MANEJO DE PERMISOS ESPECIALES Y ATRIBUTOS DE ARCHIVOS PARA SISTEMAS GNU/LINUX

Hamnlet G. Rivera Lagares

<hrivera@codigolibre.org>

Junio del 2008

Fundación Código Libre Dominicano

http://www.codigolibre.org

Índice general

1. Los bits SUID, SGID y sticky bit **1**

1.1. Introducción 1

1.1.1. Conceptos preliminares 1

1.2. El SUID 1

1.3. EL SGID 3

1.4. El sticky bit 5

1.4.1. Estableciendo permisos especiales 7

2. Permisos preestablecidos con umask **8**

2.1. El comando umask 8

3. Normas prácticas para aumentar la seguridad **10**

3.1. nosuid, nodev, noexec 10

3.2. Sistemas de ficheros en red 10

3.3. umask 11

3.4. Limitar recursos 11

3.5. wtmp, utmp 11

3.6. lastlog, faillog 12

3.7. Permisos de escritura 12

3.7.1. Ficheros extraños 13

3.7.2. Ficheros peligrosos 13

4. Atributos de un archivo **14**

4.1. Formato del mandato chattr 16

4.1.1. Atributos 16

4.2. Ejemplos 17

Capítulo1

Los bits SUID, SGID y sticky bit

1.1. Introducción

Para evitar que otros usuarios accedan a nuestros ficheros en los sistemas UNIX se establecen ficheros para indicar quien y cómo pueden acceder a esos ficheros. Estos permisos juegan un papel importante ya que en los sistemas UNIX todos son ficheros. Cuando se escribe algo en el monitor para el sistema operativo se esta escribiendo en un fichero, de igual forma cuando se leen las pulsaciones del teclado el sistema operativo esta leyendo un fichero, el correspodiente al teclado. Gracias a este típo de abstracción tenemos un sistema altamente configurable.

1.1.1. Conceptos preliminares

Habitualmente, los permisos de los archivos en UNIX se corresponden con un número en octal que varía entre 000 y 777; sin embargo, existen unos permisos especiales que hacen variar ese número entre 0000 y 7777 se trata de los bits de permanencia **sticky bit (1000), SGID (2000) y SUID (4000).**

1.2. El SUID

El bit de **SUID** o **setuid** se activa sobre un fichero añadiéndole 4000 a la representación octal de los permisos del archivo y otorgándole además permiso de ejecución al propietario del mismo; al hacer esto, en lugar de la **x** en la primera terna de los permisos, aparecerá una s o una **S** si no hemos otorgado el permiso de ejecución correspondiente (en este caso el **bit** no tiene efecto):

root@matematico:~# chmod 4777 /tmp/archivo1

root@matematico:~# chmod 4444 /tmp/archivo2

root@matematico:~# ls -l /tmp/archivo1

-rwsrwxrwx 1 root other 0 Feb 9 17:51 /tmp/archivo1

root@matematico:~# ls -l /tmp/archivo2

-r-Sr- -r- - 1 root other 0 Feb 9 17:51 /tmp/archivo2

root@matematico:~#

El bit **SUID** activado sobre un fichero indica que todo aquél que ejecute el archivo va a tener durante la ejecución los mismos privilegios que quién lo creó; dicho de otra forma, si el administrador crea un fichero y lo setuida, todo aquel usuario que lo ejecute va a disponer, hasta que el programa finalice, de un nivel de privilegio total en el sistema. Hay que aclar que para que este permiso funcine, el archivo debe tener el permiso de ejeción para el propietario del fichero. Este permiso solo es aplicable a ficheros binarios. Podemos verlo con el siguiente ejemplo:

root@matematico:/home/minixgnu# vim testsuid.c

```
#include <stdio.h>
#include <unistd.h>
 main()
{
 printf("UID: %d, EUID: %d\n",getuid(),geteuid());
 }
```

root@matematico:/home/minixgnu# gcc -o testsuid testsuid.c

Ahora le asignamos el permiso adecuado:

root@matematico:/home/minixgnu# chmod u+s testsuid

o bien pudo haber sido en modo octal:

root@matematico:/home/minixgnu# chmod 4755 testsuid

root@matematico:/home/minixgnu# ls -l testsuid

-rwsr-xr-x 1 root root 4305 Feb 10 02:34 testsuid

root@matematico:/home/minixgnu# su minixgnu

minixgnu@matematico:~$ id

uid=500(minixgnu) gid=500(users) groups=500(users)

minixgnu@matematico:~$ ./ testsuid

UID: 500, EUID: 0

Podemos comprobar que el usuario minixgnu, sin ningún privilegio especial en el sistema, cuando ejecuta nuestro programa setuidado de prueba está trabajando con un **EUID** (Effective **UID**) 0, lo que le otorga todo el poder del administrador (fijémonos que éste último es el propietario del ejecutable); si en lugar de este código el ejecutable fuera una copia de un shell, el usuario minixgnu tendría todos los privilegios del root mientras no finalice la ejecución, es decir, hasta que no se teclee exit en la línea de comandos.

1.3. EL SGID

Todo lo que acabamos de comentar con respecto al **bit setuid** es aplicable al bit **setgid** pero a nivel de grupo del fichero en lugar de propietario. En lugar de trabajar con el **EUID** del propietario, todo usuario que ejecute un programa setgidado tendrá los privilegios del grupo al que pertenece el archivo. Para activar el **bit** de **setgid** sumaremos **2000** a la representación octal del permiso del fichero y además habremos de darle permiso de ejecución a la terna de grupo; si lo hacemos, la **s** o **S**(si el permiso de ejecución no está activado) aparecerá en lugar de la **x** en esta terna. Si el fichero es un directorio y no un archivo plano, el bit **setgid** afecta a los ficheros y subdirectorios que se creen en él: estos tendrán como grupo propietario al mismo que el directorio setgidado, siempre que el proceso que los cree pertenezca a dicho grupo.

Los bits de **setuid** y **setgid** dan a UNIX una gran flexibilidad, pero constituyen al mismo tiempo la mayor fuente de ataques internos al sistema (entendiendo por ataques internos aquellos realizados por un usuario autorizado o no desde la propia máquina, generalmente con el objetivo de aumentar su nivel de privilegio en la misma). Cualquier sistema UNIX tiene un cierto número de ejecutables setuidados y/o setgidados. Cada uno de ellos, como acabamos de comentar, se ejecuta con los privilegios de quien lo creó (generalmente el **root** u otro usuario con ciertos privilegios) lo que directamente implica que cualquier usuario tiene la capacidad de lanzar tareas que escapen total o parcialmente al control del sistema operativo: se ejecutan en modo privilegiado si es el administrador quien creó los ejecutables.

Evidentemente, estas tareas han de estar controladas de una forma exhaustiva, ya que si una de ellas se comporta de forma anormal (un simple core dump) puede causar daños irreparables al sistema; tanto es así que hay innumerables documentos que definen, o lo intentan, pautas de programación considerada segura. Si por cualquier motivo un programa setuidado falla se asume inmediatamente que presenta un problema de seguridad para la máquina, y se recomienda resetear el **bit** de **setuid** cuanto antes.

Está claro que asegurar completamente el comportamiento correcto de un programa es muy difícil, por no decir imposible; cada cierto tiempo suelen aparecer fallos

(bugs) en ficheros setuidados de los diferentes clones de UNIX que ponen en peligro la integridad del sistema. Entonces, por qué no se adopta una solución radical, como eliminar este tipo de archivos? Hay una sencilla razón, el riesgo que presentan no se corre inútilmente, para tentar al azar, sino que los archivos que se ejecutan con privilegios son estrictamente necesarios en UNIX, al menos algunos de ellos. Veamos un ejemplo.

Un fichero setuidado clásico en cualquier clon es **/bin/passwd**, la orden para que los usuarios puedan cambiar su contraseña de entrada al sistema. No hace falta analizar con mucho detalle el funcionamiento de este programa para darse cuenta que una de sus funciones consiste en modificar el fichero de claves**(/etc/passwd o /etc/shadow)**. Está claro que un usuario per se no tiene el nivel de privilegio necesario para hacer esto (incluso es posible que ni siquiera pueda leer el fichero de claves), por lo que frente a este problema tan simple existen varias soluciones.

Podemos asignar permiso de escritura para todo el mundo al fichero de contraseñas, podemos denegar a los usuarios el cambio de clave o podemos obligarles a pasar por la sala de operaciones cada vez que quieran cambiar su contraseña. Parece obvio que ninguna de ellas es apropiada para la seguridad del sistema (quizás la última lo sea, pero es impracticable en máquinas con un número de usuarios considerable). Por tanto, debemos asumir que el **bit de setuid en /bin/passwd** es imprescindible para un correcto funcionamiento del sistema. Sin embargo, esto no siempre sucede así, en un sistema UNIX instalado out of the box el número de ficheros setuidados suele ser mayor de cincuenta; sin perjudicar al correcto funcionamiento de la máquina, este número se puede reducir a menos de cinco, lo que viene a indicar que una de las tareas de un administrador sobre un sistema recién instalado es minimizar el número de ficheros setuidados o setgidados. No obstante, tampoco es conveniente eliminarlos, sino simplemente resetear su bit de **setuid** mediante **chmod**.

Para asignar el **SGID** en modo octal debemos sumar dos al primer byte. Como dijimos anteriormente, si este permiso esta asignado, cuando lo veamos aparecerá una **s** o **S** dependiendo de si el permiso de ejecución está activado o no.

root@matematico:~$ chmod 2752 arhivo

root@matematico:~$ ls -l arhivo

-rwxr-s-w- 1 root root Feb 10 20:33 archivo

root@matematico:~# ls -l /bin/ping*

-rwsr-xr-x 1 root bin 14064 May 10 2007 /bin/ping

-rwsr-xr-x 1 root bin 14064 May 10 2007 /bin/ping6

root@matematico:~# chmod -s /bin/ping

root@matematico:~# ls -l /bin/ping

-rwxr-xr-x 1 root bin 14064 May 10 2007 /bin/ping

También hemos de estar atentos a nuevos ficheros de estas características que se localicen en la máquina; demasiadas aplicaciones de UNIX se instalan por defecto con ejecutables setuidados cuando realmente este bit no es necesario, por lo que a la hora de instalar nuevo software o actualizar el existente hemos de acordarnos de resetear el bit de los ficheros que no lo necesiten. Especialmente grave es la aparición de archivos setuidados de los que el administrador no tenía constancia (ni son aplicaciones del sistema ni un aplicaciones añadidas), ya que esto casi en el 100 % de los casos indica que nuestra máquina ha sido comprometida por un atacante. Para localizar los ficheros con alguno de estos bits activos, podemos ejecutar la siguientes ordenes:

```
root@matematico:~# find / \( -perm -4000 -o -perm -2000 \) -type f -exec \
> ls -l \{\} \;
```

Incluso podría crear una base de datos de programas SUID con

```
root@matematico:# find / -type f \( -perm -04000 -o -perm -02000 \)>/var/suid
```

y posteriormente verificar si ha aparecido alguno nuevo con el siguiente script:

```
for fich in find / -type f \( -perm -04000 -o -perm -02000 \)
do
if ! grep $fich /var/suid
then
echo "$fich es un nuevo fichero con SUID"
fi
done
echo “Actualiza la base de datos si es necesario”
```

1.4. El sticky bit

El sticky bit o bit de permanencia se activa sumándole **1000** a la representación octal de los permisos de un determinado archivo y otorgándole además permiso de ejecución; si hacemos esto, veremos que en lugar de una **x** en la terna correspondiente al resto de usuarios aparece una **t** (si no le hemos dado permiso de ejecución al archivo, aparecerá una **T**):

```
root@matematico:~# chmod 1777 /tmp/archivo1
root@matemtatico:~# chmod 1774 /tmp/archivo2
```

root@matematico:~# ls -l /tmp/archivo1

-rwxrwxrwt 1 root other 0 Feb 9 17:51 /tmp/archivo1

root@matematico:~# ls -l /tmp/archivo2

-rwxrwxr-T 1 root other 0 Feb 9 17:51 /tmp/archivo2

Si el bit de permanencia de un fichero está activado (recordemos que si aparece una T no lo está) le estamos indicando al sistema operativo que se trata de un archivo muy utilizado, por lo que es conveniente que permanezca en memoria principal el mayor tiempo posible; esta opción se utilizaba en sistemas antiguos que disponían de muy poca RAM, pero hoy en día prácticamente no se utiliza. Lo que si que sigue vigente es el efecto del **sticky bit** activado sobre un directorio, en este caso se indica al sistema operativo que, aunque los permisos normales digan que cualquier usuario pueda crear y eliminar ficheros (por ejemplo, un 777 octal), sólo el propietario de cierto archivo y el administrador pueden borrar un archivo guardado en un directorio con estas características. Este bit, que sólo tiene efecto cuando es activado por el administrador (aunque cualquier usuario puede hacer que aparezca una **t** o una **T** en sus ficheros y directorios), se utiliza principalmente en directorios del sistema de ficheros en los que interesa que todos puedan escribir pero que no todos puedan borrar los datos escritos, como **/tmp/** o **/var/tmp/**. Si el equivalente octal de los permisos de estos directorios fuera simplemente 777 en lugar de 1777, cualquier usuario podría borrar los ficheros del resto. Si pensamos que para evitar problemas podemos simplemente denegar la escritura en directorios como los anteriores también estamos equivocados: muchos programas como compiladores, editores o gestores de correo asumen que van a poder crear ficheros en **/tmp/** o **/var/tmp/**, de forma que si no se permite a los usuarios hacerlo no van a funcionar correctamente; por tanto, es muy recomendable para el buen funcionamiento del sistema que al menos el directorio **/tmp/** tenga el bit de permanencia activado.

Al principio habiamos comentado que el equivalente octal de los permisos en UNIX puede variar entre **0000** y **7777**. Hemos visto que podíamos sumar **4000, 2000** o **1000** a los permisos normales para activar respectivamente los **bits setuid, setgid** o **sticky**. Por supuesto, podemos activar varios de ellos a la vez simplemente sumando sus valores. En la situación poco probable de que necesitáramos todos los bits activos, sumaríamos 7000 a la terna octal 777:

root@matematico:~# chmod 0 /tmp/archivo

root@matematico:~# ls -l /tmp/archivo

---------- 1 minixgnu minixgnu 0 Feb 9 05:05 /tmp/archivo

root@matematico:~# chmod 7777 /tmp/archivo

root@matematico:~# ls -l /tmp/archivo

-rwsrwsrwt 1 minixgnu minixgnu 0 Feb 9 05:05 /tmp/archivo

Si en lugar de especificar el valor octal de los permisos queremos utilizar la forma simbólica de chmod, utilizaremos **+t** para activar el bit de permanencia o **sticky bit**, **g+s** para activar el de **setgid** y **u+s** para hacer lo mismo con el de **setuid**; si queremos resetearlos, utilizamos un signo - en lugar de un + en la línea de órdenes.

1.4.1. Estableciendo permisos especiales

Los posibles valores de todo lo que hemos dicho hasta ahora, serían los siguientes:

- - - - - - - - -	= 0	Predeterminado, si permisos especiles
- - - - - - - - t	= 1	Bit de persistencia, sticky bit
- - - - - s - - -	= 2	Bit sgid de grupo
- - - - - s - - t	= 3	Bit sgid y sticky bit
- - s - - - - - -	= 4	Bit suid
- - s - - - - - t	= 5	Bit suid y sticky bit
- - s - - s - - -	= 6	Bit suid y sgid
- - s - - s - - t	= 7	Bit suid, sgid, sticky bit

Algo que se debe tener en cuenta, es lo que decidamos establecer con permisos de bit **SUID** y **SGID**, ya que al establecerlos cualquier usuario podrá ejecutarlos como si fuera el propietario original de ese archivo o programa. Esto puede tener consecuecias de seguridad muy severas en su sistema. Es buena práctica coniderar si es conveniente que un usuario normal ejecute apliaciones que son propias del superusario (**root**) a través del cambio de bits **SUID** o **SGID**. Es mejor alternativa usar los comandos **sudo** y **su**.

Capítulo2

Permisos preestablecidos con umask

2.1. El comando umask

El comando **umask** establece la máscara de permisos de archivos y directorios. Es decir, los nuevos directorios y archivos que se crean obtienen el valor de los permisos a partir de los valores de **umask**.

minixgnu@matematico:~$ umask
0002
(o en formato simbólico con la opción -S)

minixgnu@matematico:~$ umask -S
u=rwx,g=rwx,o=rx

Lo anterior indica que un directorio y archivos ejecutables se crearán con los permisos 775 y los archivos comunes con los permisos 664. Esto se logra restando de 777 el valor de **umask** (777-002) y (666 - 002) respectivamente. El primer valor de umask corresponde para los valores de **SUID, GUID o sticky bit** que por defecto es cero 0.

minixgnu@matematico:~$ umask

0002

Creamos un archivo y según la máscara debemos tener 666-002 ó rw-rw-r - -

minixgnu@matematico:~$ touch muestra

minixgnu@matematico:~$ ls -l muestra

-rw-rw-r- - minixgnu minixgnu 0 Abr 25 13:33 muestra

Ahora creemos un directorio y según la máscara debemos tener 777–002=755 o rwxrwxr-x

minixgnu@matematico:~$ mkdir nuevo

minixgnu@matematico:~$ ls -ld nuevo

drwxrwxr-x 2 minixgnu minixgnu 4096 Abr 25 13:55 nuevo

Para establecer el valor de la máscara, simplemente se usa el mismo comando **umask** seguido del valor de máscara que se desee:

minixgnu@matematico:~$ umask 0022

Para fiajarlo en la sección, entonces es necesario agragarlo al .bash_profile o .bash_rc de nuestro home de usuario.

Capítulo3

Normas prácticas para aumentar la seguridad

Aunque sea necesario tener claros los conceptos y dedicarle algo de tiempo a una correcta planificación, tampoco los peligros expuestos tienen por qué asustar a nadie. Todas las distribuciones de GNU/Linux traen unos valores por defecto que son más que razonables para cubrir unas necesidades normales.

3.1. nosuid, nodev, noexec

Salvo casos excepcionales, no debería haber ninguna razón para que se permita la ejecución de programas **SUID SGID** en los directorios home de los usuarios. Esto lo podemos evitar usando la opción **nosuid** en el archivo **/etc/fstab** para las particiones que tengan permiso de escritura por alguien que no sea **root**. También puede ser útil usar **nodev** y **noexec** en las particiones de los direcctorios personales de y en **/var**, lo que prohíbe la creación de dispositivos de bloques o caracter y ejecución de programas.

3.2. Sistemas de ficheros en red

Si exporta sistemas de archivos vía **NFS**, asegúrese de configurar **/etc/exports** con los accesos lo más rectrictivos posibles. Esto significa no usar plantillas, no permitir acceso de escritura a **root** y montar como sólo lectura siempre que sea posible.

3.3. umask

Configura la máscara de creación de ficheros para que sea lo más restrictiva posible. Son habituales 022, 033, y la más restrictiva 077, y añadirla a **/etc/profile**. El comando **umask** se puede usar para determinar el modo de creación de ficheros por defecto en su sistema. Es el complemento octal a los modos de fichero deseado. Si los ficheros si crean sin ningún miramiento de estado de permisos, el usuario de forma inadvertida, podrá asignar permisos de lectura o escritura a alguien que no debería tenerlos.

De form típica, el estado de **umask** incluye 022, 027 y 077, que es lo más restrictivo. Normalmente **umask** se pone en **/etc/profile** y por tanto se aplica a todos los usuarios del sistema. Por ejemplo, puede tener una línea parecida a la siguiente:

Pone el valor por defecto de umask del usuario

umask 033

Esté seguro de que el valor **umask** de **root** es 077, lo cual desactiva los permisos de lectura, escritura y ejecución para otros usuarios, salvo que explicítamente use chmod(1). Si está usando, y utiliza su esquema de creación de identificador de grupos y usuarios, sólo es necesario usar 002 para **umask**. Esto se debe a que al crear un usuario se crea un grupo exclusivo para ese usuario.

3.4. Limitar recursos

Ponga el límites al sistema de ficheros en lugar de ilimitado como está por defecto. Puede controlar el límite por usuario utilizando el módulo **PAM** de límite de recursos y **/etc/pam.d/limits.conf**. Por ejemplo, los límites para un grupo **users** podrá parecer a esto:

@users hard core 0
@users hard nproc 50
@users hard rss 5000

Esto dice que se prohiba la creación de ficheros core, restringe el número de procesos a 50, y restringe el uso de memoria por usuario a 5M.

3.5. wtmp, utmp

wtpm este archivo es un fichero binario (su contenido no puede ser leeido directamente con el comando cat o similares) que almacena información relativa a cada

conexión y desconexión al sistema, Podemos ver su contenido con órdenes como **last**. Los registros guaradados en este archivo y tambien el el fichero **wtpm**, tienen el formato de la estructura **utpm**, que contien infomación como nombre de usuario, la línea por la que accede, el lugar desde donde lo hace y la hora de acceso. Para ver la estructura del clon UNIX que estas trabjando puede ejecutar (**man getutent**).

utpm este archivo es un fichero binario con información de cada usuario que está conectado en un momento dado..., el programa **/bin/login** genera un registro en este fichero cuando un usario conecta, mientras que **init** lo elimina cuando desconecta. Para visualizar el contenido de este archivo podemos utilizar ordenes como (**last, w, who**).

Los ficheros **/var/log/wtmp** y **/var/run/utmp** contienen los registros de conexión de todos los usuarios de su sistema. Se debe mantener su integridad, ya que determinan cuándo y desde dónde entró en su sistema un usuario o un potencial intruso. Los ficheros deberían tener los permisos 644, sin afectar a la normal operación del sistema.

3.6. lastlog, faillog

El fichoero **lastlog** es un archivo binario guardado gneralmente en **/var/log/lastlog** y que contiene un resgistro para cada usuario con lo fecha y la hora de su última conexión, podemos visualizar estos datos para un usuario dado mediante **who** o **finger**.

El fichero **faillog** es equivalente al anterior, pero en lugar de guardar información sobre la fecha y hora del último acceso al sistema, lo hace del útimo intento de acceso de cada usuario; una conexión es fallida si el usuario teclea incorrectamente su contraseña. Esta inforamación se muestra la siguiente vez que dicho usuario entra correctamente al sistema.

3.7. Permisos de escritura

Los ficheros con permiso de escritura global, particularmente los del sistema, pueden ser un agujero de seguridad si un cracker obtiene acceso a su sistema y los modifica. Además, los directorios con permiso de escritura global son peligrosos, ya que permiten a un cracker añadir y borrar los ficheros que quiera. Para localizar los ficheros con permiso de escritura global, use el siguiente comando:

root@matematico:# find / -perm -2 -print

y esté seguro de saber por qué tienen esos permisos de escritura. En el curso normal de una operación los ficheros tendrán permisos de escritura, incluidos algunos de **/dev** y enlaces simbólicos.

3.7.1. Ficheros extraños

Los ficheros sin propietario también pueden ser un indicio de que un intruso ha accedido a su sistema. Puede localizar los ficheros de su sistema que no tienen propietario o que no pertenecen a un grupo con el comando:

root@matematico:# find / -nouser -o -nogroup -print

3.7.2. Ficheros peligrosos

La localización de ficheros **.rhosts** debería ser uno de los deberes regulares de la administración de su sistema, ya que estos ficheros no se deberían permitir en sus sistema. Recuerde que un cracker sólo necesita una cuenta insegura para potencialmente obtener acceso a toda su red. Puede localizar todos los ficheros **.rhosts** de su sistema con el siguiente comando:

root@matematico:# find /home -name .rhosts -print

Nota: Antes de cambiar permisos en cualquier sistema de ficheros, esté seguro de que entiende lo que hace. Nunca cambie permisos de un fichero simplemente porque parezca la forma fácil de hacer que algo funcione. Siempre debe determinar porqué el fichero tiene esos permisos y el propietario antes de modificarlos.

Capítulo4
Atributos de un archivo

En el sistema de ficheros ext2 y ext3 de GNU/Linux existen ciertos atributos para los archivos que pueden ayudar a incrementar la seguridad de un sistema. De todos ellos, de cara a la seguridad algunos no nos interesan demasiado, pero otros sí que se deben tener en cuenta. Uno de los atributos interesantes quizás el que más es **a**. Tan importante es que sólo el administrador tiene el privilegio suficiente para activarlo o desactivarlo. El atributo **a** sobre un archivo indica que este sólo se puede abrir en modo escritura para añadir datos, pero nunca para eliminarlos. En qué influye esto con la seguridad?

Es simple, cuando un intruso ha conseguido el privilegio suficiente en un sistema atacado, lo primero que suele hacer es borrar sus huellas para esto existen muchos programas (denominados zappers, rootkits...) que, junto a otras funciones, eliminan estructuras de ciertos ficheros de **log** como **lastlog, wtmp** o **utmp**. Así consiguen que cuando alguien ejecute **last, who, users, w** o similares, no vea ni rastro de la conexión que el atacante ha realizado a la máquina. Evidentemente, si estos archivos de **log** poseen el atributo **a** activado, el pirata y sus programas lo tienen más difícil para borrar datos de ellos. Ahora viene la siguiente cuestión, si el pirata ha conseguido el suficiente nivel de privilegio como para poder escribir y borrar en los ficheros (en la mayoría de UNIXs para realizar esta tarea se necesita ser **root**), simplemente ha de resetear el atributo **a** del archivo, eliminar los datos comprometedores y volver a activarlo.

Obviamente, esto es así de simple, pero siempre hemos de recordar que en las redes habituales no suelen ser atacadas por piratas con un mínimo nivel de conocimientos, sino por los intrusos más novatos de la red, tan novatos que generalmente se limitan a ejecutar programas desde sus flamantes Windows sin tener ni la más remota idea de lo que están haciendo en UNIX, de forma que una protección tan elemental como un fichero con el flag **a** activado se convierte en algo imposible de modificar para ellos, con lo que su acceso queda convenientemente registrado en el sistema. Otro atributo del sistema de archivos ext2 y ext3 es **i** (fichero inmutable). Un archivo

con este flag activado no se puede modificar de ninguna forma, ni añadiendo datos ni borrándolos, ni eliminar el archivo, ni tan siquiera enlazarlo mediante **ln**. Igual que sucedía antes, sólo el administrador puede activar o desactivar el atributo **i** de un fichero. Podemos aprovechar esta característica en los archivos que no se modifican frecuentemente, por ejemplo muchos de los contenidos en **/etc** (en donde se encuentran ficheros de configuración, scripts de arranque..., incluso el propio fichero de contraseñas si el añadir o eliminar usuarios tampoco es frecuente en nuestro sistema); de esta forma conseguimos que ningún usuario pueda modificarlos incluso aunque sus permisos lo permitan. Cuando activemos el atributo **i** en un archivo hemos de tener siempre en cuenta que el archivo no va a poder ser modificado por nadie, incluido el administrador, y tampoco por los programas que se ejecutanen la máquina; por tanto, si activáramos este atributo en un fichero de log, no se grabaría ninguna información en él, lo que evidentemente no es conveniente.

También hemos de recordar que los archivos tampoco van a poder sen enlazados, lo que puede ser problemático en algunas variantes de GNU/Linux que utilizan enlaces duros para la configuración de los ficheros de arranque del sistema. Atributos que también pueden ayudar a implementar una correcta política de seguridad en la máquina, aunque menos importantes que los anteriores, son **s** y **S**. Si borramos un archivo con el atributo **s** activo, el sistema va a rellenar sus bloques con ceros en lugar de efectuar un simple unlink, para así dificultar la tarea de un atacante que intente recuperarlo; realmente, para un pirata experto esto no supone ningún problema, simplemente un retraso en sus propósitos.

Por su parte, el atributo **S** sobre un fichero hace que los cambios sobre el archivo se escriban inmediatamente en el disco en lugar de esperar el **sync** del sistema operativo. Aunque no es lo habitual, bajo ciertas circunstancias un fichero de **log** puede perder información que aún no se haya volcado a disco. Imaginemos por ejemplo que alguien se conecta al sistema y cuando éste registra la entrada, la máquina se apaga súbitamente; toda la información que aún no se haya grabado en disco se perderá. Aunque como decimos, esto no suele ser habitual , si nuestro sistema se apaga frecuentemente sí que nos puede interesar activar el bit **S** de ciertos ficheros importantes. Ya hemos tratado los atributos del sistema de ficheros ext2 y ext3 que pueden incrementar la seguridad de GNU/Linux; vamos a ver ahora, sin entrar en muchos detalles (recordemos que tenemos a nuestra disposición las páginas del manual) cómo activar o desactivar estos atributos sobre ficheros, y también cómo ver su estado.

Para lo primero utilizamos la orden **chattr**, que recibe como parámetros el nombre del atributo junto a un signo + o **-**, en función de si deseamos activar o desactivar el atributo, y también el nombre del archivo correspondiente. Si lo que deseamos es visualizar el estado de los diferentes atributos, utilizaremos **lsattr**, cuya salida indicará con la letra correspondiente cada atributo del archivo o un signo - en el

caso de que el atributo no esté activado.

4.1. Formato del mandato chattr

chattr [-RV] +-=[AacDdijsSu] [-v versión] archivos

La opción **R** cambia recursivamente los atributos de los directorios y sus contenidos.

El operador = hace que solamente hayan los atributos especificados.

4.1.1. Atributos

El atributo **A** hace que la fecha del último acceso en el archivo, no sea modificada.

El atributo **D** cuando se trata de un directorio, establece que los datos se escriben de forma sincrónica en el disco. Es decir los datos se escriben inmediatamente en lugar de esperar la operación correpondiente del sistema operativo. Es equivalente a la opción **dirsync** del comando **mount**, pero aplicada a un conjunto de archivos. Este atributo trabaja igual que el atributo **S**, con la diferencia de que este último trabaja sobre archivos y el primero sobre directorios.

El atributo **j** en sistemas de archivos ext3, cuado se montan con las opciones **data=ordered** o **data=writeback**, se establece que el archivo será escrito el registro por diario (**Journal**). Si el sistema de archivos se monta con la opción **data=journal**, todo el sistema de archivo se escribe en el registro por diario y por tanto el atributo no tiene efecto.

En atributo **u** cuando un archivo con este atributo es eliminado, sus contenidos son guardados permitiendo recuperar el archivo con alguna herramienta para tal prpósito.

4.2. Ejemplos

El siguiente ejemplo lista los atributos del archivo prueba.txt.

minixgnu@matematico:~$ lsattr prueba.txt

- - - - - - - - - - - - - - - - prueba.txt

minixgnu@matematico:~$ chattr +a prueba.txt

Si volvemos a listar los atributos del archivo, obtendremos la siguiente salida

minixgnu@matematico:~$ lsattr prueba.txt

- - - - - a - - - - - - - - - - prueba.txt

Para eliminar el modo de solo adjuntar del ejemplo anterio, solo tenemos que ejectar lo siguiente:

minixgnu@matematico:~$ chattr - prueba.txt

Ahora en el siguiente mandato se establecerá que el archivo se le agregará los atributos **S** y **s**

minixgnu@matematico:~$ chattr +Ss prueba.txt

Y posteriormente al listar sus atributos obtenemos lo siguiente:

minixgnu@matematico:~$ lsattr prueba.txt

s - S - - - - - - - - - - - - - - prueba.txt

El siguinte mandato le agrega el atributo inmnutable al archivo prueba.txt.

minixgnu@matematico:~$ chattr +i prueba.txt

Amigo lector le invito a pruebar con cada una de las opciones de éste comando, ya que esto le servirá de práctica y para le mejor entendimiento de lo que se ha escrito en este documento.

Espero este tutorial pueda ser útil a los amigos de la comuniadad del Software Libre.

[1] **Licencia de este documento**