



Accueil / News / Logiciels Libres / Copy Fail (CVE-2026-31431) : Synthèse technique sur cette faille Linux

News - Logiciels Libres

Copy Fail (CVE-2026-31431) : Synthèse technique sur cette faille Linux

Adrien.D | 03/05/2026 | Logiciels Libres | 6 Commentaires | 5502



Bonjour à tous,

Dans cet article, que je prends le temps d'écrire le 3 mai (donc avec les infos que j'ai au 3 mai), je souhaite vous parler plus en détail de cette vulnérabilité Copy Fail.

Je n'ai pas vu d'infos techniques en français sur le sujet, et j'ai voulu approfondir mes connaissances et le fonctionnement de cette faille. J'espère que ça vous servira.

Tout en bas, je vous mets les sources associées à mes recherches.

Il y a beaucoup d'incompréhensions dessus, et c'est une vulnérabilité que je considère comme nettement plus grave que ce que la plupart des analyses superficielles laissent entendre. Classifiée **CVE-2026-31431** avec un score CVSS de 7.8/10, elle permet à n'importe quel processus tournant sur une machine Linux de devenir root, et ce, sur l'intégralité des distributions majeures depuis 2017. Le proof-of-concept public fait 732 octets de Python. En une commande, on devient root. C'est incroyable !

Anatomie d'une faille : une conjonction, pas un bug isolé

Petite pensée à Olivier Poncet avec la formulation "Anatomie d'une faille"

Ce qui rend CopyFail particulièrement intéressante d'un point de vue technique, c'est qu'elle n'est pas le résultat d'un bug unique comme on l'a parfois vu. C'est la conjonction de trois évolutions indépendantes du noyau Linux qui, ensemble, créent une faille exploitable.

Voilà la chronologie en quelques phrases :

2011 : Le wrapper cryptographique AEAD

En 2011, un wrapper cryptographique AEAD (Authenticated Encryption with Associated Data) est introduit dans le noyau pour gérer IPsec (le protocole qui chiffre chaque paquet réseau individuellement, très utilisé dans les VPN).

Ce wrapper, a un comportement un peu crade mais toléré : lors du déchiffrement d'un bloc, il écrit 4 octets en dehors de sa zone mémoire déclarée. C'est du dépassement de tampon, certes, mais à ce stade c'est totalement confiné à l'intérieur du noyau. Il n'y a aucune interface avec les processus utilisateurs. Aucun risque pratique.

2015 : Les sockets AF_ALG

Quatre ans plus tard, une nouvelle famille de sockets fait son apparition : les AF_ALG sockets.

Leur rôle ? Offrir à l'espace utilisateur une interface directe vers le sous-système cryptographique du noyau (y compris vers le handler AEAD décrit ci-dessus. Désormais, un processus non-privilegié peut ouvrir un socket, demander au noyau de chiffrer ou déchiffrer des données via AEAD, et récupérer le résultat. C'est toujours fonctionnel, mais on vient d'exposer vers l'extérieur un composant qui avait jusqu'ici une petite bizarrerie interne.

2017 : Le commit problématique (72548b093e3)

C'est ici que tout bascule !

En 2017, un commit introduit dans `algif_aead.c` ce qui ressemble à une optimisation de performance : la mémoire d'entrée et la mémoire de sortie du traitement AEAD sont fusionnées et donc on traite les données en place (in-place processing).

Le problème : cela crée un chemin dans lequel les 4 octets qui étaient précédemment écrits de façon inoffensive en dehors des limites, se retrouvent maintenant à déborder dans des zones mémoire contrôlables depuis l'espace utilisateur. La faille est en place. Elle sera donc introduite à ce moment là, sans être détectée, pendant 9 ans !

A propos du mécanisme d'exploitation : le page cache comme vecteur

Pour comprendre comment l'exploitation fonctionne concrètement, il faut maîtriser deux concepts fondamentaux du noyau Linux. Je vais le faire assez court mais je pense assez complet pour illustrer le rôle qu'à le page cache dans cette affaire.

Le **page cache** est une portion de la RAM que le noyau dédie au stockage en mémoire des contenus de fichiers lus depuis le disque. Quand on lit `/usr/bin/su`, son contenu est chargé en RAM dans le page cache. La prochaine fois qu'un processus en a besoin, le noyau ne va pas relire le disque, il sert directement depuis la RAM.

On le sait, la RAM est beeeeeeeaucoup plus rapide qu'un disque, même un SSD ! Donc autant l'utiliser et en profiter !

Ce qui est problématique ici : le page cache est global au système. Il n'est pas par processus, ni par conteneur (ce qui pose souci si on a du docker par exemple). Il est au niveau du noyau, partagé par tout le monde.

L'appel système `splice()` est un appel système qui permet de déplacer des données entre deux file descriptors sans repasser par l'espace utilisateur. Couplé aux AF_ALG sockets, il devient le vecteur parfait : on peut manipuler des données dans le noyau sans jamais exposer un pointeur ou une adresse en espace utilisateur.

Y a une p'tite page de manuel qui explique bien ça : <https://www.man7.org/linux/man-pages/man2/splice.2.html>

La chaîne d'attaque

Voici ce que fait l'exploit en pratique :

Etape 1 : Ouverture d'un socket AF_ALG avec le type AEAD.

Etape 2 : Lecture d'un binaire SUID (par exemple `/usr/bin/su`) dans le page cache. Ce binaire est en mémoire, sa copie disque est intouchée.

Etape 3 : Grâce au débordement des 4 octets dans `algif_aead.c` et à `splice()`, l'exploit écrit des données contrôlées dans le page cache, à un offset précis correspondant au contenu du binaire en mémoire. Il le fait par blocs de 4 octets.

Etape 4 : Le contenu en mémoire de `/usr/bin/su` est maintenant modifié (remplacé par un shellcode qui lance un shell).

Etape 5 : La copie sur disque est intacte. Le noyau ne sait pas que le page cache a été altéré.

Etape 6 : Quand n'importe qui exécute `su`, le noyau ne relit pas le disque mais il charge la version en mémoire, la version compromise. `su` étant SUID root, le shell se lance en tant que root.

Petite note sur le SUID : Le SUID est un bit de permission qui permet à un utilisateur d'exécuter un programme avec les privilèges du propriétaire de ce fichier le temps de son exécution. `su` ayant comme propriétaire "root" le programme se lance en tant que root !

C'est le "s" du "rws" qui identifie le SUID. Voici sur ma Fedora le résultat :

Code BASH :

Copier vers le presse-papier

La seule façon d'invalider l'exploit est de vider explicitement le page cache (sinon ça se fait au bout d'un certain temps) à travers `/proc/sys/vm/drop_caches`. Mais attention ça vide tout le cache et ça va dégrader les performances de la machine.

La gravité de cette faille est réelle : pourquoi 7.8 est trompeur selon moi ?

Le score CVSS de 7.8 reflète le fait qu'une exécution de code locale est nécessaire. C'est techniquement juste. Mais c'est là que la plupart des analyses s'arrêtent, et c'est là qu'elles sous-estiment massivement la menace.

Ce que "local" signifie vraiment ici : il ne s'agit pas d'un accès shell interactif, ni d'un compte utilisateur sur la machine. Entendez par là qu'on peut l'exploiter sans se connecter avec une session utilisateur en ssh sur la machine.

Je m'explique : il s'agit de n'importe quel processus en cours d'exécution. Par exemple, apache httpd peut exploiter cette faille, un interpréteur PHP aussi, un module vérolé nodejs aussi, ou un pipeline, en fait tout ce qui peut lancer un processus sur la machine.

Alors OUI, CopyFail seul n'est pas exploitable à distance. Mais CopyFail n'est jamais seul dans un environnement réel. Le scénario d'attaque possible peut être le suivant : 1 Remote Code Execution (RCE) dans n'importe quelle application + CopyFail = root complet sur la machine !!!

Donc, en combinant ce vecteur avec CopyFail, on obtient une élévation de privilège complète sans aucune interaction avec un compte utilisateur.

Donc on comprends que les hébergeurs, les machines hébergeant des conteneurs, les environnements mutualisés avec des Joomla, Drupal ou WordPress avec des plugins vérolés ou pas à jour tous sont des cibles directes.

Vu que les conteneurs sont à la mode, rappelons que le page cache est au niveau du noyau, pas du conteneur. Cette faille est donc également un vecteur de "container escape" : depuis un conteneur Docker ou même podman rootless, on peut modifier le page cache du noyau hôte et obtenir root sur la machine physique. Vous comprenez donc le problème 😊

Pas d'antivirus, pas de conditions

Ce qui fait le cauchemar des sysadmins aujourd'hui, c'est que la modif étant dans le page cache, un antivirus n'y verra rien du tout. Seuls des EDR (comme je le dis toujours, ces fameux super antivirus) peuvent contrer ce genre d'attaque, car les EDR analysent les comportements des processus, des processus entre eux et de leurs interactions, en plus des fichiers ouverts, modifiés, etc.. D'ailleurs, sur des systèmes vulnérables, c'est l'EDR qui chez nous nous a sauvé les fesses, lors de mes tests, en attendant que le patch arrive chez Red Hat !

Mais les EDR ont un coût. Pour des PME ou des particuliers qui ont leur VPS ou leur homelab, on n'a pas d'EDR.

Aussi, il n'y a aucune condition de version à respecter. Les LPE (en français les Escalades de Privilèges en Local) classiques nécessitent souvent un alignement précis de version noyau ou de décalage mémoire ou d'un contexte précis. Ici, le même script Python de 732 octets fonctionne sur Ubuntu 24.04, Red Hat Enterprise Linux 9 et 10, Suse Linux Enterprise 16, ... donc l'exploit est "toujours fonctionnel"

ici, même si le code fourni est en python, c'est pour le PoC, mais vous pouvez reproduire la même chose en Rust même !

La découverte : une intuition assistée par IA

Je l'ai dit dans mon dernier article et dans ma vidéo, la découverte est fascinante.

Depuis la publication de la vidéo, j'ai pris le temps de me renseigner sur le "découvreur". Merci à Korben pour son article d'ailleurs !

La faille a été trouvée par Li Tang (ou Taang Lee), chercheur chez Théorie, une société de sécurité offensive réputée ayant notamment fini dans le top 3 du challenge AI Cyber Challenge du DARPA. Son approche est intéressante : il avait une intuition que le sous-système crypto du noyau Linux comportait des chemins d'accès non explorés susceptibles de permettre une élévation de privilège. Il a utilisé l'outil interne de Théorie, AI Xint Code, pour explorer le sous-système, non pas pour que l'IA trouve la faille à sa place, mais pour lui faire gagner du temps. Une analyse qui lui aurait pris une journée a été faite en une heure. Un bon exemple de ce que l'IA apporte vraiment à la recherche offensive : de l'accélération au service d'un expert, pas un remplacement de l'expertise !

La gestion de la divulgation : le vrai scandale

C'est là que les choses deviennent problématiques, et que j'estime que la situation est bien plus grave qu'une simple faille technique. Et vous allez comprendre pourquoi au moment de la publication "publique" de la faille, les distributions n'étaient pas prêtes.

Selon le site de CopyFail :

23 mars : Bug signalé à l'équipe sécurité du noyau Linux

24 mars : Accusé de réception

25 mars : Premiers patchs proposés et analysés

1er avril : Correctif principal intégré dans la branche de dev (RC de 7.0)

11 avril : Versions 6.18.22 et 6.19.12 corrigées

22 avril : CVE attribuée

29 avril : Divulgation publique ... et personne n'est prêt !

Personne n'est prêt : un problème de coordination

La mailing list Linux-distros est le mécanisme prévu pour permettre aux distributions (Debian, Ubuntu, Red Hat, SUSE, Oracle Linux, etc...) de travailler ensemble sur des problèmes de sécurité de façon non publique. Il me semble qu'il y a une période "d'embargo" de 14 jours, mais plus sûr.

L'idée c'est de pouvoir backporter les correctifs sur leurs branches du noyau maintenues, les compiler, les tester, puis les déployer de façon coordonnée. Car en effet, Ubuntu, Red Hat ne suivent pas forcément les mêmes versions du noyau que les officielles.

Notez qu'en 2026, Red Hat maintient (officiellement) RHEL10, RHEL 9 et RHEL 8. Canonical maintient Ubuntu 16.04 18.04 20.04 22.04 24.04 et 26.04. Il y a probablement la même chose chez Debian et SUSE mais je connais moins leur cycle de vie exact.

Les distributions vivent plus longtemps que les périodes LTS des noyaux Linux, et ceux-ci ne montent pas en version au cours du cycle de vie de la distribution.

Ici, les distributions n'ont pas été notifiées via ce canal. Elles ont appris l'existence de la faille en même temps que nous en fait, soit le 29 avril.

Au moment de la divulgation publique, seules les versions 7.0, 6.18 et 6.19 du noyau avaient un correctif.

Les autres versions LTS du noyau (6.12, 6.6, 6.1, 5.15 et 5.10) n'en avaient pas encore.

Debian utilise par exemple le 6.12 pour trixie, le 6.1 pour bookworm et 5.10 pour bullseye qui est old old stable.

Les correctifs ont dû être développés en urgence par l'équipe noyau et les distributions en même temps que le PoC tournait déjà partout.

Le correctif lui-même a été commis de manière à ressembler à une optimisation ou un fix mineur le 11 avril (je vous mets le lien en dessous de la mailing list du kernel). C'est une pratique parfois légitime pour éviter qu'un attaquant ne l'analyse avant le déploiement, mais qui dans ce cas a aussi contribué à retarder la prise de conscience générale.

Mitigation et correctifs

Prévisualiser

* Code de vérification

Combien font 4 + cinq ? (écrire le nombre en lettres)

Envoyer

Défaut

Visiteur
Visiteur



04/05/2026 à 10h48

#1977

Très bon article, j'ai juste repéré une typo : nouay -> noyau.

Je suis d'accord pour dire que ce genre de faille est bien plus dangereux que sa notation le laisse penser. Il faut s'attendre à trouver d'autres surprise avec les analyses par IA

Visiteur
Visiteur



04/05/2026 à 11h38

#1978

N'y a-t-il pas de coût à payer pour se passer du module algif_aead lors de la mitigation ?

Adrien.D
Administrateur



04/05/2026 à 15h27

#1979

Coquille corrigée !

Visiteur
Visiteur



04/05/2026 à 19h37

#1980

Merci pour ce detail très instructif !

Visiteur
Visiteur



04/05/2026 à 21h59

#1981

Très bon article.

Le weekend prochain, tu vas le passer à analyser le weekend du premier mai cauchemar du dDos iranien sur les services de canonical?

Le mec qui était d'astreinte sur son bateau de pêche a dû apprécier..

dyonysos
Visiteur



05/05/2026 à 14h44

#1982

Merci pour l'analyse détaillée mais néanmoins très compréhensible de cet exploit.