La latence d'incident. Pourquoi un artefact local invalidé continue d'agir quand on l'a laissé en place

Suite de l'article sur la CVE-2026-31431. Le cas du verrou sysctl résiduel sur kernel WSL2 6.6.87.2 comme terrain empirique d'un concept opératoire : la latence d'incident comme angle mort de la sécurité opérationnelle quand un artefact local invalidé n'a pas été désinstallé dans la session qui l'a produit…

…Quand les bonnes pratiques de maintien en conditions opérationnelles se rappellent à vous.

Introduction

L'article précédent posait l'écart de promotion comme distance mesurable entre le périmètre éprouvé d'une recommandation publique et le périmètre où elle est prescrite. Il visait les artefacts reçus de l'extérieur, où l'autorité émettrice n'a pas accès à la cible de l'organisation utilisatrice. Le terrain empirique était CVE-2026-31431, dite Copy Fail, et la démonstration tenait à montrer que quatre variantes de la recommandation publique avaient été comportementalement invalidées sur kernel WSL2 6.6.87.2 dans la session du 30 avril 2026. La mitigation effective avait été produite ailleurs, par filtre seccomp BPF chirurgical sur le syscall socket(AF_ALG, …), validée par invariant kernel /proc/<pid>/status.

Cet article concerne ce qui s'est passé après. Plus précisément, ce qui s'est passé pendant les 24 jours suivants, en silence, dans un coin du système où personne ne regardait, et qui s'est rappelé brutalement à l'attention le 24 mai 2026 à l'occasion d'une coupure d'électricité du réseau public ayant provoqué l'arrêt brutal de la machine hôte.

La thèse tient en une phrase. *Un artefact local invalidé qui n'a pas été désinstallé continue de produire des effets indépendants de la vulnérabilité qu'il prétendait corriger, et la latence entre son activation et son incident d'origine empêche l'attribution opérationnelle.*

Le domaine de validité est circonscrit. L'argument concerne les artefacts de mitigation locale appliqués en réponse à une vulnérabilité publique, dans des environnements où la cadence d'application précède la disponibilité d'un patch officiel. Il est démontré sur le

cas particulier d'une directive sysctl kernel.modules_disabled=1 appliquée le 30 avril 2026 dans le contexte Copy Fail, et sur l'incident qu'elle a produit le 24 mai 2026 après une coupure électrique. Il se généralise aux mitigations qui modifient l'état persistant du système (sysctl, modules, capabilities, namespaces, services) sans procédure de rollback documentée et testée. Il ne concerne ni la conduite générale de la cybersécurité opérationnelle, ni la doctrine de patch management, qui ont leurs propres cadres établis.

La démonstration suit quatre mouvements : l'artefact local et son cycle de vie complet, la séquence empirique 30 avril vers 24 mai, le concept de latence d'incident, le critère opérationnel qui en découle.

Mouvement 1. L'artefact local et son cycle de vie complet

L'article précédent traitait des artefacts reçus de l'extérieur. Mais une organisation qui exploite un environnement de production produit aussi ses propres artefacts, particulièrement dans les fenêtres d'urgence où la recommandation publique est invalidée et où il faut concevoir une mitigation alternative en quelques heures. Ces artefacts locaux ont une généalogie différente. L'autorité émettrice est l'organisation elle-même. Le périmètre éprouvé coïncide théoriquement avec le périmètre prescrit, puisque l'artefact est produit pour la cible exacte sur laquelle il s'applique. L'écart de promotion devrait être nul. Sur ce point, la doctrine implicite a raison plus souvent que pour les artefacts externes.

Mais cette absence d'écart de promotion à l'application ne dit rien du cycle de vie de l'artefact après son application. Et c'est ici que se loge un angle mort distinct, qui mérite un concept propre.

La distinction qui tranche est simple. *Appliquer un artefact* n'est pas *terminer un artefact*. L'application installe l'artefact dans l'état persistant du système. La terminaison nettoie l'artefact quand il n'est plus pertinent, qu'il a été remplacé par une autre mitigation, ou qu'il a été invalidé comportementalement. La terminaison n'est pas optionnelle dans la phase d'expérimentation, qui est précisément la phase où la majorité des artefacts sont invalidés. Mais la doctrine opérationnelle traite généralement l'application comme un acte et la terminaison comme une conséquence implicite. C'est exactement l'inverse. L'application produit un événement observable et tracé dans un changelog. La terminaison demande une procédure explicite, documentée, et testée. Le déséquilibre entre les deux est structurel.

Quand une mitigation locale est testée et déclarée inopérante, deux trajectoires opérationnelles s'ouvrent. La première consiste à passer à la tentative suivante en laissant l'artefact en place, dans l'idée qu'il ne fait pas de mal puisqu'il a été invalidé. La seconde consiste à le désinstaller proprement avant de tester la tentative suivante. La première trajectoire est dominante en pratique, parce qu'elle est plus rapide et que la

pression temporelle de la fenêtre d'urgence pèse plus que l'hygiène d'environnement. La seconde demande une discipline supplémentaire qui ne produit aucune valeur immédiate. La première est exactement celle qui produit la latence d'incident.

Un artefact invalidé est invalidé sous un certain régime de fonctionnement. Modifier ce régime peut révéler des effets que l'invalidation comportementale n'avait pas testés, parce que personne ne savait qu'ils étaient là.

Mouvement 2. La séquence empirique 30 avril vers 24 mai

L'Article VI documente quatre tentatives successives de mitigation publique sur Copy Fail, toutes invalidées par le même test syscall. La tentative 2 mobilisait le verrou sysctl kernel.modules_disabled=1. Le verrou interdit le chargement de modules kernel à l'exécution, ce qui était censé empêcher le rechargement automatique d'algif_aead par request_module(). L'invalidation comportementale a montré que le verrou n'empêchait pas le primitif d'amorçage, parce que les modules étaient déjà chargés en mémoire et que le verrou ne purgeait pas la mémoire. La tentative 2 a été documentée comme inopérante dans l'Article VI, et la démarche est passée à la tentative 3 puis 4, jusqu'à la mitigation seccomp BPF qui a tenu.

Mais le fichier /etc/sysctl.d/99-cve-2026-31431.conf contenant la directive kernel.modules_disabled=1 est resté en place. Il avait été créé pour la tentative 2, il a survécu à son invalidation, et il n'a pas été retiré dans la session qui l'a invalidé.

Pendant 24 jours, du 30 avril au 24 mai, le verrou est resté actif au sens sysctl. Mais son effet pratique était masqué. Les modules iptable_nat, iptable_filter, nf_conntrack, br_netfilter et ip6_tables étaient déjà chargés en mémoire kernel WSL2 depuis l'installation initiale de Docker en mars 2026. Tant que la mémoire kernel n'était pas vidée par un cold reboot, les modules restaient disponibles, et les services qui en dépendaient (Docker 29.3.0, Tailscale, containerd plugin erofs, conteneur Open WebUI) fonctionnaient nominalement. Le journal Docker du 6 mai 2026 montre un daemon parfaitement opérationnel, avec pull d'image, networking bridge actif, gestion de containers. Aucun symptôme apparent de l'artefact résiduel.

Le 24 mai 2026 à 22h13, une coupure du réseau électrique public a provoqué l'arrêt brutal de la machine hôte Windows Server 2025 qui faisait tourner WSL2. Le retour du courant a redémarré Windows. La tâche planifiée WSL2-AI-Startup, configurée sur trigger boot, a tenté d'invoquer wsl.exe -d Ubuntu-24.04 -- /home/jeromev/scripts/start-all.sh à 22h13:11, soit 8 secondes après le boot Windows. Pour des raisons qui restent à investiguer en post-mortem séparé, la commande a échoué silencieusement, et WSL2 n'a réellement démarré qu'à 23h41 sur intervention manuelle d'ouverture de terminal.

Pendant ces 88 minutes, la démo publique twingital-ventures.com/realisations/toxtwin a été indisponible. Cette indisponibilité est un sujet propre à traiter par ailleurs. Ce qui nous intéresse ici, c'est ce qui s'est produit quand WSL2 a finalement démarré.

Au boot frais, la mémoire kernel WSL2 a été initialisée vierge.

- Les modules netfilter pré-chargés depuis mars ont disparu avec la mémoire.
- Le verrou kernel.modules_disabled=1, persistant dans /etc/sysctl.d/, a été appliqué par systemd-sysctl.service pendant le boot.
- À partir de ce moment, le kernel WSL2 refusait toute charge de module, y compris pour iptable_nat qui était nécessaire à Docker.
- Le daemon Docker, démarré par systemd, a tenté la création de sa chaîne NAT, a reçu en retour l'erreur iptables v1.8.10 (legacy): can't initialize iptables table 'nat': Table does not exist, et a planté en boucle pendant trois tentatives avant que systemd ne le marque comme failed.
- Tailscale, démarré en parallèle, a échoué de la même manière sur ses règles IPv6 avec le message modprobe: ERROR: could not insert 'ip6_tables': Operation not permitted. Le plugin erofs de containerd a refusé de charger pour la même raison.

Quatre symptômes apparemment distincts, dans quatre services indépendants. Une seule cause racine. La cause racine n'était pas la coupure électrique. La coupure n'a fait que révéler une condition qui était présente depuis 24 jours. Sans la coupure, le système aurait continué à fonctionner indéfiniment dans son état masqué, jusqu'à ce qu'un autre événement de cold reboot (mise à jour Windows, redémarrage planifié, arrêt manuel) produise le même résultat.

Le diagnostic a pris environ deux heures, dont l'essentiel à explorer des pistes erronées : bug Docker 29, régression du kernel WSL2 Microsoft, conflit nftables vs iptables-legacy, problème de capabilities namespace. Ces pistes étaient toutes plausibles, et certaines étaient localement vraies (le kernel WSL2 ne permet effectivement pas le modprobe runtime hors d'un contexte privilégié). Mais aucune n'était la cause racine. La cause racine était un fichier de 26 octets dans /etc/sysctl.d/, créé pour une mitigation invalidée 24 jours plus tôt, jamais nettoyé.

Mouvement 3. La latence d'incident comme concept opératoire

Le cas Copy Fail prolongé rend explicite ce qui restait implicite dans la doctrine du port de promotion et de l'écart de promotion. Tout artefact persistant introduit dans l'état du système possède une dimension temporelle propre, distincte de son moment d'application et de son moment d'invalidation. Cette dimension n'a pas de nom dans la

doctrine opérationnelle standard, et c'est précisément pourquoi elle est ignorée. Je propose de la nommer la *latence d'incident*.

La latence d'incident désigne la distance temporelle entre l'activation d'un artefact persistant et l'incident qu'il produit, dans la mesure où cette distance brouille l'attribution causale. La définition est restrictive sur deux points. D'abord, elle suppose un artefact qui modifie l'état du système, pas une opération transitoire. Une commande shell qui s'exécute et se termine ne produit pas de latence d'incident. Une directive sysctl, un module noyau chargé, une règle iptables, un fichier de configuration en place, un service systemd activé, oui. Ensuite, elle suppose un brouillage de l'attribution. Si l'artefact produit son incident immédiatement, dans la session qui l'a installé, la cause est évidente. Si l'incident se déclenche un mois plus tard à l'occasion d'un événement décorrélé, l'attribution naturelle se porte sur l'événement, pas sur l'artefact.

La distinction qui tranche ici sépare deux régimes causaux que la pratique opérationnelle confond systématiquement. *La cause événementielle* est ce qui se voit. La coupure électrique, le redémarrage planifié, la mise à jour kernel, le pic de charge, tout événement observable qui précède immédiatement l'incident. C'est cette cause que l'investigation post-mortem habituelle reconstitue, parce qu'elle est tracée dans les logs et qu'elle a une signature temporelle nette. *La cause d'état* est ce qui agit. C'est la condition de possibilité de l'incident, présente dans le système avant l'événement, et qui le rend possible. La cause événementielle est nécessaire mais non suffisante pour produire l'incident. La cause d'état est nécessaire et coproduit la suffisance.

Dans le cas du 24 mai, la cause événementielle est la coupure électrique. La cause d'état est le fichier 99-cve-2026-31431.conf. Sans la coupure, l'incident ne se produit pas. Mais sans le fichier, la coupure n'aurait produit qu'une interruption transitoire suivie d'une remontée nominale des services. La cause d'état est ce qui a transformé un événement banal en incident structurel.

Trois propriétés caractérisent la latence d'incident :

1. *Le silence pendant la phase de masquage*. Pendant que l'effet de l'artefact est neutralisé par un autre élément de l'état système (dans le cas présent, les modules en mémoire), l'artefact ne produit aucun symptôme observable. Aucun log ne signale sa présence. Aucun test fonctionnel ne le détecte. L'organisation peut conduire des audits, des revues de configuration, des tests de pénétration, et l'artefact passera inaperçu parce qu'il est *opérationnellement transparent* tant que la condition de masquage tient.
2. *Le déclenchement par événement décorrélé*. L'incident se produit à l'occasion d'un événement qui n'a aucun rapport apparent avec l'artefact résiduel. La coupure électrique n'a rien à voir avec une mitigation crypto. Une mise à jour de kernel n'a rien à voir avec une règle iptables résiduelle. Un redémarrage planifié n'a rien à voir avec un service mort-vivant. La décorrélation apparente entre

l'événement déclencheur et l'artefact résiduel est précisément ce qui produit le brouillage d'attribution.

3. *L'attribution erronée par défaut*. L'investigation naturelle se concentre sur l'événement déclencheur. La coupure électrique devient le sujet du post-mortem. Les remédiations envisagées concernent la résilience face aux coupures, l'amélioration des séquences de boot, le hardening des services. Toutes ces remédiations peuvent être justifiées par ailleurs, mais aucune ne traite la cause d'état. L'artefact résiduel reste en place. L'incident peut donc se reproduire, à l'occasion d'un autre événement décorrélé, avec une autre signature de surface qui orientera vers une autre investigation erronée.

Ces trois propriétés expliquent pourquoi la latence d'incident n'est pas un problème rare ou anecdotique. Elle est la conséquence prévisible de l'asymétrie structurelle entre application et terminaison d'artefact qu'on a posée au mouvement 1. Tant que cette asymétrie persiste, la latence d'incident persiste. Et tant qu'elle n'est pas nommée, elle ne peut pas être investiguée.

Mouvement 4. Le critère opérationnel. La réversibilité explicite comme condition de l'expérimentation

Le concept de latence d'incident appelle un critère opérationnel, qui s'articule sans contradiction avec le port de promotion (Article V) et l'écart de promotion (Article VI). Mais il ne s'y ajoute pas comme une exigence supplémentaire qu'on aurait pu omettre. Il en est la condition de tenue dans le temps.

Le port de promotion suppose qu'un artefact est promu de l'expérimentation vers la dépendance normative après franchissement délibéré et tracé. Cette doctrine suppose implicitement qu'un artefact non promu reste dans l'expérimentation, ou en sort par retrait. Mais dans la pratique observée, un artefact non promu peut rester *en place* dans le système opérationnel sans être ni promu ni retiré, dans un statut hybride qui n'a pas de nom dans la doctrine. C'est ce statut hybride qui produit la latence d'incident. Le port de promotion comme dispositif vérificationnel devient un guichet à sens unique si l'organisation n'a pas de procédure miroir de retrait.

L'écart de promotion désigne la distance entre périmètre éprouvé et périmètre prescrit pour les artefacts externes. Un raisonnement parallèle s'applique aux artefacts internes : la distance entre périmètre éprouvé au moment de l'application et périmètre effectif au moment où l'artefact agit. Cette distance est elle aussi non nulle, parce que l'état du système au moment où l'artefact agit peut être différent de l'état au moment où il a été éprouvé. Dans le cas du 24 mai, la mémoire kernel pleine de modules était l'état au moment de l'éprouvement (30 avril). La mémoire kernel vide était l'état au moment où

l'artefact a agi (24 mai). L'artefact a agi dans un état où il n'avait pas été éprouvé, parce que personne n'avait pensé à tester ce qu'il faisait dans cet état.

Le critère qui en découle se formule ainsi : *tout artefact persistant doit avoir une procédure de retrait documentée et testée avant son application, pas après*. Cette formulation est exigeante, et elle l'est délibérément. Elle reconnaît que la phase d'urgence ne se prête pas à la production simultanée d'une procédure de retrait propre. Elle exige donc que la procédure de retrait soit pensée *avant* la phase d'urgence, comme partie intégrante de la boîte à outils de mitigation locale. Une organisation qui n'a pas de procédure pré-rédigée pour retirer une directive sysctl ne devrait pas appliquer de directive sysctl. Une organisation qui n'a pas de procédure pour décharger un filtre seccomp BPF ne devrait pas attacher de filtre seccomp BPF.

Le corollaire opérationnel est plus contraignant encore : *toute mitigation locale invalidée doit être nettoyée dans la session opérationnelle qui l'a invalidée*. Pas reportée à une session ultérieure de cleanup. Pas inscrite dans un backlog de dette technique. Nettoyée immédiatement, dans la même session, par la même personne, avec le même contexte mental. Le report est exactement ce qui produit la latence d'incident. Le contexte mental qui permet de comprendre pourquoi un artefact peut être retiré sans risque est éphémère. Une fois la session terminée, ce contexte se dissout, et le retrait devient une opération autonome qui demande de reconstituer l'historique, ce qui coûte plus cher que le nettoyage immédiat n'aurait coûté.

L'objection prévisible à ce critère est qu'il déplace la charge sur les opérateurs qui interviennent dans la fenêtre d'urgence, en ajoutant à la pression du moment l'exigence d'une hygiène supplémentaire. L'objection est juste, et c'est le point. La charge n'a jamais été ailleurs. Elle était simplement reportée sur le futur opérateur, généralement inconnu, qui ferait face à l'incident produit par l'artefact résiduel sans contexte pour le comprendre. La doctrine de la latence d'incident ne déplace pas la charge, elle rend visible la charge qui était déjà là et que la pratique courante laissait s'accumuler en silence.

Une seconde objection prévisible est que le critère, appliqué strictement, conduit à une paralysie opérationnelle dans les fenêtres d'urgence où le temps manque pour produire des procédures de retrait propres avant chaque tentative. L'objection est partiellement juste. Le critère doit être appliqué avec discernement sur la nature de l'artefact. Une commande shell transitoire ne demande pas de procédure de retrait. Une règle iptables ajoutée à chaud demande au minimum une note de session. Une directive sysctl persistante demande une procédure de retrait pré-rédigée. Un module noyau chargé demande un test de déchargement. Un service systemd activé demande une procédure de désactivation testée. La granularité du critère doit être proportionnée à la persistance et au rayon d'effet de l'artefact. Mais elle ne peut pas être nulle pour les artefacts à fort rayon d'effet.

L'expérimentation doctrinale se distingue de la production par sa réversibilité explicite, pas par son intention. L'intention de tester sans engagement long n'est pas une protection si le test laisse des traces persistantes que personne ne nettoie. La réversibilité explicite, c'est-à-dire la capacité documentée et testée à revenir à l'état antérieur, est la propriété qui distingue un environnement d'expérimentation d'un environnement de production qui ignore qu'il est en production. Sans cette propriété, l'expérimentation est une mise en production déguisée, et tous les engagements de fiabilité, de sécurité et de gouvernance qui devraient s'appliquer à un environnement de production s'appliquent en réalité, sans que l'organisation en ait conscience.

Conclusion

Un artefact appliqué dans une fenêtre d'urgence, déclaré inopérant après test comportemental, et laissé en place par défaut d'une procédure de retrait, produit un risque dont la signature temporelle est différente du risque que la mitigation visait à corriger. La vulnérabilité d'origine, dans le cas Copy Fail, n'a jamais été exploitée sur l'environnement considéré, parce que la mitigation effective seccomp BPF tenait. Mais l'artefact résiduel de la tentative invalidée a produit, 24 jours plus tard, une interruption complète de services indépendants de la crypto, à l'occasion d'un événement banal. L'incident n'a pas été causé par Copy Fail. Il a été causé par la trace que Copy Fail a laissée dans le système quand l'organisation a cessé de s'occuper de Copy Fail.

Cette dissymétrie entre la vulnérabilité d'origine et l'incident produit par sa mitigation résiduelle est la propriété la plus déstabilisante de la latence d'incident. Elle implique que la sécurité opérationnelle d'un environnement ne peut pas être évaluée comme la somme de ses mitigations actives. Elle doit être évaluée comme la somme de ses mitigations actives plus la somme des traces persistantes laissées par les mitigations passées, dans leurs interactions avec l'état courant du système. Cette seconde somme est, dans la plupart des organisations, inconnue. Elle n'est ni inventoriée ni testée. Elle s'accumule silencieusement dans les fichiers /etc/sysctl.d/, dans les /etc/modules-load.d/, dans les services systemd, dans les hooks SELinux, dans les capabilities, dans les namespaces, dans tous les lieux où l'état du système se modifie sans cycle de vie complet documenté.

La doctrine du port de promotion et de l'écart de promotion concernait la promotion d'artefacts en dépendance normative. Celle de la latence d'incident concerne ce qui se passe quand un artefact n'a pas été promu mais n'a pas non plus été retiré. C'est, opérationnellement, la situation la plus fréquente dans les environnements de production hétérogènes. Elle est aussi la moins traitée par les cadres de sécurité standard. Sa prise en charge n'est pas optionnelle pour les domaines régulés, où l'attribution erronée d'un incident à sa cause événementielle visible peut conduire à des

remédiations qui ne traitent pas le risque réel, et à une accumulation de dette résiduelle qui prépare le prochain incident à signature différente.

L'expérimentation ne se distingue de la production que par sa réversibilité explicite. Tant qu'un artefact n'a pas de procédure de retrait testée, il est en production. Le savoir conscient de cette équivalence est la condition d'une hygiène d'environnement qui tienne dans le temps. La supposer acquise par défaut est ce qui produit la latence d'incident.