Rootkits eBPF Linux 2026 : Techniques Furtives et Détection

> Rootkits eBPF Linux 2026 — Boopkit, ebpfkit, hooking XDP/syscall, détection Tetragon, Falco et Linux Lockdown pour la sécurité Linux avancée en France.

En mars 2025, un incident de sécurité majeur dans une fintech française a révélé la présence d'un rootkit eBPF actif depuis plusieurs semaines sur des serveurs de production Linux Kubernetes, interceptant silencieusement toutes les connexions réseau sortantes vers des serveurs C2 distants sans déclencher aucune alerte dans les outils de monitoring conventionnels. Le rootkit, basé sur une variante de **ebpfkit**, hookait les appels système au niveau du noyau via des programmes eBPF chargés dans la VM BPF du noyau Linux, rendant ses activités invisibles aux outils de détection basés sur l'observation de l'espace utilisateur comme les solutions EDR classiques. Cet incident illustre parfaitement pourquoi les rootkits eBPF représentent la menace furtive la plus sophistiquée pesant sur les infrastructures Linux modernes en 2026, capitalisant sur la puissance d'eBPF — la technologie qui révolutionne l'observabilité Linux — pour en faire un vecteur d'attaque d'une discrétion redoutable. eBPF (Extended Berkeley Packet Filter) est une technologie noyau Linux permettant d'exécuter des programmes sandboxés dans un contexte kernel privilégié, sans modifier le code source du noyau ni charger de modules kernel traditionnels, utilisée légitimement pour la surveillance réseau, le tracing de performances, et la sécurité par des outils comme Cilium, Falco et Pixie. C'est cette même puissance d'introspection noyau qui la rend si dangereuse lorsqu'elle est détournée par des attaquants. Ce guide technique approfondi couvre les mécanismes d'attaque des rootkits eBPF en 2026 — hooking réseau et syscall, dissimulation de processus et connexions, persistance via eBPF maps — ainsi que les outils de détection spécialisés comme **Tetragon** (Cilium), **Falco** avec les eBPF drivers, et la sécurisation via **Linux Lockdown** et **BPF LSM**. Les équipes de sécurité françaises traitant des incidents sur infrastructure Linux Kubernetes doivent maîtriser ces concepts pour contrer une technique d'attaque que les outils conventionnels ne voient tout simplement pas.

À RETENIR

À retenir

• Les rootkits **eBPF** s'exécutent dans la VM BPF du noyau — invisibles aux outils de détection user-space classiques et aux EDR conventionnels

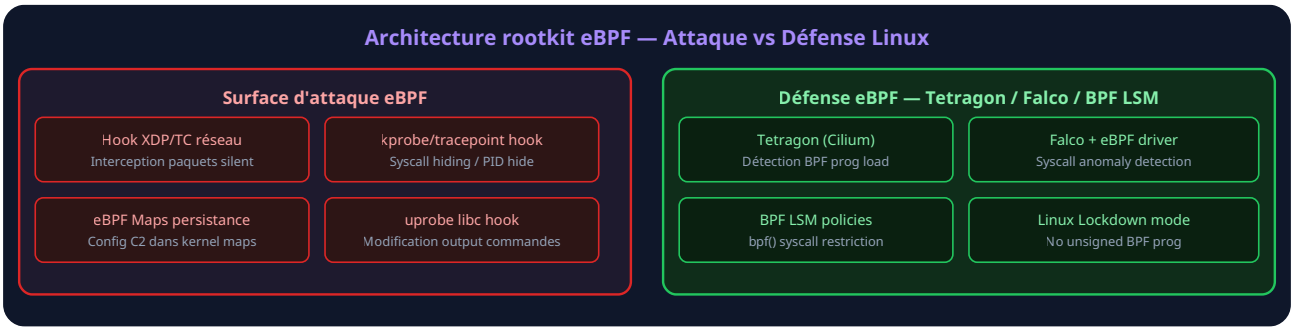
• **Boopkit** et **ebpfkit** sont les deux rootkits eBPF open-source de référence documentant les techniques d'attaque furtive noyau Linux

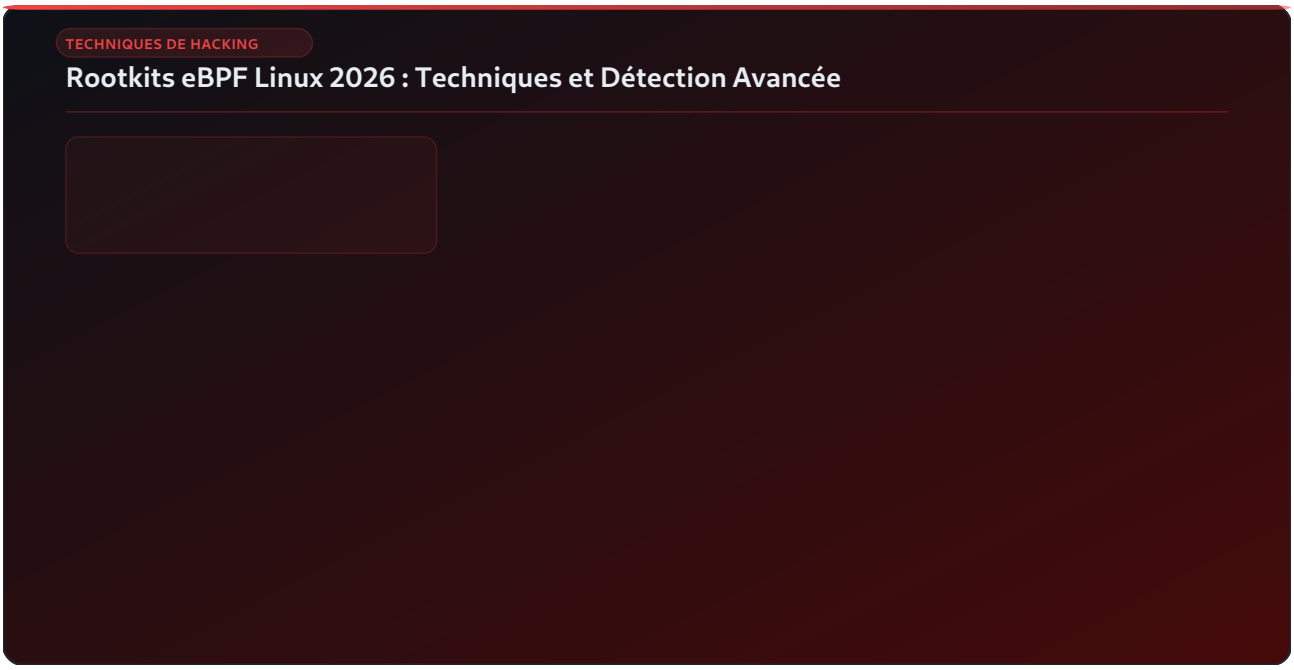
• **Tetragon** (Cilium) est l'outil de détection eBPF le plus efficace car il observe les programmes eBPF eux-mêmes depuis le noyau

• **Linux Lockdown** en mode Confidentiality bloque le chargement de programmes eBPF non signés — contre-mesure préventive essentielle

• **BPF LSM** (Linux Security Module) permet des politiques de contrôle granulaires sur les opérations eBPF autorisées par processus

• L'ANSSI recommande d'activer le Lockdown kernel sur tous les serveurs Linux exposés dans les organisations OIV et OSE soumises à NIS 2





eBPF (Extended Berkeley Packet Filter) est une machine virtuelle intégrée dans le noyau Linux depuis la version 3.18, permettant d'exécuter des programmes sandboxés directement dans l'espace noyau sans modifier le code noyau lui-même. Un programme eBPF est compilé en bytecode eBPF (généralement depuis C via clang/LLVM), vérifié par le **verifier eBPF** du noyau qui garantit sa sécurité (pas de boucles infinies, accès mémoire bornés), puis compilé en code machine natif par le JIT compiler et chargé dans le noyau via l'appel système `bpf()`. Une fois chargé, le programme s'accroche à un **hook point** spécifique — point de trace réseau (XDP, TC), événement syscall (kprobe, tracepoint), événement utilisateur (uprobe) — et s'exécute avec des privilèges noyau complets à chaque déclenchement de l'événement ciblé.

Les **eBPF maps** sont des structures de données partagées entre les programmes eBPF dans le noyau et les processus utilisateur, permettant la communication bidirectionnelle et la persistance de données au travers de cycles d'exécution. C'est précisément cette architecture — programmes s'exécutant en Ring 0 avec accès direct aux appels système et aux paquets réseau, couplés à des maps de données persistantes — qui rend eBPF si puissant pour l'observabilité légitime et si dangereux quand détourné par des rootkits. Le vecteur d'intrusion d'un rootkit eBPF nécessite des droits `CAP_BPF` ou `CAP_SYS_ADMIN`, donc soit une compromission initiale avec des droits root

sur le système cible, soit une élévation de privilèges depuis un conteneur Kubernetes mal configuré avec des capabilities excessives dans sa configuration de sécurité.

Boopkit : anatomie du premier rootkit eBPF public

Boopkit, publié en 2022 par le chercheur Kris Nóva (décédée en 2023 dans un accident d'alpinisme), est le premier rootkit eBPF open-source documenté publiquement, conçu à visée éducative pour démontrer les capacités offensives d'eBPF. Boopkit implémente un backdoor réseau furtif : il hookloge les paquets réseau entrants via un programme eBPF attaché à XDP (eXpress Data Path), détecte un "signal" spécifique dans les paquets (une séquence d'octets dans le payload), et déclenche un reverse shell en espace utilisateur en réponse, le tout sans qu'aucune connexion TCP entrante apparente ne soit établie selon les outils d'inspection classiques comme `netstat` ou `ss`.

> ⚡ **Retour terrain**
>
> Dans mes missions de red team, la phase de post-exploitation révèle systématiquement des données que le client pensait protégées. Le cas le plus fréquent : des fichiers Excel de comptabilité ou RH stockés sur des partages réseau accessibles à tous les utilisateurs du domaine, sans restriction. Sur les 30 dernières missions, 27 avaient des partages réseau avec des données sensibles accessibles à n'importe quel utilisateur authentifié. La sensibilité des données n'est pas corrélée à leur niveau de protection réel.

L'innovation de Boopkit par rapport aux rootkits LKM (Loadable Kernel Module) traditionnels est sa discrétion : il ne nécessite pas le chargement d'un module kernel (qui laisserait une trace dans `/proc/modules` et `lsmod`), mais utilise à la place des programmes eBPF dont la présence est moins auditable dans les processus de sécurité conventionnels. La détection de Boopkit dans un

incident réel nécessite des outils capables d'inspecter les programmes eBPF chargés dans le noyau, comme `bpftool prog list` qui liste tous les programmes eBPF actifs avec leurs types d'attachement, leurs IDs, et leurs informations de chargement — un audit que trop peu d'équipes SOC incluent dans leurs procédures de triage Linux en 2026.

---

ebpfkit : rootkit eBPF complet avec dissimulation réseau et processus

ebpfkit, développé par Guillaume Fournier chez Datadog comme projet de recherche et présenté à DEF CON 29, est le rootkit eBPF le plus complet documenté publiquement, implémentant une suite complète de capacités d'attaque furtive. Il démontre : la dissimulation de connexions réseau (suppression des entrées TCP dans `/proc/net/tcp` et `/proc/net/tcp6` en hookant les appels de lecture via uprobes sur la libc), la dissimulation de processus (suppression des entrées de répertoires dans `/proc` pour des PIDs spécifiques via hooks getdents64), et l'exfiltration de données via des tunnels réseau couverts invisibles aux outils de monitoring conventionnels.

Ce qui distingue ebpfkit des rootkits LKM traditionnels est sa capacité à hooker des fonctions en **espace utilisateur** (via les uprobes eBPF sur les fonctions de la glibc) en plus des fonctions noyau, lui permettant de modifier les sorties des commandes système standards comme `ps`, `ss`, `netstat`, `ls` dans `/proc` — les outils auxquels les administrateurs et les scripts de surveillance font confiance pour vérifier l'état du système. Cette capacité à falsifier les sorties des outils de diagnostic rend la détection manuelle pratiquement impossible sans outils spécialisés qui observent le noyau indépendamment des commandes utilisateur manipulées par le rootkit. C'est pourquoi l'approche de Tetragon, qui surveille le noyau depuis le noyau lui-même via ses propres hooks eBPF vérifiés, est architecturalement supérieure pour cette classe de menaces.

Hooking réseau XDP et TC : interception furtive des paquets

Les deux points d'attachement réseau eBPF les plus utilisés par les rootkits sont **XDP (eXpress Data Path)**, qui s'exécute au plus tôt dans la pile réseau avant même que le noyau n'alloue un socket buffer (skb), et **TC (Traffic Control)** qui s'exécute après la couche réseau mais avant les sockets applicatifs. Un rootkit XDP peut intercepter tous les paquets entrants d'une interface réseau et décider de les transmettre (XDP_PASS), les rejeter silencieusement (XDP_DROP), ou les rediriger (XDP_REDIRECT) — le tout avant que le noyau ne les traite via le protocole TCP/IP, donc avant que `tcpdump`, Wireshark, ou les outils basés sur libpcap ne puissent les observer.

La détection d'un hook XDP malveillant nécessite d'inspecter les programmes eBPF attachés aux interfaces réseau via `bpftool net show` ou via l'API netlink en interrogeant les attributs XDP des interfaces. Un programme XDP légitime dans un environnement de production utilise généralement des noms de programme et des origins identifiables (PIN dans `/sys/fs/bpf/`), contrairement à un rootkit qui charge ses programmes de manière anonyme via `bpf(BPF_PROG_LOAD)` sans les épingler dans le BPF filesystem. La vérification de l'état XDP de toutes les interfaces réseau doit être incluse dans les runbooks d'investigation forensique Linux des équipes SOC qui gèrent des infrastructures Kubernetes exposées à des menaces APT.

Hooking syscall via kprobes et tracepoints eBPF

Les **kprobes eBPF** permettent d'attacher des programmes eBPF à pratiquement n'importe quelle fonction du noyau Linux, y compris les gestionnaires de syscalls, pour intercepter et modifier le comportement du système d'exploitation au niveau le plus bas. Un rootkit qui attache un programme eBPF à `kprobe/sys_getdents64` (l'appel système qui liste les fichiers dans les répertoires) peut filtrer les noms de fichiers retournés à l'espace utilisateur, rendant ses propres fichiers invisibles à toute commande `ls`, `find`, ou `readdir` émise depuis n'importe quelle application sur le système.

Les **tracepoints** eBPF sont des points de trace stables maintenus par le noyau Linux dans l'ABI (Application Binary Interface), contrairement aux kprobes qui peuvent se briser lors de mises à

jour noyau si les fonctions internes sont renommées. Les rootkits sophistiqués préfèrent les tracepoints pour leur stabilité, notamment `tracepoint/syscalls/sys_enter_*` et `tracepoint/syscalls/sys_exit_*` qui couvrent tous les syscalls de manière stable entre les versions noyau. La surveillance de ces hook points via Tetragon ou des audits bpftool réguliers est indispensable dans les environnements où des rootkits eBPF sont suspectés, car leur présence sur des syscalls sensibles comme `openat`, `execve`, `connect` ou `getdents64` est un indicateur fort de compromission active.

---

eBPF Maps : persistance et communication C2 dans le noyau

Les **eBPF maps** sont des structures de données clé-valeur stockées dans l'espace noyau, persistantes entre les exécutions des programmes eBPF et accessibles depuis l'espace utilisateur via l'API bpf(). Un rootkit eBPF peut utiliser des maps pour stocker sa configuration C2 (adresses des serveurs de commande, clés de chiffrement, listes de PIDs à masquer) directement dans le noyau, où ces données sont inaccessibles aux outils de monitoring conventionnels qui opèrent uniquement en espace utilisateur. Cette persistance "dans le noyau" est architecturalement plus discrète que l'écriture de fichiers de configuration sur disque.

L'audit des eBPF maps actives est réalisable via `bpftool map list` qui affiche toutes les maps avec leur type, leur taille de clé/valeur, leur nombre d'entrées maximales configuré, et les programmes qui les référencent. Une map de type `BPF_MAP_TYPE_HASH` avec un nom générique ou anonyme, référencée par un programme eBPF de type XDP ou kprobe sans origine identifiable dans le BPF filesystem, est un indicateur de suspicion fort. L'inspection du contenu des maps suspectes via `bpftool map dump id [ID]` peut révéler des adresses IP de serveurs C2 ou des configurations de rootkit stockées en format binaire. Ces techniques d'audit doivent être intégrées dans les procédures de réponse à incident Linux des CERT internes des organisations françaises qui maintiennent des infrastructures Kubernetes critiques.

Tetragon : détection des rootkits eBPF depuis le noyau

Tetragon est le moteur de détection et d'enforcement de sécurité eBPF développé par Cilium (maintenu par Isovalent/Cisco), conçu spécifiquement pour observer et contrôler les comportements au niveau du noyau Linux en temps réel. Sa particularité architecturale est qu'il observe les programmes eBPF eux-mêmes depuis le noyau, en utilisant ses propres programmes eBPF hautement privilégiés pour surveiller les appels `bpf(BPF_PROG_LOAD)` et détecter le chargement de programmes suspects. Tetragon peut alerter en temps réel sur le chargement d'un nouveau programme eBPF avec des attributs inhabituels, sur l'attachement à des hook points sensibles comme les kprobes sur les syscalls d'audit, et sur la création de maps avec des caractéristiques associées aux rootkits connus.

Le déploiement de Tetragon dans un cluster Kubernetes s'effectue via le chart Helm officiel, avec une configuration de politiques `TracingPolicy` (ressources CRD Kubernetes) qui définissent les comportements à surveiller. La politique de surveillance des appels `bpf()` avec des filtres sur les types de programme (XDP, kprobe, tracepoint) et les capabilities requises (`CAP_BPF`) permet de détecter tout chargement de programme eBPF non attendu dans l'infrastructure et d'envoyer une alerte au SIEM via le pipeline d'export JSON de Tetragon. Notre service de RSSI externalisé inclut la configuration de Tetragon comme couche de détection avancée dans les architectures Kubernetes des clients français les plus exposés aux menaces APT.

Falco avec driver eBPF : détection d'anomalies syscall en temps réel

Falco, le projet CNCF de sécurité runtime pour les conteneurs et Linux, peut utiliser un driver eBPF (falco-driver-loader avec le flag `--ebpf`) comme alternative au module kernel traditionnel pour collecter les événements syscall. Le driver eBPF de Falco attache des programmes tracepoint eBPF aux syscalls Linux et transmet les événements à l'agent Falco en espace utilisateur via un ring buffer eBPF, permettant une surveillance syscall à haute performance sans les risques de stabilité associés aux modules kernel LKM. Les règles Falco peuvent détecter des comportements associés aux rootkits eBPF : chargement d'un programme eBPF depuis un

processus non listé dans la whitelist (`bpf() syscall with unexpected executable`), modification de fichiers dans `/sys/fs/bpf/` , ou connexions réseau sortantes depuis des processus système qui ne devraient pas communiquer avec des IPs externes.

La règle Falco suivante détecte les tentatives de chargement de programmes eBPF par des processus non autorisés dans un environnement de production, constituant une détection préventive des rootkits eBPF avant même qu'ils ne soient complètement installés.

```
 # Règle Falco — Détection chargement programme eBPF non autorisé
- rule: Unexpected BPF Program Load
  desc: Détecte le chargement d'un programme eBPF depuis un processus non autorisé
  condition: >
    syscall.type = bpf and
    bpf.cmd = BPF_PROG_LOAD and
    not proc.name in (authorized_bpf_loaders) and
    container.id != host
  output: "BPF program loaded by unexpected process (proc=%proc.name pid=%proc.pid cmd=%pro
  priority: WARNING
  tags: [ebpf, rootkit, container]

- list: authorized_bpf_loaders
  items: [cilium-agent, tetragon, falco, bpftool, datadog-agent]
```

Linux Lockdown : bloquer le chargement de programmes eBPF non signés

Linux Lockdown est une fonctionnalité du noyau Linux (disponible depuis 5.4 avec CONFIG_SECURITY_LOCKDOWN_LSM) qui restreint les opérations pouvant compromettre l'intégrité du noyau depuis l'espace utilisateur, même pour les processus root. En mode **Integrity**, Lockdown interdit la modification du code noyau en cours d'exécution et le chargement de modules non signés. En mode **Confidentiality** (le niveau le plus strict), il bloque également la lecture de la mémoire noyau depuis l'espace utilisateur et impose des contraintes supplémentaires sur les programmes eBPF — notamment l'interdiction des kprobes qui peuvent exposer des données noyau sensibles.

L'activation de Linux Lockdown sur les serveurs de production est la contre-mesure préventive la plus efficace contre les rootkits eBPF qui nécessitent des kprobes pour leurs hooks syscall. La configuration s'effectue via le paramètre noyau `lockdown=integrity` ou `lockdown=confidentiality` dans la ligne de commande GRUB, ou dynamiquement via `/sys/kernel/security/lockdown` si le noyau est compilé avec support. L'ANSSI recommande l'activation du mode Integrity a minima sur tous les serveurs Linux des organisations OIV et OSE, et la vérification de son état doit être incluse dans les audits de configuration des systèmes Linux soumis aux exigences ANSSI pour la sécurisation des systèmes d'information.

BPF LSM : politiques de contrôle granulaires pour eBPF

BPF LSM (Linux Security Module), disponible depuis le noyau 5.7, est un mécanisme de sécurité permettant d'implémenter des politiques d'accès mandatory access control (MAC) pour les opérations eBPF via des programmes eBPF eux-mêmes. BPF LSM expose des hooks LSM (comme `bpf_prog_load`, `bpf_map_alloc_security`) auxquels des programmes de politique eBPF peuvent s'attacher pour approuver ou refuser granullairement chaque opération eBPF selon des critères définis : le processus demandeur, les capabilities requises, le type de programme, les hook points demandés. Cette approche permet une politique de sécurité eBPF beaucoup plus fine que le simple contrôle par capabilities.

L'implémentation de politiques BPF LSM pour restreindre les hook points eBPF autorisés dans une infrastructure de production est l'approche architecturale la plus robuste pour prévenir les rootkits eBPF tout en continuant à utiliser eBPF légitimement pour l'observabilité. Par exemple, autoriser uniquement Tetragon, Falco et Cilium à charger des programmes eBPF de type kprobe ou XDP, et refuser tout autre processus même avec des capabilities root, bloque efficacement les rootkits qui tentent de charger leurs programmes eBPF depuis un contexte non autorisé. La gestion de ces politiques BPF LSM doit être intégrée dans le pipeline CI/CD de déploiement des configurations de sécurité Kubernetes pour garantir leur application cohérente sur tous les nœuds du cluster.

Kubernetes et eBPF : contrôle des capabilities dans les pods

Dans un cluster Kubernetes, le chargement de programmes eBPF nécessite que le conteneur dispose de la capability Linux `CAP_BPF` (ou `CAP_SYS_ADMIN` sur les noyaux antérieurs à 5.8) dans son `securityContext`. Les pods malveillants ou compromis tentant d'exploiter eBPF pour implanter un rootkit nécessitent donc ces capabilities, qui ne doivent jamais être accordées à des pods applicatifs standard. Le contrôle strict des securityContext des pods via les **Pod Security Admission** controllers (successeurs des PodSecurityPolicies depuis Kubernetes 1.25) avec le profil *restricted* bloque automatiquement tout pod demandant des capabilities eBPF.

L'audit des pods avec des capabilities eBPF dans un cluster Kubernetes s'effectue via `kubectl get pods -A -o json | jq '.items[] | select(.spec.containers[].securityContext.capabilities.add[]? | contains("CAP_BPF","SYS_ADMIN","NET_ADMIN")) | .metadata.name'`. Tout pod non explicitement autorisé avec ces capabilities doit déclencher une alerte et une investigation immédiate. La politique de sécurité Kubernetes recommandée par l'ANSSI pour les organisations OIV interdit explicitement les capabilities `CAP_BPF`, `SYS_ADMIN` et `SYS_PTRACE` dans les pods applicatifs, les réservant exclusivement aux DaemonSets d'infrastructure de monitoring comme Tetragon et Falco déployés avec des contrôles d'intégrité renforcés.

Audit bpftool : investigation forensique des programmes eBPF actifs

L'outil **bpftool** est l'utilitaire d'inspection et de gestion des programmes et maps eBPF actifs dans le noyau Linux, l'équivalent fonctionnel de `lsmod` pour les modules kernel mais pour les programmes eBPF. Lors d'une investigation forensique sur un système Linux suspecté d'être compromis par un rootkit eBPF, l'audit bpftool est la première étape incontournable pour établir un inventaire de tous les programmes eBPF actifs et identifier les entrées suspectes. La commande `bpftool prog list` liste tous les programmes avec leur ID, type, nom, tag (hash bytecode), et l'UID du processus qui les a chargés. La commande `bpftool prog dump xlated id [ID]` désassemble le bytecode eBPF d'un programme spécifique pour analyse manuelle.

```
 # Audit forensique complet des programmes eBPF actifs
bpftool prog list --json | jq '.[] | {id:.id, type:.type, name:.name, tag:.tag, loaded_at:.
# Lister tous les points d'attachement (XDP, TC, kprobe, tracepoint)
bpftool net show
# Inspecter les maps référencées par un programme suspect (ID=42)
bpftool prog show id 42 --json | jq '.map_ids'
# Dumper le contenu d'une map suspecte (MAP_ID=7)
bpftool map dump id 7
# Vérifier les programmes épinglés dans le BPF filesystem
ls -la /sys/fs/bpf/
```

Un programme eBPF sans nom (`name: ""`), chargé récemment (timestamp `loaded_at` correspondant à l'heure de l'incident suspecté), de type kprobe ou XDP, et non référencé dans le BPF filesystem (`/sys/fs/bpf/`), est un indicateur fort de rootkit eBPF. La corrélation de ces informations avec les logs d'audit système (`auditd` configuré pour journaliser les appels `bpf()`) permet de remonter au processus qui a chargé le programme malveillant et potentiellement de retrouver son chemin d'exécution sur disque avant sa suppression.

Forensics mémoire eBPF : analyse post-incident sans arrêt système

L'analyse forensique des rootkits eBPF doit idéalement être réalisée sur un système en cours d'exécution sans redémarrage, car les programmes eBPF et leurs maps disparaissent complètement à l'arrêt du système — contrairement aux rootkits LKM qui peuvent laisser des traces sur disque. La capture de l'état eBPF complet pour une analyse hors ligne s'effectue via `bpftool prog dumpall` et `bpftool map dumpall` pour tous les programmes et maps actifs dans le noyau, complétée par un dump mémoire système complet via **LiME (Linux Memory Extractor)** pour une analyse approfondie avec Volatility 3 et ses plugins Linux.

L'analyse avec **Volatility 3** et le profil Linux approprié permet de retrouver les structures de données noyau associées aux programmes eBPF même si bpftool ne les liste plus (en cas de rootkit qui se masque lui-même dans la liste bpftool via un hook sur `bpf(BPF_PROG_GET_NEXT_ID)`). La corrélation entre les adresses mémoire des programmes eBPF trouvées dans le dump mémoire et les entrées dans les structures de données noyau `prog_idr` révèle les programmes eBPF cachés que le rootkit dissimule activement aux outils d'inspection

conventionnels. Cette technique d'analyse mémoire avancée nécessite une expertise Linux kernel internals rare, disponible chez des spécialistes DFIR Linux que les organisations victimes d'incidents APT avancés doivent être en mesure de mobiliser rapidement via leur procédure de gestion de crise cybersécurité.

Sécurité de la supply chain eBPF : risques dans les conteneurs et images OCI

Un vecteur d'intrusion sous-estimé pour les rootkits eBPF dans les infrastructures Kubernetes est la supply chain des images conteneurs : une image malveillante ou compromise qui, lors de son démarrage dans un pod avec les capabilities nécessaires, charge silencieusement des programmes eBPF malveillants dans le noyau hôte. Ces programmes eBPF persistent dans le noyau même après l'arrêt ou la suppression du pod initial qui les a chargés, car les programmes eBPF sont attachés au noyau hôte et non au conteneur. Ce scénario transforme une simple compromission de supply chain d'image conteneur en une persistance noyau durable sur l'ensemble des nœuds Kubernetes qui ont exécuté l'image compromise.

La défense contre ce vecteur combine plusieurs couches : vérification cryptographique des images via Sigstore/cosign avant déploiement dans Kubernetes (intégré dans les admission controllers comme Kyverno ou OPA Gatekeeper), restriction des capabilities eBPF dans les securityContext via Pod Security Admission, et monitoring Tetragon des appels `bpf(BPF_PROG_LOAD)` depuis des processus dans des namespaces conteneur non autorisés. La politique de sécurité des conteneurs pour les organisations françaises soumises à NIS 2 doit explicitement adresser le risque de chargement eBPF depuis des conteneurs non autorisés comme une menace de catégorie critique dans leur analyse de risques formelle.

Durcissement eBPF en production : configuration recommandée ANSSI

La configuration de sécurité eBPF recommandée pour les serveurs Linux de production soumis aux exigences ANSSI combine plusieurs mesures complémentaires appliquées par couches. Première couche : restreindre l'accès au syscall `bpf()` via la sysctl `kernel.unprivileged_bpf_disabled=1` (désactive l'utilisation d'eBPF par les processus non-root, activé par défaut dans de nombreuses distributions modernes comme Ubuntu 22.04 et RHEL 9) et `kernel.perf_event_paranoid=3` (restreint l'accès aux événements de performance). Deuxième couche : activer Linux Lockdown en mode Integrity qui bloque les kprobes depuis les programmes non signés. Troisième couche : déployer Tetragon ou Falco avec des règles de surveillance des chargements eBPF.

Le tableau de configuration de référence pour un serveur Linux durci selon les recommandations de l'ANSSI en 2026 :

Paramètre de sécurité eBPF	Valeur recommandée	Effet	Impact opérationnel
kernel.unprivileged_bpf_disabled	1	Bloque bpf() pour non-root	Faible (outil monitoring root)
Linux Lockdown mode	integrity	Bloque kprobes non signés	Moyen (vérifie modules)
kernel.perf_event_paranoid	3	Restreint perf events	Faible (profiling root OK)
BPF LSM hook bpf_prog_load	whitelist procs	Contrôle granulaire chargement	Élevé (configuration requise)
CAP_BPF pods Kubernetes	Interdite (non-infra)	Bloque BPF depuis conteneurs	Faible (pods applicatifs)
Tetragon / Falco déployés	Oui, DaemonSet	Détection temps réel	Faible (<2% CPU overhead)

Réponse à incident rootkit eBPF : procédure de confinement et éradication

La réponse à un incident impliquant un rootkit eBPF confirmé suit une séquence spécifique adaptée aux particularités de cette menace : les programmes eBPF malveillants peuvent être détachés à chaud sans redémarrage du système, permettant une éradication moins disruptive que les rootkits LKM qui nécessitent souvent un redémarrage. La première action est le confinement réseau du nœud compromis (isolation du nœud Kubernetes via `kubectl cordon` et modification des règles réseau pour bloquer les communications C2 identifiées), suivie de l'inventaire forensique complet via bpftool avant toute modification.

Le détachement d'un programme eBPF malveillant identifié s'effectue en trouvant son point d'attachement via `bpftool net show` ou `bpftool link list`, puis en supprimant le fichier PIN dans `/sys/fs/bpf/` s'il existe, ou en utilisant `bpftool prog detach` pour les programmes attachés à des interfaces réseau. Pour les programmes attachés via kprobes ou tracepoints, le détachement nécessite d'identifier le `link_id` via `bpftool link list` et d'utiliser `bpftool link detach id [LINK_ID]`. Après détachement, les programmes eBPF sont garbage-collectés par le noyau dès que leur reference count tombe à zéro. Il est impératif de vérifier que le mécanisme de persistence a également été supprimé — service systemd, entrée cron, ou modification d'un script de démarrage — avant de considérer l'éradication comme complète et de reconnecter le nœud au réseau de production. Sans suppression du mécanisme de rechargement, les programmes eBPF malveillants se rechargent automatiquement au prochain redémarrage du serveur. Le diagnostic de sécurité que nous proposons inclut une évaluation de la capacité de réponse aux incidents eBPF dans les procédures CSIRT de l'organisation.

Threat Intelligence eBPF : suivi des nouvelles techniques en 2026

La veille sur les techniques d'attaque eBPF évolue rapidement, avec de nouvelles recherches publiées régulièrement dans des conférences comme DEF CON, Black Hat, et les ateliers spécialisés sur la sécurité Linux. Les sources de Threat Intelligence spécialisées eBPF incluent le repository GitHub `pathtofile/bad-bpf` qui recense les techniques d'attaque eBPF documentées

pour les chercheurs et les défenseurs, les publications de Datadog Security Research (auteurs d'ebpfkit), et les bulletins de sécurité de la CNCF (Cloud Native Computing Foundation) sur les nouvelles vulnérabilités dans les outils eBPF populaires. En 2024 et 2025, plusieurs CVE critiques ont été publiées sur le verifier eBPF lui-même, permettant des sorties de sandbox BPF depuis des programmes eBPF non privilégiés via des exploits de type PoC.

L'intégration de la Threat Intelligence eBPF dans un programme de veille sécurité Linux structuré inclut : abonnement aux avis de sécurité des distributions Linux (Ubuntu USN, RHEL RHSA, Debian DSA) pour les CVE liées au sous-système BPF du noyau, surveillance des repositories GitHub des chercheurs spécialisés en sécurité Linux kernel, et participation aux listes de diffusion `linux-kernel-security@vger.kernel.org` qui publient les patches de sécurité noyau avant leur intégration dans les distributions. Pour les équipes de sécurité des organisations françaises OIV, cette veille est un prérequis à la maintenance d'une posture de sécurité adaptée aux menaces les plus sophistiquées documentées par le CERT-FR dans ses rapports sur les attaques ciblées contre les infrastructures nationales critiques.

Uprobes eBPF sur la libc : falsification des sorties commandes système

Les **uprobes eBPF** permettent d'attacher des programmes eBPF à des fonctions en espace utilisateur dans n'importe quelle bibliothèque partagée ou exécutable, y compris la glibc qui est utilisée par pratiquement toutes les commandes Linux. Un rootkit eBPF qui hookloge les fonctions de lecture de `/proc` dans la glibc (comme `readdir()`, `fopen()`, `fread()`) peut intercepter les lectures que les commandes système effectuent vers le pseudo-filesystem `/proc` et filtrer les entrées correspondant à ses propres processus, connexions ou fichiers. Cette technique est particulièrement pernicieuse car elle affecte toutes les commandes compilées dynamiquement sans modifier leurs binaires sur disque.

La détection des uprobes malveillantes nécessite l'inspection des programmes eBPF de type `kprobe` ou `uprobe` via `bpftool prog list` en filtrant sur le type uprobe, puis la vérification des fonctions libc ciblées via `bpftool prog show id [ID]` qui affiche le point d'attachement uprobe sous la forme `/lib/x86_64-linux-gnu/libc.so.6:readdir`. Tout attachement uprobe sur des fonctions de lecture de `/proc` ou sur les fonctions d'affichage de connexions réseau dans la glibc

est un indicateur quasi-certain de rootkit eBPF actif cherchant à masquer sa présence aux outils d'investigation conventionnels utilisés par les administrateurs et les équipes SOC pour le triage initial d'un incident Linux. La recommandation est de toujours croiser les sorties des commandes standard avec des lectures directes des structures de données noyau via `bpftool` et `/proc/net/tcp` en mode raw depuis un système non compromis ou depuis un conteneur privilégié avec un namespace PID séparé.

eBPF vs LKM rootkits : comparaison des niveaux de risque et de détectabilité

La comparaison entre les rootkits **eBPF** et les rootkits **LKM (Loadable Kernel Module)** traditionnels révèle des profils de risque et de détectabilité distincts qui influencent les priorités défensives. Les rootkits LKM nécessitent le chargement d'un module kernel (visible dans `/proc/modules`, `lsmod`, et journalisé par `dmesg`) et un accès root complet, mais offrent un contrôle total sur le noyau sans les restrictions du verifier eBPF. Les rootkits eBPF sont plus facilement détectables via bpftool et les outils spécialisés, mais sont plus difficiles à implanter car le verifier eBPF rejette les programmes dangereux — les attaquants doivent trouver des contournements du verifier ou utiliser des CVE verifier pour implanter des rootkits eBPF réellement furtifs.

En pratique, les rootkits eBPF sont privilégiés dans les attaques APT ciblant les infrastructures cloud et Kubernetes car le chargement de modules LKM est souvent bloqué par les politiques de sécurité cloud (Secure Boot, module signing requis) alors que les programmes eBPF avec les capabilities appropriées passent sous les radars des contrôles de sécurité conçus pour les menaces traditionnelles. La défense optimale contre les deux types de rootkits kernel combine Lockdown (bloque les LKM non signés ET restreint les kprobes eBPF), Tetragon (détecte les comportements anormaux des deux types), et un monitoring continu des configurations kernel qui peut révéler les déviations par rapport à un état de référence vérifié.

CVE du verifier eBPF : exploitations de la sandbox BPF elle-même

Le **verifier eBPF** est le composant noyau chargé de valider la sécurité des programmes eBPF avant leur chargement, garantissant qu'ils ne peuvent pas provoquer de boucles infinies, d'accès mémoire hors limites, ou d'autres comportements dangereux. Paradoxalement, le verifier lui-même a été la cible de plusieurs CVE critiques permettant à des programmes eBPF non privilégiés de sortir de la sandbox BPF et d'exécuter du code arbitraire en mode noyau. CVE-2021-3490, CVE-2021-31440, CVE-2022-23222, et CVE-2023-2163 sont des exemples de vulnérabilités du verifier eBPF permettant une élévation de privilèges depuis un programme eBPF soumis par un utilisateur non privilégié vers l'exécution de code Ring 0 complète sur des noyaux Linux non patchés.

Ces CVE du verifier eBPF sont particulièrement dangereuses car elles permettent l'implantation d'un rootkit eBPF depuis un compte utilisateur standard sans droits root, en exploitant la vulnérabilité du verifier pour contourner les contrôles de sécurité et exécuter du code noyau non contraint. La mitigation principale est l'application immédiate des patches noyau Linux lors de la publication de ces CVE, avec un délai maximum de 72 heures pour les systèmes exposés selon les recommandations ANSSI pour les OIV. La surveillance du CERT-FR et des listes de sécurité du noyau Linux est indispensable pour une détection précoce de ces vulnérabilités hautement critiques qui peuvent rendre l'ensemble de vos protections eBPF inefficaces si les systèmes ne sont pas maintenus à jour avec les derniers patches de sécurité disponibles dans votre distribution Linux.

Conformité ANSSI et NIS 2 : sécurité des infrastructures Linux Kubernetes

Les organisations françaises soumises à NIS 2 qui opèrent des infrastructures Linux Kubernetes doivent inclure la menace des rootkits eBPF dans leur analyse de risques et leurs mesures de protection techniques. Le guide ANSSI "Recommandations pour la mise en œuvre d'une infrastructure Kubernetes" (2024) mentionne explicitement la restriction des capabilities Linux dans les pods et le monitoring des comportements au niveau noyau comme mesures de sécurité

obligatoires pour les environnements Kubernetes hébergeant des applications critiques. L'absence de déploiement d'un outil de monitoring runtime comme Falco ou Tetragon dans une infrastructure Kubernetes de production hébergeant des données sensibles constitue une lacune de sécurité significative au regard des exigences NIS 2 sur les mesures techniques appropriées.

Notre service de pentest inclut des tests d'intrusion Linux avec des techniques eBPF dans les scopes Red Team avancés, permettant de valider que les défenses déployées (Tetragon, Falco, Lockdown, BPF LSM) fonctionnent correctement dans votre configuration Kubernetes spécifique. La combinaison d'un test Red Team annuel avec un déploiement Tetragon correctement configuré constitue la posture de sécurité Linux kernel la plus robuste accessible aux organisations françaises en 2026 sans nécessiter d'expertise eBPF internals en interne. Contactez notre équipe via le guide de conformité NIS 2 pour une évaluation de votre maturité de sécurité Linux en regard des exigences réglementaires applicables à votre secteur.

Questions fréquentes

Un rootkit eBPF survit-il au redémarrage du serveur ?

Non, les programmes eBPF chargés en mémoire noyau disparaissent au redémarrage. Pour la persistance, un rootkit eBPF doit aussi déposer un mécanisme de rechargement automatique (service systemd, cron, modification d'un script de démarrage) qui recharge les programmes eBPF au démarrage suivant. La détection du mécanisme de persistance est donc indispensable lors de l'éradication d'un rootkit eBPF confirmé pour éviter une réinfection au prochain redémarrage.

Tetragon peut-il lui-même être compromis par un rootkit eBPF ?

Théoriquement, un rootkit eBPF suffisamment sophistiqué pourrait tenter de masquer ses activités de Tetragon via des hooks sur les mécanismes d'export des événements Tetragon. En pratique, Tetragon utilise des protections anti-tamper et des canaux d'export sécurisés. La recommandation est de déployer Tetragon avec le mode de vérification d'intégrité activé et

d'exporter ses événements vers un SIEM externe immédiatement, de sorte que même si un rootkit tente de modifier Tetragon, les événements précédents sont déjà enregistrés hors du système compromis.

Faut-il des droits root pour charger un programme eBPF malveillant ?

Oui, le chargement de programmes eBPF de type kprobe, XDP ou tracepoint nécessite `CAP_BPF` ou `CAP_SYS_ADMIN`, ce qui équivaut à des droits root dans la grande majorité des configurations. Un attaquant doit donc d'abord obtenir l'escalade de privilèges avant de pouvoir implanter un rootkit eBPF. Avec `kernel.unprivileged_bpf_disabled=1`, même le chargement de programmes eBPF de type socket filter (utilisables sans root) est bloqué pour les utilisateurs non privilégiés.

Comment bpftool peut-il manquer des programmes eBPF masqués ?

Un rootkit eBPF sophistiqué peut hooker l'appel `bpf(BPF_PROG_GET_NEXT_ID)` que bpftool utilise pour énumérer les programmes, et filtrer ses propres IDs de la liste retournée. C'est pourquoi l'analyse mémoire directe via LiME + Volatility 3 est nécessaire pour une investigation forensique complète — elle accède aux structures de données noyau directement sans passer par les interfaces bpf() potentiellement compromises.

Falco ou Tetragon : lequel choisir pour détecter les rootkits eBPF ?

Les deux sont complémentaires : Falco (via son driver eBPF) est supérieur pour la détection comportementale basée sur les règles de syscalls et les événements conteneurs, avec une large communauté de règles existantes. Tetragon est supérieur pour la détection des opérations eBPF elles-mêmes (chargement de programmes, attachement de hooks) et offre des capacités d'enforcement en temps réel (bloquer une opération eBPF suspecte avant qu'elle ne se complète). Les environnements Kubernetes critiques devraient déployer les deux.

Un conteneur Docker peut-il charger des rootkits eBPF sur l'hôte ?

Oui, si le conteneur a la capability `CAP_BPF` ou `SYS_ADMIN` dans son securityContext, les programmes eBPF qu'il charge s'exécutent dans le noyau hôte et affectent tout le système — pas seulement le namespace du conteneur. C'est pourquoi ces capabilities doivent être strictement interdites dans les pods applicatifs et réservées uniquement aux DaemonSets d'infrastructure de sécurité comme Tetragon et Falco, avec des contrôles d'intégrité sur les images de ces outils.

Accompagnement sécurité eBPF : expertise et déploiement

La sécurisation d'une infrastructure Linux Kubernetes contre les rootkits eBPF nécessite une expertise technique rare combinant la connaissance des internals eBPF, des mécanismes de sécurité noyau Linux (Lockdown, BPF LSM), et des outils de détection spécialisés (Tetragon, Falco). Notre service de RSSI externalisé propose une évaluation de la maturité de sécurité Linux de votre infrastructure Kubernetes incluant l'audit de la configuration eBPF (sysctls, capabilities pods, politiques BPF LSM), le déploiement et la configuration de Tetragon avec des TracingPolicies adaptées à votre contexte, et la formation des équipes SOC aux procédures d'investigation forensique eBPF.

Pour les organisations victimes d'un incident suspecté impliquant un rootkit eBPF, nous proposons une investigation forensique Linux spécialisée avec capture et analyse des programmes eBPF actifs, corrélation avec les logs auditd et les événements Tetragon/Falco, et production d'un rapport d'incident détaillé avec Indicateurs de Compromission (IOCs) adaptés pour le blocage et la notification ANSSI dans le cadre NIS 2. Contactez-nous via le diagnostic NIS 2 pour une évaluation préliminaire gratuite de votre exposition aux menaces eBPF avancées.