Contrôle d'accès e le POSIX Access Control Lists(ACL)

CS435 - Administration de Système

Dan Pham et Fabrício Nascimento

Octobre 2009

Introduction

Quand on désire contrôler l'accès aux données dans un système de fichiers, il y a plusieurs moyens d'y parvenir. Par défaut, les systèmes POSIX (Portable Operation System Interface)[2, 3] ont un mécanisme qui permet d'associer chaque entité avec un ensemble de règles, lequel est composé par une séquence d'octet qui exprime les droits du propriétaire, de son groupe et des autres utilisateurs.

Ce mode, traditionnel, assez simple est capable de résoudre les problèmes les plus fréquents. Par contre, il pose des limitations aux administrateurs de systèmes qui pour exprimer leurs besoins doivent employer des configurations non évidentes. Certaines applications choisissent de développer leur propre système de droit comme le serveur FTP Proftp[4] pour résoudre ce problèmes de droits.

Pour remédier à ces limitations, les systèmes UNIX peuvent employer les ACL. Cet article présente une exposition sur les ACL POSIX, ses modes de fonctionnement, ses qualités et désavantages. Le texte s'inspire de l'article d'Andreas Gruembacher[1] qui a fait partit de l'équipe ayant ajouté le support aux ACL dans le noyaux Linux pour les systèmes de fichiers ext2 et ext3, qui sont les plus utilisés dans les monde UNIX.

1 Le POSIX 1003.1

Traditionnellement les systèmes qui implémentaient la norme POSIX avaient un système simple et puissant de permissions mais qui cependant posait certains problèmes. En effet, les différentes versions d'ACL disponibles étaient incompatibles entre elles.

Pour normaliser les problème de sécurité sur les systèmes POSIX (ACL en faisant partie), un groupe a été formé pendant la définition de la famille de normes POSIX 1003.1. Les premiers documents POSIX qui ont pris en compte ces questions étaient les documents 1003.1e (*System Application Programming Interface*) et 1003.2c (*Shell and Utilities*), cependant, le premier draft était trop ambitieux. En effet, le groupe responsable pour la normalisation avait divisé ses efforts sur un grand nombre de domaines qui comportaient les *Access Control Lists* (ACL), les *Audit*, les *Capability*,les *Mandatory Access Control* (MAC), et l'*Information Labeling*[1].

En Janvier de 1998[1] le financement pour ce projet à été suspendu, par contre, le travail n'était pas prêt. De toute façon le dixsèptieme draft a quand même été rendu public[5].

Après cette publication, des systèmes UNIX appelés "*trusted*" (Trusted Solaris, Trusted Irix, Trusted AIX) ont été développés à partir du draft 17. Ces systèmes ne sont pas complètement compatibles entre eux. Heureusement aujourd'hui la plupart des systèmes UNIX et UNIX-like supportent les ACL. Ces implémentations sont usuellement compatibles avec le draft 17. Le projet TrustedBSD implémente aussi les ACL sur les système BSD. Les ACL sont apparues sur les Macs en 2003 avec la RELEASE MAC FreeBSD.

Les ACL sont une évolution du système de permissions traditionnel présent dans pratiquement tous les systèmes UNIX, alors, avant d'expliquer les ACL on va d'abord parler du modèle traditionnel.

Système de permissions traditionnel

Le modèle traditionnel POSIX offre trois classes d'utilisateurs qui sont : le propriétaire (*owner*), le groupe propriétaire (*group*) et les autres utilisateurs (*others*). Chaque groupe a un octet que indique les permissions de lecture (***r**ead*), d'écriture (***w**rite*) et d'exécution (*e**x**ecute*).

Après les trois octets peut venir le *Set User Id*, *Set Group Id* et le *Sticky Bit* qui peuvent être utilisés dans certain cas. Il faut faire attention avec le *Sticky Bit*, il permet aux utilisateurs normaux d'exécuter les utilitaires comme l'administrateur(*root*), donc une faille de sécurité dans une application utilisant le *Sticky Bit* peut compromettre le système entier.

Seul le *root* peut créer les groupes et changer les associations de groupes. Il peut aussi changer les propriétaires.

Les ACL

Chaque ACL est une ensemble de règles d'accès. Dans une modèle de sécurité utilisant les ACL, si une entité fait une requête pour accéder aux données, il faut consulter les la liste d'ACL pour savoir si nous avons la permission pour l'opération demandé. Les règles possibles peuvent être consultées dans le tableau ci-dessous(**??**).

Les types de ACL	
Type d'entrée	**format**
Propriétaire	user : :rwx
Utilisateur nommée	user :name :rwx
Groupe propriétaire	group : :rwx
Groupe nommée	group :name :rwx
Masque	mask : :rwx
Autres	other : :rwx

Les règles sont formées par un indicateur de classe (comme les classes du système traditionnel), l'identificateur pour préciser de quel utilisateur ou groupe on parle puis les octets de permissions.

Avec cette représentation le sens de la classe du groupe a été redéfini comme le limite supérieur de les permission de chaque entrée dans la classe du groupe. C'était a dire que les entrée du groupe et du utilisateur nommées seront désigner à entrée du groupe. Aussi, c'est importante rappeler que cette choix permettre se prémunir contre les application qui ne sont pas conscient de les ACL. [1].

Les ACL équivalentes au mode simple de permissions s'appellent les ACL minimales. Si les ACL possèdent des entrées supplémentaires, ont les appelle ACL étendues. Toutes les ACL étendues doivent avoir une entrée masque et peuvent contenir théoriquement autant d'entrées que l'on désire. On verra après que ce numéro d'entrée peut-être limitée pour chaque implémentation et qu'il est aussi important pour les performances.

Dans les ACL étendues on peut avoir des entrées avec plusieurs utilisateurs et/ou groupes, quelques de cette entrées peut-être contenir permissions qui la classe groupe n'aurais pas, alors, on peut avoir une inconsistance basée en le cas qui les permissions du groupe propriétaire sont diffèrent de les permission de la classe groupe.

On peut résoudre ce problème avec un masque. Comme on peut l'observer dans le figure (1), il y a deux cas : Les ACL minimales où la classe groupe est référencée pour l'entrée du groupe propriétaire. Les ACL étendues de la

[1] Fabricio : Je ne comprend pas bien cette affirmation, je laisse ici le teste orignial : These named group and named user entries are assigned to the group class, which already contains the owning group entry. Different from the POSIX.1 permission model, the group class may now contain ACL entries with different permission sets, so the group class permissions alone are no longer sufficient to represent all the detailed permissions of all ACL entries it contains. Therefore, the meaning of the group class permissions is redefined : under their new semantics, they represent an upper bound of the permissions that any entry in the group class will grant. This upper bound property ensures that POSIX.1 applications that are unaware of ACLs will not suddenly and unexpectedly start to grant additional permissions once ACLs are supported.

classe groupe seront trouvées en faisant un masque avec les permissions du groupe propriétaire et les permissions des utilisateurs nommés mais si l'entrée de permission du groupe propriétaire possède des droits supérieurs au masque ils seront conservés. Cela rend difficile le calcul de classe groupe.

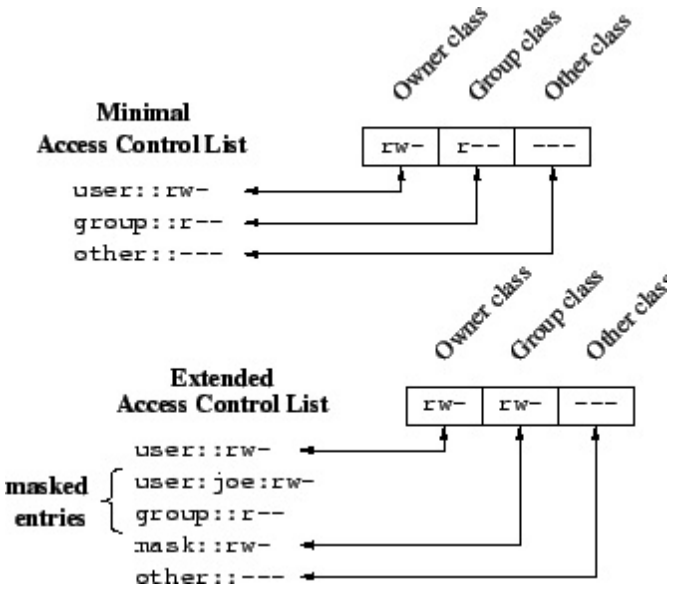

FIG. 1 – caption

Pour assurer le cohérance, quand une application change les permissions (par exemple le commande *chmod*) les ACL sont modifiée de façon a reproduire cette modification.

On a dit que la permission de la classe groupe est calculée comme la limite supérieure de tous les entrées dans le classe group. Avec les ACLs minimaux cette computation est simple, par contre, avec les ACLs étendu, on a besoin de masquer les permissions. Comme l'exemple de le tableau (**??**), les entrées de permission qui sont partie de la classe de groupe et qui aussi sont présente dans l'entrée masque sont applique effectivement. Si une permission était absent dans le masque, c'est a dire que aucun entrée de group (qui non le groupe du propriétaire) peut avoir ce permission, on dit dans ce cas qui la entrée est masquée.

La masque de permissionL		
Type	**Format**	**Permission**
Utilisateur nommée	user :jean :r-x	r-x
Masque	mask : :rw-	rw-
Permission Effective		r–

Algorithme de vérification

Pour vérifier les droits d'accès d'une objet du système de fichier, il y a une algorithme assez simple.

Algorithm 1 Vérifie se une utilisateur peut ou ne peut pas accéder une objet du système de fichier

if the user ID of the process is the owner **then**
  the owner entry determines access
else if the user ID of the process matches the qualifier in one of the named user entries **then**
  this entry determines access
else if one of the group IDs of the process matches the owning group and the owning group entry contains the requested permissions **then**
  this entry determines access
else if one of the group IDs of the process matches the qualifier of one of the named group entries and this entry contains the requested permissions **then**
  this entry determines access
else if one of the group IDs of the process matches the owning group or any of the named group entries, but neither the owning group entry nor any of the matching named group entries contains the requested permissions **then**
  this determines that access is denied
else
  the other entry determines access.
end if
if the matching entry resulting from this selection is the owner or other entry and it contains the requested permissions **then**
  access is granted
else if the matching entry is a named user, owning group, or named group entry and this entry contains the requested permissions and the mask entry also contains the requested permissions (or there is no mask entry) **then**
  access is granted
else
  access is denied.
end if

Héritage mécanisme

Le système POSIX règle non seulement les accès aux objets du système, mais aussi sur le mécanisme d'Héritage. Les ACL sont partagés en deux types, les *access ACL* (qu'on a vu jusqu'á maintenant) et les *default ACL* qui comprennent les règles d'héritage.

Quand on parle de l'héritage, on parle des droits qui sont attribue aux objet du système pendant au moment où ils sont crées. Il y un seul type d'objet qui peut être associe avec les *default ACL* ; les répertoire. Il faut dire que il n'y a

aucune sens en donne *default ACL* pour les fichiers, alors que, aucune objet du système peut être crée dans une fichier, aussi, il faut rappeler que les *Default ACL* n'ont pas aucune implication sur les *access ACL*.

Si un répertoire est crée dans une autre, si le première répertoire a *default ACL*, d'accord avec le mécanisme de héritage, le deuxième aurais le même ACL qui le première (*default* et *access*). Les objets qui ne sont pas répertoire, devons hérite les *access ACL* seulement.

Chaque *system call* que crée les objets du système de fichier a une *mode parameter*. Ce paramètre peut contenir neuve octet de permission pour chaque classe (propriétaire, groupe et les autres). Les permission effective de chaque objet crée est l'intersection de les permission défini pour les *default ACL* et les spécification dans le *mode parameter*.

Le système traditionnel a une commande pour designer les modes de permission par défaut pour les nouveau fichiers et répertoires : le commande *umask*. Quand il n'y a aucune *default ACL*, le permission effective est détermine par le paramètre de mode moins les permission configuré avec *umask*.

2 ACL en use

Dans cette session on verrais les uses des ACL dans les système d'aujourd'hui.

2.1 ACL Kernel Patches

Les ACL *patches* ont été ajouter dans le noyaux Linux depuis November 2002. Cette *patches* implémentent le POSIX 1003.1e brouillon 17 et elles ont été ajoute dans le version 2.5.46 du noyaux. Donc le support ACL et aussi présent dans le dernière version du noyaux aujourd'hui. Depuis 2004 le support aux ACL étions disponible pour les système de fichier Ext2, Ext3, IBM JFS, ReiserFS et SGI XFS. Les ACL sont supporte aussi pour le système NFS, par contre, il y a quelques problèmes de sécurité connu[8].

Aujourd'hui c'est assez simple pour ajouter le supporte aux ACL dans les distribution Linux comme Ubuntu ou Debian. On verrais les pas pour ajouter ce supporte après.

2.2 Mac OSX

Le système de exploitation Mac OSX (10.6.2 Snow Leopard dans le moment de écriture de ce article) a aussi les supporte aux ACL complètement intégrée dans l'interface de utilisateur (2).

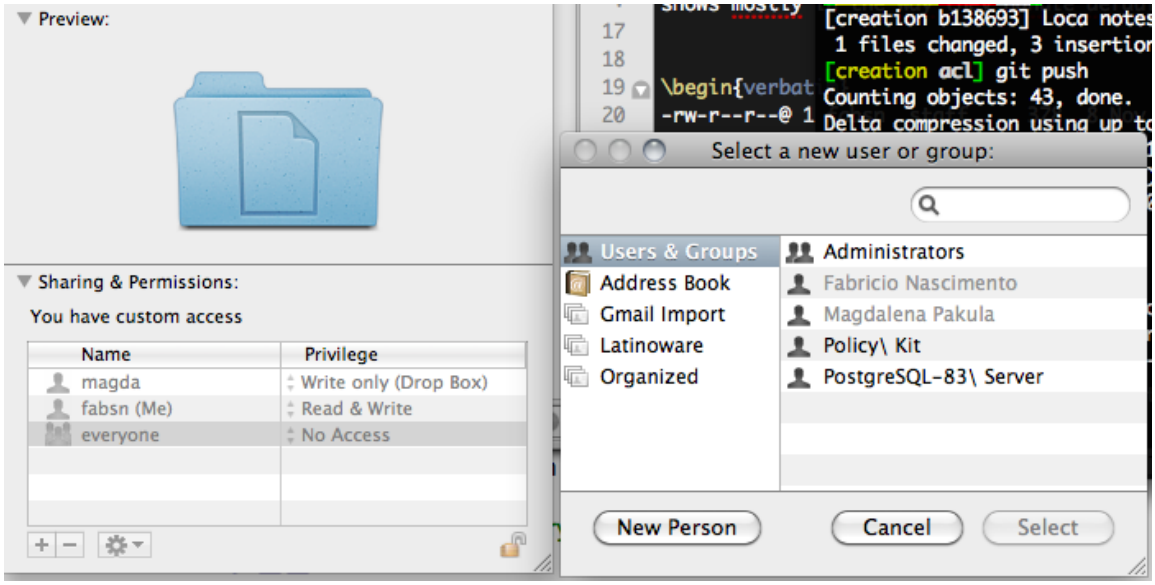

Fig. 2 – Mac OSX Snow Leopard ACL Interface

Using ACL in Linux

Les dernière version des distribution Debian ou Ubuntu, comme Ubuntu 9.10, dèjá vient avec le supporte aux ACL. Dans le Ubuntu 8.10 l'application Nautilus, qui est responsable pour la visualisation du système de fichier, contenait une interface pour les ACL, apparentement l'interface a été discontinue et le Nautilus du Ubuntu 9.10 n'en y a pas encore. Les pas pour ajouter le supporte dans le Ubuntu 9.10 sont :

```
1) Installer le paquet des acl.
user@ubuntu:$ sudo apt-get install acl

2) Ajouter le option 'acl' au système de fichier correcte dans le /etc/fstab, comment:
UUID='gros sequence' /dev/hda6 /home ext3 rw,auto,acl 0 1

3) Remounter le systeme de fichier avec le nouvelle option
user@ubuntu:$ sudo mount /home -o remount
```

Ajouter ACL aux fichiers

On peut utiliser le commande 'ls -la' pour regarde les permission. Si une fichier contient information de sécurité avancée (comme *access list*) on va voir le "character" '+', comment dans le sortie du command 'ls' ci-dessous (2.2). Une fichier avec '@' était dire que le fichier a quelque EAs.

```
-rw-r--r--@ 1 fabsn  staff     378  8 Nov 15:29 Makefile
-rw-r--r--@ 1 fabsn  staff     618  8 Nov 15:59 README
-rw-r--r--@ 1 fabsn  staff      31  8 Nov 15:15 draft-header
-rw-r--r--@ 1 fabsn  staff      24  8 Nov 15:15 header
drwxr-xr-x@ 2 fabsn  staff     102  8 Nov 15:26 img
-rw-r--r--  1 fabsn  staff     972  8 Nov 15:57 rapport-draft.aux
-rw-r--r--  1 fabsn  staff   18129  8 Nov 15:57 rapport-draft.log
drwxrwxr-x+ 3 fabsn  staff   1024  8 Nov 20:23 repertoire
```

Pour voir les ACL on doit utilise le commande *getfacl*. Regarde que les information sont ajoute d'accord avec les définition dans l'introduction sur les ACL dans la tabelle 1.

```
fabsn@vadmin:/media/esisar$ getfacl repertoire/
# file: repertoire/
# owner: root
# group: root
user::r-x
user:daemon:rwx
user:bin:rwx
user:fabsn:rwx
```

```
user:nobody:rwx
group::r-x
group:admin:rwx
group:fabsn:rwx
mask::rwx
other::r-x
```

Aussi on a le commande *setfacl* pour modifier, ou ajouter les permission ACL. Le commande dessous par exemple modifie (-m) les permission du utilisateur *fabsn* pour le répertoire.

```
setfacl -m u:fabsn:r-x repertoire
```

Exemple

HERE COMES THE EXAMPLE :-).

3 Implémentassions

Les ACLs sont fréquemment implémentent comme extensions du noyaux, ça veut dire modules dans le LINUX contexte. L'objectif de ce cession c'est explique superficiellement les questions concernant les implémentions de les ACL. Le discussion doit lancer la base pour les évaluations de performance et les problèmes dans les sessions prochaines.

"Les ACLs sont morceaux d'information de taille variable qui sont associe avec les objet du système de fichier"[1]. Plusieurs implémentations de cette modelé sont possible. Par exemple, avec Solaris dans le système de fichier UFS[6] chaque *inode* peut être avoir une ACL. Si il en a, il doit avoir l'information *i_shadow*, une pointeur pour une *shadow inode*. Les *shadow inode* sont comment fichiers régulières d'utilisateurs. Diffèrent fichier avec les mêmes ACL peut avoir pointeurs pour le même *shadow inodes*. Les information des ACL sont garde dans les bloque de donné de chaque *shadow inodes*.

La capacité de associer morceaux d'information avec fichiers est utilisé pour plusieurs fonctions du système de exploitation, alors, en la plupart de ces systèmes *UNIX-like* (Linux comprendre) on trouve les Attributs Étendu (*Extended Attributes (EAs)*). Les ACL sont implémentent avec ce mécanisme.

Le linux page de manuel[1] *attr(5)* contienne une explication précise sur les EAs dans linux, au notre but, suffi dire que comme les variables des processus, les EAs sont pairs (nom, valeur) associe de manier persistant avec les objet du système de fichier et que les appel de système linux, dans le espace de utilisateur, sont employé pour opérer sur les information de ces pairs dans le espace de adresse du noyaux. Aussi pour l'implémentation de cette infrastructure dans les système FreeBSD il faut voir le article de Robert Watson[7]. Cette article content aussi une comparaison de plusieurs implémentation de ces système.

Dans le monde linux, aouter le supporte aux ACL avec une version limité des EA offre plusieurs avantages : Facilité de implémentation, opération atomique et interface *stateless* que laisse aucun surcharge à cause de les *file handlers*. On verrais après dans le cession de performance, que l'efficience est assez importante pour être oublier quand on parle de les donnés fréquentent accès comme les ACL.

3.1 Les EAs e les système de fichiers

Dans le monde UNIX, chaque système de fichier a une diffèrent implémentation pour les EAs. On peut penser que une solution partage pour tous les système pouvait être meilleur. Par exemple, si on prendre la une solution simple comme chaque objet du système de fichier qui a les EAs, a une répertoire avec une fichier qui a le clés EA comment le nom et le contenu comment le valeur. Cette implémentation consommerait beaucoup de espace, étant donné que les blocks du système de fichier seront gaspiller pour conserver petit morceaux de donné, aussi ce solution perdrait les temps pour chercher ces information a chaque accès de fichier. Aux frais de ces problèmes chaque système tire profit de ces qualités pour ajouter le supporte aux EAs.

Ext (2,3 et 4)

Les ACL dans Ext suis le principe linux : "La plus simple solution que marche" et pour cette raison on quelques limitations. Autres solutions existent, par contre, elle sont difficile de ajouter au noyaux de manière satisfaire[9].

La solution actuel ajoute aux *i_node* une entrée que s'appelle *i_file_acl*. Cette entrée, si diffèrent de 0, est une pontier pour une EAs block. Cette EAs block a les informations de nom et valeur de tous les ACL du fichier indique pour cette *i_node*.

Le mécanisme a aussi une optimisation. Deux fichier avec le même ensemble de ACL point vers le même EA block. Le système guard un *hash map* avec les *checksum* dus blocks EA et leurs adresse. Chaque block a aussi un compteur de référence, comme les lien *hard*. Ce mécanisme aussi détermine que ce compteur la ne peut pas avoir plus que 1024 références. Il s'agit de une mesure de sécurité en cas de perte de les donne.

Aussi une limitation est imposé, tous les donnes EAs de une fichier doivent occuper une bloque EAs que peut avoir la tailler de 1, 2 ou 4 KBs.

JFS

Dans JFS, les EAs sont ajouter pour l'utilisation de les *extent*. Cette structure n'est que une consécutive liste de bloques. Pour les EAs, ça veut dire que chaque pair (nom,valeur) est garde en séquence, chaque valeur du pair ne peut pas être plus grande que 64kb. Si les EAs sont assez petite, elle pouvons être guard dans le même lieu que les information du fichier. De ce façon, il n'y a pas les limitation de ext3.

XFS

Dans XFS

ReiserFS

HGFS+

Samba

NFS

Conclusion

Références

[1] Andreas Gruenbacher, *POSIX Acess Control Lists on Linux*. http ://www.suse.de/ agruen/acl/linux-acls/online/, 2003.

[2] IEEE Std 1003.1-2001 (Open Group Technical Standard, Issue 6), Standard for Information Technology–Portable Operating System Interface (POSIX) 2001. ISBN 0-7381-3010-9. http ://www.ieee.org/

[3] IEEE 1003.1e and 1003.2c : Draft Standard for Information Technology–Portable Operating System Interface (POSIX)–Part 1 : System Application Program Interface (API) and Part 2 : Shell and Utilities, draft 17 (withdrawn). October 1997. http ://wt.xpilot.org/publications/posix.1e/

[4] Mark Lowes : Proftpd : A User's Guide March 31, 2003. http ://proftpd.linux.co.uk/

[5] Winfried Trümper : Summary about Posix.1e. Publicly available copies of POSIX 1003.1e/1003.2c. February 28, 1999. http ://wt.xpilot.org/publications/posix.1e/

[6] Jim Mauro : Controlling permissions with ACLs. Describes internals of UFS's shadow inode concept. SunWorld Online, June 1998.

[7] Robert N. M. Watson : Introducing Supporting Infrastructure for Trusted Operating System Support in FreeBSD. BSDCon 2000, Monterey, CA, September 8, 2000. http ://www.trustedbsd.org/docs.html

[8] Andreas Grünbacher : Linux Extended Attributes and ACLs. Session "Known Problems and Bugs". http ://acl.bestbits.at/problems.html

[9] Andreas Dilger : [RFC] new design for EA on- disk format. Mailing list communication, July 10, 2002. http ://acl.bestbits.at/pipermail/acl-devel/2002-July/001077.html

[10] Austin Common Standards Revision Group. http ://www.opengroup.org/austin/