ;
- Gestion des flux d’information complexes (combinaison de plusieurs types de flux) ;
- Utilisation d’un second moniteur de référence : PIGA-HIPS (non abordé ici).

Contrôle d'accès mandataire : pourquoi ?

- Parce qu'il faut faire de la sécurité positive !
- Possibilité de définir une politique de sécurité précise ;
- Confinement des programmes (même ceux tournant en Administrateur) ;
- Possibilité de définir des objectifs de sécurité et de les vérifier !

Quelques exemples sous Linux...

- grsecurity :
  - Basé sur *Trusted Path Execution* (chemin de confiance pour les binaires) ;
  - Fourni sous la forme d'un patch noyau ;
  - Mode apprentissage ;
  - Renforcement des diverses protections grâce à **PaX** ;
- SELinux :
  - Utilise les *Linux Security Modules* ;
  - Implémente le principe de *Type Enforcement* ;
  - Basé sur la notion de rôle ;
  - Nombreux outils aidant à la création de modules ;
  - Créé par la NSA, maintenu par Red Hat ;
- D'autres : SMACK, TOMOYO, RSBAC.

Et sous Windows 7 ?

- Mandatory Integrity Control (Windows Vista et supérieur)
  - Associe un niveau d'intégrité aux objets/processus du système ;
  - Simplification du modèle de Biba (protection en intégrité) ;
  - Pas de possibilité de définir une politique de sécurité ;
  - A déjà montré ses faiblesses.

    `http://www.securelist.com/en/blog/337/TDL4_Starts_Using_0_Day_Vulnerability`
- Travaux autour du contrôle d'accès :
  - Etudes sur des points précis du fonctionnement de Windows (RPC par exemple) ;
  - Restreints au modèle DAC.

Contexte de sécurité

- Pour les sujets (processus) : défini par transition ;
- Pour les objets (fichiers, sockets, dossiers...) : défini dans le `file_context` (au niveau du système de fichiers, dans les *extended attributes*) ;
- Un contexte de sécurité :
  - une identité : spécifique à SELinux mais liée à l'UID Linux de l'utilisateur ;
  - un rôle : permet d'accéder à des types et donc de pouvoir faire certaines actions (modèle RBAC) ;
  - un type (objet)/domaine (sujet) : sur celui-ci que va se baser la politique ;
  - Une ou plusieurs catégories ;
  - Une ou plusieurs sensibilités.

Type Enforcement

- Confinement des processus : réduction de l'impact sur le système en cas de compromission de l'application ;
- Réduction des privilèges (même pour les processus privilégiés) ;
- Basé sur la notion de types ;
- Politique textuelle, théoriquement portable d'un système à un autre ;
- Problèmes : politique longue et difficile à écrire.

Pourquoi labelliser ?

- Pour caractériser les ressources ;
- Sous SELinux, le fichier *file_context* :
  - Associe à chaque ressource un contexte de sécurité qui lui est unique ;
  - Permet la portabilité de la politique de contrôle d'accès ;
  - Stocké dans le système de fichiers.
- Sous Windows :
  - Pas de fichier à maintenir ;
  - Flux NTFS ? Perte de compatibilité avec le *FAT32*.

Fonctionnement

- Labellisation au moment de l'interception de l'appel système ;
- Utilisation des variables d'environnement dans le but de décrire génériquement la ressource ;
- Utilisation de types spécifiques pour décrire la ressource ;
  - .exe $\rightarrow$ _exec_t ;
  - .dll $\rightarrow$ _dll_t ;
  - Répertoire $\rightarrow$ _dir_t et représentation du chemin (presque) complet ;
- Reconstruction de l'activité du système.

Intérêt de la labellisation dynamique


```
C :\Windows\System32\cmd.exe
\Device\HarddiskVolume2\Windows\System32\cmd.exe
\??\C :\Windows\System32\cmd.exe

%systemroot%\System32\cmd.exe  system_u :object_r :cmd_exec_t
```

```
C :\Windows\System32\drivers

%systemroot%\System32\drivers  system_u :object_r :|system32|drivers_dir_t
```

Politique MAC : construction / automatisation

- Associer à chaque type sujet un ensemble d’action sur des objets ;
- Longue à écrire ;
- Utilisation des logs pour générer les règles plus facilement.

Exemple de création de règle


```
audit(1285243100 :4599)
avc :denied { read execute write create getattr } for pid=1576
comm="%systemroot%\explorer.exe" ppid=1528 path="%systemroot%\system32\cmd.exe"
scontext=system_u :system_r :explorer_t tcontext=system_u :system_r :cmd_exec_t
tclass=file
```

```
allow explorer_t cmd_exec_t :file  { read execute write create getattr }
```

Implémentation du *driver*

- Driver posant des hooks sur la SSDT (32 bits) ;
  - Facile à implémenter ;
  - Offre un contrôle de tout le système.
- Filter-driver (64 bits) : en cours de développement ;
  - Modèle préconisé par Microsoft ;
  - Totale portabilité ;
- Génération des logs par le driver.

Analyse de la protection

- Contrôle des flux d’information directs ;
- Détection des violations de propriétés simples ;
- Blocage de l’installation de malware à différents points.

Exemple : exécution d’un malware

- On considère que le binaire installant le malware est déjà sur la machine ;
- On va montrer les différents points d’interception ;
- Garder à l’esprit que toute action qui n’est pas dans la politique est interdite ;
- Le driver est donc mis en mode apprentissage.

Exemple : exécution d'un malware

```
type=AVC msg=audit(129774391393331470,214) avc :denied { execute } for pid=2344
com="%systemroot%\explorer.exe" ppid=2304
path="%systemdrive%\users\bob\desktop\8ce6d0d9f6906f3bf0233a3090226f11.exe"
scontext=system_u :system_r :explorer_t
tcontext=system_u :object_r :8CE6D0D9F6906F3BF0233A3090226F11_exec_t tclass=load
```

```
type=AVC msg=audit(129774392137591499,7148) avc :denied { write } for pid=3888
com="%systemdrive%\users\bob\desktop\8ce6d0d9f6906f3bf0233a3090226f11.exe"
ppid=2344 path="%systemdrive%\programdata\vwqgjwsurthvme.exe"
scontext=system_u :system_r :8CE6D0D9F6906_t tcontext=system_u :object_r :file_t
tclass=file
```

```
type=AVC msg=audit(129774392138104454,7158) avc :denied { execute } for
pid=3888
com="%systemdrive%\users\bob\desktop\8ce6d0d9f6906f3bf0233a3090226f11.exe"
ppid=2344 path="%systemdrive%\programdata\vwqgjwsurthvme.exe"
scontext=system_u :system_r :8CE6D0D9F6906_t
tcontext=system_u :object_r :VwQGJwSURThVmE_exec_t tclass=load
```

Analyse des performances

- Tests de performances réalisés en machine virtuelle ;
- Impact sur le lancement des applications ;
- Mesure plus précise à faire !

Performances

Processus et Arguments	Sans IPS	Avec IPS
Firefox : google.fr	2 s	4 s
Firefox : jeuxvideo.com	9 s	11 s
Internet Explorer : google.fr	3 s	5 s
Internet Explorer : msn.com	4 s	6 s
Windows Media Player	2 s	4 s
Windows Media Player : mp3	223 s	225 s
Windows Media Player : avi	12 s	13 s
Adobe Reader 9.0	1 s	2 s
Adobe Reader 9.0 : file 832 Ko	1 s	2 s
Adobe Reader 9.0 : file 18.5 Mo	2 s	4 s
Adobe Reader 9.0 : file 294 Mo	2 s	4 s

Table : Résultats des tests de performances

Limites

- Hook sur la SSDT ;
- Traitement du registre ; (tout a le même type)
- Labellisation dynamique :
  - Désignation unique des exécutables ;
  - Solutions : prise en compte du chemin complet ou séquence d’interaction.

Conclusion

- Implémentation du *Type Enforcement* sur système Windows ;
- Les premiers résultats sont concluants : sécurité et performance ;
- Des outils sont fournis pour aider à l'administration ;
- La labellisation dynamique offre une vraie portabilité décorrélée du système de fichiers ;

Perspectives

- Faire un filter-driver complet (s'affranchir des hooks sur la SSDT) ;
- Utilisation des SID pour stocker les contextes sujet (identité) ;
- Gérer les transitions ;
- Ajout de PIGA-HIPS pour gérer aussi les flux indirects.

energie atomique • energies alternatives

Questions ?

Merci !!!!