Universidad Nacional Autónoma de México
Facultad de Ingeniería
Arquitectura Cliente Servidor
Ortega de la Paz Rafael | 420054085
03 de Septiembre del 2026

Tarea 2: Administración de Linux — Permisos, fdisk y bit setuid

Introducción

En el presente trabajo se documenta la ejecución práctica de los ejercicios propuestos en la Tarea 2 de la asignatura, relacionados con la administración de permisos en Linux, el comando fdisk y el bit setuid. Se muestra evidencia de cada paso realizado en el sistema operativo, junto con las respuestas a los cuestionamientos planteados.

Investigación previa

¿Qué hace el comando fdisk? El comando fdisk es una utilidad de administración de discos disponible en los sistemas tipo Unix/Linux que permite visualizar, crear, eliminar y modificar particiones dentro de una tabla de particiones de un disco duro o unidad de almacenamiento. Al operar directamente sobre el dispositivo de bloque del disco (por ejemplo /dev/sda), su uso para modificar particiones requiere privilegios de superusuario, ya que una manipulación incorrecta puede provocar la pérdida de información del sistema (Shotts, 2019; Negus, 2020).

¿Qué hace la opción -l de fdisk? La opción -l (list) indica al comando fdisk que únicamente liste las tablas de particiones existentes en los discos detectados por el sistema, mostrando información como el tamaño del disco, el tipo de tabla de particiones y el listado de particiones con su respectivo identificador de sistema de archivos. A diferencia del uso interactivo de fdisk, esta opción no permite modificar el disco, solamente desplegar su información y finalizar la ejecución (Shotts, 2019).

Actividad 1 — Permisos, fdisk y bit setuid

Se abrieron dos terminales en el sistema: una con el usuario root y otra con el usuario sin privilegios (rafa). Al ejecutar el comando whoami en ambas terminales se confirmó la identidad de cada sesión, tal como se observa en la siguiente evidencia.

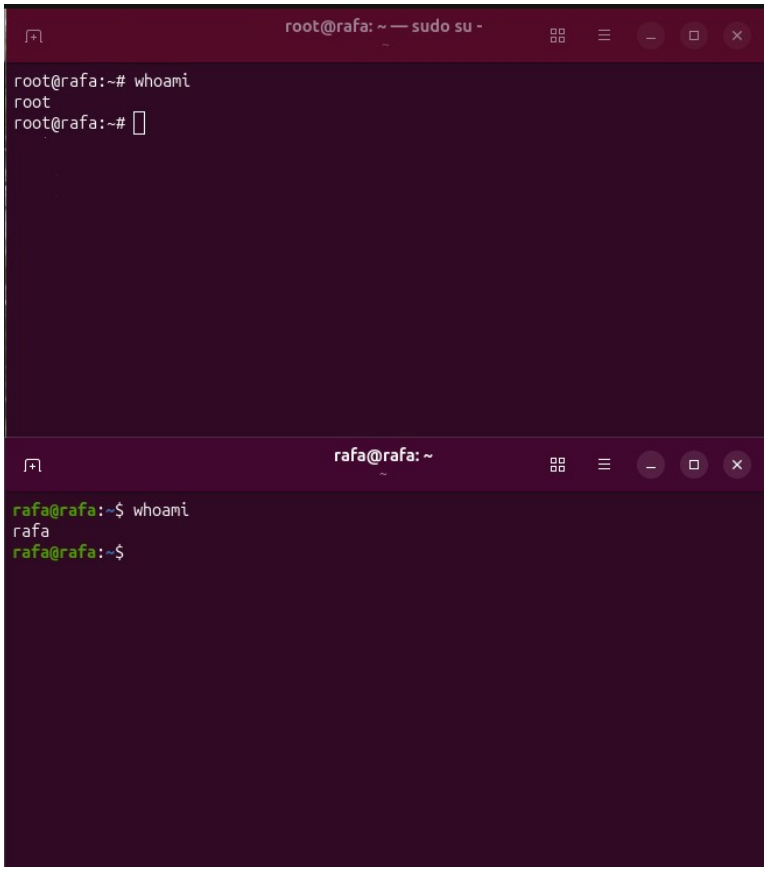


Imagen 1: Verificación de usuario con whoami en ambas terminales

Posteriormente, desde la terminal del usuario sin privilegios se ejecutó el comando fdisk -l, obteniendo el mensaje de error "Permission denied" para cada uno de los dispositivos de almacenamiento, ya que este usuario no cuenta con los permisos necesarios para leer directamente los dispositivos de bloque.

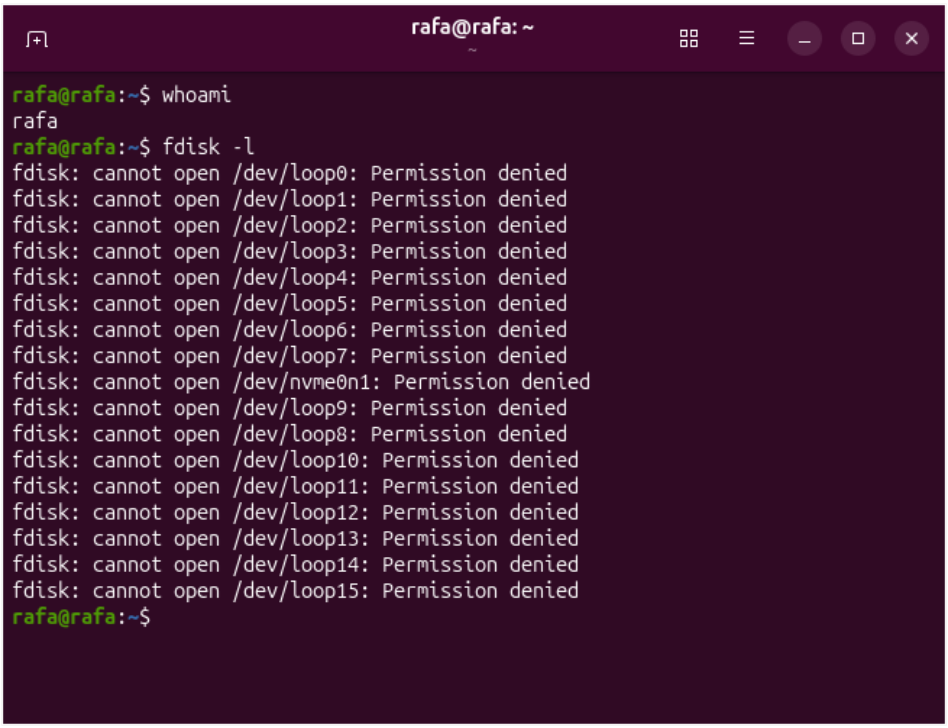


Imagen 2: Ejecución de fdisk -l como usuario sin privilegios (permiso denegado)

A continuación, desde la terminal de root se localizó la ruta del ejecutable con which fdisk, obteniendo /usr/sbin/fdisk, y se revisaron sus permisos originales con el comando ls -l, confirmando los permisos estándar -rwxr-xr-x con propietario y grupo root. Enseguida se modificó el bit setuid del ejecutable mediante el comando chmod 4511 /usr/sbin/fdisk, y se corroboró el cambio con ls -l, observando que el permiso del propietario cambió de rwx a r-s, lo que indica que el bit setuid quedó activado.

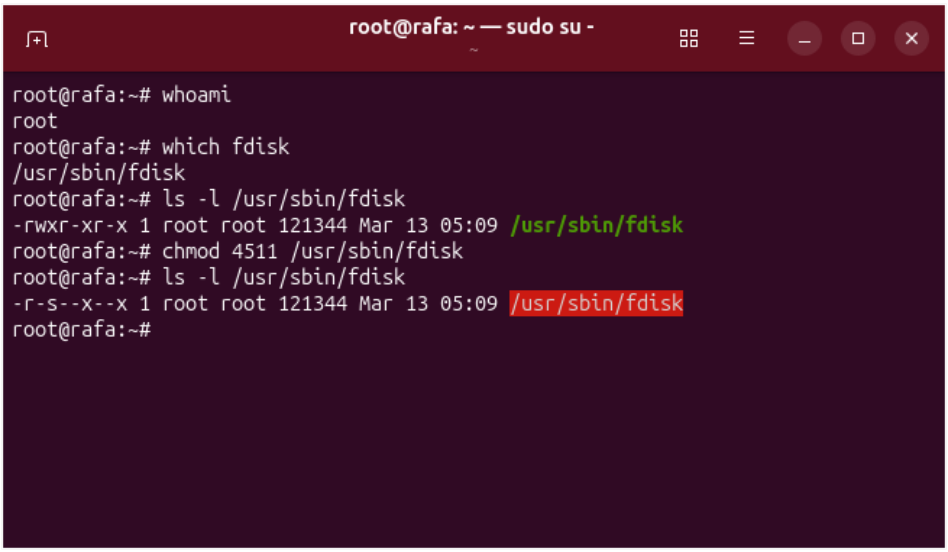
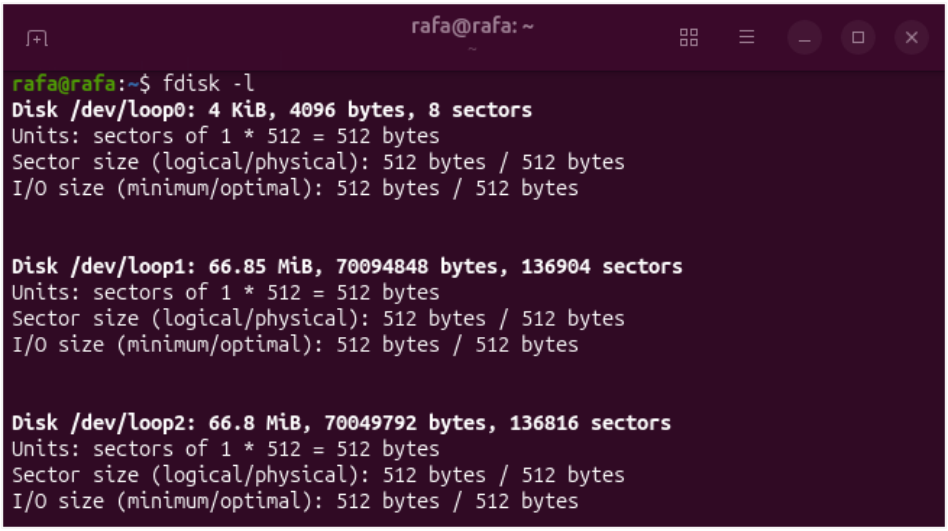


Imagen 3: Ruta y permisos originales de fdisk, y activación del bit setuid

Al regresar a la terminal del usuario sin privilegios y ejecutar nuevamente fdisk -l, el comando se ejecutó exitosamente, mostrando la información de las particiones del disco. Esto se debe a que, gracias al bit setuid, el proceso se ejecuta con los privilegios efectivos del propietario del archivo (root) y no con los del usuario que lo invocó.



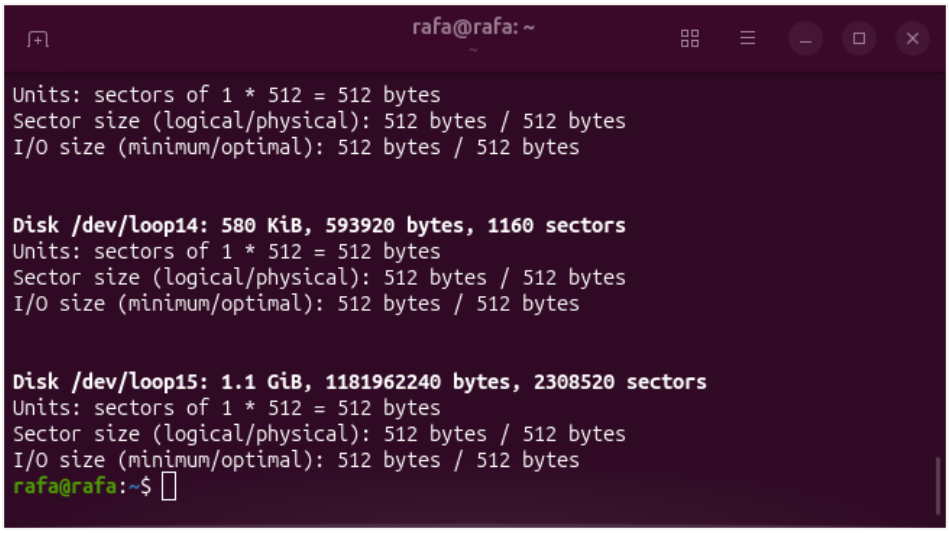


Imagen 4: Ejecución exitosa de fdisk -l tras activar el setuid

Finalmente, desde la terminal de root se restauraron los permisos originales del comando con chmod 755 /usr/sbin/fdisk, y se verificó el resultado con ls -l, confirmando que el bit setuid quedó desactivado y los permisos volvieron a -rwxr-xr-x.

Imagen 5: Restauración de los permisos originales del comando fdisk

Actividad 2 — Programa ids.c

Se creó el archivo ids.c con un programa en lenguaje C que imprime el ID de usuario real y efectivo, así como el ID de grupo real y efectivo, utilizando las funciones getuid(), geteuid(), getgid() y getegid(). El programa se compiló con el comando gcc ids.c -o ids y se ejecutó por primera vez con el usuario propietario del archivo (rafa).

Imagen 6: Creación, compilación y primera ejecución del programa ids

En esta primera ejecución, el Real user ID y el Effective user ID mostraron el mismo valor (1000), dado que el programa se ejecutó con el mismo usuario que es dueño del ejecutable.

Posteriormente, desde la terminal de root se cambió el propietario del ejecutable a root con chown root:root ids y se activó el bit setuid con chmod 4755 ids. Al verificar con ls -l se confirmaron los nuevos permisos -rwsr-xr-x con propietario root.

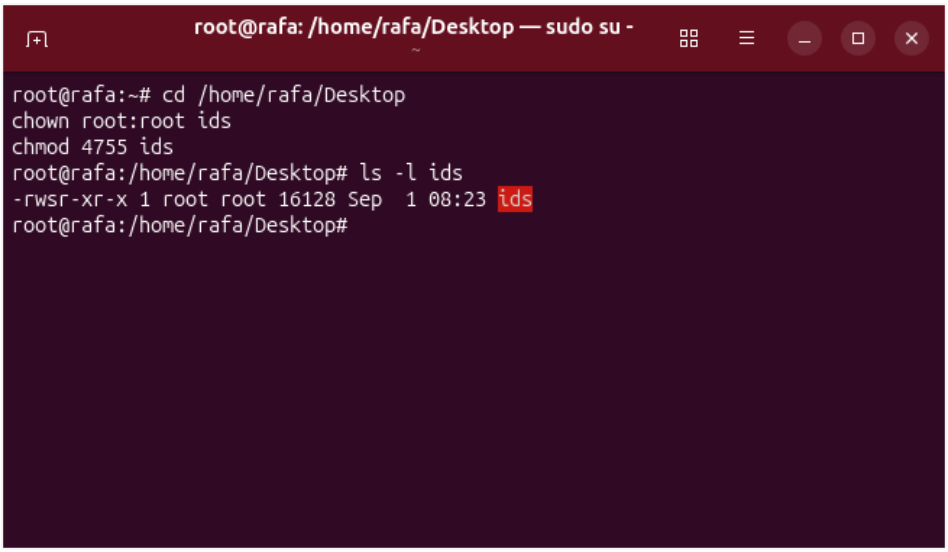


Imagen 7: Cambio de propietario y activación del setuid en el ejecutable ids

Al ejecutar nuevamente el programa desde la terminal del usuario sin privilegios, se observó la diferencia esperada: el Real user ID permaneció en 1000 (el usuario que ejecutó el programa), mientras que el Effective user ID cambió a 0, correspondiente a root. Esto demuestra que, gracias al bit setuid, el proceso adquiere los privilegios efectivos del propietario del archivo, aunque haya sido invocado por otro usuario.

Imagen 8: Ejecución de ids con setuid activo, mostrando Real UID distinto de Effective UID

Conclusión

A través de este ejercicio se comprobó de manera práctica el funcionamiento del bit setuid en Linux: un mecanismo que permite que un proceso se ejecute con los privilegios del propietario del archivo en lugar de los privilegios del usuario que lo invoca. Esto explica tanto la restricción inicial de fdisk -l para usuarios sin privilegios como la diferencia observada entre el UID real y el UID efectivo en el programa ids.c, y evidencia la importancia de administrar cuidadosamente este tipo de permisos por sus implicaciones de seguridad.

Referencias

Negus, C. (2020). *Linux Bible* (10.ª ed.). John Wiley & Sons.

Shotts, W. E. (2019). *The Linux command line: A complete introduction* (2.ª ed.). No Starch Press.