Uso de Listas de Control de Acceso (ACLs) en Linux

Ignacio Arenaza Nuño

Proyecto escomposlinux.org (iarenaza@escomposlinux.org)

El presente documento muestra las posibilidades del modelo de permisos basado en listas de control de acceso o ACLs. Una vez discutido el modelo, se muestra como implementar dicho modelo de seguridad en un sistema GNU/Linux y como se pueden aprovechar las posibilidades que este modelo ofrece, tanto en el servicio básico de acceso a ficheros como por medio de un sistema de compartición de ficheros con SAMBA.

1. Listas de Control de Acceso

1.1. Introducción

Una de las características que se echan en falta en los sistemas Linux actuales, especialmente en entornos corporativos, es la posibilidad de especificar los permisos de acceso a los ficheros con mayor granularidad. Los sistemas Linux siguen el modelo de permisos Unix tradicional segementando el tipo de acceso en tres categorías:

- El propietario del fichero (User)
- El grupo al que pertenece el fichero (Group)
- El resto de usuarios del sistema que no están en ninguna de las dos categorías anteriores (Other).

también conocido como modelo UGO (User, Group, Other).

Sin embargo, estas tres categorías se revelan insuficientes en una gran cantidad de situaciones, donde desearíamos poder especificar permisos diferenciados para varios usuarios o grupos determinados.

Aquí es donde entran en juego los permisos basados en Listas de Control de Acceso, más conocidos como ACLs. En este sistema de permisos los ficheros no tienen un juego fijo de permisos (como en el modelo tradicional, que tiene tres permisos y sólo tres), sino que los permisos del fichero son en realidad una lista de Entradas de Control de Acceso (o ACEs). Cada una de estas ACEs contiene un par (usuario/grupo, permiso) que indica un tipo de acceso determinado para un usuario o grupo, y el conjunto de todas ellas forman la ACL que marca el tipo de acceso permitido en un fichero.

Este sistema es el utilizado entre otros, por los sistemas de ficheros NTFS (de Windows NT y sucesores), el sistema UFS de Solaris y el sistema HFS de HP-UX.

1.2. ACLs en Linux

Hay un dato que se suele desconocer sin embargo y es que el sistema de ficheros ext2, desde su diseño original, previó la inclusión de este tipo de sistemas de control de acceso y estaban incluídos los enganches (*hooks*) necesarios para su implementación posterior, de forma 100% transparente y compatible hacia atrás.

Pues bien, desde hace varios años existen varios proyectos para incorporar los sistemas basados en ACL en los diferentes sistemas de ficheros soportados por Linux, y especialmente en ext2 (por ser el tipo de sistema de ficheros más extendido hasta el momento en los sistemas Linux).

Uno de ellos es el proyecto RSBAC (http://www.rsbac.de/), cuyos objetivos son mucho más ambiciosos (realmente mucho más, su objetivo es conseguir un sistema Linux con un nivel de seguridad equivalente al nivel B1 del antiguo Libro Naranja -TCSEC-), pero que incorpora también una implementación 100% operativa de ACLs para ext2.

Otro de ellos es el proyecto Linux Extended Attributes and Access Control Lists (http://acl.bestbits.at/), que originalmente tenía como objetivo incorporar el sistema de ACLs al sistema de ficheros ext2 (y posteriormente ext3 cuando este apareció). Posteriormente, al ser el sistema de ACLs elegido por el equipo de Samba (http://www.samba.org/) para su implementación de ACLs sobre ext2 (para poder ofertar recursos compartidos via SMB con ACLs al igual que los sistemas Windows NT y posteriores) ha sido el candidato oficial de ACLs en ext2 para su inclusión definitiva en el kernel. De hecho, desde la versión 2.5.23 forma parte del kernel estándar.

Para ello se ha hecho un esfuerzo de coordinación y estandarización bastante grande con los desarrolladores de otros sistemas de ficheros como XFS y JFS (principalmente, aunque no sólo con estos) que también soportaban ACLs desde su origen. La idea era ofertar una capa abstracta común en el VFS (*Virtual File System*) de forma que las aplicaciones y el resto del sistema operativo trabajasen por igual, y con la misma API, con cualquiera de los sistemas de ficheros que soportasen ACLs (ext2/ext3, XFS, JFS, ReiserFS, etc.).

El resultado: ya están disponibles los sistemas de ficheros con ACLs en el núcleo estándar, en su versión de desarrollo. Sin embargo, no es necesario esperar hasta la estabilización del actual núcleo de desarrollo para disfrutar de las ventajas de las ACLs. Todos los sistemas de ficheros mencionados arriba están disponibles bien como parte del kernel estándar, bien como parches, para las versiones estables del núcleo. Y son parches con calidad de producción, con lo cual son perfectamente utilizables en entornos cuya estabilidad sea una condición indispensable.

En el caso del sistema de ficheros ext2/ext3, que son el objetivo del proyecto Linux Extended Attributes and Access Control Lists, las ACLs son incluso transparentes para aquellos núcleos que no lleven

incorporados los parches necesarios, de forma que si accidentalmente se arranca el sistema con un núcleo sin soporte para ACLs no ocurre absolutamente nada, salvo obviamente que no disponemos de las características avanzadas de las ACLs y sólo tenemos a nuestra disposición el modelo de permisos tradicional.

A condición de que tengamos un ejecutable de e2fsck que soporte ACLs, incluso si el núcleo no lo soporta, podemos ejecutar e2fsck sobre el sistema de ficheros de forma segura. En caso de tener un e2fsck sin soporte de ACLs, el único problema que tendremos en este caso es la pérdida de las ACLs, pero nunca la pérdida de datos.

Las versiones de e2fsprogs (el paquete donde se incluyen las utilidades de ext2) a partir de la version 1.28 ya incorporan soporte de serie para Atributos Extendidos (Extended Attributes o EAs), que es la característica del sistema de ficheros necesaria para poder implementar las ACLs. En el sitio del propio proyecto se pueden encontrar parches para algunas versiones anteriores de e2fsprogs (http://acl.bestbits.at/download.html#E2fsprogs), en caso de ser necesario.

1.3. Cómo incorporar ACLs a nuestro sistema Linux

¿Qué debemos hacer por tanto para disfrutar de ACLs en nuestros sistemas de ficheros ext2/ext3 ya mismo? Lo que sigue es un resumen de las instrucciones (http://acl.bestbits.at/steps.html) dadas en el propio sitio del proyecto, donde se indica paso a paso todo el proceso a seguir. Voy a presuponer en este caso que nuestro sistema no dispone de sistemas de ficheros con EAs y ACLs, con lo cual me voy a saltar los pasos de copia de respaldo de las ACLs actuales (y la limpieza del sistema).

1. Lo primero es obtener una copia de e2fsprogs que soporte EAs y ACLs. La version 1.28 o posterior servirá. En caso contrario, hay que obtener los parches mencionados arriba y las fuentes de nuestra version actual de e2fsprogs (seguramente nuestra distribución tendrá disponibles las fuentes en su repositorio habitual) y construir de nuevo los ejecutables e instalarlos en el sistema. Alternativamente podemos optar por instalar directamente una versión 1.28 o posterior directamente, sin parchear la actualmente usada en nuestra distribución.
2. A continuación debemos compilar un kernel con soporte para EAs y ACLs. Para ello debemos descargar los parches correspondientes a nuestro núcleo (http://acl.bestbits.at/download.html#Kernel) y aplicarlos de la siguiente forma (el directorio linux/ indica la ubicación donde tenemos desempaquetadas las fuentes del núcleo):

   ```
   $ cd linux/
   $ zcat ../linux-a.b.c-xattr-x.y.z.diff.gz | patch -p1
   $ zcat ../linux-a.b.c-acl-x.y.z.diff.gz | patch -p1
   ```

   Una vez parcheado es necesario incluir en la compilación las siguientes opciones al menos (disponibles en la categoria File Systems):

   ```
   CONFIG_FS_POSIX_ACL=y       (POSIX Access Control Lists)
   CONFIG_EXT3_FS_XATTR=y      (Ext3 extended attributes)
   CONFIG_EXT3_FS_POSIX_ACL=y  (Ext3 POSIX Access Control Lists)
   CONFIG_EXT2_FS_XATTR=y      (Ext2 extended attributes)
   ```

```
CONFIG_EXT2_FS_POSIX_ACL=y  (Ext2 POSIX Access Control Lists)
```

Los nombres pueden variar ligeramente de una versión del kernel a otra. Las opciones xxx_XATTR son para activar los Atributos Extendidos, y las opciones xxx_POSIX_ACL para activar las ACLs. Los valores xxx_EXT2_xxx son para sistemas de ficheros ext2 y los valores xxx_EXT3_xxx para ext3.

Existen otras dos opciones más (al menos en el kernel 2.4.19, que es el que estoy usando para escribir esto):

I. Ext2 extended attribute block sharing (IG_EXT2_FS_XATTR_SHARING): que permite el uso de un mismo bloque común para almacenar los atributos extendidos de varios nodos-i, en el caso de que dichos atributos sean indénticos en todos los nodos-i.

II. Ext2 extended user attributes (CONFIG_EXT2_FS_XATTR_USER): que permite a los procesos de usuario almacenar atributos extendidos adicionales en los nodos-i. Por ejemplo para almacenar cualquier tipo de información adicional que dichos procesos deseen, como el juego de caracteres usuado en el fichero (o cualquier otra cosa que se nos ocurra).

Por supuesto, los dos valores anteriores existen de idéntica forma para el sistema de ficherox ext3.

3. El siguiente paso es construir las utilidades que sirven para gestionar los AEs y las ACLs de los ficheros: getfattr, setfattr, getfacl, setfacl, etc. Tenemos que descargar el paquete attr (http://acl.bestbits.at/download.html#Attr) que es necesario para poder construir despues el paquete acl (http://acl.bestbits.at/download.html#Acl), que incluye las utilidades de gestión de las ACLs en sí (el paquete attr incluye además algunas utilidades de gestión de atributos extendidos).

   Es conveniente revisar primero si existen versiones ya empaquetas para nuestra distribución de Linux (tanto Debian GNU/Linux como RedHat, entre otras, las tienen), y que sea una versión de las mismas que sea compatible con la revisión del parche aplicada al núcleo que acabamos de construir

   En caso de no tenerlas en nuestra distribución, no ser compatibles o simplemente querer tener la última versión disponible compilada por nosotros mismos, estos son los pasos a seguir (necesitaremos el entorno de desarrollo autoconf y todas la utilidades de compilación/desarrollo habituales, más algún retoque manual):

   I. Desempaquetar las utilidades de gestión de Atributos Extendidos (attr-2.0.10.src.tar.gz en el momento de escribir esto):

   ```
   tar xzvf attr-2.0.10.src.tar.gz
   cd attr-2.0.10
   ```

   A continuación tenemos que compilar las herramientas en sí. Si nuestra distribución es RedHat o Debian GNU/Linux, las utilidades vienen con los ficheros de especificación y control

necesarios para crear paquetes nativos (ver el fichero doc/INSTALL).

En caso contrario, debemos compilar todo desde cero con las siguientes instrucciones (en el directorio raíz de los ficheros extraídos):

```
autoconf
./configure
make
su root
make install install-lib install-dev
```

Este último método dejará todos los ejecutables más las bibliotecas de funciones de manejo de AEs bajo /usr/local. En general el fichero doc/INSTALL da las indicaciones necesarias para obtener los ejecutables en cualquiera de los casos (incluídas algunas indicaciones para deshabilitar el código de depuración, etc).

II. Desempaquetar las utilidades de gestión de ACLs (acl-2.0.18.src.tar.gz en el momento de escribir esto):

```
tar xzvf acl-2.0.18.src.tar.gz
cd acl-2.0.18
```

A continuación tenemos que compilar las herramientas en sí. Como en el caso anterior, si nuestra distribución tiene ya compiladas estas utilidades y con compatibles con la versión de los parches incluídos en el núcleo, lo más sencillo es usar los paquetes nativos de la distribución.

En caso contrario, debemos compilar todo desde cero con las siguientes instrucciones (en el directorio raíz de los ficheros extraídos):

```
autoconf
./configure
make
su root
make install install-lib install-dev
```

Este último método dejará todos los ejecutables más las bibliotecas de funciones de manejo de ACLs bajo /usr/local. Como en el caso anterior, el fichero doc/INSTALL da las indicaciones necesarias para obtener los ejecutables en cualquiera de los casos.

III. Por último debemos obtener una versión del paquete fileutils parcheado para soportar EAs y ACLs. De lo contrario no podremos copiar los ficheros con sus AEs y ACLs, sacar en los listados la indicación de que hay acls adicionales a los permisos tradicionales (ya que se siguen manteniendo los permisos tradicionales), etc.

En el propio sitio del proyecto se puede encontrar el parche (http://acl.bestbits.at/download.html#Fileutils) para algunas versiones del paquete fileutils. En general esta es la parte más complicada del proceso, al ser un paquete completamente externo al sistema de ficheros y las utilidades ext2/ext3. Los problemas se presentarán si la versión de fileutils que vamos a utilizar no se corresponden exactamente a las indicadas en los parches, ya que tendremos que integrar parte de los cambios a mano, y eso siempre es más complicado.

Una vez compilado e instalado todo lo mencionado en los puntos anteriores ya tenemos todos los elementos necesarios para poder disfrutar de los EAs y las ACLs en nuestros sistemas de ficheros ext2/ext3. Sólo nos queda un último paso, indicar al núcleo que en un determinado sistema de ficheros deseamos usar ACLs (y EAs).

Para ello debemos editar el fichero /etc/fstab y añadir una opción adicional a la de aquellos sistemas de ficheros a los que queremos activar las ACLs : **acl**. También podemos añadir una opción para indicar explícitamente que no queremos usar las ACLs en un sistema de ficheros, aun si dicho sistema de ficheros contiene ACLs: **noacl**.

Existe un juego de opciones adicional para activar o desactivar el uso de los AEs de usuario (si hemos optado por compilarlos en nuestro núcleo): **user_xattr** y **nouser_xattr**. Por cierto, que los valores por defecto para todos los sistemas de ficheros ext2/ext3 en caso de no especificar nada son **noacl** y **nouser_xattr**

Una vez hecho lo anterior, sólo nos resta arrancar con el nuevo núcleo que acabamos de compilar y *¡voilá!*, ya tenemos nuestras ACLs disponibles y listas para usar.

2. Uso de las ACLs

Una vez que hemos compilado el kernel con soporte para ACLs y que hemos compilado las herramientas necesarias para poder trabajar con ellas (además de parchear aquellas utilidades de gestión de ficheros y sistemas de ficheros para que reconozcan las ACLs y las respeten y sepan interpretarlas), podemos ya dedicarnos a la gestión de las propias ACLs en si.

Para ello disponemos de dos utilidades principalmente:

- getfacl: que nos permite consultar las ACLs de un fichero dado.
- setfacl: que nos permite modificar las ACLs de un fichero dado.

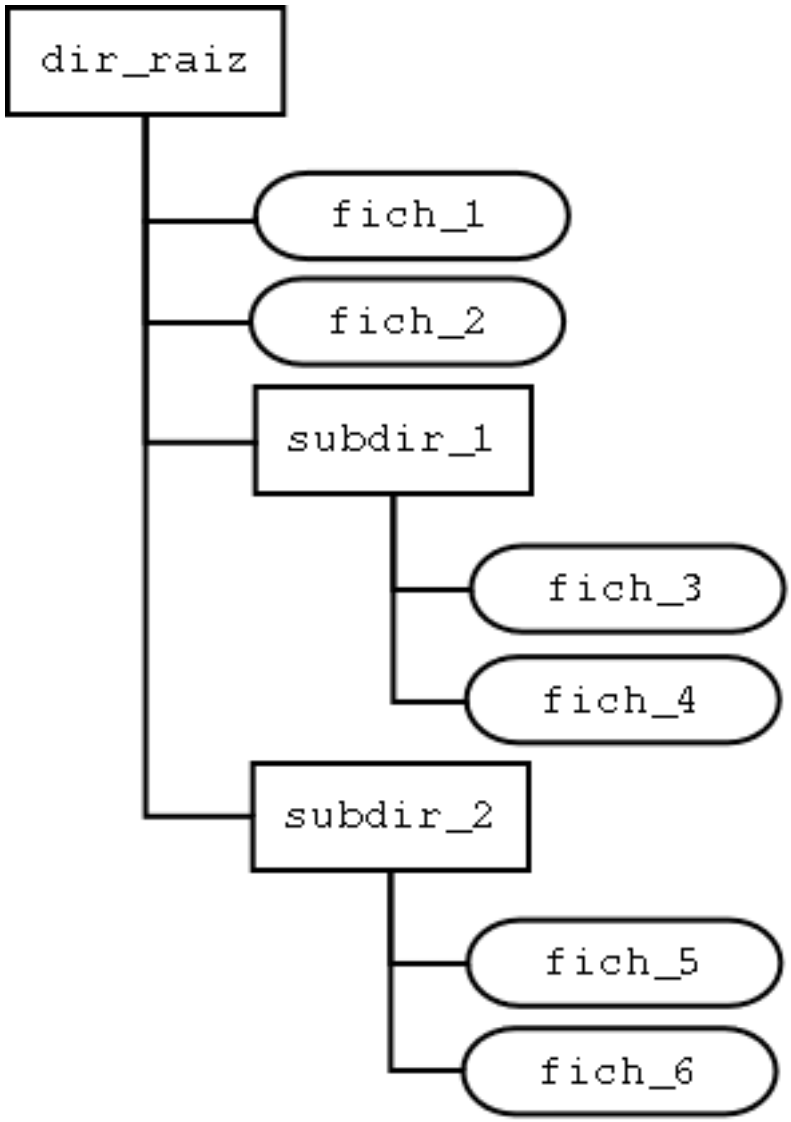
Ambas utilidades se hayan perfectamente documentadas en sus respectivas páginas del manual (setfacl(1) (http://acl.bestbits.at/cgi-man/setfacl.1) y getfacl(1) (http://acl.bestbits.at/cgi-man/getfacl.1)),

y podemos encontrar una pequeña introducción en la página del manual de acl(5) (http://acl.bestbits.at/cgi-man/acl.5), vamos a ver aquí algunos ejemplos sencillos que ilustrarán el uso básico de estas utilidades.

2.1. Creación de una ACL básica.

Existen casos en los cuales el tradicional juego de pemisos UGO (User, Group, Others) no es suficiente. Casos en los que desearíamos que más de un usuario o más de un grupo pudiese tener acceso a un fichero, pero con permisos diferenciados. Con el modelo UGO esto no es posible, puesto que sólo tenemos sitio para los permisos de un único Usuario o un único Grupo. Todo lo demás cae en el Other (el resto). Con las ACLs esto es sencillo.

Vamos a presuponer que tenemos un directorio que contiene una serie de ficheros y dos subdirectorios (a su vez con ficheros) como se indica en la siguiente figura:



Tenemos asimismo los siguiente tipos de usuarios diferenciados:

1. Los usuarios del grupo de sistemas de información, que deben tener acceso completo a todos los directorios y ficheros, para su mantenimiento. Llamaremos a este grupo *sistemas*.
2. Los usuarios del grupo de desarrollo (llamaremos a este grupo *desarrollo*), que deben tener acceso de lectura y escritura (y ejecución en su caso) en el primer subdirectorio y todos sus ficheros y subdirectorios. En este directorio es donde se desarrollan las nuevas versiones del software que usa el tercer grupo.

   Asi mismo debe tener acceso de lectura/escritura al segundo de los subdirectorios para implantar las nuevas versiones estables del software.

3. Los usuarios de explotación. Este grupo debe tener acceso en modo lectura (y eventualmente ejecucion) sobre los ficheros de la aplicacion. Llamaremos a este grupo *explotacion*.
4. El resto de usuarios del sistema no deben tener ningún tipo de acceso a ninguno de los ficheros o subdirectorios.

Con las condiciones anteriores es imposible usar el modelo de permisos tradicional UGO, puesto que tenemos tres grupos de usuarios diferentes, sin contar el resto de usuarios del sistema. Eso significa que con un sólo grupo de usuarios con permisos asignados por directorio o fichero no es posible acomodar los permisos para los tres grupos.

Utilizando ACLs es fácil resolver el problema, puesto que podemos asignar un juego de permisos diferente para cada grupo de usuarios. Para ello debemos crear las ACLs necesarias para cada grupo de usuarios y directorio o fichero.

1. Tenemos que dar permisos completos al grupo de sistemas sobre todos los ficheros y directorios implicados. Para ello usamos setfacl de la siguiente forma:

```
setfacl -b -k -R dir_raiz
setfacl -R -m g:sistemas:rw
```

   Vayamos por partes con la sintaxis de setfacl:

   - En el primer caso usamos la opcion -b para borrar la ACL que ya pudiera tener el directorio raíz. Usamos también la opción -k para borrar la ACL por defecto que pudiera tener el directorio raíz (más sobre esto después), y por último usamos la opción -R para aplicar los cambios de forma recursiva a todo lo que cuelga del directorio raíz. Con esto conseguimos tener todo limpio y listo para empezar.

     Teóricamente la primere vez no hace falta limpiar la ACL puesto que aún no hemos asignado ninguna ACE, pero de paso aprovechamos para mostrar como se hace :)

   - En el segundo caso indicamos de nuevo que queremos aplicar de forma recursiva los cambios (opción -R) pero esta vez le decimos que queremos modificar (-m) la ACL del objeto en cuestión. En este caso es necesario además indicar el valor de la ACE que deseamos añadir o modificar. *setfacl* distingue entre asignar una ACL (opción -s) en la cual se eliminan las ACEs existentes y se añade la ACE especificada en la orden, y modificar una ACL (opción -m) en la cual podemos modificar o añadir una ACE.

     En este caso queremos añadir una nueva ACE. Para ello debemos escribimos la ACE que nos interesa:

     `g:sistemas:rw`

Todas las ACEs tienen tres componentes separadas por ':' (en el caso de las operaciones de borrado de ACEs el tercer componente es opcional). El primero de los componentes indica si se trata de un ACE de usuario (valor u) o de grupo (valor g). Incluso es posible asignar ACEs al grupo de usuarios *resto* (other), pero esto no suele ser habitual, así que omitiremos la sintaxis (se pueden encontrar más detalles en la página del manual de setfacl(1)).

El segundo de los componentes es el nombre de usuario o grupo al que se le aplica la ACE. Se puede dar el nombre simbólico o el valor numérico del uid o gid correspondiente. El tercer componente es el valor del permiso asociado a esta ACE, y puede ser una combinación cualquiera de las letras r, w, x y -, o un valor numérico octal (como en chmod).

Por tanto, en nuestro caso tenemos que es una ACE de grupo (g), que se aplica al grupo de sistemas y que le estamo dando los permisos de lectura y escritura (rw).

2. A continuacion tenemos que dar permisos al grupo de desarrollo tanto en el directorio subdir_1 y todo su contenido, como en el directorio subdir_2 y todo su contenido. Sin embargo no tenemos que dar acceso a los ficheros que cuelgan directamente de dir_raiz. Para ello usamos setfacl con la sintaxis explicada en el punto anterior:

```
setfacl -R -m g:desarrollo:rw dir_raiz/subdir_1
setfacl -R -m g:desarrollo:rw dir_raiz/subdir_2
```

3. Por último tenemos que dar permisos al grupo de explotación en el directorio subdir_2 y todo su contenido:

```
setfacl -R -m g:explotacion:rx dir_raiz/subdir_2
```

Lo normal en este punto es comprobar cuales son los permisos reales que tienen cada uno de los ficheros y directorios existentes, para ver si efectivamente se ajustan a los requisitos planteados.

Para ello podemos usar getfacl para ver cuales son las ACL de cada uno de los directorios o ficheros implicados. La sintaxis es sencilla:

```
getfacl fichero ...
```

Si lo usamos para ver el resultado de las órdenes anteriores tenemos:

```
# getfacl dir_raiz
# file: dir_raiz
# owner: root
# group: root
```

```
user::rwx
group::r-x
group:sistemas:rw-
mask::rwx
other::r-x
```

El listado de permisos que obtenemos al ejecutar getfacl se compone de entradas de tipo user y group, además de las entradas para mask y other. En el caso de las entradas user y group, tendremos siempre una entrada para el propietario del fichero y el grupo del fichero (son las líneas que no indican un usuario o grupo concreto) y tantas líneas adicionales como ACEs de ese tipo hayamos asginado al fichero o directorio. La entrada other es la entrada del permiso tradicional *other* del modelo UGO.

La entrada mask es especial. Esa entrada, que se puede manipular con setfacl permite especificar el máximo de permisos que se pueden asignar en dicho fichero con las ACEs de usuario y grupo.

Veamos ahora las ACL del resto de ficheros y directorios:

```
# file: dir_raiz/fich_1
# owner: root
# group: root
user::rw-
group::r--
group:sistemas:rw-
mask::rw-
other::r--

# file: dir_raiz/fich_2
# owner: root
# group: root
user::rw-
group::r--
group:sistemas:rw-
mask::rw-
other::r--

# getfacl dir_raiz/dir_1
# file: dir_raiz/dir_1
# owner: root
# group: root
user::rwx
group::r-x
group:sistemas:rw-
group:desarrollo:rw-
mask::rwx
other::r-x

# getfacl dir_raiz/dir_1/fich*
# file: dir_raiz/dir_1/fich_3
# owner: root
# group: root
user::rw-
group::r--
group:sistemas:rw-
mask::rw-
other::r--

# file: dir_raiz/dir_1/fich_4
# owner: root
# group: root
user::rw-
group::r--
group:sistemas:rw-
group:desarrollo:rw-
mask::rw-
other::r--

# getfacl dir_raiz/dir_2
# file: dir_raiz/dir_2
# owner: root
# group: root
user::rwx
group::r-x
group:sistemas:rw-
group:desarrollo:rw-
group:explotacion:r-x
mask::rwx
other::r-x

# getfacl dir_raiz/dir_2/fich*
# file: dir_raiz/dir_2/fich_5
# owner: root
# group: root
user::rw-
group::r--
group:sistemas:rw-
group:desarrollo:rw-
group:explotacion:r-x
mask::rw-
```

```
# group: root
user::rw-
group::r--
group:sistemas:rw-
group:desarrollo:rw-
mask::rw-
other::r--
```

```
other::r--

# file: dir_raiz/dir_2/fich_6
# owner: root
# group: root
user::rw-
group::r--
group:sistemas:rw-
group:desarrollo:rw-
group:explotacion:r-x
mask::rw-
other::r--
```

2.2. ACLs por defecto

Todo lo anterior está muy bien, pero tiene un problema: los ficheros y directorios nuevos que se creen no van a tener todas esas ACLs con los valores adecuados, y por tanto tendremos que estar cada dos por tres ajustando los valores de las ACLs con las órdenes anteriores. Evidentemente eso no es operativo en absoluto. En nuestra ayuda llegan las ACLs por defecto (en realidad habría que hablar de ACEs por defecto, pero la documentación habla de ACLs por defecto, así que mantendremos la misma terminología para no crear más confusión).

Las ACLs por defecto nos permiten indicar cuál es el juego de ACEs que queremos que se apliquen automáticamente a los nuevos ficheros y directorios que se creen dentro de un directorio dado. Se dice en estos casos que los permisos se heredan desde el directorio padre.

La sintaxis es idéntica a la de una ACE normal, con la diferencia de que debemos usar además la opción -d:

```
setfacl -d -m g:sistemas:rw dir_raiz
setfacl -d -m g:desarrollo:rw dir_raiz/dir_1
setfacl -d -m g:desarrollo:rw dir_raiz/dir_2
setfacl -d -m g:explotacion:rw dir_raiz/dir_2
```

Si usamos getfacl para ver la ACL del directorio raíz por ejemplo, tenemos lo siguiente:

```
# getfacl dir_raiz
# file: dir_raiz
# owner: root
# group: root
user::rwx
group::r-x
group:sistemas:rw-
mask::rwx
other::r-x
```

```
default:user::rwx
default:group:sistemas:rwx
default:group::r-x
default:mask::rwx
default:other::r-x
```

Es decir, se añaden una serie de ACEs especiales, de tipo default que contienen la información a utilizar para los nuevos ficheros y directorios que se creen dentro de éste.

3. ACLs y Samba

A partir de la version 2.2.0 de Samba, este incorpora soporte para ACLs si el sistema operativo nativo lo soporta. Si hemos activado las ACLs como se indica arriba en nuestro núcleo, sólo debemos instalar las bibliotecas de desarrollo de atributos extendidos (libattr.a y libattr.h) y listas de control de acceso (libacl.a y libacl.h) en nuestro sistema para poder recompilar Samba con soporte para ACLs nativo. Si hemos seguido las instrucciones de compilación e instalación dadas arriba, estará todo lo necesario ya instalado.

Una vez compilado y para poder usarlo es necesario instalar las bibliotecas de enlace dinamico de atributos (libattr.so) y listas de control de acceso (libacl.so) en los lugares habituales. De nuevo, si hemos seguido las instrucciones de más arriba, ya tendremos instalado todo lo necesario.

Ahora sólo nos resta compilar samba con la opción de soporte para ACLs. Lo único necesario en este caso es, a la hora de lanzar la orden ./configure debemos añadir la opcion --with-acl-support a la lista de opciones que usamos habitualmente:

```
./configure --with-acl-support ... otras opciones adicionales
```

y compilar e instalar los nuevos binarios. A partir de este momento Samba contiene todo el codigo necesario para traducir las ACLs del protocolo SMB en ACLs nativas y viceversa, con lo cual tenemos soporte completo para el modelo de permisos tradicional de Microsoft Windows NT o posteriores.

Sin embargo, lo anterior por lo general no es suficiente si utilizamos Samba integrado en un dominio NT o 2000 (en modo mixto, ya que la versión estable de Samba -2.2.x- no tiene soporte para modo nativo de Active Directory). En este caso queremos usar la lista de usuarios y grupos del propio dominio para asignar los permisos y ACLs del recurso compartido por Samba.

Lo cual significa que dichas cuentas de usuario y grupo deben existir en el sistema operativo anfitrión donde se ejecuta Samba, ya que las ACLs del sistema operativo las crea el núcleo de Linux y por tanto debe usar UIDs y GIDs existentes en el sistema.

La solución es dar de alta todas esas cuentas en el sistema anfitrión. Pero si lo hacemos de forma manual tenemos dos grandes inconvenientes:

1. El número de cuentas puede ser muy alto (gran cantidad de trabajo) y podemos no conocer las contraseñas de buena parte (o la totalidad) de dichas cuentas.
2. A partir de este momento tenemos dos bases de datos de usuarios que mantener de forma sincronizada (manualmente), lo cual es siempre un desastre a punto de suceder.

Por suerte los chicos de Samba lo han tenido en cuenta, y a partir de la version 2.1.x de Samba han incorporado un nuevo componente a la familia de soluciones SMB: winbind.

Winbind es un módulo que se integra en el sistema Name Service Switch (NSS) y que es un componente más del sistema que puede enumerar usuarios y grupos. Se une así al sistema tradicional basado en ficheros locales y a los sistemas de gestión de usuarios basados en red como NIS, NIS+ o LDAP entre otros. Es por tanto un componente más que se compila con el resto de samba y que genera una serie de biblitecas de enlace dinámico para el sistema NSS.

Lo único que tenemos que hacer es lanzar la ejecución de Winbind (corre como un demonio de sistema), añadir en el fichero de configuración de Samba unas pocas directivas de configuración de Winbind para indicarle los rangos de UIDs y GIDs a gestionar, y configurar el Name Service Switch para que consulte a Winbind cuando no encuentre los usuarios en el resto de subsistemas configurados. Como el sistema NSS utiliza cachés de nombres, el rendimiento de Winbind es bastante rápido incluso en sistemas con unos pocos miles de usuarios (el autor lo utiliza de forma rutinaria con unas 1.000 cuentas de usuarios en una red de área local con un impacto de velocidad aceptable).

3.1. Valores de Configuración de Winbind en el fichero smb.conf

Lo primero que tenemos que hacer es editar el fichero smb.conf y añadir los parámetros que queremos que utilice winbindd (el demonio que oferta el servicio Winbid). Para ello tenemos que añadir las directivas:

```
winbind separator = "."
# usar los ujids de 10000 a 20000 para los usuarios del dominio
winbind uid = 15000-20000
# usar los ujids de 10000 a 20000 para los grupos del dominio
winbind gid = 15000-20000
# permitir la enumeracion de los usuarios y grupos del dominio
winbind enum users = yes
winbind enum groups = yes
# tiempo, en segundos, para el cacheo de la informacion
winbind cache time = 60
```

Donde:

1. **winbind separator**: indica el carácter a usar como separador entre el nombre del dominio al que pertenece Samba y el nombre de usuario o grupo individual. Winbind construye nombres de usuarios del tipo DOMINIO_separador_USUARIO y DOMINIO_separador_GRUPO. Por ejemplo: ESCOMPOSLINUX.iarenaza usando el valor indicado en el ejemplo de arriba.
2. **winbind uids**: indica el rango de UIDs del sistema anfitrion que Winbind usa para mapear los usuarios del dominio NT en los usuarios nativos del sistema local. Lo que hace Winbind es crear de forma artificial tantos usuarios locales (UIDs) como usuarios haya en el dominio, y hacer una asignacion estática entre ambos. Esta información se guarda en un fichero de bases de datos que conviene no perder (llamado winbindd_idmap.tdb, ubicado en /var/lib/samba en el caso de Debian GNU/Linux) o borrar, ya que la asignacion puede ser diferente la proxima vez que se construya la base de datos (de forma automatica por parte de Winbind si ve que esta falta).

   Es **muy importante** asegurarse de que el rango de UIDs (y GIDs para el valor siguiente) **no estén siendo ya usados** por otros subsistemas de enumeración/validación de usuarios (NIS, NIS+, LDAP, etc.)

3. **winbind gids**: idéntico al parámetro anterior, pero para el rango de identificadores de grupos GIDs) a usar. Se almacenan en el mismo fichero que antes.
4. **winbind enum users**: indica si queremos que winbind genere la lista de usuarios en respuesta a la ejecución de las funciones de biblioteca setpwent(), getpwent() y endpwent() que sirven para enumerar todos los usuarios existentes y sus datos asociados.

   En instalaciones con muchos usuarios (varios miles o decenas de miles) puede ser interesante suprimir esta enumeración (valor "no") por razones de rendimiento. Sin embargo, algunos programas se comportan de forma extraña en estos casos, con lo cual habrá que hacer pruebas en dichos casos.

5. **winbind enum groups**: Idéntico al caso anterior, pero para las funciones de biblioteca setgrent(), getgrent() y endgrent(), que sirven para enumerar los grupos existentes en el sistema anfitrión.
6. **winbind cache time**: tiempo en segundos a cachear la información sobre usuarios y grupos del dominio antes de volver a pedir los datos al controlador del dominio de nuevo.

Una vez ajustados los valores anteriores, debemos reinciar los demonios de Samba.

3.2. Configuración del Name Service Switch

Por último nos queda configurar el sistema NSS para enganchar Winbind como parte del mismo. Para ello debemos copiar las bibliotecas de enlace dinamico compiladas durante la compilación de Samba en el directorio /lib: el fichero llamado libnss_winbind.so.2 y crear un enlace simbólico a este llamado libnss_winbind.so.

Una vez copiados dichos ficheros, debemos configurar el sistema Name Service Switch para que utilice dichos servicios. Para ello editamos el fichero /etc/nsswitch.conf y añadimos el nombre winbind a las

entradas passwd: y group::

```
passwd: files ... otros ... winbind
group: files ... otros ... winbind
```

3.3. Ejecución del demonio winbindd

Para poner en marcha el demonio winbindd, basta con lanzarlo como usuario root. Si nuestro sistema tiene arranque de tipo System V, podemos agregar un script que lo lance durante el arranque del mismo (**después** de haber lanzado el demonio nmbd ya que requiere de sus servicios) y que lo detenga durante la parada del sistema.

Una vez puesto en marcha winbind como parte del sistema NSS, podemos usar las ordenes:

```
getent passwd
getent group
```

para obtener un listado de todos los usuarios y grupos locales de la máquina, listado que debería contener todos los usuarios y grupos creado (importados) por winbind:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
...
DOMINIO/aajuria:x:10011:10000:Ander Ajuria:/home/DOMINIO/aajuria:/bin/false
DOMINIO/aalberdi:x:10012:10000:Amaia AlberdI:/home/DOMINIO/aalberdi:/bin/false
DOMINIO/aalcelay:x:10014:10000:Amaia Alcelay:/home/DOMINIO/aalcelay:/bin/false
...
```

lo cual nos muestra que winbind está funcionando correctamente.

4. Licencia