

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

CS435 - Administration de Système

Dan Pham et Fabrício Nascimento

Octobre 2009

## Introduction

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

Ce mode traditionnel est assez simple et capable de adresser les problèmes plus fréquents. Par contre, il pose des limitations aux administrateurs de système, lesquels fréquemment doivent employer quelques configurations non évidentes afin d'être capable d'exprimer ces besoins. Par exemple les applications comme le serveur FTP Proftpd[4] ont ces exclusives façons de résoudre ces problèmes de droits pour accéder les objets du système de fichiers.

À cause de remédier ces limitations présentes les UNIX permettent d'employer les ACL.

Cette article présente une exposition sur les ACL POSIX, ces modes de fonctionnement, ces clés de succès et désavantages. Le texte est fortement basé en l'article de Andreas Gruenbacher[1] dont a été dans l'équipe qui a ajouté le support aux ACL dans le noyau Linux pour les systèmes de fichiers ext2 et ext3, lequel est le système de fichiers plus utilisé dans le monde UNIX.

## 1 Le POSIX 1003.1

Traditionnellement les systèmes qui implémentent les patrons POSIX avaient un système simple et puissant de permission, quand même, certains problèmes ont arrivés, et éventuellement les problèmes ont apparu.

Après savoir la nécessité de régler sur le domaine de sécurité et non seulement les ACL, un groupe a été formé pendant la définition de la famille de patrons POSIX 1003.1. Les premiers documents POSIX qui ont été considérés ces questions étaient les documents 1003.1e (*System Application Programming Interface*) et 1003.2c (*Shell and Utilities*), cependant, la première approximation de ce sujet était trop ambitieuse. Le groupe responsable pour la patronisation avait centré ces efforts dans une tâche 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 travail n'était pas près. De toute façon le dix-septième brouillon a été publié quand même[5].

Donc après cette année, les systèmes UNIX appelés "*trusted*" (Trusted Solaris, Trusted Irix, Trusted AIX) ont été développés avec quelques parts de la documentation 17. Ces systèmes ne sont pas complètement compatibles entre eux.

Aujourd'hui la plupart des systèmes UNIX et UNIX-like supportent ACL. Ces implémentations sont usuellement compatibles avec le document 17. Le projet TrustedBSD aussi avait ajouté les ACL sur les systèmes BSD. Les ACL et les MAC FreeBSD-RELEASE ont 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 (read), écrire (write) et exécution (execute). 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 tous 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 permet 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 groupes 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. <sup>1</sup>.

<sup>1</sup>Fabricio : Je ne comprend pas cette affirmation, je laisse ici le teste original : These named group and named user entries are assigned to the group class, which already contains

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

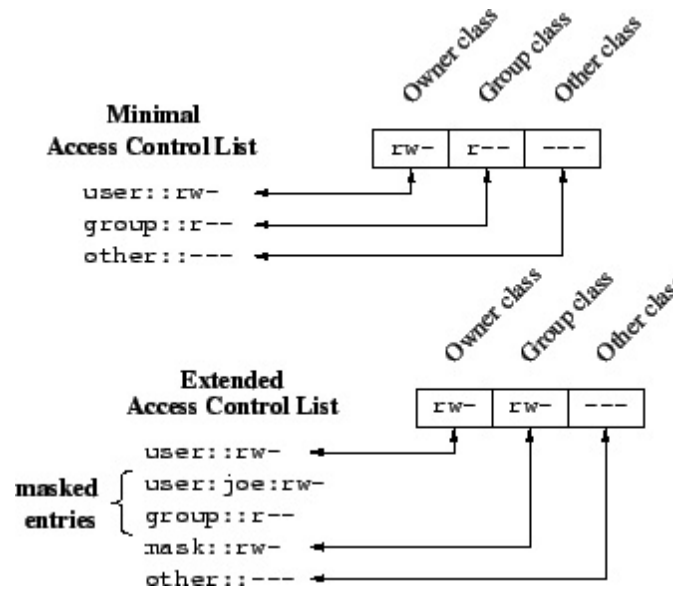


FIG. 1 – caption

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

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.

ce modification.

## Algorithme de vérification

Pour vérifier les droits d'accès de 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 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'à 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 pendant le moment en que il sont crée. Il y une seul type de 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*.

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 ACL par défaut, 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 Implementation

### Conclusion

### Références

- [1] Andreas Gruenbacher, *POSIX Access 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/>