

— OSSIR · MARDI 13 OCTOBRE 2026

Rustinel

Anatomie d'un moteur de détection endpoint multiplateforme

Théo Foucher

v1.9.1 · Apache 2.0

Windows · Linux · macOS

Sigma · YARA · IOC

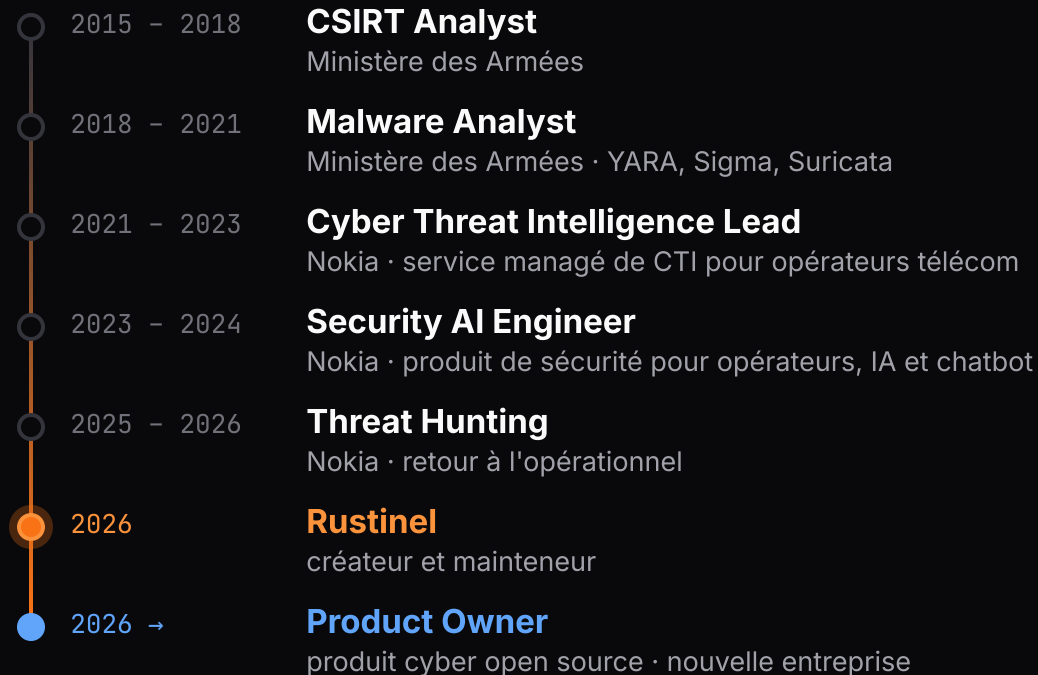


Qui suis-je ?

Théo Foucher

Threat intelligence, hunting et détection, bientôt côté produit

- 10 ans côté défense : CSIRT, analyse de malware, CTI, hunting
- Des années à écrire des règles **YARA** et **Sigma**
- Enseignant en analyse de malware et DFIR (EPITA, ENSSAT)
- Réserviste opérationnel cyberdéfense



— LE FIL ROUGE

Jusqu'où peut-on aller avec la **seule** **télémetrie native** des OS ?

Sans driver supplémentaire ni module noyau, avec les mécanismes fournis par les OS.

ETW

eBPF

Endpoint Security



Rustinel en une phrase

Des règles Sigma, YARA et IOC appliquées à la télémétrie native de Windows, Linux et macOS, avec des alertes en local.

3

OS : Windows, Linux, macOS

3

moteurs : Sigma, YARA, IOC

27

versions publiées depuis février 2026

0

driver supplémentaire ou module noyau *



* Sur Linux, des programmes eBPF s'exécutent dans le noyau : du code vérifié et borné, pas un module. Détails en partie 04.

Ce que Rustinel est, et ce qu'il n'est pas



Un moteur de détection

- Open source, un binaire par OS
- Détection et alertes locales par défaut
- Labs, detection engineering, tests de SIEM
- Machines sans EDR



Pas une plateforme EDR complète

- Pas d'anti-tamper
- Pas de blocage avant exécution
- Pas de console de gestion
- La réponse active existe, mais agit après coup

Au programme

02 Le problème et le paysage

03 Architecture générale

04 Comment Rustinel observe chaque OS

05 Le moteur de détection et ses règles

06 Démo

07 Écosystème, limites et ce qui vient

08 Questions

Trois OS, trois mondes

W Windows

ETW et Event Log

ETW : le système de traçage intégré à Windows. Des fournisseurs (providers) y émettent des événements typés.

Riche, mais une partie dépend de la politique d'audit.

L Linux

eBPF, tracepoints, auditd

eBPF : de petits programmes, vérifiés par le noyau, attachés à des points d'observation. Pas d'API de sécurité unique.

Avec eBPF, c'est vous qui écrivez le collecteur.

M macOS

Endpoint Security

Endpoint Security : l'API officielle d'Apple, qui notifie les outils de sécurité des événements (processus, fichiers...).

Apple décide de ce qui est visible, avec des autorisations à obtenir.

Même question sur chaque machine : **qui a lancé quoi, avec quelle ligne de commande, et qu'a-t-il touché ?** Trois réponses différentes.

Sigma comme langage commun

```
title: Découverte via whoami
logsource:
  category: process_creation
detection:
  selection:
    Image|endswith:
      - '\whoami.exe'
      - '/whoami'
  condition: selection
```

Kernel-Process

ETW · Windows

exec (eBPF)

tracepoint · Linux

ES exec

Endpoint Security · macOS

Champs Sysmon

Image
CommandLine
ParentImage
User · ProcessId

La vraie question : **une règle qui se charge peut-elle réellement se déclencher sur cet OS ?**

Ce qui existe déjà

OUTIL	OS	SOURCE	RÔLE
Sysmon / Sysmon for Linux	Windows, Linux	Driver / eBPF	Collecte, pas de moteur de détection
Aurora (Nextron)	Windows	ETW	Détection Sigma, propriétaire (Lite gratuite)
Fibratus	Windows	ETW	Détection, langage propre + YARA, open source
Falco	Linux	eBPF / module	Détection, langage propre, conteneurs
Tetragon	Linux	eBPF	Observation et blocage, avec ou sans Kubernetes
osquery	Multi	API OS	Requêtes SQL sur l'état du système
Wazuh	Multi	Logs + agent	Plateforme HIDS/SIEM avec serveur
Velociraptor	Multi	API OS + ETW	DFIR et hunting, collecte autonome possible

Panorama au 12 octobre 2026. Voir la documentation de chaque outil.

Où se place Rustinel

Même philosophie qu'**Aurora** et **Fibratus** sur Windows : ETW natif, pas de driver. Rustinel l'étend à **Linux** et **macOS**.



Multiplateforme

Un moteur, un format de config, les mêmes champs sur trois OS.



Sigma, YARA, IOC

Comportement (Sigma), fichiers (YARA), indicateurs atomiques (IOC), nativement.



Local par défaut

Alertes ECS NDJSON locales.
Envoi optionnel par webhook.

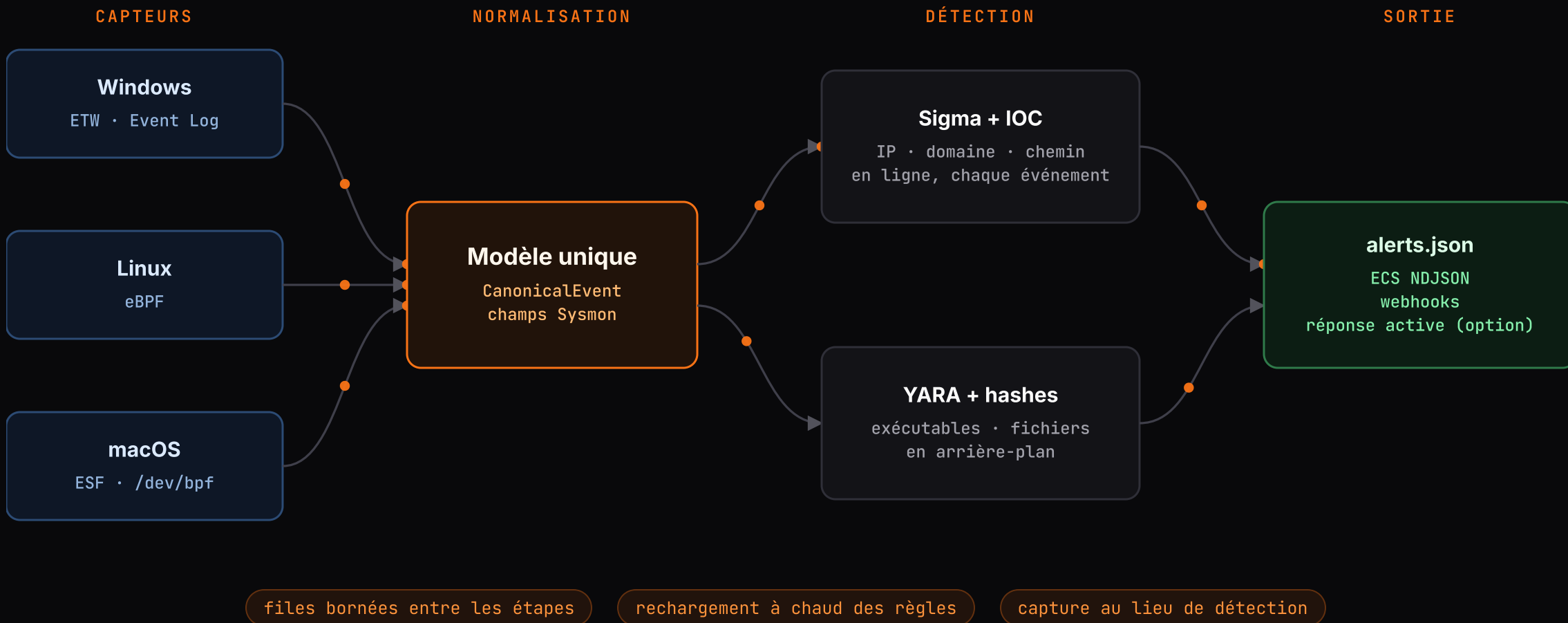


Capture et replay

Enregistrer une fois, rejouer à chaque évolution des règles.

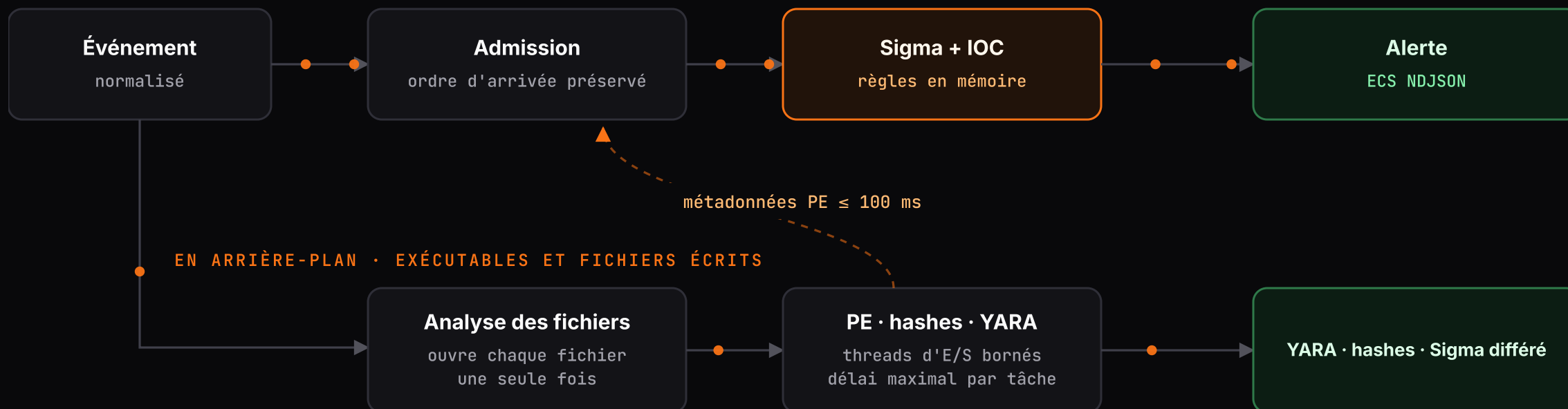


Un pipeline, trois capteurs



Deux vitesses de détection

EN LIGNE · CHAQUE ÉVÉNEMENT



Quand la file est pleine



186

acceptés

23

perdus, mais comptés

14/14

occupation maximale

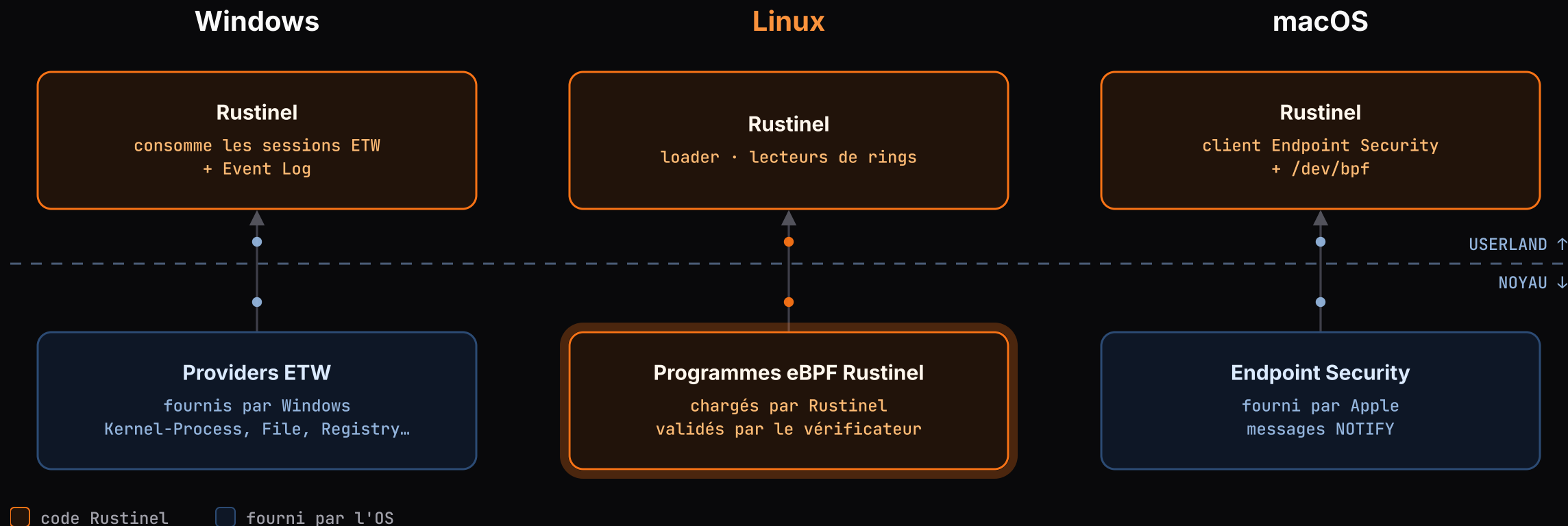
1 Le choix

File pleine : l'événement est abandonné et la perte est comptée, sans bloquer le capteur.

2 La raison

Bloquer un callback ETW ou un lecteur de ring eBPF déplace la perte **dans le noyau**, où elle est bien moins visible.

Qui fait quoi, de chaque côté de la frontière



Ce qu'on gagne, ce qu'on perd

+ Gains

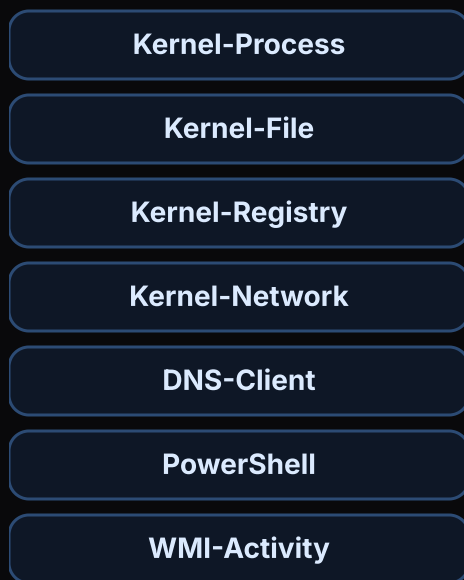
- Pas de driver à faire signer par Microsoft
- Moins de code privilégié à maintenir
- Installation guidée
- Pas de driver propriétaire à adapter

- Pertes

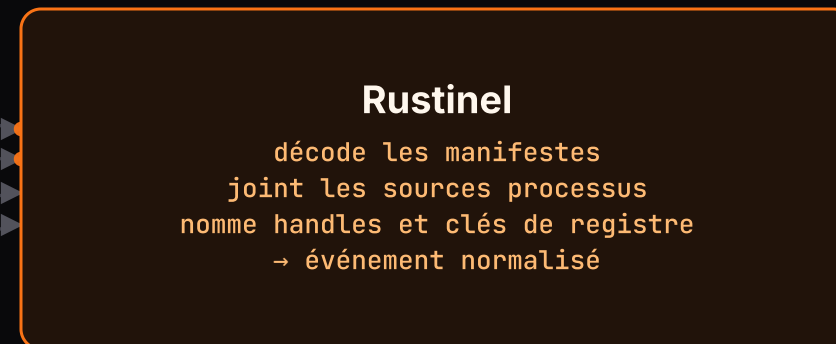
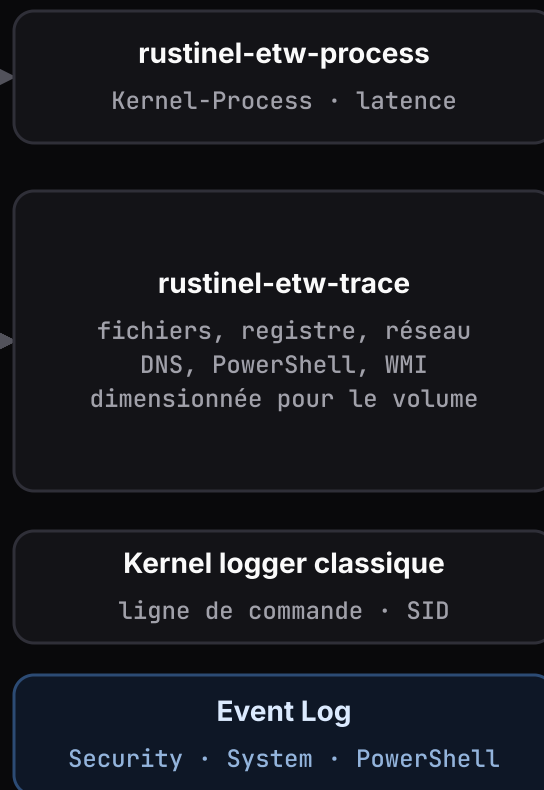
- Choix d'observer : la réponse agit *après* l'événement
- Pas d'anti-tamper
- On ne voit que ce que l'OS expose
- Ex. : le provider ETW Threat-Intelligence est réservé aux processus PPL antimalware

ETW et Event Log

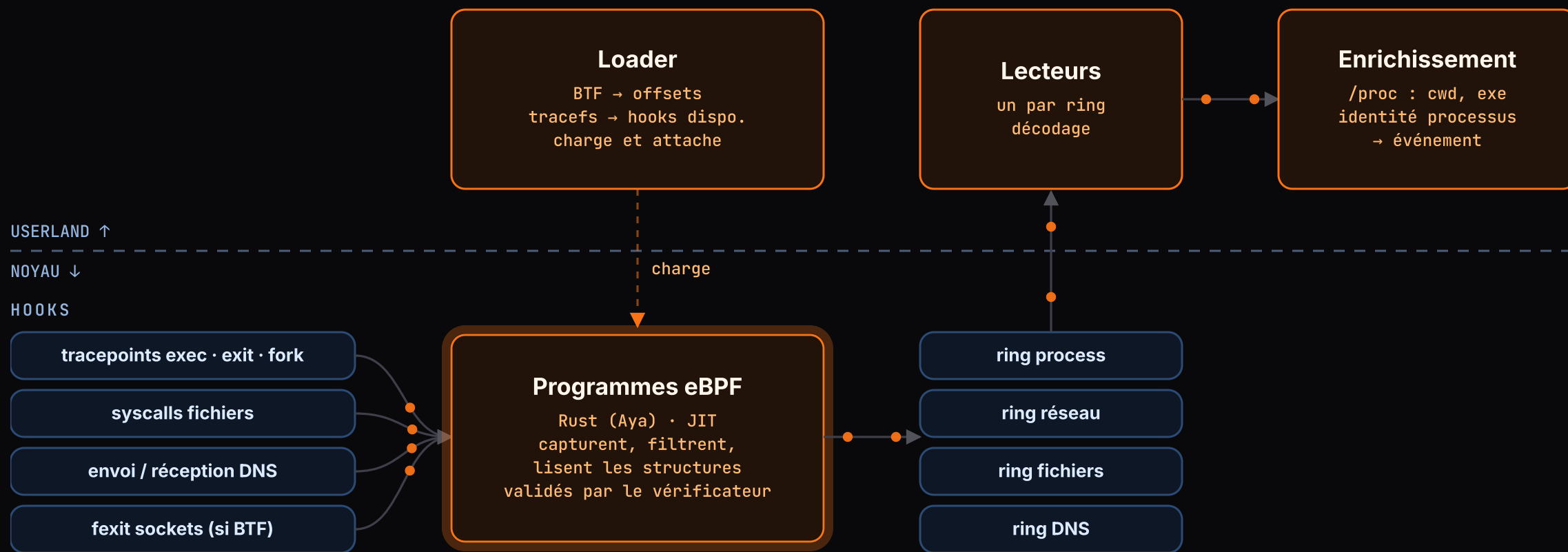
PROVIDERS ETW



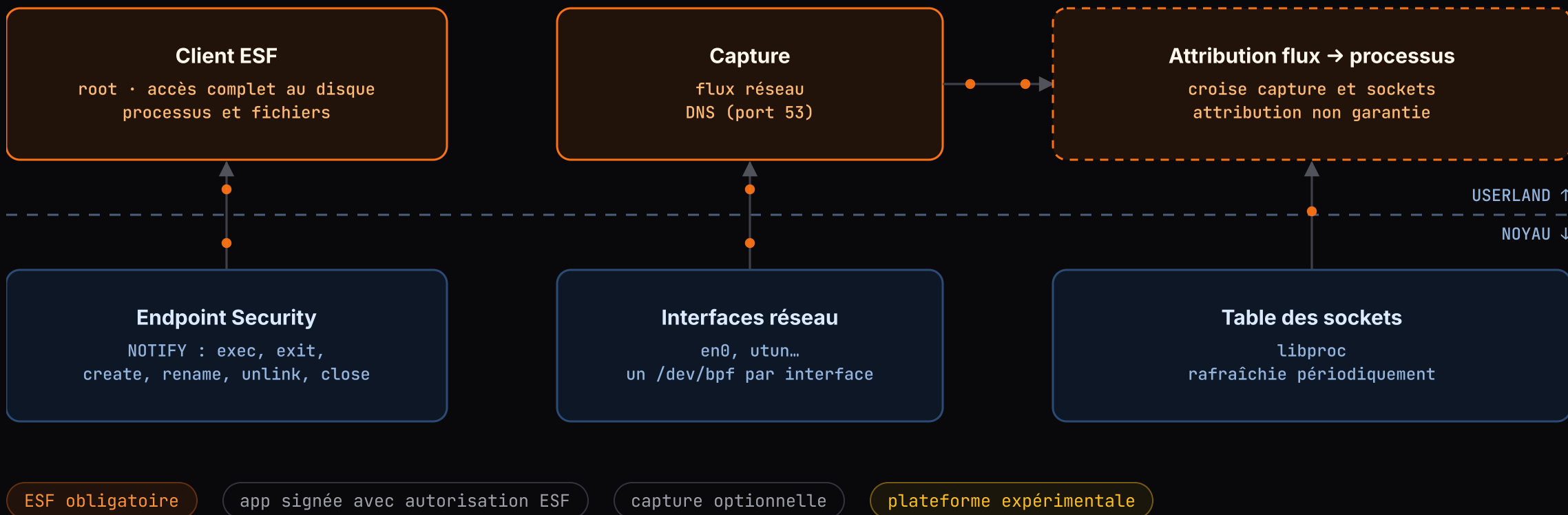
SESSIONS ET CANAUX



eBPF : noyau et espace utilisateur



Endpoint Security et /dev/bpf



Trois capteurs, trois couvertures

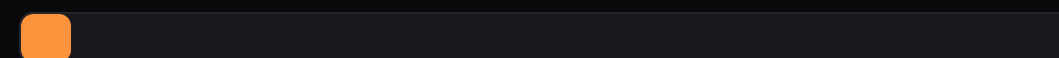
	WINDOWS	LINUX	MACOS
Mécanisme	ETW + Event Log	eBPF	ESF + /dev/bpf
Code dans le noyau	Non	Oui	Non
Collecte	Très large	Process, réseau, fichiers, DNS	Process, fichiers, réseau, DNS
Statut	Stable	Stable	Expérimental

RÈGLES SIGMAHQ CHARGEABLES • 3 783 TESTÉES EN CI

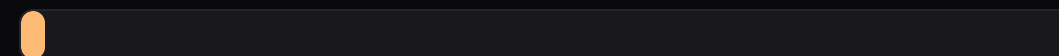
Windows 2 709



Linux 178



macOS 86



Mêmes noms de champs ≠ même couverture.

Trois moteurs, trois questions

Σ Sigma

Que fait ce processus ?

Détection comportementale et corrélations avec RSigma. Règles sur les hashes en différé.

en ligne ou en différé

IOC

Est-ce déjà connu ?

Détection atomique : une valeur connue, sans contexte. Hashes, imphash, IP, domaines, chemins.

IP • domaine • chemin en ligne

Y YARA

Que contient ce fichier ?

Détection de fichiers : chaînes, métadonnées et code dans les exécutables et fichiers écrits. Mémoire selon la configuration et le type d'exécution.

arrière-plan • ressources limitées

Packs prêts à l'emploi : **rustinel-rules**, installés par `rustinel setup`. Rechargement à chaud, et en cas d'échec, les règles précédentes restent actives.

Le contenu : rustinel-rules

R rustinel · le moteur

Collecte la télémétrie et évalue les règles.

v1.9.1

Apache 2.0

Σ rustinel-rules · le contenu

Sigma, YARA et IOC, versionnés à part. Une règle = un fichier, un identifiant stable. Les packs ne font que les référencer.

v0.4.0

DRL 1.1



	ESSENTIAL	PEU DE FP	ADVANCED	FP MOYENS	HUNTING	FP ÉLEVÉS
Windows	19 règles		39		42	
Linux	10		26		27	
macOS	expérimental	5	21		.	

Catalogue **signé** (minisign), version minimale du moteur déclarée par pack, mise à jour par rustinel rules update.

Cette règle peut-elle se déclencher ?

```
$ rustinel sigma doctor --platform windows
```

can-fire

Source prise en charge. Les champs mesurés fiables ne donnent que des réserves.

degraded

Évaluation limitée par un champ conditionnel, dérivé ou tronqué, jamais mesuré (ex. OriginalFileName).

can-never-fire

Collecteur absent, champ jamais rempli ou dépendance manquante.

Analyse de compatibilité. Un test reste nécessaire pour valider la détection.

SigmaHQ compte 3 783 règles : **2 709** se chargent sur Windows, **178** sur Linux, **86** sur macOS. Le nombre de règles **n'est pas** la couverture.

CHAMPS MESURÉS FIABLES • process_creation

Windows CommandLine · Image · ParentImage · ParentProcessId

Linux CommandLine · Image · ParentImage · ParentCommandLine

macOS CommandLine · Image

Tout autre champ reste une limite tant qu'il n'est pas mesuré.

PACK WINDOWS DE LA DÉMO • 17 RÈGLES

8

9

■ can-fire (avec réserves) ■ dégradées ■ 0 can-never-fire

Detection as Code

La règle • extrait simplifié

```
title: Encoded PowerShell (demo)
logsource:
  category: process_creation
  product: windows
detection:
  selection:
    Image|endswith: '\powershell.exe'
    CommandLine|contains:
      - ' -EncodedCommand '
  condition: selection
level: high
```

Le test atomique • encoded_powershell.ps1

```
# Test de la règle complète
# Charge utile bénigne : un simple Write-Host.

$enc = [Convert]::ToBase64String(
    [Text.Encoding]::Unicode.GetBytes(
        'Write-Host rustinel-atomic-encoded'))

& powershell.exe -EncodedCommand $enc

# Alerte attendue, sinon échec du test
```

À chaque changement, la CI **valide** la règle, exécute le test sur une version fixe de Rustinel, et lance `sigma doctor` sur les packs.

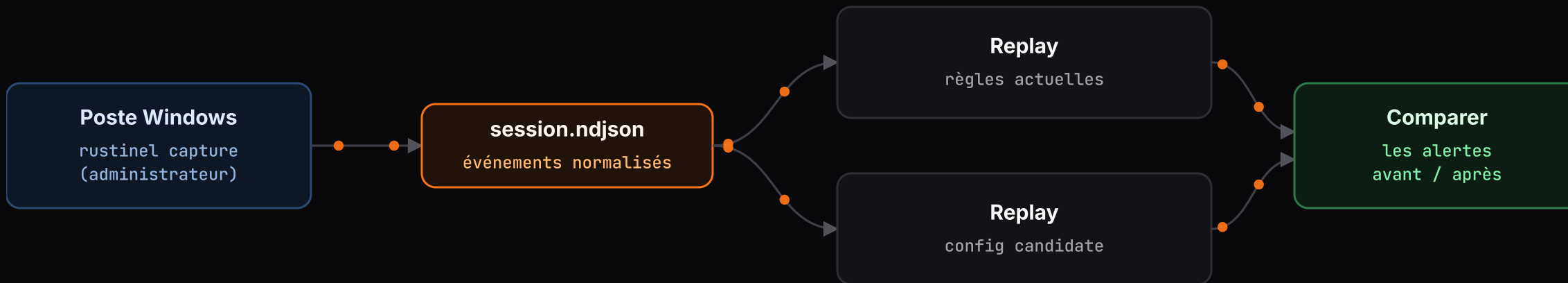
Les commandes principales

rustinel - zsh

```
$ rustinel setup           # configure et installe le service
$ rustinel run             # lance la détection au premier plan
$ rustinel doctor          # vérifie la santé de l'agent
$ rustinel sigma doctor    # vérifie la compatibilité des règles
$ rustinel capture         # enregistre les événements
$ rustinel replay          # réévalue une capture sans droits admin
$ rustinel rules update    # met à jour le pack de règles
$ rustinel update          # met à jour le binaire
```

```
$ █
```

Une capture, plusieurs jeux de règles



```
PS> rustinel capture --output session.ndjson # Windows (admin)
# Ensuite, sans droits administrateur
$ rustinel replay session.ndjson
$ rustinel replay session.ndjson --config candidate.toml
```

Événements uniquement : pas de scan YARA ni de calcul de hashes en replay.

● DÉMO

Rustinel en vrai

1 Alertes en direct

whoami · PowerShell encodé · mshta

2 Capture et replay

même trace, deux jeux de règles

3 sigma doctor

verdict par règle sur le pack Windows

Un écosystème open source européen



Trois projets open source, quatre pays européens, et des auteurs qui contribuent aussi au code de Rustinel.

Ce qui vient

Télémétrie

Surtout Windows, puis Linux. macOS à l'étude.

Fiabilité

Détecter les pertes d'événements et les capteurs coupés.

Moteur

Détection parallèle, ancêtres de processus, IOC avec expiration.

Règles

Environ 150 règles à écrire, avec des tiers fondés sur des preuves.

Le backlog est public. Une contribution : **une règle, plus un test.**

Détecter, tester et connaître ses limites

✓ Cas d'usage

- Labs blue / purple team
- Detection engineering
- Tester un pipeline SIEM
- Postes sans EDR

! Limites

- Couverture inégale selon l'OS
- macOS expérimental
- Pas de prévention, pas d'anti-tamper

→ Contribuer

- Retours terrain
- Tests de détection
- Travail sur les plateformes



— MERCI

Questions ?

Un moteur de détection qu'on peut lire, exécuter, tester, et essayer de contourner.

github.com/Karib0u/rustinel

github.com/Karib0u/rustinel-rules

docs.rustinel.io • rustinel.io



Annexes

Slides de secours pour les questions : Rust, callbacks vs natif, détails Windows et Linux, artefacts, observer et sécuriser l'agent.

Pourquoi Rust

1 Un seul binaire

Pas de runtime à installer. Les mêmes sources pour les trois OS.

2 Sûreté mémoire

L'agent tourne en root ou SYSTEM et parse des données contrôlables par un attaquant : payloads ETW, fichiers PE, règles.

3 Coût prévisible

Pas de ramasse-miettes. Files et caches limités pour maîtriser les ressources.

Le même langage jusque dans le noyau : les programmes eBPF sont écrits en **Rust avec Aya**.

Intercepter ou observer

EDR AVEC DRIVER • CALLBACK SYNCHRONE



RUSTINEL • TÉLÉMÉTRIE NATIVE, ASYNCHRONE



Un choix de collecte : ETW, points de trace Linux et notifications ESF. Sur macOS, les événements AUTH permettraient aussi de bloquer.

Reconstruire le contexte

FileCreate FileObject=0xA1 C:\Users\bob\AppData\x.dll **nommé**

FileWrite FileObject=0xA1 (aucun chemin) **sans chemin**

↳ index FileObject → nom

FileWrite TargetFilename C:\Users\bob\AppData\x.dll **reconstruit**

Même mécanisme pour les clés de registre. Les objets ouverts avant le démarrage sont nommés grâce à un **instantané initial**.

P Processus

Kernel-Process et le kernel logger classique sont joints par PID, PID parent et horodatage.

! Limites

Audit Security et logging PowerShell dépendent de la politique de la machine.

Non collectés : accès inter-processus, threads distants, named pipes, drivers.

Un binaire, plusieurs versions du noyau



Hooks disponibles découverts dans `tracefs`. Un hook absent ne dégrade que sa fonctionnalité.

C Collecte

Processus, fichiers, DNS en clair. Réseau : tuple lu dans la structure socket du noyau.

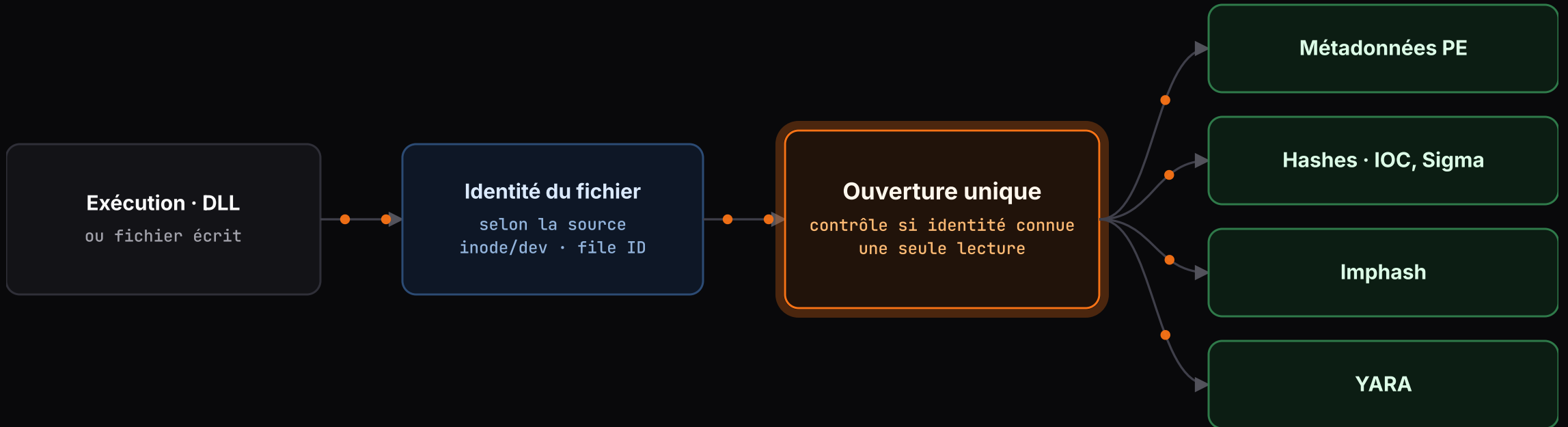
E Enrichissement

Identité processus via l'heure de naissance noyau. `/proc` pour cwd et exe. Inode et device mesurés dans le noyau.

! Limites

Noyau 5.8 minimum. Pas de DNS chiffré (DoH, DoT). Pas de lecture d'auditd.

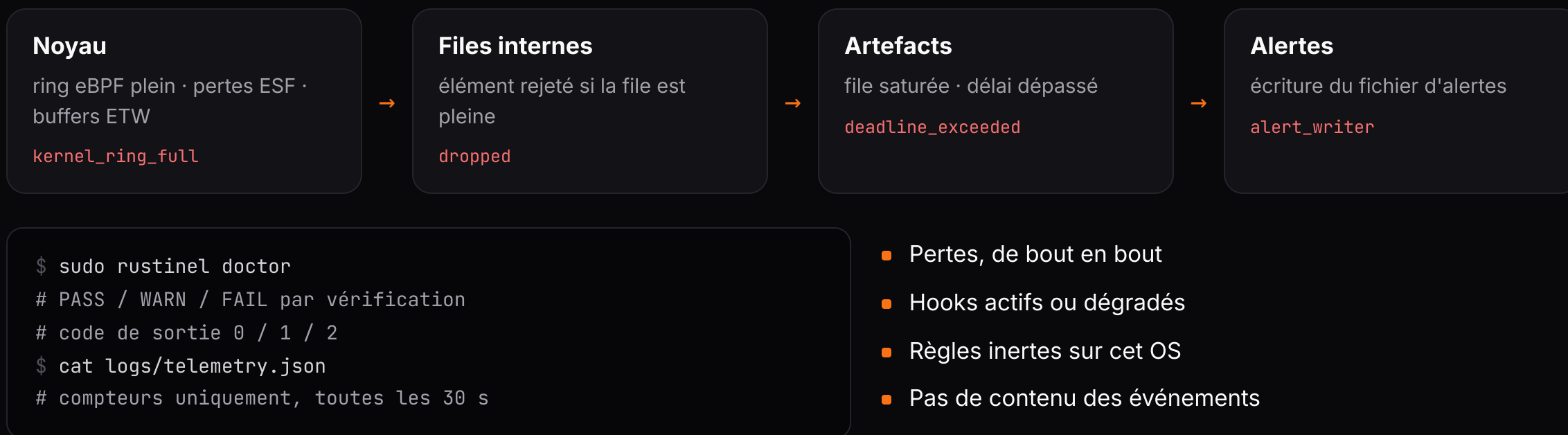
Ouvrir une fois, analyser quatre fois



Fichiers écrits : scan seulement avec une identité. Sous Windows, elle est relevée **après l'événement**.

Mémoire des processus : un worker YARA **séparé**, après un délai, sous un budget partagé.

Savoir ce que l'on ne voit pas



Un agent privilégié est une cible

1 Modèle de menace

Un processus root ou SYSTEM qui lit des configs, des règles et des fichiers de la machine.

2 Confiance des entrées

Refuse les règles, IOC et configs modifiables par un autre compte.

3 Lecture des fichiers

Identité validée à l'ouverture. `openat2 + RESOLVE_IN_ROOT` pour rester dans la racine du processus.

4 Réponse active

Handle Windows ou `pidfd` Linux : une réutilisation de PID ne redirige pas le kill. Jamais PID 0 à 4 ni lui-même.

Assumé : un attaquant **déjà administrateur** peut arrêter l'agent.