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

CS435 - Administration de Système

Dan Pham et Fabrício Nascimento

Octobre 2009

Introduction

Quand l'objective c'est contrôle l'accès sur les données dans une système de fichiers, il y a plusieurs formes de règlement. Par défaut, les systèmes POSIX (Portable Operatin System Interface)[2, 3] ont une mécanisme que permettre de associer chaque entité avec un ensemble de règle, lequel est composé par une séquence d'octet que exprime le droit du propriétaire, de son groupe et des autres utilisateurs.

Ce mode traditionnel est assez simple et capable de adresser les problème plus fréquents. Par contre, il pose des limitation aux administrateur de système, lesquels fréquemment doivent employer quelques configuration non évidentes afin d'être capable de exprimer ces besoins. Par exemple les application comme le serveur FTP Proftp[4] ont ces exclusive façon de résoudre ces problèmes de droits pour accéder les objets du système de fichier.

À couse de remédier ces limitation présente les UNIX permettent d'employer les ACL.

Cette article présente une exposition sur les ACL POSIX, ces mode de fonctionnement, ces clefs de succès et désavantages. Le texte est fortement basé en l'article de Andreas Gruembacher[1] dont a été dans l'équipe que as ajouté le support aux ACL dans le noyaux Linux pour les système de fichier ext2 et ext3, lequel est le système de fichier plus utilisé dans les monde UNIX.

1 Le POSIX 1003.1

Traditionnellement les système qui implémentions les patron POSIX avaient une système simple et puissante de permission, quand même, certain problèmes ont arrive, et éventuellement les problèmes ont apparaître.

Après savoir la nécessite de régler sur le domaine de sécurise et non seulement les ACL, une groupe a était forme pendant la définition de la famille de patron POSIX 1003.1. Les premières documents POSIX qui ont été considère ces question étaient les document 1003.1e (*System Application Programming Interface*) et 1003.2c (*Shell and Utilities*), cependant, la première approximation de ce sujet était trop ambitieuse. Les groupe responsable pour le patronisation avait centre ces effort dans une tas assez grande de choses, lesquelles comprenant *Access Control Lists* (ACL), *Audit*, *Capability*, *Mandatory Access Control* (MAC), et *Information Labeling*[1].

En Janvier de 1998[1] le financement était fini, par contre, le travaille n'était pas prés. De toute façon le dixsèptieme brouillon a été publique quand même[5].

Donc après cette an, les système UNIX appelé "*trusted*" (Trusted Solaris, Trusted Irix, Trusted AIX ont été développer avec quelque parts de la documentation 17. Ces systèmes ne sont pas complètement compatible entre eux.

Aujourd'hui la plupart des système UNIX et UNIX-like support ACL. Ces implémentassions sont usuellement compatible avec le document 17. Le projet TrustedBSD aussi avait ajouter les ACL sur les système BSD. Les ACL e les MAC FreeBSD-RELEASE avaient apparu en 2003.

Les base des ACL sont lancé sur le système traditionnel présent usuellement dans presque tous les système UNIX, alors, avant de préciser sur les ACL on parlerais du modelé traditionnel.

Système de permission Traditionnel

Le modelé traditionnel POSIX offre trois group de utilisateur qui sont le propriétaire, le group e les autres. Chaque group a une octet que indique les permission de lecture (**r**ead), écrire (**w**rite) et exécution (e**x**ecute). La première classe fournit les permission pour le utilisateur que rempli le rôle de propriétaire, ensuite, vient les droits pour le groupe principal du propriétaire enfin les droites pour touts les autres utilisateurs.

Après les trois octets peut venir le *Set User Id*, *Set Group Id* et *Sticky bit*, lesquelles sont utilisé dans certain cases. Il faut faire attention avec le *Sticky Bit*, il permit les utilisateur normale d'exécuter les utilitaire comment le administrateur(*root*), par contre, quelque manque de sécurité peut compromettre le système entière.

Seulement le *root* peut créer les groups e changer les association de groupes. Celui-là que aussi peut changer les propriétaire.

Les ACL

Dans une modèle de de sécurise ACL, si quelque agent faire une requête pour accéder aux donnés, il faut consulte les ACL pour une entrée que permettre l'opération demandé. Chaque ACL est une ensemble de règles d'accès. Les règles possible peut-être regarder 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

Le format des règles sont formée par une indicateur de classe (comme les classe du système Traditionnel), une quantificateur pour spécifier de quel utilisateur ou group on parle et les octet de permission.

Avec cette représentation le sens de la classe du groupe a redéfini comment le limite supérieur de les permission de chaque entrée dans la classe du groupe. Ça arrive parce 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 qui sont équivalent avec le mode simple de permission de fichier s'appellent minimale et ils ont trois entrée. Si les ACL peut avoir plusieurs entrée ont les appelle étendu. Tous les ACL étendu doivent avoir l'entrée masque et peut contenir théoriquement n'importe combien de entrée. On verrais après que ce numéro d'entrée peut-être limitée pour chaque implémentassions et que aussi il est important pour la performance.

Dans les ACL minimale les permission de classe groupe sont égale à les permission de groupe propriétaire, cependant, dans les ACL étendu on peut avoir les entrée 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 résoudre ce problème avec le masque. Comme on peut observer dans le figure 11, il y a deux cas : les permission de la classe groupe seront déguise pour

[1] Fabricio : Je ne comprend pas 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.

la masque tandis que les permission de l'entrée du groupe propriétaire encore définit les permission pour le groupe propriétaire.

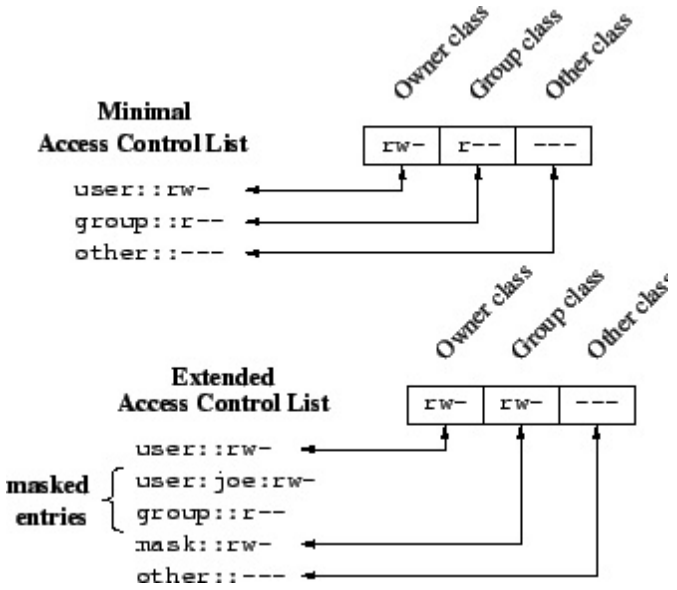

FIG. 1 – caption

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

On a dit que le permission de la classe groupe sont calculée comme le limite supérieur de tous les entrée 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 2**??**, les entrée 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 de une objet du système de fichier il y a une algorithme assez simple.[2]

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 patron POSIX règle non seulement sur les accès des objet du système, mais aussi sur le mécanisme de Héritage. Les ACL sont partage en deux type, les *access ACL* (que on a vu jusqu'á maintenant) et les *default ACL* qui comprendre les règles de héritage.

Quand on parle de l'héritage on parle de les droits qui sont attribue aux objet du système pendand le moment en que il sont crée. Il y une seul type de

[2] Il faut traduire cette algorithme lá :-)

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 une 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 Linux

Using ACL in Linux

The Linux getfacl and setfacl command line utilities do not strictly follow POSIX 1003.2c draft 17, which shows mostly in the way they handle default ACLs. See section 6.

```
-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

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
```

3 Implémentassions

ACLs sont fréquemment implémentent comment extensions du noyaux, 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 systeme de fichier"[1]. Plusieurs implémentassions de cette modelé sont possible. Par exemple, avec Solaris dans le système de fichier UFS[6] chaque *inode* peut être avoir an 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) implémentent les Attributs Étendu (*Extended Attributes (EAs)*) et les ACL sont implémentent avec ce mécanisme.

Le linux page de manuel[1] *attr(5)* contienne une explication plus 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.

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.

Il faut dire que la version plus limité des EA peut offrir plusieurs avantages : Il sont plus facile de implémettre, les opération sont atomic, l'interface est *stateless* et pour cette raison il n'y a aucun surcharge cause pour le travaille avec *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

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