

— OSSIR · TUESDAY, OCTOBER 13, 2026

Rustinel

Anatomy of a **cross-platform** endpoint
detection engine

Théo Foucher

v1.9.1 · Apache 2.0

Windows · Linux · macOS

Sigma · YARA · IOC

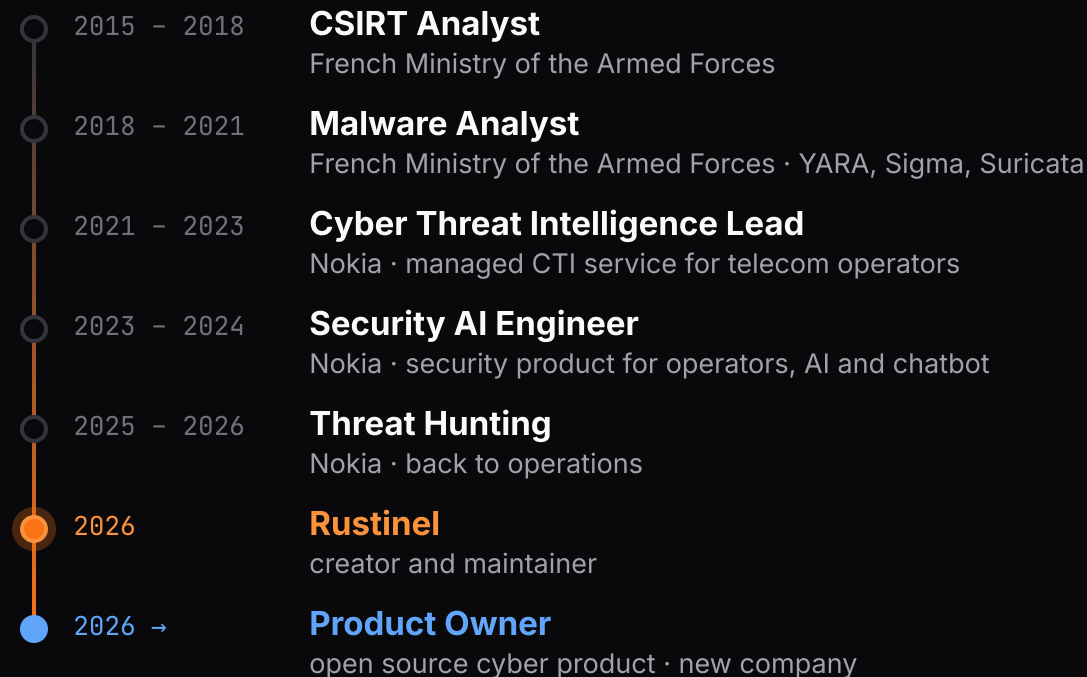


Who am I?

Théo Foucher

Threat intelligence, hunting and detection, moving to product

- 10 years on the defensive side: CSIRT, malware analysis, CTI, hunting
- Years of writing **YARA** and **Sigma** rules
- Lecturer in malware analysis and DFIR (EPITA, ENSSAT)
- Cyber defense operational reservist



— THE COMMON THREAD

How far can you go with **native OS telemetry alone?**

No additional driver or kernel module, using the mechanisms provided by each OS.

ETW

eBPF

Endpoint Security



Rustinel in one sentence

Sigma, YARA and IOC rules applied to native Windows, Linux and macOS telemetry, with alerts kept local.

3

OSes: Windows, Linux, macOS

3

engines: Sigma, YARA, IOC

27

releases since February 2026

0

additional driver or kernel module *



* On Linux, eBPF programs run in the kernel: verified, bounded code, not a module. More in part 04.

What Rustinel is, and what it is not

✓ A detection engine

- Open source, one binary per OS
- Local detection and alerts by default
- Labs, detection engineering, SIEM testing
- Machines with no EDR

✗ Not a full EDR platform

- No anti-tamper
- No pre-execution blocking
- No management console
- Active response exists, but acts after the fact

What's ahead

02 The problem and the landscape

03 Overall architecture

04 How Rustinel observes each OS

05 The detection engine and its rules

06 Demo

07 Ecosystem, limits and what's coming

08 Questions

Three OSes, three worlds

W Windows

ETW and Event Log

ETW: Windows' built-in tracing system. Providers emit typed events into it.

Rich, but part of it depends on the audit policy.

L Linux

eBPF, tracepoints, auditd

eBPF: small programs, checked by the kernel, attached to observation points. No single security API.

With eBPF, you write the collector yourself.

M macOS

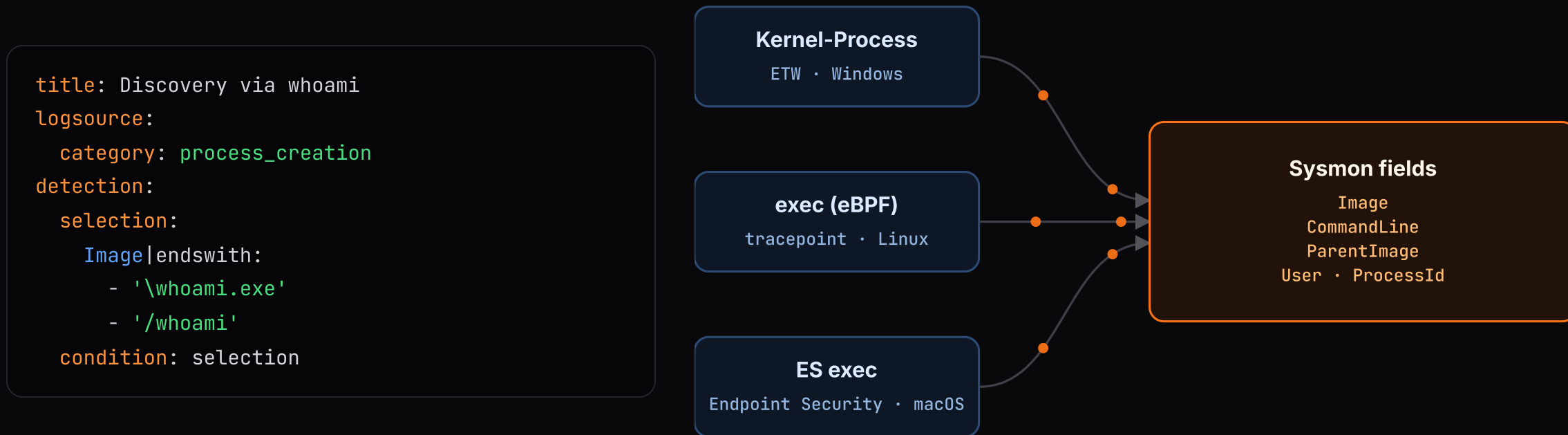
Endpoint Security

Endpoint Security: Apple's official API, which notifies security tools of events (processes, files...).

Apple decides what is visible, and you need permissions.

Same question on every machine: **who launched what, with which command line, and what did it touch?** Three different answers.

Sigma as a common language



The real question: **can a rule that loads actually fire on this OS?**

What already exists

TOOL	OS	SOURCE	ROLE
Sysmon / Sysmon for Linux	Windows, Linux	Driver / eBPF	Collection, no detection engine
Aurora (Nextron)	Windows	ETW	Sigma detection, proprietary (free Lite)
Fibratus	Windows	ETW	Detection, own language + YARA, open source
Falco	Linux	eBPF / module	Detection, own language, containers
Tetragon	Linux	eBPF	Observation and blocking, with or without Kubernetes
osquery	Multi	OS APIs	SQL queries on system state
Wazuh	Multi	Logs + agent	HIDS/SIEM platform with a server
Velociraptor	Multi	OS APIs + ETW	DFIR and hunting, standalone collection available

Overview as of October 12, 2026. See each tool's documentation.

Where Rustinel fits

Same philosophy as **Aurora** and **Fibratus** on Windows: native ETW, no driver. Rustinel extends it to **Linux** and **macOS**.



Cross-platform

One engine, one config format, the same fields on three OSes.



Sigma, YARA, IOC

Behavior (Sigma), files (YARA), atomic indicators (IOC), natively.



Local by default

Local ECS NDJSON alerts.
Optional webhook delivery.

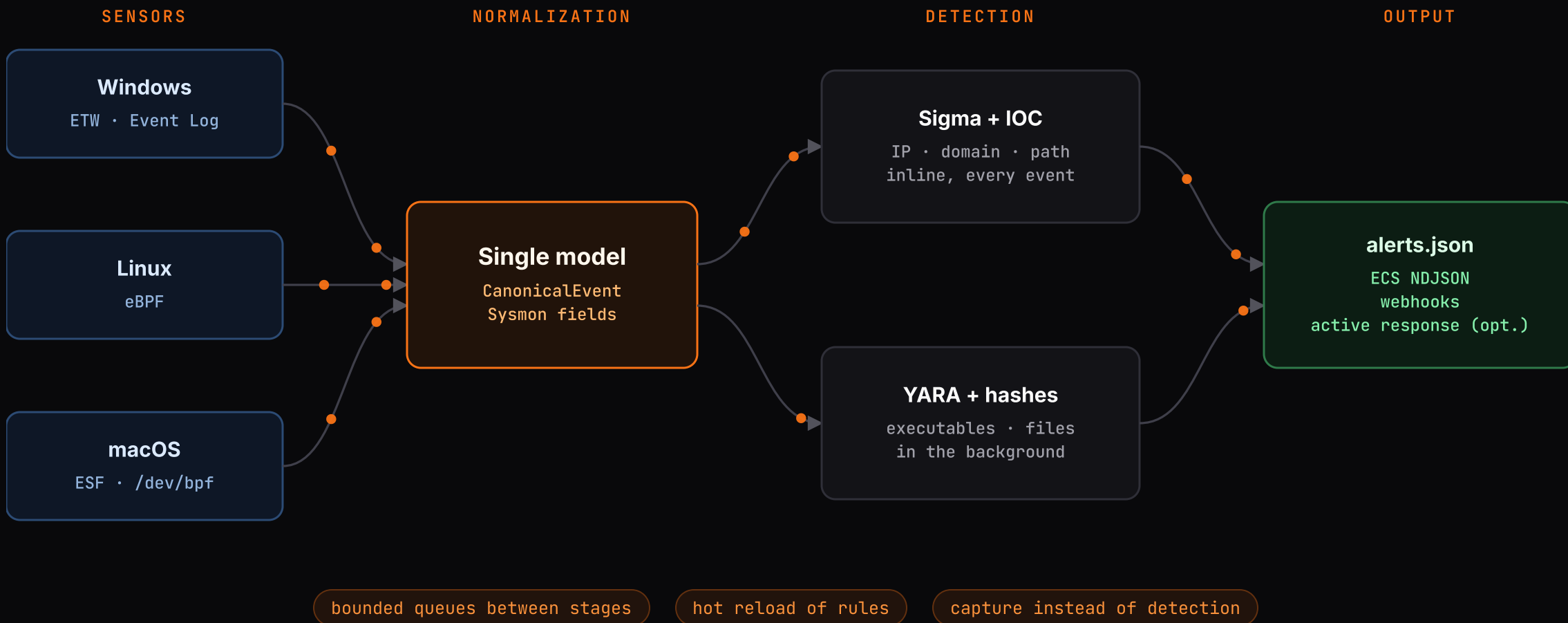


Capture and replay

Record once, replay every time your rules change.

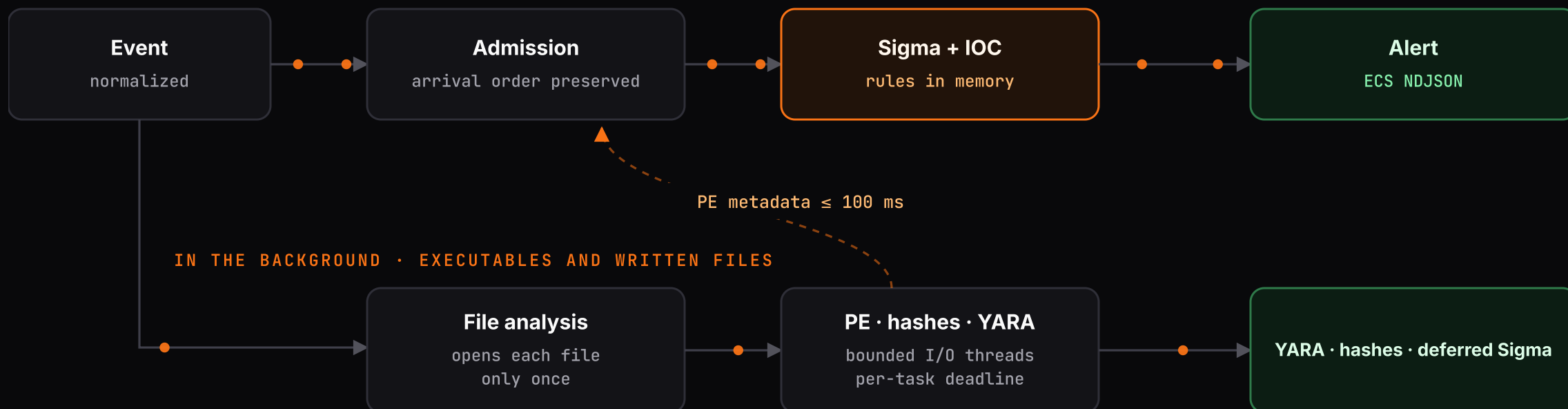


One pipeline, three sensors



Two detection speeds

INLINE · EVERY EVENT



When the queue is full



186

accepted

23

dropped, but counted

14/14

max depth (high water mark)

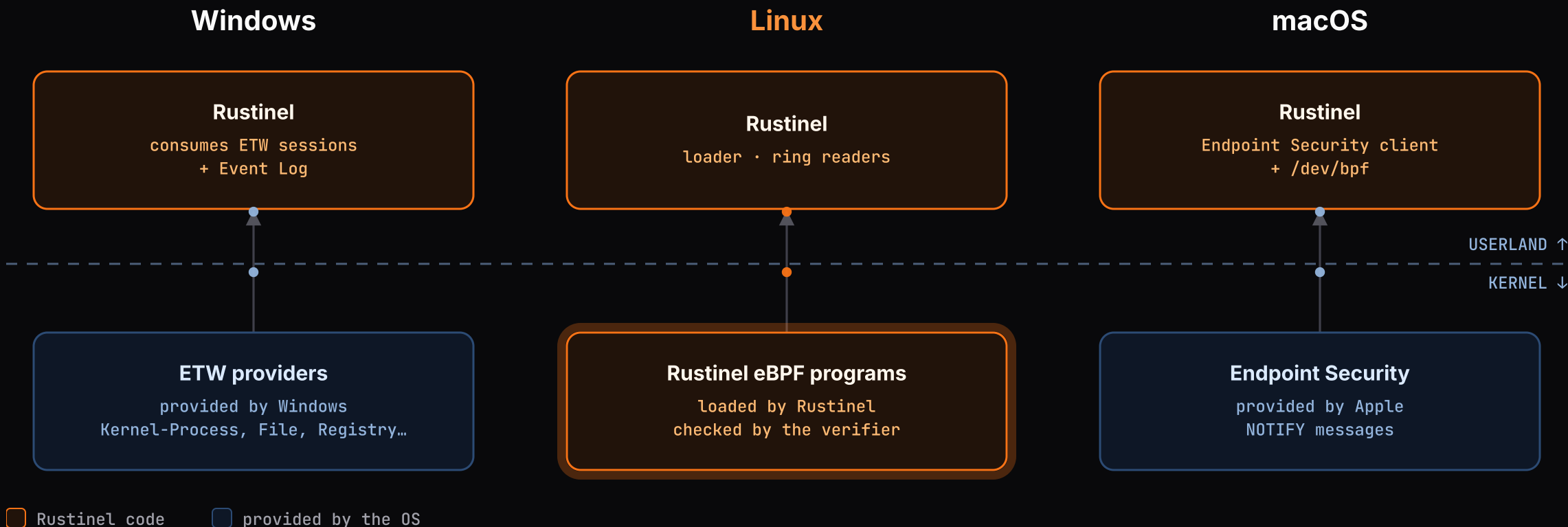
1 The choice

Queue full: drop the event and count the loss, without blocking the sensor.

2 The reason

Blocking an ETW callback or an eBPF ring reader moves the loss **into the kernel**, where it is much less visible.

Who does what, on each side of the boundary



What you gain, what you lose

+ Gains

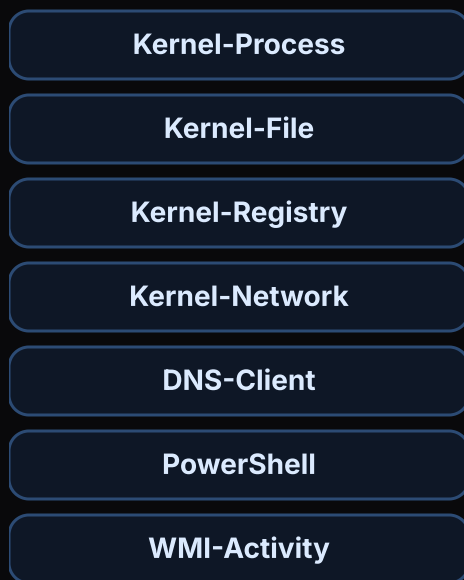
- No driver to get signed by Microsoft
- Less privileged code to maintain
- Guided installation
- No custom driver to adapt

- Losses

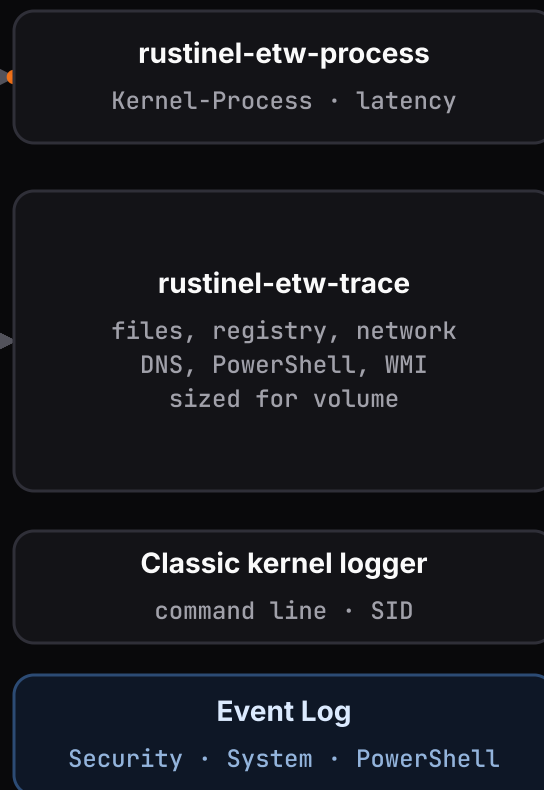
- Observation by design: response acts *after* the event
- No anti-tamper
- You only see what the OS exposes
- E.g. the ETW Threat-Intelligence provider is reserved for PPL antimalware processes

ETW and Event Log

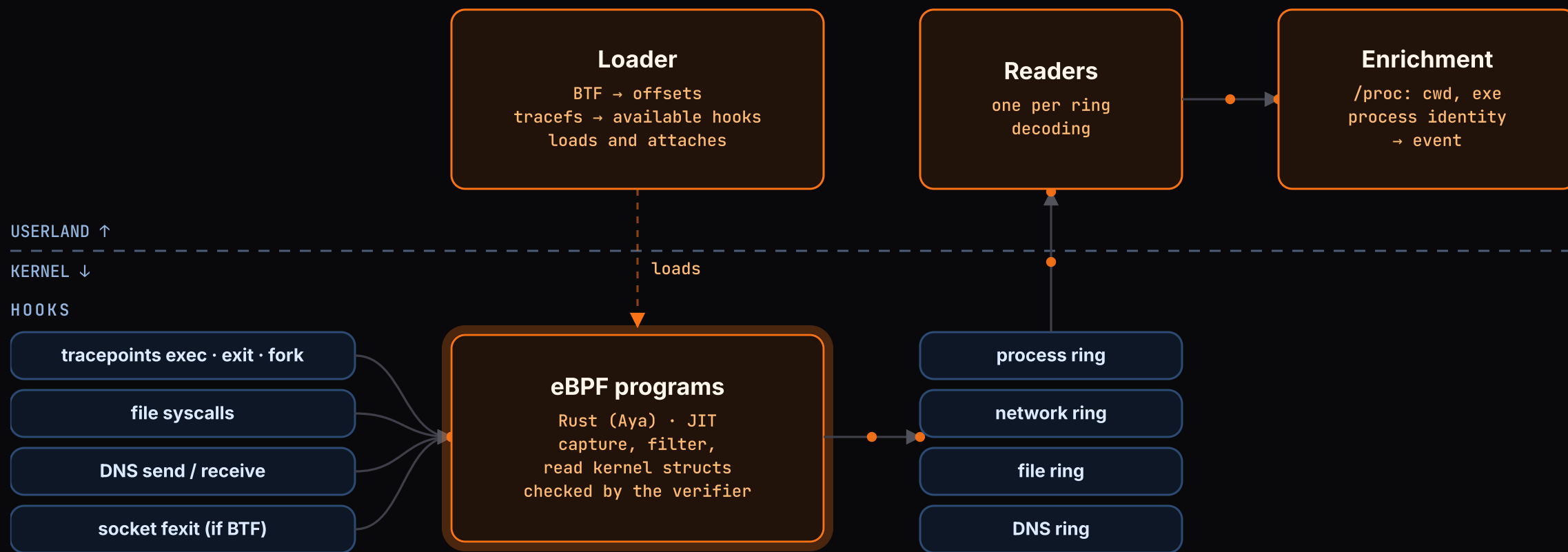
ETW PROVIDERS



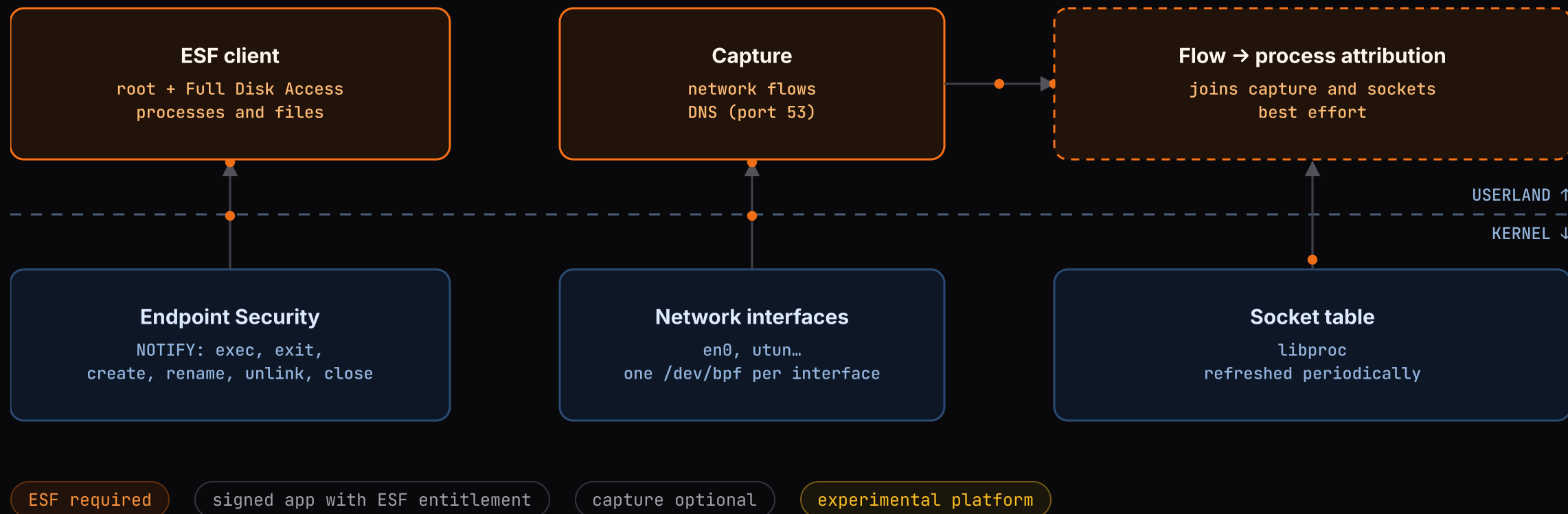
SESSIONS AND CHANNELS



eBPF: kernel and user space



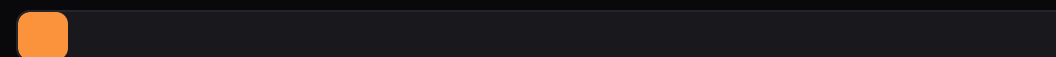
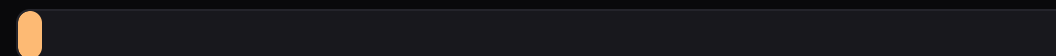
Endpoint Security and /dev/bpf



Three sensors, three coverages

	WINDOWS	LINUX	MACOS
Mechanism	ETW + Event Log	eBPF	ESF + /dev/bpf
Code in the kernel	No	Yes	No
Collection	Very broad	Process, network, files, DNS	Process, files, network, DNS
Status	Stable	Stable	Experimental

LOADABLE SIGMAHQ RULES • 3,783 TESTED IN CI

Windows 2,709**Linux** 178**macOS** 86

Same field names ≠ same coverage.

Three engines, three questions

Σ Sigma

What is this process doing?

Behavior and correlations, evaluated by RSigma. Hash-based rules run in a deferred pass.

inline or deferred

IOC

Is it already known?

Atomic detection: one known value, no context. Hashes, imphash, IPs, domains, paths.

IP • domain • path inline

Y YARA

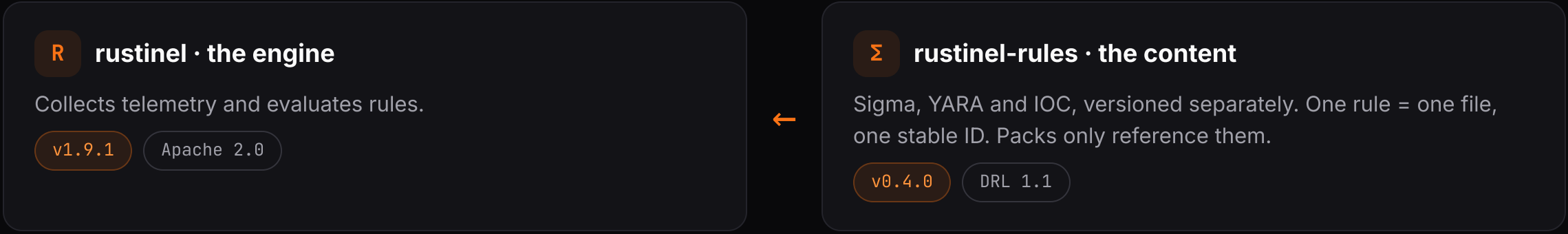
What does this file contain?

File detection: strings, metadata and code in executables and written files. Memory depends on configuration and execution type.

background • limited resources

Ready-made packs: **rustinel-rules**, installed by `rustinel setup`. Hot reload, and a failed reload keeps the previous rules.

The content: rustinel-rules



	ESSENTIAL LOW FP	ADVANCED MEDIUM FP	HUNTING HIGH FP
Windows	19 rules	39	42
Linux	10	26	27
macOS experimental	5	21	.

Signed catalog (minisign), minimum engine version declared per pack, updated with rustinel rules update.

Can this rule actually fire?

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

can-fire

Supported source. Fields measured as reliable only add caveats.

degraded

Evaluates, but a field is conditional, derived or truncated and was never measured (e.g. OriginalFileName).

can-never-fire

Missing collector, field never filled, or missing dependency.

Compatibility analysis. A test is still needed to validate detection.

SigmaHQ has 3,783 rules: **2,709** load on Windows, **178** on Linux, **86** on macOS. Rule count **is not** coverage.

FIELDS MEASURED AS RELIABLE • process_creation

Windows CommandLine · Image · ParentImage · ParentProcessId

Linux CommandLine · Image · ParentImage · ParentCommandLine

macOS CommandLine · Image

Every other field stays a limitation until it is measured.

DEMO WINDOWS PACK • 17 RULES



can-fire (with caveats) degraded 0 can-never-fire

Detection as Code

The rule • simplified excerpt

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

The atomic test • encoded_powershell.ps1

```
# Test for the full rule
# Benign payload: a plain Write-Host.

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

& powershell.exe -EncodedCommand $enc

# Expected alert, otherwise the test fails
```

On every change, CI **validates** the rule, runs the test against a fixed Rustinel version, and runs `sigma doctor` on the packs.

The main commands

rustinel - zsh

```
$ rustinel setup           # configures and installs the service
$ rustinel run             # runs the agent in the foreground

$ rustinel doctor          # checks agent health
$ rustinel sigma doctor    # checks rule compatibility

$ rustinel capture         # records events
$ rustinel replay          # re-evaluates a capture without admin rights

$ rustinel rules update    # updates the rules pack
$ rustinel update          # updates the binary
```

```
$ █
```

One capture, different rule sets



```
PS> rustinel capture --output session.ndjson # Windows (admin)
# Then, without administrator rights
$ rustinel replay session.ndjson
$ rustinel replay session.ndjson --config candidate.toml
```

Events only: no YARA scans or hash computation during replay.

● DEMO

Rustinel for real

1 Live alerts

whoami · encoded PowerShell · mshta

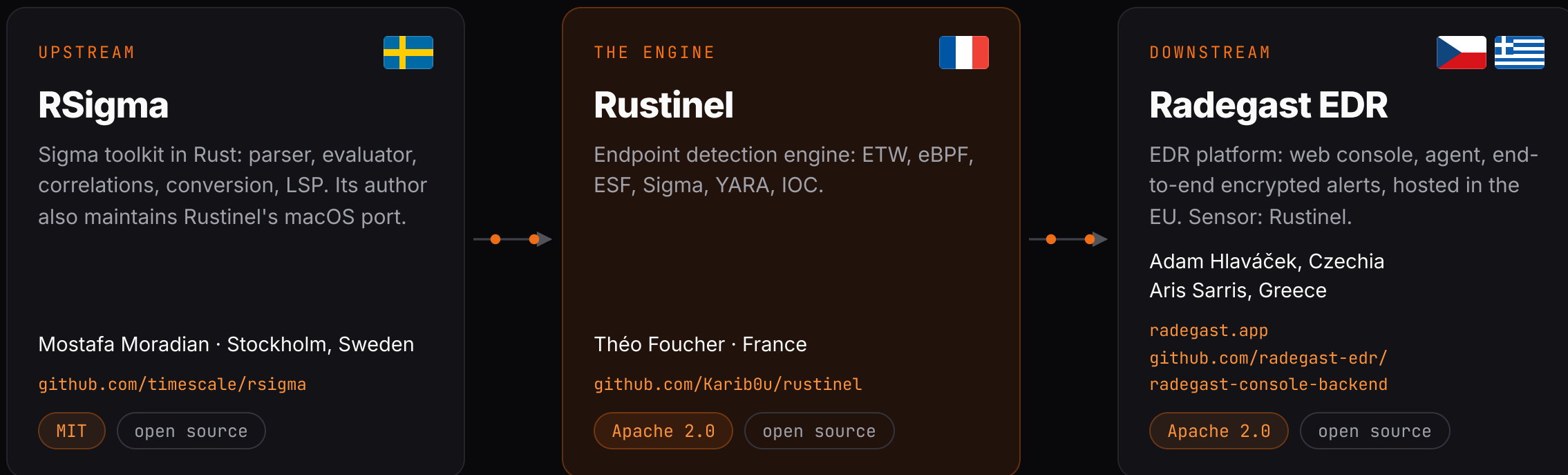
2 Capture and replay

same trace, two rule sets

3 sigma doctor

per-rule verdict on the Windows pack

A European open-source ecosystem



Three open-source projects, four European countries, and authors who also contribute to Rustinel's code.

What's coming

Telemetry

Mostly Windows,
then Linux. macOS
under study.

Reliability

Detect dropped
events and disabled
sensors.

Engine

Parallel detection,
process ancestry,
IOCs with expiry.

Rules

About 150 rules to
write, with evidence-
based tiers.

The backlog is public. A contribution: **one rule, plus one test.**

Detection, testing and known limits

✓ Use cases

- Blue / purple team labs
- Detection engineering
- Testing a SIEM pipeline
- Endpoints with no EDR

! Limits

- Coverage varies by OS
- macOS is experimental
- No prevention, no anti-tamper

→ Contribute

- Field feedback
- Detection tests
- Platform work



— THANK YOU

Questions?

A detection engine you can read, run, test, and try to evade.

github.com/Karib0u/rustinel

github.com/Karib0u/rustinel-rules

docs.rustinel.io • rustinel.io





Appendix

Backup slides for questions: Why Rust, callbacks vs native, Windows and Linux details, artifacts, observing and securing the agent.

Why Rust

1 A single binary

No runtime to install. The same sources for all three OSes.

2 Memory safety

The agent runs as root or SYSTEM and parses attacker-influenced data: ETW payloads, PE files, rules.

3 Predictable cost

No garbage collector. Queue and cache limits help control resource use.

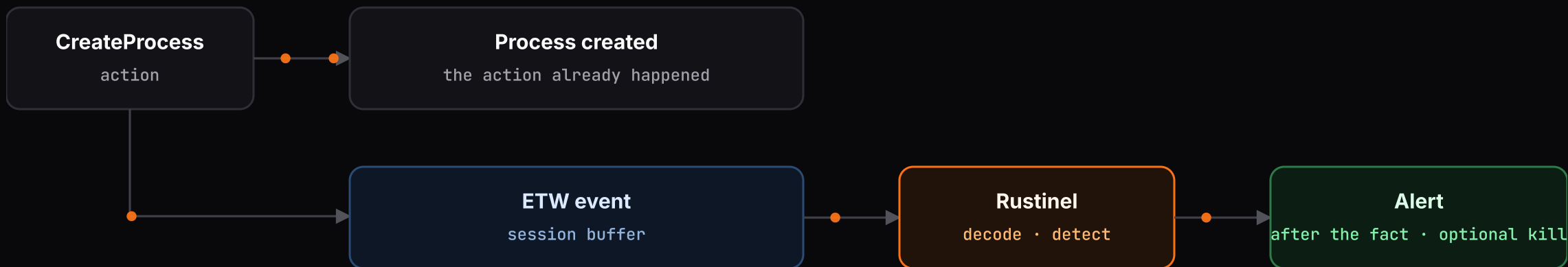
The same language all the way into the kernel: the eBPF programs are written in **Rust with Aya**.

Intercept or observe

EDR WITH A DRIVER · SYNCHRONOUS CALLBACK



RUSTINEL · NATIVE TELEMETRY, ASYNCHRONOUS



A collection choice: ETW, Linux tracepoints and ESF notifications. On macOS, AUTH events could also block actions.

Rebuilding the context

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

FileWrite FileObject=0xA1 (no path) no path

↳ FileObject → name index

FileWrite TargetFilename C:\Users\bob\AppData\x.dll rebuilt

Same mechanism for registry keys. Objects opened before startup are named from an **initial snapshot**.

P Processes

Kernel-Process and the classic kernel logger are joined on PID, parent PID and timestamp.

! Limits

Security auditing and PowerShell logging depend on the machine's policy.

Not collected: cross-process access, remote threads, named pipes, drivers.

One binary, several kernel versions



Available hooks discovered in `tracefs`. A missing hook only degrades its own feature.

C Collection

Processes, files, plain DNS. Network: tuple read from the kernel socket structure.

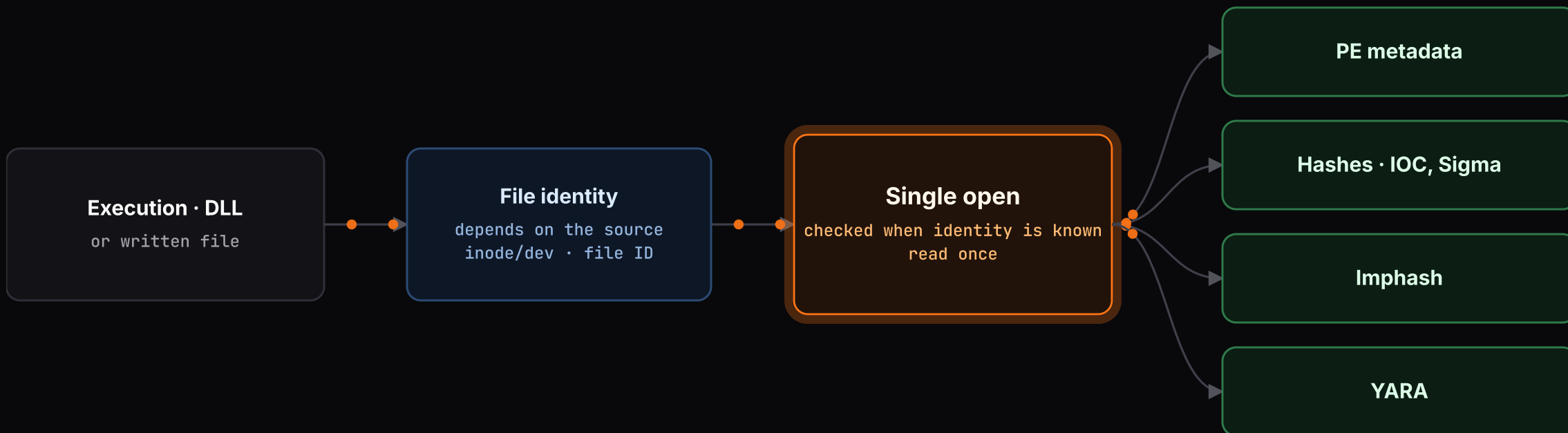
E Enrichment

Process identity via kernel birth time.
/proc for cwd and exe. Inode and device measured in the kernel.

! Limits

Kernel 5.8 minimum. No encrypted DNS (DoH, DoT). No auditd.

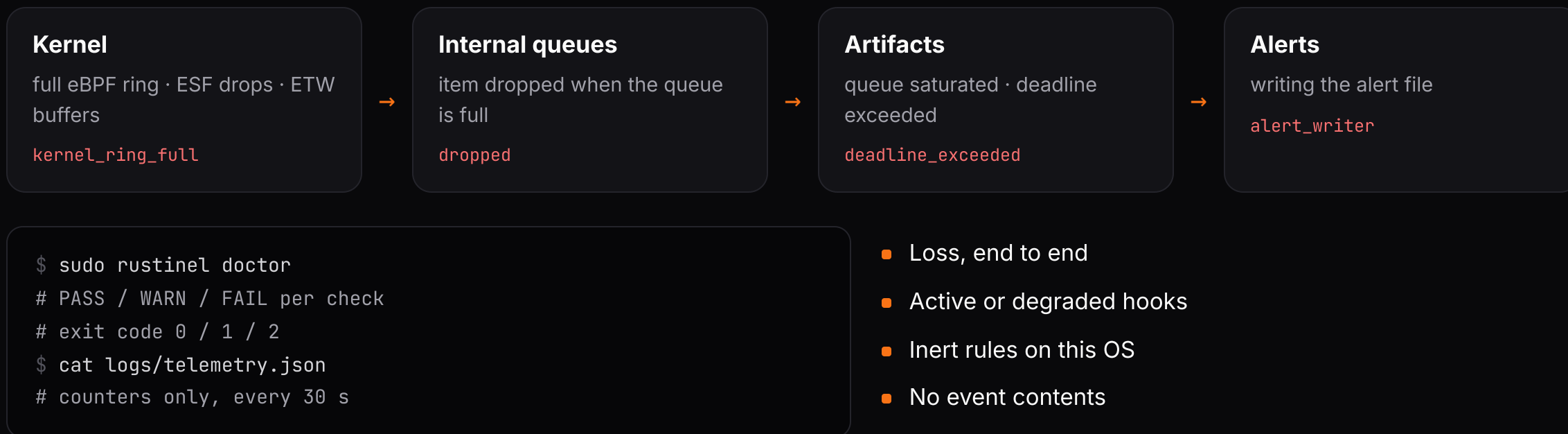
Open once, analyze four times



Written files need an identity to be scanned. Windows measures it **after the event**.

Process memory: a **separate** YARA worker, after a delay, under a shared budget.

Knowing what you cannot see



A privileged agent is a target

1 Threat model

A root or SYSTEM process that reads configs, rules and files from the machine.

2 Input trust

Refuses rules, IOCs and configs that another account could modify.

3 File reads

Identity checked on open. `openat2 + RESOLVE_IN_ROOT` to stay inside the process root.

4 Active response

Windows handle or Linux `pidfd`: PID reuse cannot redirect the kill. Never PID 0 to 4, nor itself.

Accepted: an attacker who is **already an administrator** can stop the agent.