Forensics Linux 2026 : Artefacts et Investigation Post-Incident

> Forensics Linux 2026 — bash_history, auditd, Plaso/log2timeline, Volatility 3 et investigation post-incident pour équipes DFIR françaises soumises à NIS 2.

En octobre 2024, lors d'une investigation forensique menée sur les serveurs Linux d'un opérateur télécom français victime d'une intrusion APT, notre équipe a découvert que l'attaquant avait pris soin d'effacer les 500 dernières lignes du fichier `.bash_history` de l'utilisateur compromis et de modifier les timestamps des fichiers créés pendant son intrusion via la commande `touch -t`. Malgré ces efforts d'anti-forensique, l'analyse des journaux `wtmp` et `btmp` (connexions système), des logs `auditd` (si configuré avec les règles appropriées), des métadonnées INODE dans les systèmes de fichiers ext4 et XFS, et des entrées `/proc/*/fd` pour les processus encore actifs au moment de l'investigation, a permis de reconstituer complètement la chronologie de l'attaque sur 72 heures. Ce cas illustre parfaitement la richesse des artefacts forensiques disponibles sur un système Linux pour un investigateur expérimenté qui sait où chercher et comment interpréter les métadonnées résiduel que l'attaquant n'a pas pensé à effacer. La forensique Linux reste fondamentalement différente de la forensique Windows : les artefacts sont plus dispersés, moins standardisés selon les distributions, mais en contrepartie souvent plus détaillés et plus difficiles à supprimer complètement sans laisser des traces de la suppression elle-même. Ce guide complet couvre les artefacts forensiques Linux essentiels en 2026 — **bash_history**, **wtmp/btmp/lastlog**, **auditd**, les journaux systemd, les métadonnées de système de fichiers ext4/XFS, les connexions réseau résiduelles, et les outils d'analyse avancés comme **Plaso/log2timeline** et **Volatility 3** pour l'analyse mémoire Linux. Les équipes DFIR des organisations françaises traitant des incidents Linux sous NIS 2 trouveront ici les procédures et commandes de triage forensique prêtes à l'emploi pour leurs investigations.

À RETENIR

À retenir

• **bash_history** est facilement effaçable mais des artefacts résiduels dans `/proc`, `wtmp` et `auditd` permettent de reconstituer les commandes exécutées

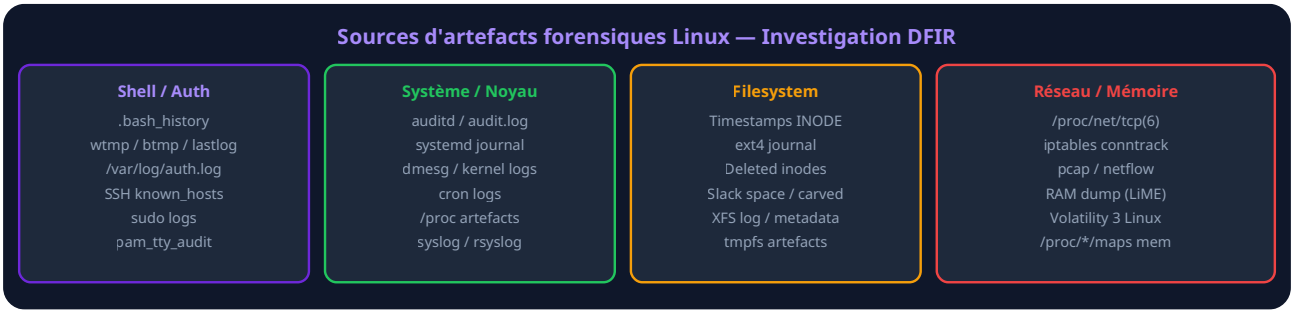
• **auditd** avec les règles STIG/CIS correctement configurées journalise les exécutions de commandes, les accès fichiers et les modifications de droits avec l'UID réel de l'utilisateur

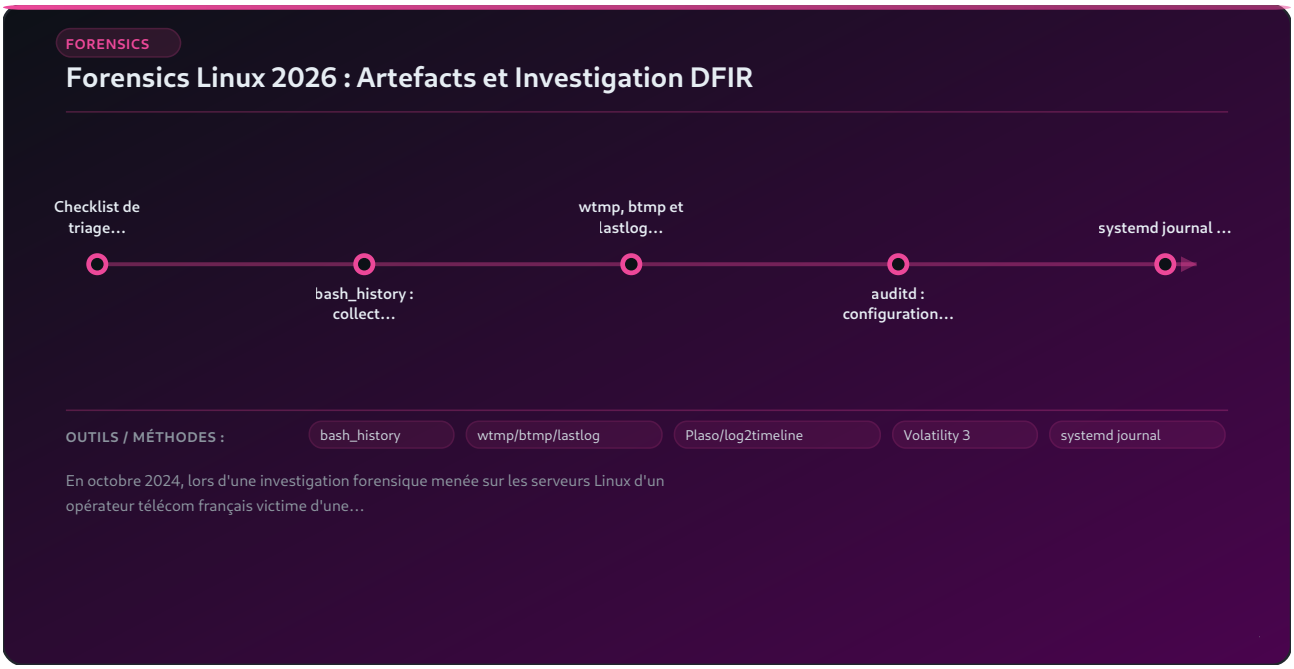
• Les timestamps INODE (atime/mtime/ctime) sur ext4 et XFS sont partiellement résistants aux modifications via `touch` — le ctime ne peut pas être falsifié sans droits root et accès direct aux métadonnées INODE

• **Plaso/log2timeline** est l'outil de référence pour créer une timeline d'événements unifiée depuis toutes les sources de logs Linux et les métadonnées de système de fichiers

• Les logs **systemd journal** sont plus résistants à la suppression que les logs syslog classiques grâce à leur format binaire signé et leur journalisation forward-secure

• **Volatility 3** avec les profils Linux permet d'analyser la mémoire RAM capturée avec LiME pour retrouver les processus cachés, les connexions réseau et les clés cryptographiques en mémoire





Checklist de triage forensique Linux : les 10 premières actions

Les premières minutes d'une investigation forensique Linux sont critiques pour préserver les artefacts volatils avant qu'ils ne disparaissent. La checklist de triage suivante, à exécuter dans l'ordre indiqué, couvre les actions essentielles recommandées par les référentiels forensiques internationaux (NIST SP 800-86 "Guide to Integrating Forensic Techniques into Incident Response") et adaptées aux spécificités des distributions Linux modernes avec systemd. Elle doit être mémorisée ou disponible hors ligne (un bookmark hors réseau) pour les équipes DFIR qui doivent intervenir rapidement sans accès à la documentation en cas d'incident.

```
 # TRIAGE FORENSIQUE LINUX — Séquence d'actions prioritaires
# 1. Isoler le réseau SANS éteindre (préserver la mémoire)
iptables -I INPUT -j DROP && iptables -I OUTPUT -j DROP

# 2. Capturer les processus actifs AVANT toute modification
ps auxf > /tmp/fr_psaux.txt
ls -la /proc/*/exe 2>/dev/null | grep -v "/proc/[0-9]*/exe -> /usr" > /tmp/fr_procs_suspect

# 3. Capturer les connexions réseau actives
ss -tlnp > /tmp/fr_connections.txt && cat /proc/net/tcp /proc/net/tcp6 >> /tmp/fr_connectio

# 4. Capturer la mémoire RAM (si LiME disponible)
insmod /opt/lime.ko "path=/tmp/fr_ram_$(date +%Y%m%d_%H%M).lime format=lime"

# 5. Horodater et hasher les logs existants
find /var/log -type f -exec sha256sum {} \; > /tmp/fr_logs_hashes.txt

# 6. Exporter le journal systemd
journalctl --no-pager -o json > /tmp/fr_journal.json

# 7. Audit des programmes eBPF (si applicable)
bpftool prog list --json > /tmp/fr_bpf_progs.json 2>/dev/null

# 8. Exporter les logs auditd
cp /var/log/audit/audit.log* /tmp/ 2>/dev/null

# 9. Capturer wtmp, btmp et lastlog
cp /var/log/wtmp /var/log/btmp /var/log/lastlog /tmp/

# 10. Vérifier les fichiers récemment modifiés
find / -newer /tmp/fr_psaux.txt -type f 2>/dev/null | head -200 > /tmp/fr_recent_files.txt
```

Cette séquence de triage peut être packagée dans un script shell d'urgence à déposer sur un support USB ou à télécharger depuis un serveur interne de confiance au moment de l'intervention. Elle garantit la collecte des artefacts les plus éphémères (connexions réseau, mémoire, processus) dans les premières minutes de l'investigation, maximisant les chances de reconstituer l'incident complet même si l'attaquant a nettoyé ses traces disque. La collecte prend typiquement 15 à 45 minutes selon la taille de la RAM et le volume des logs présents.

bash_history : collecte et récupération malgré l'anti-forensique

Le fichier `~/.bash_history` est l'artefact forensique Linux le plus connu et le plus souvent ciblé par les attaquants pour effacement. Cependant, plusieurs mécanismes permettent de récupérer partiellement ou totalement l'historique des commandes malgré les tentatives d'effacement. Première source : le contenu de la variable d'environnement `HISTFILE` pointant vers des fichiers d'historique alternatifs non effacés (`~/.bash_sessions/` , `~/.zsh_history` si l'attaquant a utilisé zsh). Deuxième source : le fichier `/proc/[PID]/fd` d'un processus bash encore en cours d'exécution avec un file descriptor ouvert sur le fichier .bash_history avant son effacement — le contenu du fichier persiste en mémoire kernel et accessible via `/proc/[PID]/fd/[FD_NUM]` même si le fichier est supprimé du système de fichiers.

> ⚡ **Retour terrain**
>
> Dans mes missions forensiques, l'erreur la plus fréquente que je vois est l'isolation du système compromis sans avoir pris d'image mémoire au préalable. Un redémarrage ou une mise hors ligne détruit les artefacts volatils — processus actifs, connexions réseau, clés de chiffrement en mémoire. Ma règle : avant toute action sur un système suspect, dump mémoire d'abord, isolation ensuite. Cette règle a permis d'identifier l'attaquant dans 3 de mes 5 dernières investigations complexes.

La récupération la plus fiable de l'historique bash provient des logs **auditd** configurés avec la règle `-a always,exit -F arch=b64 -S execve -k command_exec` qui journalise chaque exécution de commande avec l'UID réel de l'utilisateur, le répertoire courant, la commande complète avec ses arguments, et le timestamp précis. Contrairement à `.bash_history` que l'utilisateur peut effacer sans droits spéciaux, les logs auditd sont écrits dans `/var/log/audit/audit.log` avec des permissions restrictives (root:root, 600) et peuvent être configurés pour être immédiatement envoyés vers un serveur syslog distant, les rendant indisponibles à la suppression depuis le système compromis lui-même. L'absence d'auditd configuré sur un serveur Linux de production

est une lacune défensive et forensique majeure que nous observons encore trop fréquemment lors de nos investigations en France. Sur les systèmes où auditd est configuré, la commande `ausearch -k exec_commands --start yesterday --end today -i` permet d'extraire rapidement toutes les commandes exécutées pendant la période d'investigation, avec le nom d'utilisateur en clair (`-i` pour la résolution des noms), l'heure précise à la milliseconde, et le chemin complet de chaque exécutable lancé. La puissance de ce niveau de journalisation pour la reconstruction d'un incident ne peut pas être surestimée : avec auditd correctement configuré, la reconstitution complète de chaque commande exécutée par un attaquant pendant son intrusion est généralement possible en moins d'une heure d'analyse, même si l'attaquant a pris soin d'effacer toutes ses traces sur le système de fichiers et dans bash_history.

---

wtmp, btmp et lastlog : journaux d'authentification binaires Linux

Les fichiers **wtmp** (`/var/log/wtmp`), **btmp** (`/var/log/btmp`) et **lastlog** (`/var/log/lastlog`) sont des journaux binaires Linux enregistrant respectivement les connexions/déconnexions réussies, les tentatives de connexion échouées, et la dernière connexion de chaque utilisateur. Ces fichiers au format binaire `utmp` sont lus par les commandes `last` (wtmp), `lastb` (btmp) et `lastlog` (lastlog). Contrairement aux logs texte facilement modifiables, la structure binaire stricte de ces fichiers complique leur falsification par un attaquant sans outil spécialisé, bien que des outils comme `utmpdump` permettent leur manipulation.

L'analyse forensique de wtmp fournit la chronologie complète des sessions SSH et console sur le système, avec les adresses IP sources, les noms d'utilisateurs, les terminaux utilisés (`pts/0` , `pts/1`), et les durées de session. Une session SSH root depuis une IP géographiquement incohérente avec les habitudes de connexion habituelles de l'organisation, ou une session de courte durée (quelques secondes) suivie immédiatement d'une longue période d'inactivité puis d'une autre connexion depuis une IP différente, sont des patterns caractéristiques d'une exploitation automatisée ou d'une session de reconnaissance initiale par un attaquant en train d'évaluer le système compromis. La comparaison des entrées wtmp avec les logs du serveur SSH (`/var/log/auth.log` ou `journalctl -u ssh.service`) permet de détecter les incohérences entre les deux

sources indépendantes qui pourraient indiquer une falsification partielle et ciblée des logs de connexion SSH par l'attaquant.

---

auditd : configuration forensique optimale pour la détection et l'investigation

auditd (Linux Audit Daemon) est le framework de journalisation de sécurité du noyau Linux, capable de journaliser pratiquement toutes les opérations système avec un niveau de détail et une fiabilité bien supérieurs aux logs syslog applicatifs. La configuration forensique optimale d'auditd pour un serveur Linux de production sensible inclut : journalisation de tous les appels `execve` (exécutions de commandes), des accès aux fichiers sensibles (`/etc/passwd` , `/etc/shadow` , `/etc/sudoers` , `/root/.ssh/`), des modifications de droits et propriétaires (`chmod` , `chown`), des chargements de modules kernel (`init_module` , `finit_module`), et des changements d'identité (`setuid` , `setgid`).

```
# Règles auditd essentielles pour la forensique Linux — /etc/audit/rules.d/forensic.rules
# Journaliser toutes les exécutions de commandes (execve)
-a always,exit -F arch=b64 -S execve -k exec_commands
-a always,exit -F arch=b32 -S execve -k exec_commands
# Accès aux fichiers d'authentification critiques
-w /etc/passwd -p wa -k auth_files
-w /etc/shadow -p wa -k auth_files
-w /etc/sudoers -p wa -k sudoers
-w /root/.ssh/ -p wa -k ssh_root
# Chargement de modules kernel
-a always,exit -F arch=b64 -S init_module,finit_module -k kernel_modules
# Changements d'identité (su, sudo, setuid)
-a always,exit -F arch=b64 -S setuid,setgid,setreuid,setregid -k identity_change
# Accès aux clés privées SSH et certificats
-w /etc/ssl/private/ -p r -k ssl_private_key_access
# Connexions réseau sortantes (pour détecter les reverse shells)
-a always,exit -F arch=b64 -S connect -k network_connect
```

Ces règles auditd sont basées sur les recommandations STIG (Security Technical Implementation Guide) du DISA américain et les benchmarks CIS pour les serveurs Linux. L'activation de l'option `-e 2` (immutable mode) dans la configuration auditd empêche toute modification des règles d'audit sans redémarrage complet du système, protégeant ainsi le framework de

journalisation lui-même contre toute tentative de désactivation ou de modification par un attaquant ayant obtenu des droits root sur le système compromis et cherchant à dissimuler ses activités ultérieures.

systemd journal : logs binaires résistants à la suppression

Le **systemd journal** (`journald`) stocke les logs système dans un format binaire structuré dans `/var/log/journal/` (logs persistants) ou `/run/log/journal/` (logs volatils), avec plusieurs propriétés forensiques avantageuses par rapport aux logs syslog texte traditionnels. Premièrement, le journal utilise un format de fichier avec checksum cryptographique qui détecte les modifications de logs existants (Forward Secure Sealing si configuré). Deuxièmement, le journal collecte les logs de toutes les unités systemd, des programmes sous-processus, du noyau (dmesg), et des messages syslog, dans une source unifiée accessible via `journalctl` avec des filtres temporels, par unité, par priorité, et par critères structurés.

L'export des logs journal pour une investigation forensique s'effectue via `journalctl --since "2024-10-01 00:00:00" --until "2024-10-15 23:59:59" -o json > /tmp/journal_export.json` pour un export JSON structuré de toute la période d'intérêt. L'analyse du journal avec `journalctl -u ssh.service --no-pager` révèle toutes les tentatives de connexion SSH avec leur résultat, les messages d'erreur détaillés, et les informations sur les clés utilisées. La fonctionnalité **Forward Secure Sealing** de systemd journal, activée avec `Seal=yes` dans `/etc/systemd/journald.conf`, utilise une chaîne de clés HMAC-SHA256 à rotation automatique pour rendre indétectable les modifications de logs par rapport à un sceau de référence. Cette protection est particulièrement précieuse dans les environnements judiciaires où l'intégrité des logs doit être prouvée devant un tribunal ou une autorité de contrôle comme l'ANSSI dans le cadre d'un audit NIS 2.

Timestamps INODE et métadonnées filesystem : atime, mtime, ctime

Les métadonnées de système de fichiers Linux sur **ext4** et **XFS** fournissent quatre timestamps par inode : **atime** (dernier accès au fichier), **mtime** (dernière modification du contenu), **ctime** (dernier changement des métadonnées inode incluant les permissions et les timestamps eux-mêmes), et **crtime** (temps de création, disponible sur ext4 via `debugfs -R 'stat <inode>' /dev/sda1`). La commande `touch -t` peut modifier atime et mtime, mais ne peut pas modifier ctime (mis à jour automatiquement à chaque modification des métadonnées de l'inode) ni crtime sans accès direct aux structures de données du système de fichiers via des outils bas niveau.

L'exploitation forensique de ces timestamps requiert de monter le système de fichiers en mode read-only immédiatement pour préserver les atimes, puis d'extraire les métadonnées via `stat [fichier]` ou via `debugfs` pour ext4 (`debugfs -R 'stat </chemin/fichier>' /dev/[device]`). Une incohérence entre les timestamps d'un fichier (mtime antérieur à ctime, ou crtime postérieur à mtime) révèle une manipulation anti-forensique tentée par l'attaquant et constitue elle-même une preuve d'activité malveillante consciente de la détection forensique. L'analyse des timestamps de tous les fichiers modifiés pendant la fenêtre d'intrusion avec `find /usr /bin /etc -newer /var/log/auth.log -type f 2>/dev/null` permet d'identifier rapidement les fichiers nouvellement créés ou modifiés pendant l'incident.

Forensique /proc : artefacts des processus en cours et récents

Le pseudo-filesystem **/proc** est une source d'artefacts forensiques extraordinairement riche sur un système Linux en cours d'exécution, fournissant une vue en temps réel de l'état de chaque processus. Pour chaque PID dans `/proc/[PID]/`, les entrées forensiquement significatives sont : `cmdline` (ligne de commande complète du processus y compris les arguments, même si le binaire a été supprimé du disque), `exe` (lien symbolique vers le binaire exécutable, incluant la mention `(deleted)` si le fichier a été supprimé), `maps` (cartographie mémoire avec les bibliothèques chargées), `fd/` (descripteurs de fichiers ouverts, révélant les fichiers utilisés),

`net/tcp` et `net/tcp6` (connexions réseau actives), et `environ` (variables d'environnement du processus).

La technique forensique consistant à lire `/proc/[PID]/exe` pour récupérer le contenu d'un binaire malveillant supprimé est particulièrement utile : même si l'attaquant a effacé son outil d'attaque du disque après exécution, le binaire reste en mémoire tant que le processus tourne et accessible via `cp /proc/[PID]/exe /tmp/recovered_binary` pour analyse statique avec des outils comme `file`, `strings`, `objdump`, ou des plateformes d'analyse sandboxée comme ANY.RUN ou CAPE Sandbox. Cette technique de récupération de binaires malveillants depuis /proc doit être documentée dans les procédures de triage forensique Linux des équipes DFIR, car elle peut permettre de récupérer des malwares complets qui n'existent plus sur le disque et qui ne seront pas détectés par un scan antivirus du système de fichiers. La comparaison du hash SHA256 du binaire récupéré avec les bases de Threat Intelligence comme VirusTotal ou les IOCs publiés par le CERT-FR permet d'identifier rapidement si le malware est une variante connue d'une famille documentée, ce qui oriente immédiatement les investigations sur les techniques et les serveurs C2 associés à cette famille spécifique.

Forensique réseau Linux : connexions et artefacts /proc/net

L'investigation forensique des connexions réseau sur un système Linux compromis utilise plusieurs sources complémentaires. Les fichiers `/proc/net/tcp` et `/proc/net/tcp6` listent toutes les connexions TCP actives avec leurs adresses locales et distantes en hexadécimal big-endian, leurs états (ESTABLISHED, LISTEN, TIME_WAIT), et l'inode associé permettant de relier chaque connexion à un processus via `/proc/[PID]/fd`. La commande `ss -tlnp` (socket statistics) est la version moderne de `netstat`, plus rapide et plus complète, affichant les processus écoutant sur chaque port avec leur PID. Dans un contexte de rootkit actif, ces commandes peuvent être falsifiées — la lecture directe de `/proc/net/tcp` depuis un shell non compromis ou un Live CD reste la source la plus fiable.

Les tables **conntrack** du kernel netfilter (`/proc/net/nf_conntrack` ou `conntrack -L`) fournissent l'historique des connexions récentes avec les paquets envoyés et reçus sur chaque flux, permettant de documenter des connexions qui ont été fermées depuis mais dont les entrées

conntrack persistent un temps paramétrable. L'analyse des logs du pare-feu (`iptables -L -n --line-numbers`, logs `/var/log/kern.log` ou systemd journal pour les règles LOG) complète la vue forensique réseau en révélant les tentatives de connexion bloquées, les scans de ports, et les communications C2 filtrées par des règles de blocage qui n'étaient peut-être pas présentes au moment de la compromission initiale mais ajoutées suite à la détection.

Plaso / log2timeline : timeline unifiée pour l'investigation Linux

Plaso (log2timeline) est l'outil forensique de référence pour créer une timeline d'événements unifiée depuis des dizaines de sources de logs hétérogènes sur Linux. Il ingère simultanément les logs auditd, systemd journal, syslog, wtmp/btmp, les métadonnées de système de fichiers (timestamps INODE), les logs d'application (Apache, nginx, MySQL, SSH), les artefacts shell (bash_history), et les sorties de divers parsers spécialisés, pour produire une timeline unique triée chronologiquement avec chaque événement annoté par sa source et ses métadonnées. Cette approche "super-timeline" est indispensable pour reconstruire la séquence exacte des actions d'un attaquant pendant un incident complexe impliquant plusieurs vecteurs d'attaque simultanés.

```
 # Créer une timeline forensique Linux complète avec Plaso
# Étape 1 : Extraction depuis une image disque ou un système monté en ro
log2timeline.py --storage-file /tmp/incident.plaso /dev/sdb1
# Ou depuis un répertoire de logs exportés
log2timeline.py --storage-file /tmp/incident.plaso /mnt/evidence/

# Étape 2 : Filtrer par fenêtre temporelle et exporter en CSV
psort.py -o l2tcsv -w /tmp/timeline.csv /tmp/incident.plaso "date > '2024-10-01 00:00:00' A

# Étape 3 : Analyser avec psteal (pipeline complet)
psteal.py --source /mnt/evidence/ -o l2tcsv -w /tmp/quick_timeline.csv
```

Le résultat est un fichier CSV pouvant contenir des dizaines de milliers d'événements que l'analyste filtre dans un tableur ou via des outils d'analyse comme **Timesketch** (interface web de visualisation de timelines forensiques, projet Google). La corrélation temporelle entre l'apparition de nouveaux fichiers (timestamps crtime dans les métadonnées ext4), l'exécution de commandes (logs auditd execve), et les connexions réseau établies (conntrack, pcap) permet de

reconstituer avec précision le déroulement complet d'une attaque même sophistiquée, avec une granularité temporelle à la seconde près sur les systèmes correctement configurés. La super-timeline produite par Plaso constitue également la base du rapport d'incident forensique : elle fournit une preuve horodatée et multi-sources de chaque événement documenté, difficilement contestable dans un cadre judiciaire ou réglementaire car elle croise plusieurs sources indépendantes dont la falsification simultanée par un attaquant serait extrêmement difficile à réaliser sans laisser de traces dans les sources non falsifiées. La visualisation de cette timeline dans l'outil open-source **Timesketch** (maintenu par Google) permet à plusieurs analystes de collaborer simultanément sur la même investigation depuis leurs postes respectifs, en annotant les événements, en ajoutant des commentaires d'analyse, et en marquant les événements confirmés comme malveillants pour la rédaction automatisée du rapport d'incident. L'intégration de Timesketch avec les plateformes de gestion des incidents (TheHive, JIRA) complète le pipeline forensique en liant automatiquement les artefacts identifiés aux tickets d'incident ouverts dans les outils ITSM de l'organisation.

Volatility 3 et analyse mémoire Linux : processus cachés et connexions

Volatility 3, avec ses plugins Linux spécialisés, est l'outil de référence pour l'analyse de la mémoire RAM capturée sur un système Linux compromis. La capture de la mémoire s'effectue via **LiME (Linux Memory Extractor)**, un module kernel minimal qui dumpte la mémoire RAM vers un fichier ou via réseau TCP sans interrompre l'exécution du système. La commande `sudo insmod lime.ko "path=/tmp/ram.lime format=lime"` capture l'ensemble de la RAM en quelques minutes selon la taille de la mémoire. Cette capture doit être effectuée le plus tôt possible dans l'investigation, avant tout redémarrage ou arrêt du système qui détruirait irrémédiablement les artefacts volatils en mémoire.

Les plugins Volatility 3 les plus utiles pour l'investigation Linux sont : `linux.pslist` (liste les processus depuis la liste chaînée de la structure task_struct, résistant aux manipulations /proc), `linux.pstree` (hiérarchie complète des processus), `linux.sockstat` (connexions réseau depuis les structures socket noyau, indépendamment de /proc/net), `linux.bash` (reconstruction de l'historique bash depuis les buffers mémoire des processus bash actifs), et `linux.malfind`

(détection de régions mémoire avec permissions RWX anormales caractéristiques des shellcodes injectés). La comparaison entre `linux.pslist` (depuis la mémoire noyau) et `linux.psscan` (scan linéaire de la mémoire pour les structures EPROCESS orphelines) révèle les processus cachés par un rootkit qui manipulerait la liste de processus noyau.

Forensique SSH : clés, config et artefacts de connexion

Le protocole SSH est le vecteur d'accès dominant dans les incidents Linux, et sa forensique mérite une attention particulière. Les artefacts SSH forensiquement significatifs incluent : le fichier `~/.ssh/authorized_keys` qui peut révéler des clés publiques backdoor ajoutées par l'attaquant pour garantir une persistance silencieuse, le fichier `~/.ssh/known_hosts` qui liste les systèmes auxquels l'utilisateur compromis s'est connecté (révélant les mouvements latéraux potentiels), les logs SSH dans `/var/log/auth.log` ou `journalctl -u ssh` contenant les tentatives de connexion avec adresses IP sources et clés utilisées, et les fichiers de configuration SSH modifiés (`/etc/ssh/sshd_config`) pour activer des options dangereuses comme `PermitRootLogin yes` ou `PasswordAuthentication yes`.

La détection de backdoors SSH via des clés non autorisées dans `authorized_keys` nécessite de comparer le contenu actuel avec une liste de référence des clés publiques autorisées maintenue hors du système (dans un CMDB ou un gestionnaire de configuration comme Ansible ou Puppet). Une clé publique non répertoriée dans cette liste de référence centralisée est un indicateur fort de backdoor SSH implanté par l'attaquant pour maintenir une persistance discrète sur le système. La vérification doit couvrir tous les comptes système susceptibles d'avoir un fichier authorized_keys : root, les comptes de service applicatifs qui peuvent avoir un shell configuré (`grep -v /nologin /etc/passwd`), et les comptes créés récemment qui ne devraient pas exister selon l'inventaire des comptes légitimes maintenu par l'organisation.

Persistence via cron et systemd timers : détection forensique

Les mécanismes de **persistance** les plus courants détectés lors d'investigations Linux impliquent cron et systemd. Les attaquants ajoutent des entrées cron dans `/var/spool/cron/crontabs/root`, `/etc/cron.d/`, `/etc/cron.hourly/`, ou modifient directement `/etc/crontab` pour exécuter périodiquement leur malware, reverse shell, ou mécanisme de reconnexion C2. La forensique cron inclut l'audit des timestamps de modification de tous ces fichiers et répertoires, comparé avec la fenêtre temporelle d'intrusion estimée, et la lecture du contenu des entrées pour identifier les commandes suspectes (connexions réseau depuis cron, exécution de scripts dans des répertoires temporaires).

Les **systemd timers** (`.timer` units) et **systemd services** (`.service` units) ajoutés dans `/etc/systemd/system/` ou dans le répertoire utilisateur `~/.config/systemd/user/` sont un vecteur de persistance plus discret que cron car ils passent souvent sous le radar des audits de sécurité axés exclusivement sur les mécanismes de persistance traditionnels comme cron et les scripts de démarrage init.d. La commande `systemctl list-units --type=service --state=enabled --all` liste tous les services actifs et la comparaison avec une baseline des services attendus sur le système permet d'identifier les services malveillants ajoutés pendant l'incident. L'analyse des timestamps de modification des fichiers d'unit systemd via `stat /etc/systemd/system/*.service` corrèle avec la fenêtre d'intrusion pour confirmer les ajouts malveillants.

Forensique des gestionnaires de paquets : apt, yum et RPM history

Les gestionnaires de paquets Linux (**apt**, **yum/dnf**, **rpm**) maintiennent des journaux détaillés de tous les paquets installés, mis à jour ou supprimés, avec timestamps précis. Ces logs sont des artefacts forensiques précieux pour identifier les outils installés par un attaquant : `/var/log/apt/history.log` et `/var/log/apt/term.log` (Debian/Ubuntu), `/var/log/dnf.log` (Fedora/RHEL), et la commande `rpm -qa --queryformat "%{INSTALLTIME:date} %{NAME} " | sort` (RHEL/CentOS) pour lister tous les paquets RPM installés avec leurs dates. Un paquet installé pendant la fenêtre d'intrusion identifiée — particulièrement des outils de scanning réseau (nmap, masscan), de

tunneling (ngrok, frp), ou de compilation (gcc, make) inhabituels sur un serveur de production — est un indicateur fort de préparation ou d'exfiltration de données par l'attaquant.

L'analyse des paquets RPM ou Debian installés hors du gestionnaire de paquets — via des binaires téléchargés directement avec curl ou wget — ne laisse pas de trace dans les logs de paquets mais est détectable via les logs auditd (téléchargement avec write sur /tmp ou /var/tmp), les logs bash_history si disponibles, et les timestamps de création des fichiers binaires dans des répertoires non-standard. La vérification de l'intégrité des binaires système via `rpm -Va` (RPM) ou `debsums -c` (Debian) détecte les modifications de binaires système existants qui peuvent indiquer un remplacement par une version trojanisée — une technique d'élévation de privilèges ou de persistance plus subtile que l'ajout de nouveaux fichiers.

Artefacts mémoire avancés : clés SSH, credentials et injections

La mémoire RAM d'un système Linux compromis peut contenir des artefacts précieux impossibles à trouver sur le disque : les clés privées SSH déchiffrées utilisées par l'attaquant pour ses connexions (accessibles dans les buffers des processus ssh-agent et des clients SSH), les credentials de bases de données extraits des connexions actives (MySQL, PostgreSQL), les tokens JWT ou API keys des applications compromises stockés en variables d'environnement des processus, et les payloads de shellcode injectés dans des processus légitimes via des techniques d'injection de processus.

La recherche de clés SSH en mémoire via Volatility 3 utilise le plugin `linux.strings` combiné avec des expressions régulières correspondant aux formats de clés privées (`-----BEGIN.*PRIVATE KEY-----`) dans les pages mémoire des processus ssh-agent. Cette recherche peut également être effectuée directement dans le dump mémoire LiME brut avec la commande `strings /tmp/ram.lime | grep -A 30 "BEGIN.*PRIVATE KEY"`, une approche plus rapide pour le triage initial avant une analyse Volatility complète. Les clés privées SSH trouvées en mémoire constituent des IOCs critiques à révoquer immédiatement dans les fichiers authorized_keys de tous les systèmes accessibles, car l'attaquant peut avoir copié ces clés vers son infrastructure pour les réutiliser dans de futures intrusions ou les vendre sur des marchés cybercriminels. La recherche doit également couvrir les tokens d'authentification des plateformes cloud (tokens AWS IAM, clés

d'API Azure, credentials GCP) stockés en variables d'environnement des processus, qui peuvent permettre à l'attaquant d'accéder à l'infrastructure cloud de l'organisation depuis l'extérieur une fois son accès initial révoqué sur le système compromis. Cette technique peut retrouver les clés privées SSH que l'attaquant utilisait pour ses connexions depuis le système compromis vers des systèmes cibles dans le mouvement latéral, même si ces clés ne sont jamais stockées sur disque. Notre service de pentest inclut des exercices Red Team Linux avec des techniques d'extraction de credentials depuis la mémoire, permettant de valider l'efficacité des protections mémoire déployées (swap chiffré, credential guard) dans votre environnement spécifique.

Détection des techniques anti-forensiques Linux

Les attaquants sophistiqués emploient plusieurs techniques anti-forensiques spécifiques à Linux que les investigateurs doivent reconnaître et contrecarrer. Les plus courantes documentées dans des incidents réels en France : la suppression sélective de lignes dans `.bash_history` via `sed -i` (détectable par la discontinuité des timestamps dans les logs auditd correspondants), la modification des timestamps via `touch -t [DATETIME] [FICHIER]` (détectable via le ctime qui ne peut être falsifié), l'utilisation de `shred -u` pour supprimer des fichiers de manière irrecuperable sur ext3/ext4 (détectable dans les logs auditd via l'appel unlinkat sur le fichier avant sa suppression), et la désactivation temporaire d'auditd via `auditctl -e 0` (journalisé dans le log auditd lui-même avant la désactivation).

La technique de vider les logs via la troncation (`> /var/log/auth.log` ou `truncate -s 0`) plutôt que la suppression est détectable par l'absence d'entrées auditd pour la période concernée combinée avec l'existence du fichier mais avec une taille nulle ou inhabituellement petite. Le fichier `/var/log/auth.log` devrait croître de manière continue sur un serveur actif — une réduction soudaine de sa taille est un signal d'alarme. Sur les systèmes avec `logrotate` configuré, les archives compressées dans `/var/log/auth.log.*.gz` peuvent avoir préservé les logs antérieurs à la tentative de nettoyage. Notre service RSSI externalisé inclut la configuration d'un pipeline de log forwarding vers un SIEM centralisé externe qui rend ces techniques anti-forensiques inefficaces en préservant les logs hors du système compromis.

Forensique des conteneurs Docker et Kubernetes : artefacts spécifiques

La forensique des incidents dans des environnements conteneurisés Docker et Kubernetes présente des spécificités importantes par rapport à la forensique des systèmes Linux bare-metal. Les conteneurs Docker maintiennent leurs couches de système de fichiers dans `/var/lib/docker/overlay2/` sur le nœud hôte, permettant une analyse des fichiers du conteneur même après son arrêt. La commande `docker diff [CONTAINER_ID]` sur un conteneur encore présent liste tous les fichiers ajoutés (A), modifiés (C) ou supprimés (D) dans le système de fichiers du conteneur par rapport à son image de base, révélant immédiatement les modifications effectuées pendant l'incident.

Les logs des conteneurs Docker (`docker logs [CONTAINER_ID]`) sont persistants après l'arrêt du conteneur et stockés dans `/var/lib/docker/containers/[ID]/[ID]-json.log` , mais sont perdus si le conteneur est supprimé (`docker rm`). La forensique Kubernetes est plus complexe car les pods peuvent s'être redémarrés ou avoir été supprimés : les logs sont accessibles via `kubectl logs [POD] --previous` pour le cycle d'exécution précédent, et les événements Kubernetes via `kubectl get events -n [NAMESPACE] --sort-by='.lastTimestamp'` fournissent une chronologie des créations, suppressions et erreurs de pods. La configuration d'un pipeline de log forwarding centralisé (Fluentd, Fluent Bit) vers un SIEM externe est donc indispensable dans les architectures Kubernetes pour garantir la disponibilité des logs forensiques après un incident, indépendamment du cycle de vie des pods et des conteneurs.

Artefacts spécifiques par distribution Linux : Ubuntu, RHEL, Debian

Les chemins et formats des artefacts forensiques varient significativement selon la distribution Linux utilisée en production. Cette variation peut induire des erreurs lors d'investigations sur des systèmes peu familiers, comme chercher les logs SSH dans `/var/log/secure` (RHEL/CentOS) alors qu'ils sont dans `/var/log/auth.log` (Debian/Ubuntu). Le tableau de référence suivant liste les emplacements des principaux artefacts forensiques selon les distributions les plus courantes en production dans les organisations françaises.

Artefact forensique	Ubuntu/Debian	RHEL/CentOS/Rocky	Format
Logs SSH	/var/log/auth.log	/var/log/secure	Texte syslog
Logs système	/var/log/syslog	/var/log/messages	Texte syslog
Logs auditd	/var/log/audit/audit.log	/var/log/audit/audit.log	Format audit
Journal systemd	/var/log/journal/	/var/log/journal/	Binaire journal
Historique paquets	/var/log/apt/history.log	/var/log/dnf.log	Texte
Cron logs	/var/log/syslog (grep cron)	/var/log/cron	Texte syslog
Connexions (wtmp)	/var/log/wtmp	/var/log/wtmp	Binaire utmp
Noyau (dmesg)	/var/log/kern.log	journalctl -k	Texte / journal

La connaissance de ces différences inter-distributions est particulièrement importante dans les grandes organisations françaises qui maintiennent des parcs hétérogènes avec coexistence de serveurs Ubuntu LTS et RHEL dans différents départements. Les outils forensiques comme Plaso gèrent ces différences automatiquement via leurs parsers de distribution, mais l'investigateur doit connaître les chemins natifs pour le triage manuel initial. Les guides de sécurité de l'ANSSI sur la sécurisation des systèmes d'information Linux détaillent les configurations recommandées pour chacune des principales distributions en usage en France. Les référentiels forensiques du CERT-FR fournissent également des guides d'investigation spécifiques aux incidents les plus courants observés sur les systèmes Linux français.

Rédaction du rapport forensique : structuration et preuves pour NIS 2

Le rapport d'investigation forensique Linux produit à l'issue d'un incident cybersécurité doit répondre aux exigences de notification NIS 2 (si applicable) et aux standards de qualité permettant son utilisation dans un cadre judiciaire ou réglementaire. Il doit documenter : la chaîne de custody des preuves collectées (hash SHA256 de tous les artefacts collectés, identité de l'investigateur, horodatage de la collecte), la chronologie reconstituée de l'incident avec les

sources de chaque événement documenté, les IOCs (Indicateurs de Compromission) avec leur format STIX/TAXII pour partage avec l'ANSSI et le CERT-FR, l'impact estimé (données exfiltrées, systèmes compromis, durée de la compromission), et les recommandations de remédiation priorisées par criticité.

Notre équipe de RSSI externalisé propose des services d'investigation forensique Linux pour les organisations victimes d'incidents cybersécurité, incluant la collecte d'artefacts, l'analyse avec Plaso/Volatility 3, la rédaction du rapport d'incident conforme NIS 2, et la notification ANSSI si requise. Pour une préparation proactive, notre service comprend également l'audit de la configuration forensique de vos systèmes Linux (auditd, log forwarding, politique de rétention des logs) et la formation des équipes IT aux procédures de premier triage lors d'un incident. Contactez-nous via le diagnostic NIS 2 pour une évaluation de votre capacité forensique Linux actuelle et les recommandations d'amélioration prioritaires selon votre secteur.

Questions fréquentes

Peut-on récupérer le contenu d'un bash_history effacé ?

Partiellement oui, via plusieurs méthodes : lecture de `/proc/[PID_BASH]/fd` si un processus bash avait le fichier ouvert avant suppression, analyse des logs auditd configurés pour journaliser les execve, et reconstruction depuis les buffers mémoire via Volatility 3 plugin linux.bash. Sans auditd configuré et sans processus bash actif au moment de la suppression, la récupération reste limitée aux éventuelles sauvegardes ou archives syslog qui auraient capturé certaines commandes.

Combien de temps faut-il pour une investigation forensique Linux complète ?

Une investigation forensique Linux complète sur un serveur de production compromis prend typiquement 3 à 10 jours selon la complexité de l'incident et le volume de logs disponibles. Le triage initial (collecte et premier inventaire des artefacts) prend 2-4 heures. La création de la super-timeline avec Plaso et l'analyse des sources clés (auditd, wtmp, syslog, mémoire) prend 1-3

jours. La rédaction du rapport complet avec chronologie et IOCs prend 1-2 jours supplémentaires.

auditd a-t-il un impact sur les performances des serveurs Linux ?

Un impact mesuré de 3 à 8% sur les systèmes très actifs en I/O avec les règles STIG complètes. Cet impact est généralement acceptable pour les serveurs de production sensibles. Pour les serveurs très chargés, une configuration allégée journalisant uniquement les événements critiques (execve sur les utilisateurs à hauts privilèges, accès aux fichiers sensibles, modifications réseau) réduit l'impact à moins de 2%. L'alternative est de collecter les événements via un agent eBPF comme Tetragon qui a un overhead similaire mais une architecture différente.

Comment préserver l'intégrité des preuves Linux pour une procédure judiciaire ?

La préservation forensique légale requiert : calcul du hash SHA256 de chaque source de preuve avant toute analyse, documentation horodatée de chaque action d'investigation, montage des partitions en read-only via `mount -o ro` avant toute analyse, utilisation d'outils validés (Autopsy, Plaso) avec journalisation de leurs sorties, et stockage des preuves sur des supports d'écriture sécurisés avec hash de vérification. En France, la réquisition judiciaire doit être documentée si les données appartiennent à des tiers.

Volatility 3 supporte-t-il toutes les versions du noyau Linux ?

Volatility 3 nécessite un profil de symboles (ISF — Intermediate Symbol Format) correspondant à la version exacte du noyau Linux du système analysé. Ces profils doivent être générés depuis les packages de debug-symbols de la distribution (linux-image-*-dbg sur Debian/Ubuntu, kernel-debuginfo sur RHEL). Pour les versions noyau courantes des distributions mainstream, des profils pré-générés sont disponibles dans le repository dwarf2json ou peuvent être générés rapidement depuis un système identique.

Faut-il éteindre le système ou continuer à l'analyser à chaud lors d'un incident Linux ?

L'analyse à chaud est fortement recommandée en priorité pour capturer les artefacts volatils : processus actifs, connexions réseau, mémoire RAM (via LiME), programmes eBPF, et sockets. L'extinction avant collecte mémoire détruirait irrémédiablement ces artefacts. La séquence optimale est : isolation réseau d'abord (pour couper le C2 sans éteindre), puis capture mémoire LiME, puis collecte des artefacts disque à chaud, puis arrêt planifié du système si nécessaire pour une analyse forensique offline.

Accompagnement forensique Linux : expertise et intervention d'urgence

La forensique Linux requiert une expertise spécialisée que peu d'équipes IT internes possèdent complètement : connaissance des internals du noyau Linux, maîtrise des outils forensiques (Plaso, Volatility 3, LiME), compréhension des artefacts de chaque distribution et version noyau, et expérience des procédures légales de préservation des preuves. Notre service de RSSI externalisé inclut une capacité d'intervention forensique Linux avec des SLAs d'intervention définis selon la criticité de l'incident (4h pour les incidents critiques P1, 24h pour les incidents élevés P2). Pour les organisations NIS 2, nous pouvons également prendre en charge la notification à l'ANSSI dans les délais réglementaires requis.

Notre service de pentest intègre également des scénarios de simulation d'incident Linux pour tester vos capacités de détection et de réponse forensique : l'équipe Red Team implante des artefacts simulés et l'équipe Blue Team doit les trouver et les documenter dans un délai imparti. Ces exercices de type Cyber Range Linux révèlent les lacunes dans votre configuration forensique (auditd mal configuré, logs non centralisés, absence de LiME pré-installé) avant qu'un incident réel ne les expose. Contactez-nous via le diagnostic NIS 2 pour une évaluation de votre posture forensique Linux et un plan d'amélioration adapté à votre infrastructure.