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.

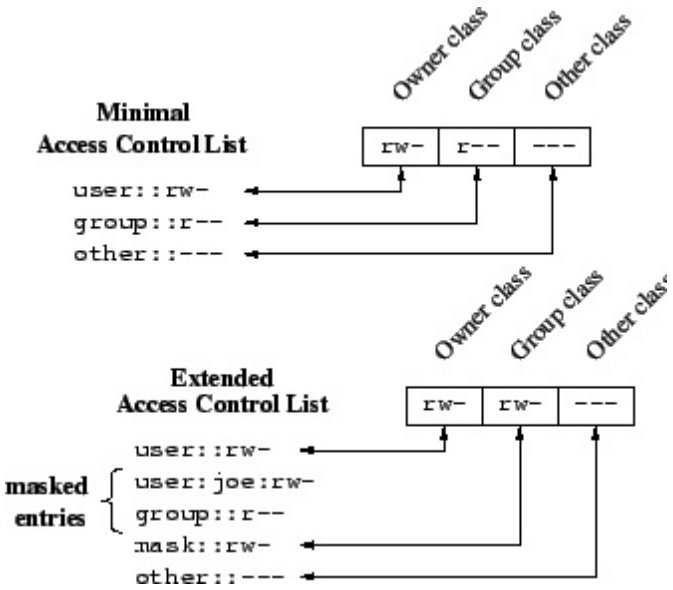

FIG. 1 – caption

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 l'identifiant de l'utilisateur du processus est le propriétaire **then**
  l'entrée du propriétaire détermine l'accès
else if l'identifiant d'utilisateur du processus correspond à une entrée d'utilisateur nommé dans la table des ACL **then**
  l'entrée détermine l'accès
else if un des identifiants de groupe du processus correspond au groupe propriétaire et l'entrée contient les permissions requises **then**
  l'entrée détermine l'accès
else if un des identifiants de groupes correspond à un des groupe nommés et cette entrée contient les permissions requises **then**
  l'entrée détermine l'accès
else if Un des identifiants de groupe du processus correspond au groupe propriétaire ou correspond à un des groupes nommés mais ni le groupe propriétaire ni aucun des groupes nommé contient les permission requises. **then**
  ceci détermine que l'accès est interdit
else
  l'entrée other détermine l'accès
end if
if l'entrée qui détermine l'accès est l'entre du propriétaire ou l'entrée other qui contient les permissions requises **then**
  l'accès est autorisé
else if l'entrée correspondante est l'utilisateur nom, ou le groupe propriétaire ou le groupe nommé et cette entrée contient les permissions requises et l'entrée masque contient aussi les permission. (ou il n'y a pas d'entrée masque) **then**
  l'accès est autorisé
else
  l'accès n'est pas autorisé
end if

Héritage mécanisme

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

Quand on parle de l'héritage, on parle des droits qui sont attribués aux objets du systèmes de fichiers au moment où ils sont crées. Il y a un seul type d'objet qui peut être associe avec les *default ACL*; les répertoires. Il faut dire

que il n'y a pas de sens pour les *default ACL* pour les fichiers car on ne peut pas créer un fichier à l'intérieur d'un fichier. Aussi les *Default ACL* et les *access ACL* sont complètement indépendant.

Si un répertoire est crée dans une autre, si le première répertoire a *default ACL*, avec le mécanisme d'héritage, le deuxième aura le même ACL que le premier (*default* et *access*). Les objets qui ne sont pas des répertoires, devons hériter les *default ACL* seulement.

Chaque *system call* qui crée les objets du système de fichier a un *mode parameter*. Ce paramètre peut contenir neuf octets de permission pour chaque classe (propriétaire, groupe et les autres). Les permissions de chaque objet créée sont l'intersection des permissions définies pour les *default ACL* et le *mode parameter*.

Le système traditionnel a une commande pour désigner les modes de permissions par défaut pour les nouveaux fichiers et répertoires : le commande *umask*. Quand il n'y a aucune *default ACL*, la permission effective est déterminé par le *mode parameter* moins les permissions configurés 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).

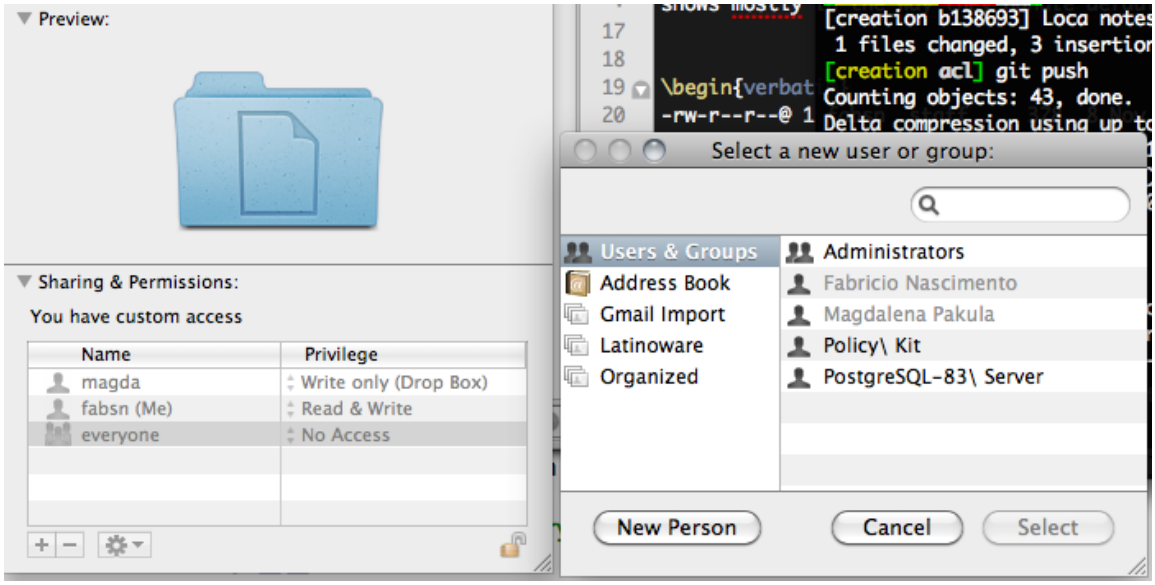

Fig. 2 – Mac OSX Snow Leopard ACL Interface

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 Implèmentassions

Les ACLs sont frèquemment implèmentèes comme extensions du noyau, c'est à dire des modules un systéme LINUX. L'objectif de cette section est d'expliquer de maniére globale l'implèmentions des ACL.

————————Le discussion doit lancer la base pour les èvaluations de performance et les problémes dans les sessions prochaines.————-

"Les ACLs sont des informations de taille variable qui sont associèes avec les objets du systéme de fichier"[1]. Plusieurs implèmentations des ACLs sont possibles. Par exemple, avec Solaris, dans le systéme de fichier UFS[6] chaque *inode* peut avoir une ACL. S'l en a une, il doit avoir l'information *i_shadow*, un pointeur pour un *shadow inode*. Les *shadow inode* sont comment fichiers règuliérs d'utilisateurs. Différent fichiers avec les mÍmes ACL peut avoir pointeurs pour le mÍme *shadow inodes*. Les information des ACL sont garde dans les bloc de donnèes de chaque *shadow inodes*.

La capacitè d'associer des informations avec des fichiers est utilisè dans plusieurs fonctions du systéme de exploitation. De ce fait, la plupart des systémes *UNIX-like* (de type Linux) on trouve les Attributs ... tendus (*Extended Attributes (EAs)*). Les ACL sont implèmentèes avec ce mècanisme.

La manpage [1] *attr(5)* contient des explications prècises sur les EAs dans Linux, ————au notre but, suffit dire que comme les variables des processus, les EAs sont des paires (nom, valeur) associèes de maniére persistantes avec les objets du systéme de fichiers et que les appels Linux, dans le espace de utilisateur, sont employè pour opèrer sur les information de ces paires dans le espace de adresse du noyaux. Aussi pour l'implèmentation de cette infrastructure dans les systéme FreeBSD il faut voir le article de Robert Watson[7]. ———————————-Cette article content aussi une comparaison de plusieurs implèmentation de ces systéme.

Dans le monde linux, ajouter le support aux ACL avec une version limitèe des EA offre plusieurs avantages : un grande facilitè d'implèmentation, ———————-opèration atomique et interface *stateless* que laisse aucun surcharge à cause de les *file handlers*. On verra aprés dans la section de performance, que l'efficience est assez importante pour Ítre oublier quand on parle de les donnès frèquentent accés comme les ACL.————————

3.1 Les EAs et les systémes de fichiers

Dans le monde UNIX, chaque systéme de fichier a une différent implèmentation pour les EAs. On peut penser qu'une solution partagèe pour l'ensemble des systémes pourait Ítre plus efficace. Par exemple, si on prend une solution simple où chaque objet du systéme de fichiers a les EAs, un rèpertoire avec un fichier qui a le clès EA comment le nom et le contenu comment le valeur. Cette implèmentation consommerait beaucoup de espace, ètant donnè que les blocks du systéme de fichier seront gaspillès pour conserver petit morceaux de donnè, aussi ce solution perdrait les temps pour chercher ces information a chaque accés

de fichier. Aux frais de ces problémes chaque systéme tire profit de ces qualitès pour ajouter le supporte aux EAs.

Ext (2,3 et 4)

Les ACL dans Ext suivent le principe linux : "La solution la plus simple qui marche" et pour cette raison subviennent quelques limitations. D'autres solutions existent, par contre, elle sont difficiles à ajouter au noyau de maniére satisfaisante[9].

La solution actuelle ajoute aux *i_node* une entrèe qui s'appelle *i_file_acl*. Cette entrèe, si différent de 0, est une ponteur sur un bloc d'EAs. Ce bloc d'EAs a les informations de nom et valeur de tous les ACL du fichier indiquè pour cet *i_node*.

Le mècanisme a aussi une optimisation. Deux fichiers avec le mÍme ensemble de ACL point vers le mÍme bloc d'EAs. Le systéme garde une *hash map* avec les *checksum* des blocs d'EAs et leurs adresse. Chaque block a aussi un compteur de rèfèrence, comme les liens *hard*. Ce mècanisme dètermine aussi que ce compteur là ne peut pas avoir plus que 1024 rèfèrences. Il s'agit d'une mesure de sècuritè en cas de perte des donnèes.

Aussi une limitation est imposèe : toutes les donnèes des EAs d'un fichier doivent occuper un bloc d'EAs ayant une taille de 1, 2 ou 4 KBs.

JFS

Dans JFS, les EAs sont ajoutèes dans une liste consècutive de blocs contigus (un extent). Cela veut dire que chaque paire (nom,valeur) est gardèe en sèquence et que chaque valeur de la paire ne peut pas Ítre plus grande que 64kb. Si les EAs sont assez petites, elles pourrons Ítre gardèes dans le mÍme lieu que les informations du fichier. De ce faÁon, il n'y a pas les limitations d'ext3.

XFS

XFS est sans aucun dout le systéme de fichiers le plus simple pour implèmenter les EA. Les paires d'EA de petite tailles sont stockèes directement dans l'inode, celles de taille moyenne sont stockèes dans les blocs feuilles de l'arbre binaire et pour celle de grande taille, dans un arbre binaire complet.————manque—— – XFS peut configurer la taille de sa table d'inodes. La taille minimale est de 256 octet et la taille maximale peut aller jusqu'à la moitiè des blocs du systéme de fichier. Dans le cas où on a une table de taille minimale, celle-ci n'est pas assez grande pour accueillir les ACLs. On doit alors les stocker de maniére externe —-au systéme de fichier—-. Si on augmente la taille de la table les ACL pourront y Ítre stockèes. Les ACL ètant trés souvent interrogèes par le systéme, cela augmente les performances en terme de temps d'accés au dètriment de l'espace disque qu'elles consomment. XFS n'a pas de mècanisme de partage des attributs. La taille individuelle des attributs est limitèe à 64Kb.

ReiserFS

ReiserFS support le *tail merging* qui permet à plusieurs fichiers de partager le mÍme bloc pour stocker leurs donnèes. Cela rend le systéme trés efficace pour s'il on posséde de nombreux fichiers de petite taille. De l'autre cÙtè cette technique consomme beaucoup de ressources CPU.

Comme le ReiserFs peut facilement manipuler des petits fichiers, les EA peuvent Ítre implèmentèes sous forme de petits fichiers. Pour chaque fichiers qui a un EA, un dossier spècial (qui est souvent cachè) est crèè avec un nom dèrivè de son inode. Dans ce dossier chaque EA est stockèe dans un fichier sèparè qui a pour nom le nom de l'attribut. Le contenu de chaque fichier est la valeur de l'attribut.

Le systéme ReiserFS n'implèmente pas le partage d'attributs mais la crèation d'une extension pour le gèrer est possible. La taille individuelle des attributs est limitèe à 64Kb.

————————-Sharing could even be implemented on a per-attribute bases, so the result would be a highly efficient and flexible solution. ————The size of individual attributes is limited to 64 KiB.

HGFS+

Samba

MicrosoftWindows supports ACLs on its NTFS file system, and in its Common Internet File System (CIFS) protocol [20], which formerly has been known as the Server Message Block (SMB) protocol. CIFS is used to offer file and print services over a network. Samba is an Open Source implementation of CIFS. It is used to offer UNIX file and print services to Windows users. Samba allows POSIX ACLs to be manipulated from Windows. This feature adds a new quality of interoperability between UNIX and Windows. The ACL model of Windows differs from the POSIX ACL model in a number of ways, so it is not possible to offer entirely seamless integration. The most significant differences between these two kinds of ACLs are : Windows ACLs support over ten different permissions for each entry in an ACL, including things such as append and delete, change permissions, take ownership, and change ownership. Current implementations of POSIX.1 ACLs only support read, write, and execute permissions. In the POSIX permission check algorithm, the most significant ACL entry defines the permissions a process is granted, so more detailed permissions are constructed by adding more closely matching ACL entries when needed. In the Windows ACL model, permissions are cumulative, so permissions that would otherwise be granted can only be restricted by DENY ACL entries. POSIX ACLs do not support ACL entries that deny permissions. A user can be denied permissions be creating an ACL entry that specifically matches the user. Windows ACLs have had an inheritance model that was similar to the POSIX ACL model. Since Windows 2000, Microsoft uses a dynamic inheritance model that allows permissions to propagate down the directory hierarchy when permissions of parent directories are modified. POSIX ACLs are inherited at file create time only. In

the POSIX ACL model, access and default ACLs are orthogonal concepts. In the Windows ACL model, several different flags in each ACL entry control when and how this entry is inherited by container and non-container objects. Windows ACLs have different concepts of how permissions are defined for the file owner and owning group. The owning group concept has only been added with Windows 2000. This leads to different results if file ownership changes. POSIX ACLs have entries for the owner and the owning group both in the access ACL and in the default ACL. At the time of checking access to an object, these entries are associated with the current owner and the owning group of that object. Windows ACLs support two pseudo groups called Creator Owner and Creator Group that serve a similar purpose for inheritable permissions, but do not allow these pseudo groups for entries that define access. When an object inherits permissions, those abstract entries are converted to entries for a specific user and group. Despite the semantic mismatch between these two ACL systems, POSIX ACLs are presented in the Windows ACL editor dialog box so that they resemble nativeWindows ACLs pretty closely. Occasional users are unlikely to realize the differences. Experienced administrators will nevertheless be able to detect a few differences. The mapping between POSIX and Windows ACLs described here is found in this form in the SuSE and the UnitedLinux products, while the official version of Samba has not yet integrated all the improvements recently made : The permissions in the POSIX access ACL are mapped to Windows access permissions. The permissions in the POSIX default ACL are mapped to Windows inheritable permissions. Minimal POSIX ACLs consist of three ACL entries defining the permissions for the owner, owning group, and others. These entries are required. Windows ACLs may contain any number of entries including zero. If one of the POSIX ACL entries contains no permissions and omitting the entry does not result in a loss of information, the entry is hidden fromWindows clients. If aWindows client sets an ACL in which required entries are missing, the permissions of that entry are cleared in the corresponding POSIX ACL. The mask entry in POSIX ACLs has no correspondence in Windows ACLs. If permissions in a POSIX ACL are ineffective because they are masked and such an ACL is modified via CIFS, those masked permissions are removed from the ACL. Because Windows ACLs only support the Creator Owner and Creator Group pseudo groups for inheritable permissions, owner and owning group entries in a default ACL are mapped to those pseudo groups. For access ACLs, these entries are Submitted for publication at the USENIX Annual Technical Conference, San Antonio, Texas, June 2003 11 mapped to named entries for the current owner and the current owning group (e.g., the POSIX ACL entry ìu : :rwî of a file owned by Joe is treated as ìu :joe :rwî). If an access ACL contains named ACL entries for the owner or owning group (e.g., if one of Joeís files also has a ìu :joe :...î entry), the permissions defined in such entries are not effective unless file ownership changes, so such named entries are ignored. When an ACL is set by Samba that contains Creator Owner or Creator Group entries, these entries are given precedence over named entries for the current owner and owning group, respectively. POSIX access ACL and default ACL entries that define the same permissions are mapped to a Windows ACL entry

that is flagged as defining both access and inheritable permissions.

NFS

The NFS protocol performs client-side caching to improve efficiency. In version 2 of the protocol, decisions as to who gets read access to locally cached data are performed on the client. These decisions are made under the assumption that the file mode permissions bits and the IDs of the owner and owning group are sufficient to do that. This assumption is obviously wrong if an extended permission scheme like POSIX ACLs is used on the server. Because NFSv2 clients perform some access decisions locally, they will incorrectly grant read access to file and directory contents cached on the client to users who are a member in the owning group in two cases. First, if the group class permissions include read access, but the owning group does not have read access. Second, if the owning group does have read access, but a named user entry for that user exists that does not allow read access. Both situations are rare. Workarounds exist that reduce the permissions on the server side so that clients only see a safe subset of the real permissions [7, 10]. No anomalies exist for users who are not a member in the owning group. There are two ways to solve this problem. One is to extend the access check algorithm used on the client. The other is to delegate access decisions to the server and possibly cache those decisions for a defined period of time on the client. The first solution would probably scale better to a high number of readers on the client side, as long as the server and all clients can agree on the access check algorithms use. Unfortunately, this apapproach falls apart as soon as servers implement different permission schemes.

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/