1 Aspectos avanzados de Permisos

1.1 Sumario

1.2 Permisos por defecto

Cuando se crea un archivo o directorio de un usuario en Linux se le asignan unos permisos por defecto. Para los directorios, los permisos de base son 0777 (rwxrwxrwx) y para los archivos son 0666 (rw-rw-rw).

Existe un comando para visualizar y controlar esta asignación de permisos, El comando **umask**. Su sintaxis es:

umask [-S] [máscara]

El -S indica que la salida del comando será simbólica La máscara **indica aquellos permisos que se restarán del total.** Es decir, es una representación complementaria a la asignación

umask sin parámetros muestra la máscara de permisos del usuario

Ejemplos:

```
umask
```

Muestra la máscara de permisos por defecto del usuario en formato complementario octal

```
umask -S
```

Muestra la máscara de permisos por defecto en notación simbólica

```
umask 022
```

El umask por defecto 0022 se utiliza para los usuarios regulares. Con esta máscara, los permisos predeterminados de los directorios son 755 , esto es, rwx para propietario, rx para grupo, rx para otros y para los archivos 644, esto es, rw para propietario, r para grupo, r para otros.

La definición de la máscara para los usuarios del sistema suele realizarse con la directiva **UMASK** el archivo /**etc/login.defs** y éste es utilizado por el módulo **PAM (Plugable Authentication Module) pam_umask.so**.

Además de los bits de permisos clásicos (rwx) existen otros 3 "especiales" que se utilizan para otros fines. Veámoslos:

1.3 Sticky bit

Es un permiso que solo afecta a directorio y protege de borrados los contenidos del mismo cuando éste es compartido. Es decir, cuando hay permisos de escritura (w) para todos los usuarios que lo comparten pero queremos **evitar que alguno de los usuarios no propietarios del directorio puedan borrar su contenido**. En los listados, con el comando ls, por ejemplo haciendo un ls -ld <directorio>, se puede ver al final de la máscara de permisos como una "t". Un ejemplo clásico del uso de este bit es en el directorio /tmp, que suele estar compartido pero con sticky bit activado.

1.4 Setuid bit

Este es un permiso especial que se aplica a archivos ejecutables. Cuando se ejecuta un programa, archivo ejecutable, se utiliza la identidad del usuario que lo invoca. Por ejemplo el usuario pepe tiene permiso para ejecutar el comando passwd, como podemos ver si hacemos un ls en el directorio /usr/bin donde se ubica ese comando:

```
-rwsr-xr-x  1 root   root       42824 sep 13 00:29 passwd
```

vemos que el permiso "x" está activado para todos. Sin embargo en el lugar en el que debiera aparecer la "x" del propietario aparece una "s", indicando que el bit setuid está activado. El significado de ésto es que cuando se ejecute el archivo se hará en nombre y con los permisos de su propietario, en este caso root. Por lo tanto, aunque pepe invoque a passwd éste se ejecutará con usuario root. ¿Y porqué se hace esto así?. Pues porque hay ciertos programas que, independientemente de quién los invoce, necesitan permisos específicos. En el caso de passwd recordad que se utilizaba para cambiar la contraseña de los usuarios del Sistema. Esto requería escribir en el archivo de contraseñas /etc/shadow y, como sabemos, a ese archivo solo tiene acceso root en modo escritura. Por este motivo necesitamos que el bit setuid esté activado en passwd.

NOTA: Por cuestiones de seguridad algunas distribuciones de Linux desactivan el comportamiento **setuid** en scripts de shell. Se conocen varios tipos de ataques de infiltración de código malicioso utilizando este mecanismo en scripts de shell. Por este motivo el comportamiento setuid está deshabilitado en scripts. En el siguiente comentario, extraído de un foro (http://unix.stackexchange.com/questions/364/allow-setuid-on-shell-scripts) se menciona este comportamiento:

```
Setuid shebang

There is a race condition inherent to the way shebang (*!) is typically implemented:

    The kernel opens the executable, and finds that it starts with *!.
    The kernel closes the executable and opens the interpreter instead.
    The kernel inserts the path to the script to the argument list (as argv[1]), and executes the interpreter.

If setuid scripts are allowed with this implementation, an attacker can invoke an arbitrary script by creating a symbolic
link to an existing setuid script, executing it, and arranging to change the link after the kernel has performed step 1
and before the interpreter gets around to opening its first argument. For this reason, most unices ignore the setuid bit
when they detect a shebang.
```

1.5 Setguid bit

Éste es un permiso análogo al anterior con la diferencia de que, en este caso, el programa se ejecuta con los permisos del grupo del propietario del archivo. En caso de que setguid esté activado veremos una s en lugar de una x en la máscara de permisos del segundo grupo de letras rwx (que afectan al grupo del propietario)

Se pueden activar todos, uno, o ninguno de estos bits setuid y setguid.

1.6 Establecimiento de los bits de Permisos Especiales

Se puede hacer de 2 formas:

Mediante un cuarto dígito hexadecimal en la máscara de permisos con valores: 1000: Para sticky **2000**: Para setuid **4000**: Para seguid

No tiene sentido especificar más de un bit especial por los siguientes motivos:

- Sticky bit solo aplica a directorios
- Setuid bit y Seguid bit se aplican a archivos ejecutables, pero no tiene sentido definir ambos bits, pues sería especificar dos identidades de ejecución diferentes. Por tanto o bien, se ejecuta como propietario, o como miembro del grupo propietario

Ejemplo:

```
chmod 1766 /home/usuario/compartido
```

Añadiría el bit sticky al directorio /home/usuario/compartido que a su vez tiene máscara de permisos 766 para u(sers)g(roup)o(thers)

En la forma relativa de especificar permisos (mediante letras):

- **t**: Indica sticky

- **s**: Indica setuid o setguid en función de dónde se ponga la s

Ejemplo:

```
chmod u+s /home/usuario/miscript
```

Si hacemos un

```
ls -la /home/usuario/miscript
```

veríamos en la máscara de permisos, con umask 022, lo siguiente:

rwsr-xr-x

Ejemplo:

```
chmod g+s /home/usuario/miscript
```

Si hacemos un

```
ls -la /home/usuario/miscript
```

veríamos en la máscara de permisos, con umask 022, lo siguiente:

rwxr-sr-x

Añadirían por tanto los bits setuid y seguid el archivo ejecutable /home/usuario/miscript. Notad que no tiene sentido especificar simultáneamente ámbos.

1.7 Atributos. Comandos chattr y lsattr

1.7.1 lsattr

Comando que permite listar los atributos de archivos y directorios establecidos mediante el comadno **chattr**. Ejemplo:

```
lsattr dir
```

Muestra los atributos de los contenidos del directorio dir

1.7.2 chattr

Este comando permite cambiar los atributos de archivos y directorios de un sistema de archivos ext2/ext3/ext4. Solo root puede invocar a este comando

La sintaxis es:

chattr [opciones] [modo] ficheros

Las opciones son:

```
-R  Recursivamente.
-V  Muestra una salida detallada
```

Los modos, similares en su uso a chmod:

```
+  se usa para añadir atributos
-  se usa para quitar atributos
=  se usa para especificar los atributos
```

Algunos de esos atributos son:

- **A** evita que se modifique el campo atime al acceder a un fichero.
- **a** sólo permite abrir el fichero para añadir datos.
- **c** el fichero se guarda automáticamente comprimido por el kernel.
- **D** cuando un directorio es modificado, los cambios son escritos síncronamente.
- **d** excluye al fichero para ser respaldado por dump.
- **i** impide modificar, eliminar, renombrar el fichero y también enlazarlo, es decir lo hace de solo lectura.
- **s** al borrar un fichero con este atributo,sus bloques son rellenados con ceros.
- **S** cuando un fichero es modificado,los cambios son escritos síncronamente.
- **u** cuando un fichero es eliminado,su contenido es guardado.

Ejemplo:

```
chattr -R +c /home/user/doc
```

De modo recursivo establece el atributo c (guardar comprimido) del archivo correspondiente

```
chattr +i test.txt
```

Hace de solo lectura el archivo test.txt

1.8 ACL (Access Control List)

Los sistemas de archivos Linux más modernos, como ext2, ext3 y ext4, permiten la definición de **ACL**, listas de control de acceso, para regular los permisos de acceso a directorios y archivos. Con la definición de ACL pueden ampliarse las posibilidades de definición de permisos del sistema estándar **rwx ugo** y ampliarla a grupos y usuarios fuera de los roles habituales de **u (user) g (group) y o (others)**.

Para poder habilitar las ACL es preciso instalar el paquete correspondiente. Ejecutamos como root, o con sudo:

```
apt install -y acl
```

El sistema de archivos al que se van a aplicar las ACL tiene que ser montado con la opción correspondiente en /etc/fstab, en autofs, o bien mediante un comando de montaje manual. A continuación se ve una entrada en **/etc/fstab** a modo de ejemplo:

```
UUID=07aebd28-24e3-cf19-e37d-1af9a23a45d4    /home    ext4   defaults,acl   0   2
```

1.8.1 Listar ACL de un directorio

Usamos el comando **getfacl**

Supongamos que tenemos en nuestros sistema un directorio **/comun/ciclos/SMR1/MME** cuyo resultado a la hora de invocar el comando **getacl**, según el comando:

```
getfacl /comun/ciclos/SMR1/MME
```

muestra una salida como la siguiente:

```
# file: .
# owner: javier
# group: profes
user::rwx
group::r-x
group:profes:rwx
group:smr1:r-x
mask::rwx
other::r-x
default:user::rwx
default:group::r-x
default:group:profes:rwx
default:mask::rwx
default:other::r-x
```

En esta lista puede verse

- el **propietario** del directorio
- el **grupo propietario** y los **otros**
- los permisos correspondientes a ambos, y también los correspondientes a otros grupos de usuarios que se han añadido a la ACL.
    - ♦ En este caso, el grupo profes tiene permisos rwx
    - ♦ el grupo smr1 permisos r-x.
- El parámetro **mask** indican los permisos más restrictivos aplicados, es decir, independientemente de lo establecido en la ACL, el valor de mask marcará el máximo permiso utilizable. La directiva mask no afecta a los permisos de user ni other.
- Las entradas **default** se aplican únicamente a directorios y afectan a los archivos contenidos dentro del directorio. Están relacionados con la herencia de permisos dentro de los directorios. Según el ejemplo anterior, la entrada para el grupo smr1, es decir los permisos r-x, no serán heredados para los contenidos del directorio, al no estar establecida la directiva default correspondiente.

NOTA: Para comprobar que en un directorio están activadas las **ACL** veremos en la salida de un **ls -la** el signo + al final de la línea de los permisos

1.8.2 Gestionar ACLs

Del mismo modo que, para obtener los ACL de un directorio, utilizamos **getfacl**, para establecer las ACL utilizaremos el comando **setfacl**. Con el comando **setfacl** se modificarán las ACLs ya aplicadas. Veamos como añadir grupos a la ACL. Ejecutamos (si es necesario como root, o con sudo)

```
setfacl -m g:green:rwx /var/www/
setfacl -m g:blue:rwx /var/www
```

Estamos añadiendo los grupos **green y blue a la ACL el directorio /var/www**, a estos se le aplicarán los permisos definidos en la propia línea de definición de la ACL. La opción -R aplicará la ACL recursivamente dentro del directorio.

En el siguiente ejemplo vemos como eliminar un grupo de la ACL:

```
setfacl -x g:green /var/www
```

También es posible transferir ACLs de un directorio a otro, o a través de un archivo de texto. Veamos algún ejemplo:

```
echo "g:green:rwx" > acl
setfacl -M acl /home/midir
```

Escribe la ACL en el archivo acl y luego lo vuelca mediante el comando setacl en el directorio correspondiente

```
getfacl dir1 | setfacl -b -n -M - dir2
```

Recupera ACLs del directorio dir1 y las establece para el directorio dir2

```
getfacl -a dir1 | setfacl -d -M- dir2
```

Copia las ACL de dir1 en la ACL por defecto de dir2

1.9 Referencias

ACL en Linux

https://manuais.iessanclemente.net/index.php/Permisos_e_listas_de_control_de_acceso:_ACLs,_Eiciel

https://help.ubuntu.com/community/FilePermissionsACLs

Volver

JavierFP 11:31 09 ene 2019 (CET)