Capítulo 2: Permisos especiales y ACL

A menudo es necesario permitir que un usuario común pueda ejecutar un binario que sea propiedad de otro usuario con los permisos de este último. También puede ocurrir que un programa en particular deba ejecutarse con privilegios administrativos sin importar quién lo corra. Para estas situaciones disponemos de los permisos especiales **SUID** y **SGID** (**S**et **U**ser **ID** y **S**et **G**roup **ID**). Este tema, junto con la posibilidad de restringir acceso a archivos y directorios y establecer atributos especiales para estos serán los objetivos de este capítulo.

SUID, SGID, y sticky bit

Si un programa posee el permiso SUID y root es el propietario de dicho archivo, el proceso que resulte de ejecutarlo tendrá permisos de superusuario y contará con los mismos accesos de root en todo el sistema. Esto pone de relieve la necesidad de restringir al mínimo posible los archivos que cuenten con este permiso especial. Típicamente, el único programa que cuenta con SUID es `passwd`, el cual permite que cada usuario del sistema pueda cambiar su propia contraseña (por defecto, no la de los demás) sin tener que depender para ello de un administrador.

Lo mismo aplica para SGID en un archivo, aunque el mismo brinda acceso a que un proceso se ejecute con los privilegios del grupo propietario de dicho archivo.

Para agregar el SUID al archivo **miejecutable.sh** podemos recurrir a dos alternativas. Una de ellas consiste en agregar un 4 al principio de los permisos a asignar (si los expresamos en forma octal) o utilizar u+s como vemos a continuación.

```
chmod 4755 miejecutable.sh
chmod u+s miejecutable.sh
```

Para establecer el SGID sobre **otroejecutable.sh** el procedimiento es similar:

```
chmod 2755 otroejecutable.sh
chmod g+s otroejecutable.sh
```

Podemos ver ambos casos en la imagen siguiente, donde el SUID y el SGID se indican con una letra s en la terna de permisos correspondientes al dueño o al grupo, respectivamente:

```
[gacanepa@server ~]$ chmod 4755 miejecutable.sh
[gacanepa@server ~]$ chmod 2755 otroejecutable.sh
[gacanepa@server ~]$ ls -l *ejecutable.sh
-rwsr-xr-x 1 gacanepa gacanepa 0 Sep 22 20:48 miejecutable.sh
-rwxr-sr-x 1 gacanepa gacanepa 0 Sep 22 20:48 otroejecutable.sh
[gacanepa@server ~]$
```

Además del archivo **/usr/bin/passwd**, también es probable que existan otros que posean el permiso SUID dependiendo del software que tengamos instalado en nuestro equipo (dicho sea de paso, otro binario que generalmente posee el bit SUID es at **/bin/at**). Además de estos dos ejemplos, el binario correspondiente al sistema **X Window** (sin la **s** al final) también posee dicho permiso (por razones obvias, todos los usuarios deben poder iniciar sesión en un entorno gráfico).

Búsqueda de archivos con permisos elevados

Como administradores del sistema, nos interesará identificar todos los archivos que cuenten con los permisos de SUID o SGID, y restringir los mismos al mínimo necesario. Una vez identificados, podremos proceder a removerlos o cambiar sus permisos a otros menos potencialmente peligrosos para la integridad del sistema.

Para realizar esta tarea podemos valernos del uso de `find` y de la posibilidad de utilizar los permisos de un archivo como criterio de búsqueda. Los siguientes dos ejemplos permiten encontrar todos los archivos que cuentan con cualquiera de los dos permisos especiales.

```
find /usr/bin -type f -perm -u+s
find /usr/bin -type f -perm -g+s
```

En las siguientes imágenes vemos el resultado de los dos comandos de arriba en un sistema CentOS 7 (la salida puede variar en otras distribuciones):

```
[root@server ~]# find /usr/bin -type f -perm -u+s
/usr/bin/umount
/usr/bin/su
/usr/bin/crontab
/usr/bin/newgrp
/usr/bin/passwd
/usr/bin/gpasswd
/usr/bin/staprun
/usr/bin/chsh
/usr/bin/chfn
/usr/bin/pkexec
/usr/bin/at
/usr/bin/sudo
/usr/bin/fusermount
```

```
[root@server ~]# find /usr/bin -type f -perm -g+s
/usr/bin/locate
/usr/bin/wall
/usr/bin/lockfile
/usr/bin/ssh-agent
/usr/bin/write
[root@server ~]#
```

Alternativamente, podríamos guardar la salida de los dos comandos anteriores en archivos de texto y luego de un tiempo volver a ejecutarlos. De esta manera, podríamos compararlos con el comando `diff` y asegurarnos de que los permisos especiales no hayan sido establecidos en ningún otro programa desde la última revisión.

Sticky bit

En un sistema Linux pueden existir directorios en los que todos los usuarios del sistema pueden crear, modificar, y borrar archivos. Como ejemplos, podemos nombrar a **/tmp** y a cualquier directorio compartido (pensemos por ejemplo en el espacio común al que pueden acceder varios usuarios en un servidor FTP). El *sticky bit* aplicado a un directorio es un permiso especial que hace que solamente el dueño de un archivo contenido en el mismo o el propietario del directorio puedan borrarlo.

Para ilustrar, crearemos el directorio **/pruebas** y le asignaremos permisos 777:

```
mkdir /pruebas
chmod 777 /pruebas
```

Como próximo paso, agreguemos un archivo vacío llamado **ejemplo1.txt**:

```
touch /pruebas/ejemplo1.txt
```

Con otra cuenta de usuario ahora intentemos borrar el archivo:

```
rm /pruebas/ejemplo1.txt
```

A continuación, podemos apreciar los comandos anteriores y confirmar que **ejemplo1.txt** fue eliminado:

```
[root@server ~]# mkdir /pruebas
[root@server ~]# chmod 777 /pruebas
[root@server ~]# exit
logout
[gacanepa@server ~]$ touch /pruebas/ejemplo1.txt
[gacanepa@server ~]$ su - fulano
Password:
Last login: Sat Sep 22 21:08:17 UTC 2018 on pts/0
[fulano@server ~]$ rm /pruebas/ejemplo1.txt
rm: remove write-protected regular empty file ‘/pruebas/ejemplo1.txt’? y
[fulano@server ~]$ ls -l /pruebas
total 0
[fulano@server ~]$
```

Veamos qué sucede si intentamos repetir los pasos anteriores luego de haberle asignado el *sticky bit* sobre **/pruebas**:

```
chmod +t /pruebas
touch /pruebas/ejemplo2.txt
rm /pruebas/ejemplo2.txt
```

Tal como observamos, la presencia del *sticky bit* en esta ocasión impide que un usuario que no sea el dueño del directorio borre un archivo que no sea de su propiedad.

```
[root@server ~]# chmod +t /pruebas
[root@server ~]# exit
logout
[gacanepa@server ~]$ touch /pruebas/ejemplo2.txt
[gacanepa@server ~]$ su - fulano
Password:
Last login: Sat Sep 22 21:24:59 UTC 2018 on pts/0
[fulano@server ~]$ rm /pruebas/ejemplo2.txt
rm: remove write-protected regular empty file ‘/pruebas/ejemplo2.txt’? y
rm: cannot remove ‘/pruebas/ejemplo2.txt’: Operation not permitted
[fulano@server ~]$
```

En general, para ingresar a un directorio (mediante el comando `cd`) y realizar operaciones dentro del mismo, es necesario contar con permisos de lectura y ejecución sobre dicho directorio y sus padres. Para borrar un archivo, se necesita contar con permisos de escritura sobre el directorio padre, y de ejecución sobre todos los directorios padre hacia arriba. Por ejemplo, para borrar el archivo **/dir1/ejemplo.txt**, es necesario poseer permisos de ejecución sobre **/**, sobre **/dir1**, y sobre este último también permisos de escritura.

ACLs: Listas de control de acceso

Los archivos y directorios tienen conjuntos de permisos configurados para el propietario del archivo, el grupo propietario del archivo y todos los otros usuarios del sistema. Sin embargo, estos permisos tienen sus limitaciones. Por ejemplo, no podemos configurar diferentes permisos para usuarios diferentes sobre un mismo archivo o directorio. Para poder hacerlo, se crearon las listas de control de acceso, más comúnmente conocidas por la sigla **ACL** (**A**ccess **C**ontrol **L**ists).

---

*Es importante aclarar que para utilizar ACLs es necesario que el sistema de archivos haya sido montado con la opción respectiva. Por defecto, los sistemas **ext4** se montan con esta funcionalidad. Podemos*

verificar esto con el comando `tune2fs` seguido de la opción `-l` y del sistema de archivos a verificar, por ejemplo, `tune2fs -l /dev/sda1`.

Existen dos tipos de ACLs:

- ACLs de acceso: lista de permisos de acceso para un archivo o directorio específico.
- ACLs predeterminadas: sólo puede ser asociado con un directorio. Si un archivo dentro de un directorio no tiene una ACL, hereda las reglas del ACL predeterminado del directorio.

Los ACLs se pueden configurar:

- Por usuario
- Por grupo
- Para usuarios que no estén en el grupo de usuarios para el archivo
- A través de la máscara de derechos efectivos

El comando `setfacl` configura ACLs para archivos y directorios. Vamos a utilizar la opción `-m` (o su equivalente `--modify`) para modifique el ACL de un archivo o directorio. Esta herramienta es la encargada de llevar a cabo el cambio de permisos de ACL (representados por `<reglas>`) sobre un archivo o directorio (`<objeto>` en el ejemplo a continuación). Su sintaxis es la siguiente:

```
setfacl -m <regla> <objeto>
```

Consideremos el siguiente ejemplo para ilustrar. Los usuarios **gacanepa** y **fulano** son miembros del grupo **familia**. Por eso, ambos tienen acceso de lectura y escritura sobre el archivo **/comunes/archivoprueba**:

```
[gacanepa@server ~]$ grep familia /etc/group
familia:x:5003:gacanepa,fulano
[gacanepa@server ~]$ ls -l /comunes
total 0
-rw-rw---- 1 root familia 0 Sep 22 21:48 archivoprueba
[gacanepa@server ~]$ echo "Escribiendo como gacanepa" > /comunes/archivoprueba
[gacanepa@server ~]$ su - fulano
Password:
Last login: Sat Sep 22 21:30:46 UTC 2018 on pts/0
[fulano@server ~]$ echo "Escribiendo como fulano" >> /comunes/archivoprueba
[fulano@server ~]$ cat /comunes/archivoprueba
Escribiendo como gacanepa
Escribiendo como fulano
[fulano@server ~]$
```

Con el esquema tradicional de permisos se vuelve un tanto complicado asignar diferentes niveles de acceso a este archivo para cada usuario. Para remediar esta situación, podemos utilizar listas de control de acceso. Supongamos que deseamos que **gacanepa** continúe teniendo permisos de escritura, pero que **fulano** solamente pueda leer. Para lograr ese objetivo, crearemos una ACL de sólo lectura en **/comunes/archivoprueba** para este último de la siguiente manera:

```
setfacl -m u:fulano:r /comunes/archivoprueba
```

Con `getfacl /comunes/archivoprueba` podemos verificar que se haya aplicado la ACL. Ahora veamos el resultado de intentar escribir en el mismo archivo luego del cambio:

```
[fulano@server ~]$ getfacl /comunes/archivoprueba
getfacl: Removing leading '/' from absolute path names
# file: comunes/archivoprueba
# owner: root
# group: familia
user::rw-
user:fulano:r--
group::rw-
mask::rw-
other::---

[fulano@server ~]$ echo "Intentando escribir como fulano otra vez" >> /comunes/archivoprueba
bash: /comunes/archivoprueba: Permission denied
[fulano@server ~]$
```

Para eliminar la ACL utilizaremos:

```
setfacl -x u:fulano /comunes/archivoprueba
```

tras lo cual el usuario **fulano** tendrá nuevamente permisos de escritura sobre **/comunes/prueba**. En el caso de que hayamos creado más de una ACL sobre el mismo archivo podemos removerlas a todas juntas empleando `setfacl -b` seguido de la ruta al archivo.

Las ACLs predeterminadas se asignan agregando la opción `-d`:

```
setfacl -dm g:familia:r /comunes
```

```
[root@server ~]# setfacl -dm g:familia:r /comunes
[root@server ~]# getfacl /comunes
getfacl: Removing leading '/' from absolute path names
# file: comunes
# owner: root
# group: familia
user::rwx
group::r-x
other::r-x
default:user::rwx
default:group::r-x
default:group:familia:r--
default:mask::r-x
default:other::r-x

[root@server ~]# touch /comunes/nuevaprueba
[root@server ~]# getfacl /comunes/nuevaprueba
getfacl: Removing leading '/' from absolute path names
# file: comunes/nuevaprueba
# owner: root
# group: root
user::rw-
group::r-x                      #effective:r--
group:familia:r--
mask::r--
other::r--
```

De esta manera, cualquier archivo que se agregue dentro de **/comunes** posteriormente heredará la misma regla (los miembros del grupo **familia** podrán leerlos, pero no escribir en ellos).

Atributos especiales

Además de los permisos de lectura, escritura, y de ejecución asociados con el dueño, el grupo propietario y el resto de los usuarios del sistema, Linux nos permite establecer otros atributos en archivos y directorios. Estas propiedades son específicas del tipo de sistema de archivos. En esta sección hablaremos de algunas de las que podemos encontrar en sistemas de archivos del tipo **extendido** (utilizaremos como ejemplo **ext4**). Mediante el uso de estos atributos podemos impedir, entre otras cosas, que un archivo sea borrado o renombrado, o que se elimine contenido de este.

Para ver la lista completa de atributos especiales disponibles podemos consultar el man page de `chattr`. Si deseamos ver cuáles atributos están establecidos en un archivo o directorio utilizaremos `lsattr` seguido de la ruta a dicho objeto.

Para empezar, crearemos dos archivos llamados **prueba1.txt** y **prueba2.txt** en nuestro directorio actual. En cada uno de ellos agregaremos luego el siguiente contenido:

```
echo "Yo soy el archivo prueba1.txt" > prueba1.txt
echo "Yo soy el archivo prueba2.txt" > prueba2.txt
```

Ahora nos interesa establecer los siguientes atributos especiales:

- *append only*, representado con una **a**
- *immutable*, representado con una **i**.

Estos atributos nos permiten, respectivamente:

- agregar contenido al archivo, pero no borrar nada.
- evitar que sea borrado, renombrado, editado, o que se agregue un enlace al mismo.

Con una cuenta con permisos de superusuario, agreguemos el atributo *append only* a **prueba1.txt** y también el *immutable* a **prueba2.txt**:

```
chattr +a prueba1.txt
chattr +i prueba2.txt
```

Ahora regresemos a nuestra cuenta de usuario común y veamos qué sucede cuando 1) tratamos de agregar y borrar contenido de **prueba1.txt**, 2) hacer lo mismo con **prueba2.txt**, 3) borrar, renombrar, editar, o crear un enlace a **prueba2.txt**:

```
[gacanepa@server ~]$ echo "Yo soy el archivo prueba1.txt" > prueba1.txt
[gacanepa@server ~]$ echo "Yo soy el archivo prueba2.txt" > prueba2.txt
[gacanepa@server ~]$ su
Password:
[root@server gacanepa]# chattr +a prueba1.txt
[root@server gacanepa]# chattr +i prueba2.txt
[root@server gacanepa]# exit
exit
[gacanepa@server ~]$ echo "Agregar contenido a prueba1.txt" >> prueba1.txt
[gacanepa@server ~]$ cat /dev/null > prueba1.txt
-bash: prueba1.txt: Operation not permitted
[gacanepa@server ~]$ echo "Agregar contenido a prueba2.txt" >> prueba2.txt
-bash: prueba2.txt: Permission denied
[gacanepa@server ~]$ mv prueba2.txt prueba2.txt.old
mv: cannot move ‘prueba2.txt’ to ‘prueba2.txt.old’: Operation not permitted
[gacanepa@server ~]$ ln prueba2.txt prueba2.link
ln: failed to create hard link ‘prueba2.link’ => ‘prueba2.txt’: Operation not permitted
[gacanepa@server ~]$
```

Las operaciones que fallaron no podrán realizarse mientras los atributos mencionados estén presentes. Finalmente, para quitarlos, deberemos hacer (también como root):

```
chattr -a prueba1.txt
chattr -i prueba2.txt
```

El uso de los atributos de archivo puede ayudarnos a impedir, por ejemplo, que un intruso que haya obtenido acceso a nuestro sistema pueda modificar o borrar los logs del sistema (¡si es que dichos archivos poseen los atributos correspondientes!)

También es posible copiar ACLs predeterminadas de un directorio a otro. Muchas veces cuando trabajamos nos encontramos con que un directorio tiene aplicadas muchas reglas de control de acceso y queremos replicar esto en otro directorio. Para esto vamos a emplear `getfacl` y `setfacl` simultáneamente:

```
getfacl directorio1 | setfacl -b -n -M - directorio2
```

El comando copia anterior la ACL de **directorio1** a **directorio2** usando la opción `-M` que permite que la salida de `getfacl` se transforme en entrada de `setfacl`. Además, recordemos que `-b` indica que se deben limpiar las ACL del segundo directorio (si existen) y mediante `-n` establecemos nuevas ACL para el mismo.