Sistemas de archivos en espacio de usuario por usuarios sin privilegios

Hugo Osvaldo Barrera

***Resumen*—Sistemas operativos de corriente principal usan núcleos monolíticos, optimizando rendimiento a costa seguridad y fiabilidad. FUSE (Filesystem in User Space) es un mecanismo mediante el cual la implementación de un sistema de archivos puede ejecutarse en espacio de usuario. Esto permite implementar sistemas de archivos que se ejecutan en un contexto de seguridad y confinamiento de fallas diferente al núcleo. Sin embargo, montar estos sistemas de archivo aún requiere privilegios de superusuario, lo cual limita su usabilidad por usuarios sin privilegios. Diseñamos e implementamos un servicio que permite a usuarios sin privilegios montar sistemas de archivos sin necesidad de escalar privilegios. La implementación (llamada "UNFUSE") permite a usuarios sin privilegios montar sistemas de archivos sin necesidad de escalar privilegios. La implementación mantiene la misma ABI (application binary interface) y API (application programming interface) que FUSE, permite su uso como sustituto directo sin necesidad de cambios de código ni recompilación en sistemas de archivos FUSE existentes.**

***Palabras Clave*—sistemas de archivos, espacio de usuario**

I. Introducción

Los sistemas operativos modernos de corriente principal utilizan núcleos monolíticos, que optimizan el rendimiento a costa de la seguridad y la fiabilidad [1]. Estos núcleos implementan la mayoría de los servicios, incluidos los servicios del sistema de archivos, dentro del propio núcleo, violando dos propiedades clave de seguridad: el confinamiento de fallos y el principio del mínimo privilegio [2].

"FUSE (File system in User space) es una interfaz de aplicación que permite montar sistemas de archivos con implementaciones que se ejecutan en el espacio de usuario, lo que posibilita el uso de sistemas de archivos que no operan en el espacio del núcleo."[3]

FUSE es suportado en Linux y variantes variantes de BSD[4] [5], por lo sistemas de archivos implementados en FUSE pueden ser fácilmente portables a varios sistemas operativos.

Montar un sistema de archivos FUSE es una operación privilegiada, dado que su implementación internamente ejecuta la llamada de sistema `mount`, la cual sólo puede ser ejecutada por el superusuario [4]. Los sistemas operativos analizados imponen una de las siguientes dos restricciones:

- Sólo el superusuario puede montar un sistema de archivo FUSE (OpenBSD, NetBSD). [4]
- Dependen de un binario setuid, permitiendo a usuarios sin privilegios ejecutar operaciones con privilegios de superusuario (Linux, FreeBSD). [3] [6]

Los ejecutables setuid habilitan que un programa se ejecute siempre con privilegios elevados y los mismos son una carga de seguridad [7]. "usar setuid-root es siempre un riesgo de seguridad" [7] y "los binarios con setuid-a-root han sido una fuente sustancial e interminable de vulnerabilidades de escalada de privilegios en sistemas Unix" [8].

En el caso de FreeBSD, otra área de preocupación es que "mount_fusefs es capaz de invocar un programa arbitrario" [6].

Para todos los sistemas operativos observados, FUSE incluye como mínimo los siguientes dos componentes: (1) un controlador del núcleo y (2) una biblioteca para implementaciones de sistemas de archivos en espacio de usuario. [3]

El controlador del núcleo de FUSE (por ejemplo, `fuse.ko` en Linux) delega las operaciones del sistema de archivos al servicio del espacio de usuario y es una parte inevitable de la base confiable en una arquitectura monolítica [3].

La biblioteca de FUSE, `libfuse`, proporciona una interfaz mediante la cual servicios de sistemas de archivos pueden interactuar con el servicio expuesto por el controlador del núcleo.

Adicionalmente, en el caso de la implementación de FUSE en Linux, los usuarios no privilegiados pueden montar un sistema de archivos a con el mencionado binario setuid `fusermount`. En estos escenarios, la libraría `libfuse` se encarga de ejecutar `fusermount` internamente.

Esta investigación busca implementar un servicio en el espacio de usuarios mediante el cual los procesos sin privilegios (y posiblemente aislados) puedan montar sistemas de archivos sin ningún tipo de escalada de privilegios. Nuestra implementación experimental busca mantener compatibilidad con implementaciones existentes, para ser utilizable como una mejora progresiva sin requerir cambios en los núcleo existentes de las implementaciones del sistema de archivos FUSE.

Para el caso de las plataformas que actualmente proporcionan un binario auxiliar setuid, buscamos eliminar la necesidad de éste y la preocupación de seguridad que conlleva, además de eventualmente permitir usar un sistema con soporte para setuid completamente deshabilitado. En el caso de las plataformas que no proporcionan tal ayudante, esto permitiría a los usuarios sin privilegios montar sistemas de archivos FUSE sin tener que introducir un nuevo binario setuid.

Los objetivos de esta investigación incluyen:

- Eliminar la necesidad de un binario setuid en implementaciones de FUSE.
- Proveer un mecanismo seguro y portable para montar sistemas de archivo FUSE como un usuario sin privilegios.
- Mantener compatibilidad de ABI y API con implementaciones existentes de `libfuse`.

II. Revisión de antecedentes

El artículo "A fully unprivileged CernVM-FS"[9] propone una implementación de sistemas de archivos completamente no privilegiados en espacio de usuario. Aunque la implementación es segura, depende de interfaces específicas de Linux para crear espacios de nombres montables y no es portable a otras plataformas Unix-like. La aplicación también requiere coordinar todos los procesos para que se ejecuten dentro un espacio de nombres de montaje.

III. Plan y metodología de la investigación

Diseñar un servicio de sistema que expone una API mediante la cual un proceso sin privilegios puede solicitar la montaje de un sistema de archivos ejecutándose en espacio de usuario, sin que el proceso solicitante eleve sus propios privilegios. El montaje en sí es realizado por el servicio, mientras que la implementación del sistema de archivos pertenece a el client.

Esta implementación se denominará "unprivileged fuse" o simplemente "unfuse".

La implementación propuesta utiliza las mismas interfaces del núcleo, por lo cual no requiere cambios en el núcleo en sí. El servicio expone un socket de dominio Unix, el cual recibe conexiones de nuevos proceso de usuarios sin privilegios que deseen montar un sistema de archivos.

Implementar una biblioteca substituto para `libfuse`, que expone la misma API y ABI que la implementación original. Esta implementación substituto, llamada `libfuse-unfuse` se conecta al socket de dominio Unix expuesta por el servicio.

Evaluar el funcionamiento y seguridad del sistema, comparando su comportamiento con el enfoque tradicional basado en binarios `setuid`.

IV. Diseño e Implementación del Servicio Unfuse

El servicio unfuse expone un socket de dominio Unix, al cual sólo podrán conectarse usuarios que tengan permisos para montar sistemas de archivos (estos permisos son simplemente administrador mediante permisos del sistema de archivo donde esté ubicado el socket).

Cuando un cliente se conecta al servicio, éste transmite la ruta al directorio destino al servicio, quien se encarga de ejecutar el montaje en sí.

Montar un sistema de archivos en una ruta controlada por un usuario sin privilegios requiere delicadeza en el control de permisos. No sólo debe el usuario ser dueño de la ruta a montar, sino que también es importante evitar condiciones de carrera entre el momento que se controlan los permisos y el momento que se realiza el montaje. Para ello, observamos que `fusermount` ejecuta una llamada a `chdir` al directorio de destino, con el fin de asegurarse que que todas las operaciones se realicen relativas al mismo directorio, indistintamente de si el archivo en la ruta original cambia entre el momento que se realiza el control y el momento que se ejecuta el montaje.

Imitaremos el mismo diseño, donde el servicio realiza una llamada a `chdir` para evitar condiciones de carrera.

Al montar un directorio debemos evitar condiciones de carrera entre el momento que controlamos permisos de un directorio y el momento que montamos un sistema de archivo sobre ese directorio. Para evitar estas condiciones de carrera, usamos el mismo enfoque que `fusermount` usamos la llamada de sistema `chdir` para cambiar el directorio actual del proceso al directorio destino donde estamos montando. Si el directorio al cual apunta la ruta originalmente provista cambia entre el momento que realizamos un control de permisos y el momento en cual llamamos a `mount(2)`, el sistema de archivo será montado en el directorio sobre el cual controlamos los permisos, indistintamente de si la ruta original aún apunta a él o a un directorio distinto.

Para la implementación de las porciones del servicio relacionadas a controles de seguridad y montar el sistema de archivos, reusamos código existente de `libfuse`. Esto nos permite rápidamente implementar una prueba de concepto sin tener que re-implementar toda la funcionalidad existente de cero.

Para la implementación de la librería substituto, usamos las firmas de las funciones en `libfuse`, con el fin de fácilmente asegurar compatibilidad de API con la librería original. Tras compilar nuestro sustituto, lo instalamos en `/usr/lib/libfuse3.so.3.16.2`, con lo cual programs que enlacen dinámicamente a `libfuse` cargarán el substituto sin requerir ningún cambio.

Cuando un cliente se conecta al socket del servicio unfuse, primero debe especificar la ruta de montaje, tras lo cual se realiza la transferencia de descriptores de archivos para FUSE. Esto permite que un cliente, en un contexto de seguridad más amplio, inicie la conexión y luego transfiera el socket a otro proceso con mayores restricciones, como puede ser un chroot, una cárcel (en FreeBSD), un espacio de nombres u otro mecanismo de aislamiento similar, que será el encargado de implementar el sistema de archivos sin necesidad de acceder al sistema de archivos global.

Intercambiamos el descriptor de archivo entre el cliente (el cual implementa el sistema de archivo), y el servicio (el cual realiza el montaje en sí), usamos la funcionalidad `SCM_RIGHTS` de `unix(7)`.

Una vez que el sistema de archivos está montado, la comunicación entre el cliente y el servidor finaliza, por lo que no hay más sobrecarga en tiempo de ejecución durante la operación del sistema de archivos.

V. Presentación y análisis de los datos

Para evaluar la efectividad de `unfuse`, se realizaron una serie de pruebas centradas en tres ejes: funcionalidad básica, aislamiento y seguridad, y compatibilidad con aplicaciones existentes basadas en FUSE.

En las pruebas de funcionalidad básica, usuarios sin privilegios lograron montar sistemas de archivos FUSE utilizando la biblioteca `libfuse-unfuse` en rutas controladas dentro de sus directorios personales. Los intentos de montar fuera de estas áreas fueron correctamente rechazados, demostrando un adecuado control de permisos.

Respecto al aislamiento, `unfuse` fue probado dentro de contenedores y entornos `chroot`, donde los clientes pudieron establecer el punto de montaje y transferir conexiones a procesos confinados sin que se produzca una elevación de

privilegios ni accesos no autorizados. Esto confirma que el servicio respeta los límites del entorno aislado.

En cuanto a seguridad, se validó que los intentos de montar en rutas no propiedad del usuario fueron denegados y que el socket Unix que expone el servicio está protegido mediante permisos del sistema de archivos. Además, se mitigaron posibles condiciones de carrera entre la verificación de permisos y el montaje, utilizando un cambio de directorio controlado.

Finalmente, la compatibilidad fue evaluada al sustituir la biblioteca original `libfuse` por `libfuse-unfuse` en aplicaciones existentes, las cuales funcionaron sin necesidad de modificaciones, incluyendo operaciones dinámicas de montaje y desmontaje.

VI. Resultados

Los resultados obtenidos indican que `unfuse` cumple eficazmente con los objetivos planteados. Permite a usuarios sin privilegios montar sistemas de archivos FUSE de manera segura, eliminando la necesidad de utilitarios con `setuid` como `fusermount`. Se mitigaron vulnerabilidades relacionadas con escalada de privilegios y condiciones de carrera usando el mismo enfoque que `fusermount`. Además, se confirmó el respeto por el aislamiento en entornos confinados como contenedores y `chroot`.

El rendimiento durante el montaje fue comparable al de las soluciones tradicionales, con una sobrecarga insignificante, y únicamente a la fase inicial de conexión al servicio. Tal como esperado, no se detectó impacto negativo en el funcionamiento de los sistemas de archivos montados.

Como limitación, se identificó la dependencia en la ejecución del servicio y la necesidad de una política adecuada para controlar el acceso al socket Unix. A su vez, esto provee a un administrador de sistema mayor flexibilidad al poder deshabilitar nuevos montajes con sólo detener al servicio.

VII. Conclusión

Este trabajo presentó el diseño, implementación y evaluación de `unfuse`, un servicio que permite montar sistemas de archivos FUSE sin privilegios elevados. A diferencia de enfoques tradicionales que requieren utilitarios con `setuid`, `unfuse` ofrece un modelo basado en un servicio intermediario que no modifica el núcleo del sistema operativo.

La solución demostró ser efectiva en escenarios tanto simples como en entornos confinados, integrándose de forma transparente con aplicaciones existentes gracias a la biblioteca `libfuse-unfuse`. La arquitectura basada en sockets Unix mantiene altos estándares de seguridad.

Para trabajos futuros, se propone integrar esta funcionalidad en `libfuse` o desarrollar una implementación más apta para entornos de producción que esta prueba de concepto.

En conjunto, `unfuse` representa una alternativa segura y moderna para el uso de sistemas de archivos en espacio de usuario bajo políticas de mínimo privilegio. Cumple con una condición necesaria, pero no suficiente, para permitir el uso de un sistema con ejecutables `setuid` deshabilitados completamente.

Referencias

[1] S. Biggs, D. Lee, and G. Heiser, "The jury is in: Monolithic os design is flawed: Microkernel-based designs improve security," in *Proceedings of the 9th Asia-Pacific Workshop on Systems (APSys '18)*. Association for Computing Machinery, 2018, p. 1–7. [Online]. Available: https://doi.org/10.1145/3265723.3265733

[2] S. Y. Lim, S. Agrawal, X. Han, D. Eyers, D. O'Keeffe, and T. Pasquier, "Securing monolithic kernels using compartmentalization," 2024. [Online]. Available: https://arxiv.org/abs/2404.08716

[3] The Linux Kernel Community, "FUSE — The Linux Kernel Documentation," 2025. [Online]. Available: https://www.kernel.org/doc/html/v6.14/filesystems/fuse.html

[4] The OpenBSD Project, *fuse_mount(3) – mount or dismount a FUSE file system*, 2018. [Online]. Available: https://man.openbsd.org/fuse_mount.3

[5] Benjamin Fleischer and the macFUSE team, "macfuse — file system integration made easy," macFUSE Project. [Online]. Available: https://macfuse.github.io

[6] The FreeBSD Project, *mount_fusefs(8) – mount a Fuse file system daemon*, 2025. [Online]. Available: https://man.freebsd.org/cgi/man.cgi?query=mount_fusefs&manpath=FreeBSD+14.2-RELEASE+and+Ports

[7] D. Dykstra, "Apptainer without setuid," in *EPJ Web of Conferences*, vol. 295. EDP Sciences, 2024. [Online]. Available: https://doi.org/10.1051/epjconf/202429507005

[8] B. Jain, C.-C. Tsai, J. John, and D. E. Porter, "Practical techniques to obviate setuid-to-root binaries," in *Proceedings of the Ninth European Conference on Computer Systems*. Association for Computing Machinery, 2014, pp. 1–14. [Online]. Available: https://doi.org/10.1145/2592798.2592811

[9] J. Blomer, D. Dykstra, G. Ganis, S. Mosciatti, and J. Priessnitz, "A fully unprivileged cernvm-fs," in *EPJ Web of Conferences*, vol. 245. EDP Sciences, 2020, p. 5. [Online]. Available: https://doi.org/10.1051/epjconf/202024507012