

RUNTIME ENFORCEMENT FOR CI/CD

Build Your Software With Confidence

Scanners list what your build declares. Provenance attests to where it came from. **Endura controls what it does.**

[Your Name]
[Job Title]
[Email Address]



01

The Problem

What runs inside your build, and what it can reach.

CONTENTS

- 01 The Problem
- 02 The Solution
- 03 The Proof
- 04 Appendix

IN THIS SECTION

01

The Build Is The Breach

02

Six Recent Compromises

03

The Runtime Enforcement Gap



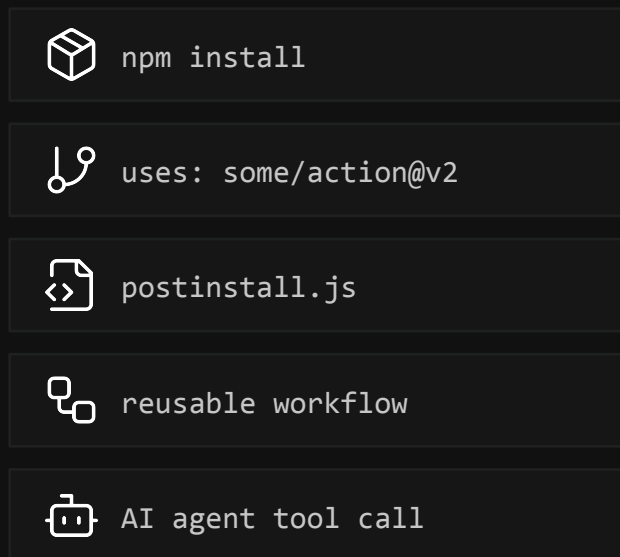
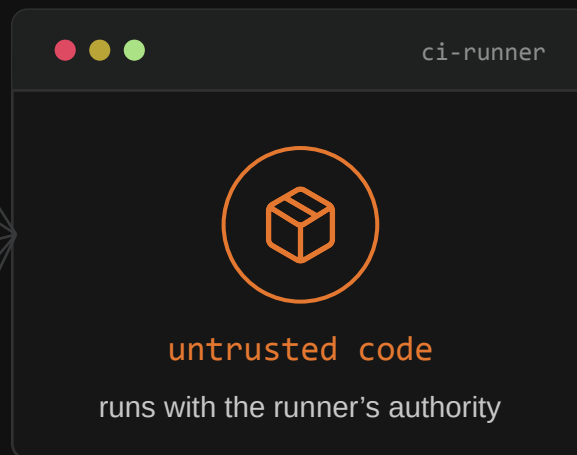
▸ THE PROBLEM

The Build Is The Breach

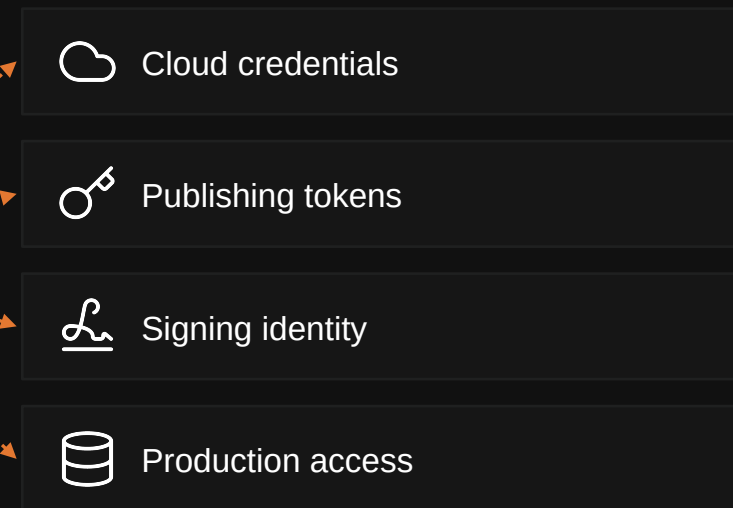
Every build runs code nobody fully reviewed, and it executes the moment it is pulled in.

UNTRUSTED CODE

THE PROBLEM



WITHIN REACH



■ Nothing watches these at runtime.

Your build inherits everything the runner can reach. Before tests, before scans, before review, it already runs with your pipeline's credentials.



Six Build-Time Compromises In Six Months

THE PROBLEM

MAR 19, 2026

Trivy GitHub Actions

76 of 77 trivy-action tags force-pushed to credential-stealing code that read runner memory.

76 of 77 tags force-pushed

APR 30, 2026

PyTorch Lightning

Attributed to the same actor as Trivy, this wave hit PyPI, trojanizing releases of a widely used ML library.

2 registries, one actor

MAY 20, 2026

Mini Shai-Hulud (@antv)

A compromised maintainer published 639 malicious versions whose payload stole GitHub Actions credentials.

639 malicious versions

MAR 31, 2026

Axios On npm

A hijacked maintainer account shipped 1.14.1 and 0.30.4 with a postinstall hook that dropped a cross-platform RAT.

~3 hrs live on npm

MAY 11, 2026

TanStack

84 malicious versions across 42 packages, published by the project's own pipeline with valid SLSA provenance.

84 malicious versions

AUG 4, 2026

ChainDrop

Pushed straight to main, the maintainer's own pipeline built and published signed, poisoned releases of keyv and cacheable.

444 npm packages hit

Sources: Aqua Security, axios, TanStack, Microsoft, Orca, Expel, and CISA advisories, 2026.



Signed. Attested. Compromised.

Existing controls answer identity and origin before code runs. None of them governs what code does once your pipeline executes it.

THE PROBLEM



Scanners list what your build declares.

SCA and SAST inventory dependencies and find known flaws in code before it ships.



Provenance attests to where it came from.

Signing and SLSA prove which workflow produced an artifact and that it was not altered in transit.



Endura controls what it does.

Runtime Sensor sets the boundary of what a build may reach and enforces it in the kernel, before actions complete.

100%

of 2026's three largest npm supply chain attacks carried valid provenance.

Necessary, but not sufficient. SCA, SAST, SBOMs, provenance, and signing answer identity and origin. Endura adds the runtime boundary they were never designed to enforce.



02

The Solution

How Endura turns real build behavior into enforced policy.

CONTENTS

- 01 The Problem
- **02 The Solution**
- 03 The Proof
- 04 Appendix

IN THIS SECTION

01

Untrusted Code, Governed

02

Enforcement In The Kernel

03

Policy And Architecture



Untrusted Code, Governed.

Endura sets the limits of what build code can reach, then enforces them in the kernel across four stages.

THE SOLUTION



01 DISCOVER_

Every Pipeline, Monitored

See every pipeline across providers, teams, and environments, all in one Team Server view.



02 DERIVE_

Every Behavior, Captured

Runtime Sensor observes files, processes, and network calls from deep inside the running kernel.



03 ENFORCE_

Every Action, Governed

Deny-by-default policy, enforced in the kernel with eBPF LSM hooks while your build is running.



04 RESPOND_

Every Violation, Actionable

Violations and their full context flow to Team Server, where teams triage and tune the policy.

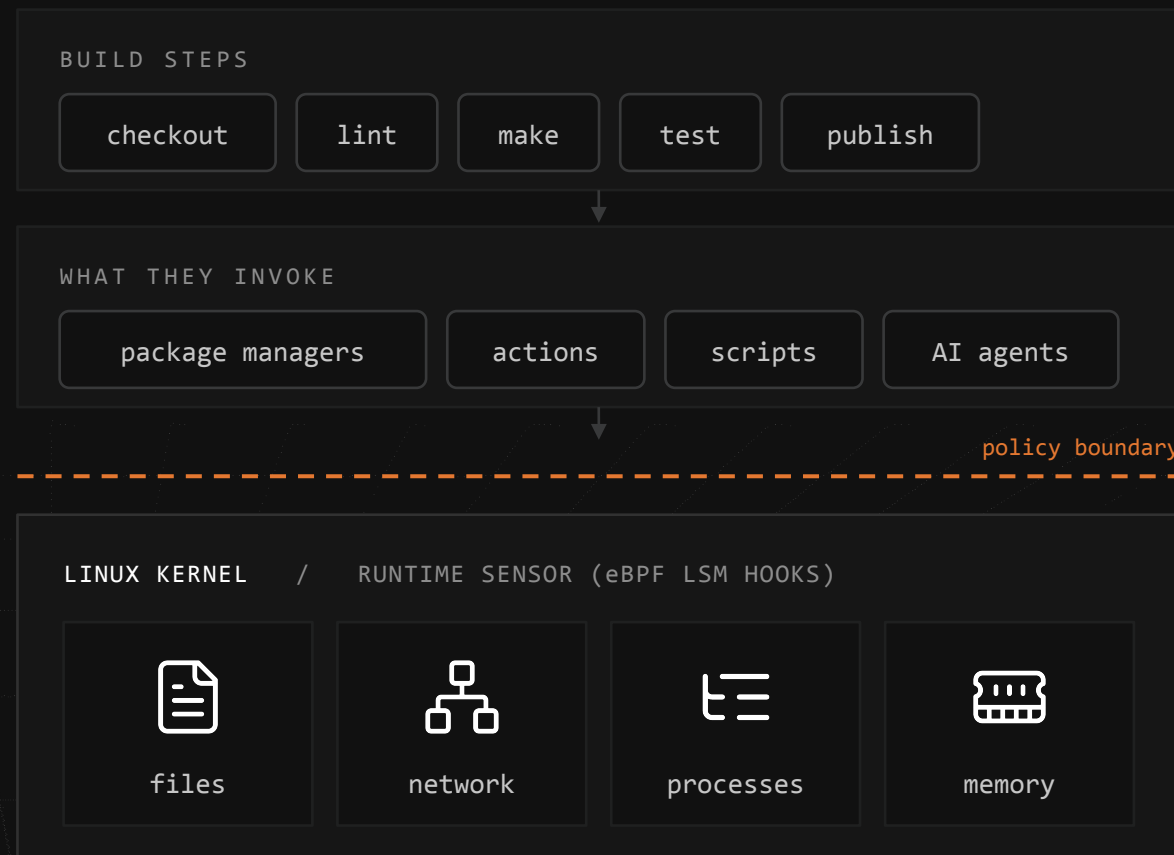
Policy comes from your builds, not a template. Endura derives the baseline from what your pipelines actually do, then enforces it in the kernel.



In The Kernel, Where Code Can't Hide

Scanners list what your build declares.
Provenance attests to where it came from.
Endura controls what it does.

Runtime Sensor runs as eBPF below the build and everything it invokes. File, network, process, and memory operations all pass through the kernel, so that is where policy is enforced.



50+

kernel-level security hooks

40+

threat techniques prevented

<5%

typical build overhead



From Runtime Behavior To Enforced Policy

Derive, refine, observe, enforce is the recommended rollout. You can also assign an enforce policy from the start.

THE SOLUTION



DERIVE_

Record What Happened

Runtime Sensor records the files, processes, network, and credential access a build actually touches.



REFINE_

Set The Boundary

Turn observed behavior into an intentional least-privilege policy. Observed does not mean approved.



OBSERVE_

Validate, No Blocking

Run the policy in observe mode. It flags violations without stopping the build, so nothing breaks.



ENFORCE_

Deny What's Outside

Switch to enforce. Deny-by-default. Anything outside the boundary is denied before it completes.

Observed does not mean approved. Refinement decides what should be allowed. Enforcement denies everything else.



What A Policy Looks Like

Policies are versioned JSON, assigned per pipeline, and reviewed like any other code.

THE SOLUTION

```
policy.json

{
  "mode": "enforce",
  "ip": {
    "connect": [
      "/usr/bin/curl|api.example.com|{80,443}",
      "all|140.82.112.0/20|443"
    ]
  }
}
```

Rules read **executable | destination | port**. Covered operations that are not listed are denied.

Example syntax from the Endura policy walkthrough; see the policy specification for the full schema.

POLICY PRIMITIVES

- **path.open**
Any file open, such as ~/.ssh or ~/.aws
- **path.create / path.write**
New files (create) and changes to existing ones (write)
- **path.execute**
Execution chain: grandparent|parent|target
- **ip.connect**
Outbound connections by program, host or CIDR, and port
- **hook.pttrace**
Debugger attachment to another running process

%workspace%

all

{a,b}

Workspace variable, explicit wildcard, brace expansion



Two Components, One Loop

Wrap your build with job start and job stop. The Sensor runs on the Linux host, below any CI/CD provider.

THE SOLUTION



01 Team Server finds pipelines.

02 Sensor fetches its policy.

03 Sensor reports violations.

04 Alerts go to your channels.



Violations, In Context

Violations arrive with the build, the rule, up to four generations of process ancestry, and the kernel's decision.

THE SOLUTION

Team Server					build #4821 · enforce	
Pipelines		Builds		Violations	Policies	
PIPELINE	RULE	RESOURCE		TIME	VIOLATION DETAIL	
payments-api	path.open	~/.npmrc		09:14:02	BUILD #4821 · payments-api	
payments-api	ip.connect	45.9.148.0/24:443		09:14:02	POLICY security-baseline v7	
web-frontend	path.execute	/bin/sh		09:12:47	RULE path.open ~/.npmrc	
data-pipeline	path.write	/usr/local/bin		08:57:10	PROCESS node > npm > postinstall.js	
mobile-ci	hook.ptrace	node (pid 1042)		08:41:33	DENIED terminated before completion	

Illustrative Team Server view.



03

The Proof

Coverage, evidence, compatibility, and what it costs.

CONTENTS

- 01 The Problem
- 02 The Solution
- 03 The Proof
- 04 Appendix

IN THIS SECTION

01

Pipeline Threat Coverage

02

Three Case Studies

03

Compatibility And Pricing



Pipeline Threats, Stopped At Execution



File Access

Block unauthorized reads, writes, and execution of files.



Memory Attacks

Block memory patching and fileless execution.



Container Breakouts

Stop unauthorized container execution and socket access.



Process Injection

Block preloading, debugger attachment, runtime tampering.



Credential Theft

Block reads of secrets, tokens, and SSH keys.



Network Exploitation

Block C2 channels and data exfiltration.



Privilege Escalation

Stop escalation through kernel capabilities.



Kernel Exploitation

Block malicious kernel modules and eBPF programs.



Reverse Shells

Detect socket-backed shells and packet sniffing.



Malicious Persistence

Stop backdoors in startup files and scheduled tasks.

Behavior-based, not signature-based. Endura denies actions outside your policy, including techniques that have no name yet.



Case Study: XZ Utils

Three steps of a real build-time backdoor, and the policy primitive that governs each.

THE ATTACK

Backdoor Triggered At ./configure

A patched `build-to-host.m4` spawned a `sed|tr|xz -d|bash` pipeline, decoding hidden shell stages from test fixtures.

1

Endura Restricts Process Execution

`path.execute` limits which shells and decoders a build step may spawn, so the decoded payload never runs.

THE ATTACK

Test Fixtures Read At Build Time

The stages read `bad-3-corrupt_lzma2.xz`, a file meant only for the post-build test harness, during `./configure`.

2

Endura Restricts File Reads

`path.open` limits which files a build step may read, so configure-time access to test fixtures is refused in the kernel.

THE ATTACK

Makefile Patched To Link Backdoor

A prebuilt `liblzma_la-crc64-fast.o` appeared with no matching `gcc` or `as`, then linked into `liblzma.so.5`.

3

Endura Restricts File Writes

`path.create` and `path.write` limit which objects a step may add or change, so injected code never reaches the final binary.

Scope: Endura limits what build code can do when it runs. It does not stop an upstream maintainer account compromise. That is a different problem.



Case Study: Shai-Hulud

A self-replicating npm worm that harvests credentials from build runners and republishes itself.

THE ATTACK

Install Hook Runs First

The worm ships inside a trojanized package. A `preinstall` script executes during `npm install`, before tests, scans, or review.

1

Endura Restricts Process Execution

`path.execute` limits what an install hook may spawn, so the bundled scanner never reaches its first stage.

THE ATTACK

Credentials Swept From The Runner

A bundled secret scanner sweeps the filesystem and environment for npm, GitHub, AWS, and Vault credentials.

2

Endura Restricts File Reads

`path.open` limits which files a step may read, so `~/.npmrc`, `~/.aws`, and `~/.ssh` stay out of reach.

THE ATTACK

Secrets Leave, Worm Republishes

Stolen tokens are pushed to an attacker-controlled public repository, then used to publish poisoned versions of further packages.

3

Endura Restricts Network Egress

`ip.connect` limits outbound connections by executable, destination, and port, cutting off exfiltration and republishing.

Scope: Endura governs what the payload may do on your runner. It does not stop the upstream account compromise that published the package.



Case Study: A Re-Enabled Action

September 2026: actions pulled in May for credential theft came back online with the payload still in place.

THE ATTACK

The Step Runs The Payload Again

Disabled May 19, restored September 16 with the malicious code still in place, then disabled again September 25. Any workflow pinned to a `version` tag ran it in between.

1

Endura Restricts Process Execution

`path.execute` lists the chains a step may run, as grandparent, parent, and target. The restored payload matches no rule, so it never starts.

THE ATTACK

Credentials Read From The Runner

The payload reads CI/CD tokens and cloud credentials from the runner filesystem and environment.

2

Endura Restricts File Reads

`path.open` limits which files a step may read, so CI tokens and cloud credentials on the runner are out of reach.

THE ATTACK

Secrets Sent To The Attacker

Harvested credentials are pushed to an attacker-controlled endpoint, before anyone notices the action is compromised again.

3

Endura Restricts Network Egress

`ip.connect` limits outbound connections by executable, destination, and port. The attacker host matches no rule, so the socket never opens.

Nothing here depends on knowing the action was compromised: there was no new version, no CVE, and no changed digest. The boundary comes from your build's own behavior.

Technical Compatibility

Runs where your builds already run, on x86_64 and ARM64 *, as SaaS or fully self-hosted.

OPERATING SYSTEMS

- AlmaLinux (9, 10)
- Alpine Linux (3)
- Amazon Linux (2023)
- Arch Linux
- CentOS Stream (10)
- Debian (12, 13)
- Fedora (44)
- Gentoo Linux
- Oracle Linux (8 UEK, 9, 10)
- RHEL (9, 10)
- Rocky Linux (9, 10)
- SUSE Enterprise (15, 16)
- Ubuntu (24, 25, 26)

CI/CD PROVIDERS

- Argo Workflows †
- AWS CodeBuild
- Azure Pipelines
- Bamboo
- CircleCI
- CloudBees CI
- GitHub Actions
- GitLab CI
- Jenkins
- Koji
- TeamCity
- Travis CI †

AUTH & NOTIFICATIONS

- CyberArk (OIDC)
- Google Identity (OIDC)
- Microsoft Identity (OIDC)
- Oracle Identity Cloud (OIDC)
- Slack
- Teams
- Email

DEPLOYMENT

- SaaS or 100% self-hosted
- Sensor: RPM, DEB, TGZ, Helm
- Team Server: Docker, Podman, Helm
- OIDC auth · RBAC controls

* 2026 roadmap

† Team Server discovery only



See It On A Real Build

A pilot runs on one pipeline and shows enforcement working end to end.

01 Deploy Into Your Pipeline

Add Runtime Sensor to an existing workflow in a few lines. The same two commands work on any CI/CD provider.

02 Derive, Then Observe

Auto-generate a least-privilege baseline in derive mode, then validate it in observe mode without blocking anything.

03 Enforce And Prevent

Switch to enforce mode, introduce out-of-policy behavior, and watch the kernel deny it before it completes.



/opt/endura/sensor

```
$ systemctl enable endura-sensor
```

```
✓ Starting Runtime Sensor
```

```
Ready to accept connections
```

```
$ endura job start --mode enforce \  
--workspace ./ci_job
```

```
> Fetching policy from Team Server
```

```
> Enforcing policy security-baseline
```

```
> DENY path.open ~/.npmrc
```

```
> DENY ip.connect 45.9.148.0/24:443
```



▸ PER PIPELINE _

Priced By What You Protect

One rate per protected pipeline. No platform fee, no onboarding fee, unlimited discovery on every plan.

Pilot

Validate on a focused set of critical pipelines.

\$500 /Pipeline/60 Days

- Up to 25 pipelines
- Full runtime enforcement
- Automated policy generation
- Unlimited discovery

START PILOT

Startup

Protect production pipelines across your CI/CD.

\$500 /Pipeline/Year

- Up to 300 pipelines
- Everything in Pilot
- Notification integrations
- Multi-provider coverage

GET STARTED

Business

Standardize runtime policy across a growing estate.

Volume Pricing

- Up to 2,500 pipelines
- Everything in Startup
- Graduated volume pricing
- Analytics and reporting

GET A QUOTE

Enterprise

Standardize across environments and models.

Custom Pricing

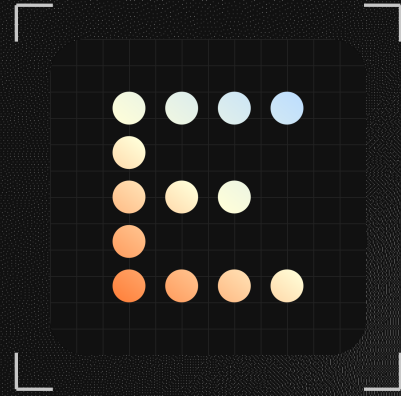
- Enterprise-scale coverage
- Everything in Business
- Managed or self-hosted
- Multi-BU deployments

CONTACT SALES

What counts as a pipeline? Your provider's own project unit: one GitHub repository, one GitLab project, one Jenkins job. One policy, one price.

endurasecurity.com/pricing





► READY WHEN YOU ARE _

Ready To See What Your Builds Are Doing?

Deploy the Sensor on one pipeline and see every file, connection,
and process your build touches.

REQUEST A DEMO

endurasecurity.com

04

Appendix

Engineering detail: failure modes, deployment, data flow, tooling, and where Endura fits.

CONTENTS

- 01 The Problem
- 02 The Solution
- 03 The Proof
- **04 Appendix**

IN THIS SECTION

01

Failure Modes And Attribution

02

Deployment And Data Flow

03

Tooling And Stack Fit



What Happens When Something Fails

Two questions decide whether a control belongs in the critical path: what happens when it breaks, and what happens when it is attacked.

AN OPERATOR STOPS THE SENSOR

BUILDS UNAFFECTED

By design, running builds are not killed. Enforcement ends for the remainder of those builds, and endura job stop then exits non-zero, so the pipeline surfaces it rather than finishing quietly.

TEAM SERVER UNREACHABLE

ENFORCEMENT CONTINUES

Enforcement continues on the local policy. Submissions resume after connectivity returns, though telemetry from the outage may be lost.

A BUILD TRIES TO STOP THE SENSOR

ROOT ONLY

Starting and stopping the Sensor are root operations while build steps run unprivileged, so nothing inside a running build can stop it.

CONFIG OR TLS TAMPERING

FAILS CLOSED

The environment file is root-owned. With certificate pinning enabled, a missing or empty CA file fails closed rather than falling back.

Endura is not a kill switch for your pipeline. Operators control the Sensor. In enforce mode, the offending process is terminated by default, not just logged.



One Sensor, Many Concurrent Jobs

A single Sensor instance enforces a separate policy for every job it protects, up to 24 at once.

```
$ endura job list
```

ID	UID	MODE	WORKSPACE
3f9a1c7e5b2d8046	1001	enforce	/build/api-service
a17c04e9b8f2d531	1001	enforce	/build/web-frontend
c82b6f0d41a9e7b3	1002	observe	/build/data-pipeline

one Sensor, three policies enforced at once

HOW A JOB IS ATTRIBUTED

- One Sensor per host. There is no per-build Sensor instance.
- Each job is anchored by its workspace path and its mount namespace (mntns) at start.
- Jobs are selected by 16-digit hex ID or workspace path. The ID resolves across mount namespaces.
- A non-root user may start a job only for a workspace they own. Root bypasses ownership, not existence.
- Policy precedence is the local policy file, then the built-in derive policy. Team Server can replace either once provider and pipeline match.

Stale jobs do not linger. Jobs can be purged on a configurable absolute timeout, and are retired fifteen minutes after a CI/CD cancellation by default.



Deployment, Requirements, And Credentials

The Sensor initiates every connection. No inbound access to build hosts is required.

INSTALL METHODS

- Native package: RPM, DEB, TGZ
- Runs as a systemd service
- Docker or Podman
- Kubernetes DaemonSet
- One Sensor per node

HOST REQUIREMENTS

- Supported kernel with eBPF
- BTF at `/sys/kernel/btf/vmlinux`
- Root access to start and stop
- 1 GB available memory or more
- Outbound HTTPS to Team Server

CONFIGURATION

- Environment file, root-owned
- Optional TLS certificate pinning

Outbound only. Team Server never opens a connection into a build host, so the Sensor adds no new inbound attack surface to your build fleet.



What The Sensor Sends To Team Server

Four categories of information leave the build host, all over the outbound Team Server channel.

01



Host Details

Attributes of the machine the Sensor runs on, such as operating system name, version, and machine ID.

02



Build And Pipeline

The build currently being protected and the pipeline it belongs to, so activity correlates with your CI system.

03



Violations

Operations denied or flagged during the build, with the context needed to triage them.

04



Sensor Logs

The Sensor's own logs, so Team Server can act as a centralized log viewer across the fleet.

Standalone mode sends no Endura telemetry off-host. Set only the license key and run without a Team Server URL: policies and results stay on the host. With Team Server, traffic is outbound HTTPS with certificate verification on by default.



Inspect And Debug On The Host

Sensor state, loaded hooks, running jobs, and the policy in force are all readable on the build host itself.

endura · build host

```
$ endura sensor status
active (online)
$ sudo endura sensor hooks # per-program state
$ sudo endura sensor logs -f

$ endura job list
$ endura job get /workspace # JSON
$ endura policy describe 3f9a1c7e5b2d8046
$ endura policy validate .endura/policy.json
```

WHO CAN RUN WHAT	
sensor status, wait	Any local user
sensor start, stop, logs, hooks	Root
job start	Workspace owner or root
job get, list, status, stop, policy describe	Job owner or root

endura sensor hooks lists every loaded eBPF program and whether it attached. An unattached row means that event class is not enforced on the host.



Where Endura Fits Your Stack

Each control below answers a real question. None was built to govern what the build does while it runs.

CATEGORY	WHEN IT ACTS	WHAT IT GOVERNS
SCA	Before the build	Known vulns in declared dependencies
SAST	Before the build	Flaws in first-party source
SBOM and provenance	Build and publish	What is in the artifact, and its origin
Package screening	Before install	Whether a package looks suspicious
Pipeline posture, ASPM	Before the build	Where the pipeline is misconfigured
Runner hardening	During the build	Network egress, with tamper detection
Production runtime	In production	What workloads do after release
Endura	During the build, in the kernel	What the build is allowed to do: file, network, process, memory, and kernel operations its hooks cover

Necessary, but not sufficient. Endura complements these controls, it does not replace them. Its nearest neighbor, runner hardening, also acts during the build but is network-centric; Endura enforces deny-by-default across more operation classes. Full comparison in the Control Stack whitepaper.

