como administrador sin tener que introducir ninguna contraseña. Tenga en cuenta que, en ambos casos, los argumentos están definidos explícitamente. Si usamos otros argumentos (por ejemplo, */sbin/fdisk /dev/hda*) esto podría causar que *sudo* nos pidiese la contraseña adecuada.

Riesgos de Seguridad

Al igual que cualquier otra funcionalidad relacionada con la seguridad de nuestro sistema, *sudo* puede abrir un montón de agujeros si no somos cuidadosos. Algunos de estos agujeros podrían ser consecuencia de no configurar *sudo* adecuadamente. Otros problemas aparecen cuando confundimos el comportamiento natural del comando que especificamos junto a *sudo*.

En otros casos, podríamos conocer el comportamiento del comando, pero simplemente pasamos por alto algunas ramificaciones de este comportamiento. Por ejemplo, una importante tarea de administración es reaccionar cuando el sistema de archivos se llena demasiado. Naturalmente, si no tenemos los permisos necesarios, vamos a tener problemas al intentar limpiar el sistema de archivos. En un caso que conozco en detalle, esto se resolvió configurando *sudo* de manera que los usuarios normales pudieran usar *gzip*. El objetivo era que pudiesen usarlo para comprimir los archivos de log y liberar espacio en disco.

Esta solución demostraba dos problemas. El primero es la postura de los administradores de sistemas al requerir que el usuario normal reaccionara al problema al vuelo en lugar de configurar el sistema de forma proactiva para que comprimiese los archivos de log automáticamente. El segundo problema es que cuando *gzip* terminaba de comprimir el archivo, tenía que borrar el original.

Para habilitar a los usuarios normales para comprimir cualquier archivo de log, se especificó *gzip* con un asterisco, lo que significaba que podría ejecutarse sobre cualquier archivo. Por lo tanto es posible que alguien comprimiese un archivo que en realidad era necesario para el funcionamiento normal del sistema. Un ejemplo muy sencillo podría ser el archivo */etc/passwd*. Si un usuario fuese a comprimir este archivo mediante *sudo*, el original desaparecería y una gran cantidad de elementos del sistema dejarían de funcionar correctamente.

Esta misma oficina estaba teniendo problemas con procesos desbocados. La solución fue permitir a los usuarios normales que matasen estos procesos mediante *sudo*. Al igual que con *gzip*, el comando *kill* se especificó con un asterisco, lo que significa que el usuario podía matar *cualquier* proceso.

Ambos casos requieren que hagamos unas suposiciones acerca de la fiabilidad de los usuarios. Por otro lado, si nuestro objetivo es hacer que el sistema sea lo más seguro posible, incluyendo evitar que la gente haga cosas que no deberían hacer, permitir al usuario que borre cualquier archivo o que mate cualquier proceso conduce a un buen número de problemas de seguridad.

El siguiente peligro viene por pasar por alto el comportamiento natural del comando que especificamos. Por ejemplo, cuando estamos resolviendo algún problema, a menudo es necesario revisar varios archivos de log y de configuración. En algunos casos, podría ser trabajo del usuario normal intentar solucionar el problema inicialmente, revisando estos archivos. En lugar de tomarnos un tiempo para definir qué archivos tendrían que leerse por qué usuarios, podríamos tener la tentación de permitir a todos los usuarios la ejecución del comando *more* para paginar cualquier archivo del sistema.

Una vez más, *cualquier* significa *cualquier*. Algunos archivos no deberían ser leídos por ningún usuario (por ejemplo, los archivos que contienen contraseñas). Incluso si confiamos lo suficiente en los usuarios para permitirles que echen un vistazo a estos archivos, tenemos que preocuparnos acerca de lo que pasaría si un intruso consiguiese acceder a la cuenta de un usuario.

Otro problema con el comando *more* es que nos deja ejecutar un *shell escape*, lo que nos permite iniciar un subshell desde un comando *more*. Debido a que *more* se inicia como administrador, por ejemplo, todos los subprocesos, incluyendo cualquier consola, se ejecutarán como administrador. Esto significa que ahora tenemos una consola que se ejecuta como root, ¡y tenemos control total sobre el sistema! Para empeorar las cosas, he visto algunas instalaciones donde esto es posible sin necesidad siquiera de introducir una contraseña para ejecutar el comando, lo que significa que podemos conseguir acceso a consolas de administrador hasta el contenido más comprometido.

Una solución a este problema es no permitir el acceso a los usuarios a programas con una opción de shell escape. Si todo lo que el usuario necesita es ver los contenidos de un archivo, puede que nos arreglemos usando el comando *cat*, que no tiene shell escapes. Si el archivo es muy largo, puede que no quepa en la pantalla usando *cat*. Esto no es un problema, porque podemos canalizar la salida a través de *more*. Debido a que el proceso *more* no se ejecuta como el usuario destino (es decir, el administrador), hacer un shell escape simplemente nos devuelve al usuario original.

Aunque este método se dirige a comandos específicos, nos puede coger desprevenidos si ejecutamos un programa que permita shell escapes donde no los esperamos. Para solucionar este problema, *sudo* proporciona mecanismos para evitar que los comandos ejecuten shell escapes. La primera opción es configurar default a *noexec*:

```
<C>Defaults noexec<C>
```

La configuración *noexec* se aplicará a todos los comandos por defecto, independientemente de qué usuario los inicie y qué usuario sea el objetivo. De forma alternativa, podríamos deshabilitar los shell escapes comando a comando con *NOEXEC*:

```
jimmo ALL=NOEXEC ↵
/usr/bin/view, /usr/bin/more
```

Sudo hace un buen trabajo al llevar el registro de quién hace qué y si tuvo éxito o no. Por defecto, *sudo* registra eventos usando el servicio syslog. Sin embargo, podemos definir explícitamente un archivo de log usando la entrada por defecto *logfile* en */etc/sudoers*. Lo que es más, podemos definir diferentes instalaciones y prioridades de syslog a nuestra conveniencia. Véase la página man de *sudoers* para más detalles.

Conclusiones

Como con otras herramientas de seguridad, la estrategia más segura para usar *sudo* es deshabilitar todos los accesos y entonces habilitar sólo lo que necesitamos, pieza a pieza. Tenga en mente la capacidad intrínseca de los shells de Linux para ejecutar comandos juntos mediante tuberías.

Antes de comenzar a diseñar nuestros procedimientos de administración del sistema para nuestro entorno, es una buena idea asegurarse de que sabemos exactamente por qué estamos usando cada comando y cada argumento que ejecutamos con *sudo*. Quizá pueda encontrar soluciones mejores (como evitar que se ejecuten procesos desbocados), o al menos maneras más seguras de especificar comandos. Peque de prudente, no de cómodo. ■