

Clean Libraries Whitepaper

Securing the Dependency Decision

Why AI-assisted development has moved the control point upstream, and what an effective control at that point has to do

ABOUT THIS PAPER

Coding agents now select, resolve and install open source dependencies faster than any human review can evaluate them.

The result is a structural gap: the moment a component is chosen and the moment it is examined have drifted weeks apart, and every control an enterprise owns sits on the wrong side of it. This paper examines how the gap forms, why late-stage scanning cannot close it, and what a control now of resolution must do - covering origin, behaviour, equivalence, evidence, and operation without connectivity.

ORIGIN

BEHAVIOUR

EQUIVALENCE

EVIDENCE

OFFLINE OPERATION

TABLE OF CONTENT

-	Executive summary	03
01	The control point has moved	03
02	Why scanning later can't close the gap	05
03	What the control must do	06
04	How Clean Libraries works	07
05	What verification means	08
06	Verdict semantics	09
07	Deployment models	10
08	Evidence and compliance	10
09	Adoption path	12
10	Conclusion	13
-	About CleanStart	13

OVERVIEW

Executive Summary

Where a developer once chose a library deliberately and a reviewer examined that choice, an AI coding agent now resolves an entire dependency tree in a single turn – fast, largely invisible, and unreviewed. Volume has risen by an order of magnitude; the human control that governed it has not.

Most enterprise tooling examines components at build, at the registry, and at deployment – all after the decision has been made and merged. Everything found there is rework, arriving at the most expensive possible moment. Closing the gap requires a control at resolution time: one that returns a decision before installation, applies identically in the IDE, CLI, agent and pipeline, and produces evidence as a by-product. To be credible it must answer three questions about every component – where it came from, what it does, and whether it is still the software it claims to be.

Clean Libraries implements this control. It maintains a catalog of verified components across eight language ecosystems, verifies each against origin, behaviour and equivalence, and returns a signed verdict at the moment a dependency is requested. It runs in managed, self-hosted, air-gapped and sovereign configurations. Every dependency it returns is attested and signed, so – alongside an attested CleanStart image – both can be joined into a single attestation chain of your own, across the container image and the language dependencies installed into it.

SECTION 01

The control point has moved

For two decades, open source governance rested on a human decision: a developer chose which library to pull in, and a reviewer looked at the diff. The slowness did useful work – it created a checkpoint at which someone considered whether a dependency belonged in the codebase.

AI-assisted development removed that checkpoint without replacing it. An agent asked to implement a feature writes the import, resolves the tree, and installs – often across dozens of packages, faster than any reviewer reads the diff. The decision still happens. Nobody makes it.

19.7%

of packages did not exist

43%

recurred across all ten queries

Research presented at USENIX Security 2025 examined 576,000 code samples across sixteen language models. More significant than the hallucination rate is its determinism: a name a model produces reliably is a name an attacker can register in advance, and the resulting package will be installed by every agent that reaches for it. One such name was found referenced in 237 GitHub repositories, propagated by machine-generated agent skill files instructing agents to install it.

And the decision is larger than it looks. What enters a build is every dependency of every dependency, resolved automatically. A handful of direct dependencies routinely expands into hundreds of components, contributed by maintainers nobody evaluated, carrying install scripts nobody read.

1.1 What you don't have

For nearly all of those components you hold a name, a version and a hash. You do not hold:

Missing	What it means
Provenance	No proof of which source commit produced the artifact, in what environment, or whether it was built from published source at all. Registry malware often exists only as a published artifact with no corresponding source code in any repository.
A behaviour record	Nothing indicates whether this version opened an unauthorized network connection, executed a new install script, or was published by a compromised account after long dormancy – the exact pattern behind several of 2026's most consequential supply-chain compromises .
A decision trail	When an auditor asks who approved a component, there is no answer because no human made the decision . An automated solver resolved and pulled it in. This distinction is critical under regulatory frameworks demanding proof of component selection control.

You cannot govern what you have never seen.

The transitive closure is the part of the supply chain nobody chose, nobody reviewed, and nobody can currently account for.

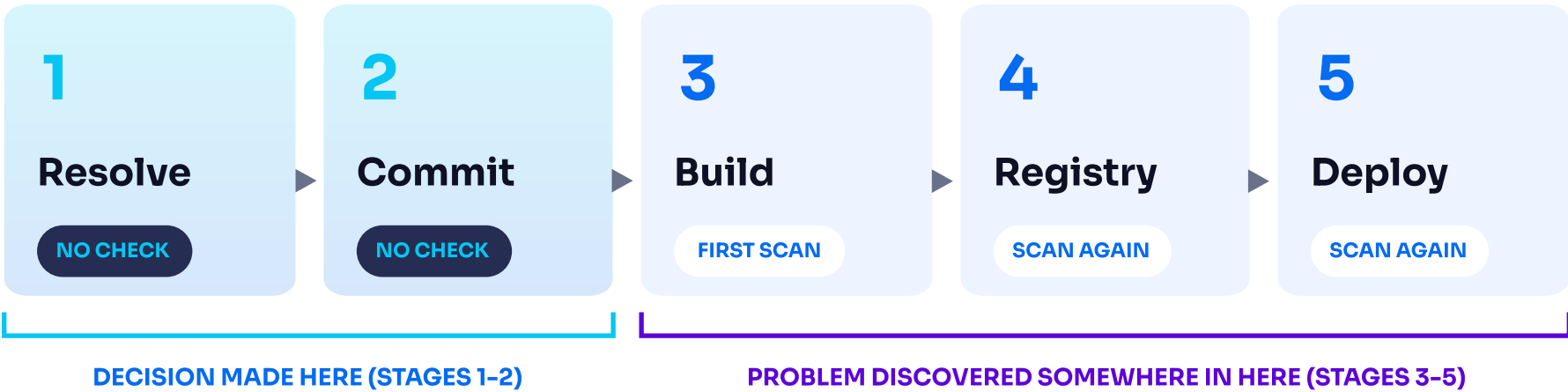
SECTION 02

Why scanning later can't close the gap

Most enterprises answer this by adding controls later: dependency scanning at build, image scanning at the registry, admission control at deploy

Each is a reasonable control, and each sits after the decision it is meant to govern

2.1 The distance problem



A finding at the build gate is never simple – it is a request to undo work. The branch has merged, other work depends on it, and the fix is rarely a version bump but a dependency conflict, a breaking change, and a regression cycle. At the registry gate it is worse: a release is waiting, and the choice is to ship with an exception or slip the date.

2.2 The four costs

Cost	How it lands
Engineering time	Developers’ context-switch back into work completed weeks earlier, at the least efficient possible moment.
Release predictability	Critical findings arrive at gates that sit immediately before a delivery commitment, converting a technical problem into a scheduling one.
Audit exposure	A scan report demonstrates that problems were found. It does not demonstrate that component selection was controlled, which is the question regulators increasingly ask.
Alert fatigue	When a build routinely returns hundreds of findings, teams stop reading them. The control persists on paper and has quietly ceased to function.

None of this argues for scanning less. Downstream scanning remains necessary as defence in depth and as evidence. It argues that scanning cannot be the primary control, because it is positioned to report on decisions, not to govern them.

SECTION 03

What the control must do

A control at the decision point has a demanding specification: fast enough not to interrupt work, correct enough to be trusted, and evidence-generating without a separate effort.

Requirement	Why it is necessary
Decision before installation	A control that reports after the fact is a scanner. To govern selection, it must return a verdict while refusal is still cheap.
Identical policy at every surface	The same request must get the same answer in the IDE, the CLI, the agent and the pipeline. Otherwise developers learn the policy at merge time, when the fix is most expensive.
Machine-consumable output	Agents must be able to call the control directly and act on the result. An open protocol rather than a bespoke SDK keeps this true for agents not yet adopted.
Verified origin	A version and a hash do not tell you where an artifact came from. The control must verify the artifact against its claimed publisher and upstream registry and reject packages whose origin cannot be established.
Verified behaviour	Rebuilding does not help when the source repository itself was compromised. Behavioural analysis covers that residual.
Demonstrated equivalence	A rebuild is only useful if it is still the same software. This must be shown, not asserted.
Evidence as a by-product	Compliance artifacts should accumulate from normal operation rather than being assembled retrospectively.
Graceful degradation	A blocked package must not leave a developer with a broken build and no path forward.
Operation without connectivity	Many regulated environments cannot transmit dependency metadata outside their boundary at all.

SECTION 04

How Clean Libraries works

Clean Libraries has four parts: a verification pipeline that produces trusted artifacts, a catalog that stores them with their evidence, a verdict service that answers requests, and the enforcement surfaces developers and agents work through.

4.1 The verification pipeline

Every request returns a signed verdict. Evidence about the package (provenance, behavioural analysis, and equivalence checks against the source repository) is judged against a versioned policy, and the result is signed with an ECDSA P-256 key held in a managed key service. The verdict and its audit record are produced before the response is returned.

Coverage spans eight ecosystems, with the most-consumed packages pre-built so the common request is answered instantly.

Components outside the pre-built set are not passed through silently. They are queued for build on demand and returned as `INSUFFICIENT_DATA` until verification completes – an explicit state, because treating unverified components as safe is the failure this control exists to prevent.

4.2 The catalog

The catalog stores verified artifacts with their evidence, indexed by exact coordinate, and is consumed through the repository infrastructure you already run – so developers pull dependencies exactly as before.

4.3 The verdict service

The verdict service is the control point. It takes a component coordinate, evaluates it against verification results and your policy, and returns one of four signed verdicts – machine-readable, with the reason and policy version attached.

4.4 Enforcement surfaces

A control only works where people and agents operate. Clean Libraries is exposed through a VS Code extension that evaluates a dependency as it is written, a CLI and JavaScript SDK that give identical results locally and in scripts, and CI/CD enforcement that gates on policy with the reason attached to any failure.

For coding agents, Clean Libraries is published as a standard Model Context Protocol (MCP) server. Rather than a bespoke plugin per agent, the control speaks the protocol agents already use, so any MCP-compatible agent connects by configuration alone. Claude Code, Cursor and Gemini CLI are tested and documented; native plugins for GitHub Copilot and AWS Kiro are in development. Adopting a new agent does not mean waiting for vendor support before it falls under policy.

The agent calls the server before installing. A hallucinated or squatted name is refused at that point, and every agent-initiated request is recorded against an agent identity – the audit trail of automated dependency decisions that current guidance points toward.

SECTION 05

What verification means

“Verified” is used loosely in this market. Clean Libraries treats it as three separate questions, each needing different evidence.

Origin

Where did it come from?

Behaviour

What does it do?

Equivalence

Is it still the same software?

5.1 Origin – where did it come from?

Clean Libraries verifies the origin of every package : the signature, the content hash, and build provenance. Packages whose origin cannot be established are not passed through. This closes the largest class of registry attack: packages that exist only as an artifact, with no reviewable source behind them.

5.2 Behavior – what does it do?

Clean Libraries analyses behaviour independently. Components run in a sandbox at install and import, and their capabilities are diffed against the previous version. Newly introduced network, filesystem or process-execution behaviour is surfaced, not assumed benign. Maintainer and publishing signals are evaluated too, which is what catches the dormancy-gap takeover: an account silent for months that suddenly publishes.

5.3 Equivalence – is it still the same software?

This is the question most rarely answered, and the most persuasive to engineers. A rebuild is only useful if it behaves like what it replaces. Clean Libraries rebuilds independently and diffs the result for reproducibility, then runs the upstream project’s own test suite against the rebuild. Where it deviates, the deviation is reported, not concealed.

Together

these support a claim neither rebuilding nor scanning makes alone: this component came from a known source, does what it is expected to do, and is still the software you intended to use.

SECTION 06

Verdict semantics

Every request returns exactly one verdict. The set is deliberately small, because ambiguity at a control point becomes discretion, and discretion at machine speed is no control at all.

Verdict	Action	Meaning
CLEAN	Admit	Origin verified, behaviour verified, equivalence to upstream demonstrated. Trusted coordinates are returned and installation proceeds.
FLAGGED	Review	An anomaly was surfaced – a newly introduced capability, a maintainer signal, or a provenance weakness. Organisational policy determines whether it proceeds.
BLOCKED	Refuse	The component fails policy. Installation is refused with the reason attached, and a governed alternative is offered where one exists.
INSUFFICIENT_DATA	Queue	Verification has not yet completed. The component is queued for build on demand rather than passed through as though it were safe.

The decision itself is auditable

Verdicts are cryptographically signed, so the decision itself is auditable – not just the artifact. You can show not only what you admitted, but under which policy, when, and on what basis.

SECTION 07

Deployment models

Verification quality does not depend on connectivity: an air-gapped deployment is a first-class configuration, not a degraded one.

Model	Description	Typical fit
Managed	CleanStart operates the verification pipeline and catalog; the organisation consumes verdicts and artifacts.	Fastest path to value; commercial and cloud-native estates.
Self-hosted	The full stack runs inside the organisation's own infrastructure and network boundary.	Enterprises with existing platform teams and internal hosting requirements.
Air-gapped	Complete function with no outbound connectivity. Catalog and verdict data are delivered through a controlled sync process.	Defence, critical national infrastructure, and classified environments.
Sovereign	Data residency held within a specified jurisdiction, with no export of dependency metadata.	Regulated industries and public-sector obligations in APAC, the EU and elsewhere.

SECTION 08

Evidence and compliance

Compliance work fails less often for lack of controls than for inability to show they operated. Clean Libraries is built so that evidence accumulates from normal use.

8.1 What is retained



Provenance

Provenance tying each artifact to a specific source commit and build environment.



Signed verdict

The signed verdict, with its reason code, timestamp, governing policy version, and audit record ID.



Policy version

The policy version in force at the time, so a verdict issued today stays checkable against the rules that produced it.



Decision trail

A decision trail recording who or what requested each component - including agent-initiated requests, linked to an agent identity.

8.2 Current obligations

The EU Cyber Resilience Act's reporting duties begin on 11 September 2026 - a 24-hour early warning and 72-hour notification for actively exploited vulnerabilities - with the design and conformity regime following on 11 December 2027. The earlier date is the operational one: an organisation already holding component-level provenance and SBOMs draws on existing records rather than opening an investigation.

A joint CISA/NSA advisory of 1 May 2026 on agentic AI points organisations toward approved component sources and human approval for high-impact automated actions. An approved source agents must consult, with an auditable link between each agent decision and an agent identity, speaks directly to that.

For FedRAMP, CMMC and similar frameworks, the hard part is cryptographic module validation across the boundary between a container image and the language dependencies installed into it. Every Clean Libraries dependency comes attested and signed; because the image is attested too, both sides of that boundary arrive with proof of where they came from - so you can join them into a single attestation chain of your own, instead of reconciling two components that carry no proof at all.

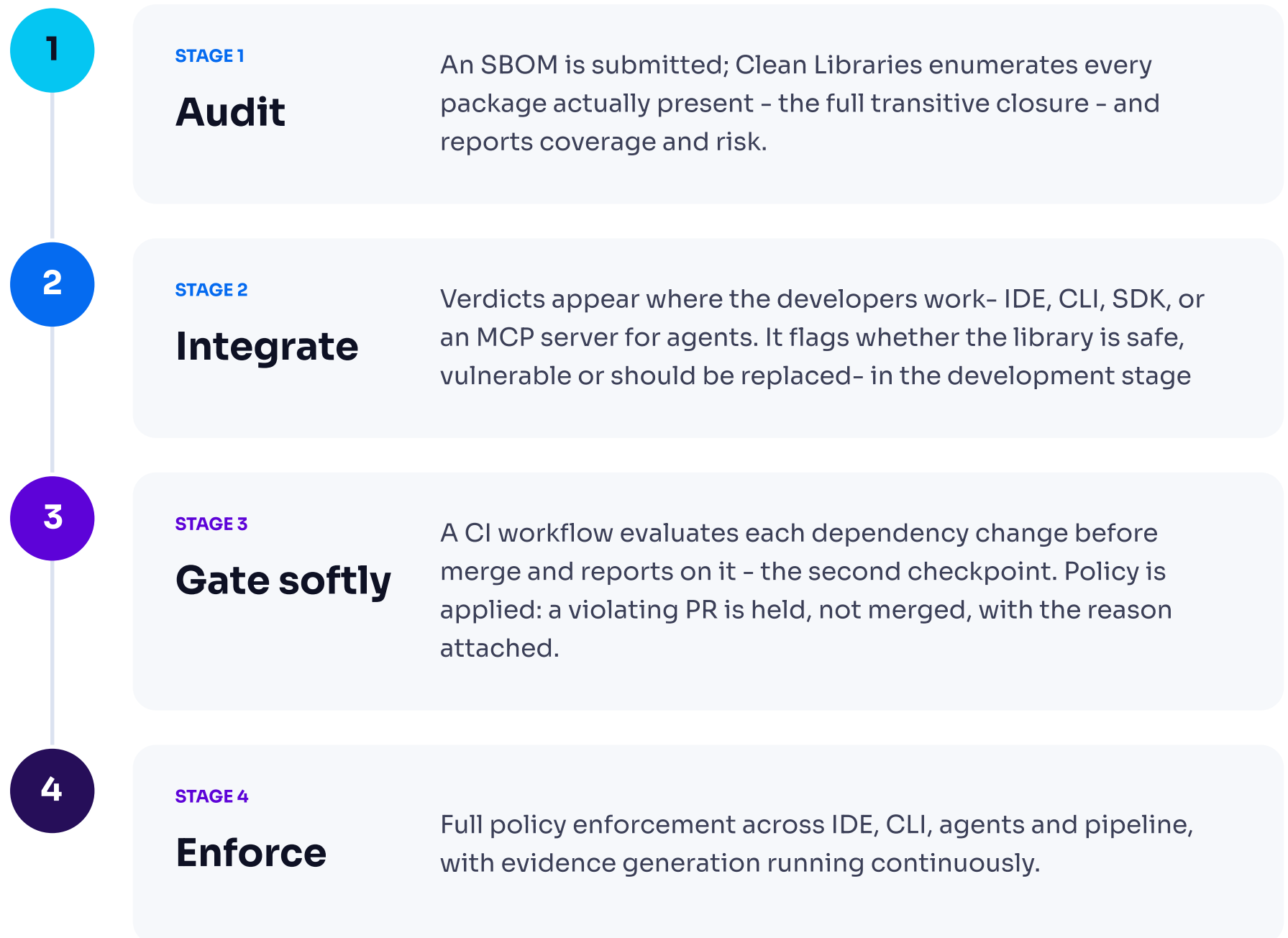
8.3 Portability

All evidence is emitted in standard formats - no proprietary artifact types, no dependency on CleanStart tooling to read it. A governance control that creates lock-in is a governance risk of its own. If you leave, the evidence leaves with you.

SECTION 09

Adoption path

Clean Libraries rolls out in phases, so no team meets an unexpected failing build.



Why start with the audit

The first phase stands on its own: most organisations have never measured their true transitive closure, and that number - usually an order of magnitude past the list a team recites from memory - is what makes the rest concrete.

SECTION 10

Conclusion

The control point for open source consumption has moved – to the moment an agent resolves a dependency, at a speed and volume human review was never built to absorb.

Scanning later is not wrong; it is positioned to describe outcomes, not govern choices, and every finding lands at the most expensive moment. The alternative is a decision returned at resolution – grounded in verified origin, behaviour and demonstrated equivalence, applied identically wherever a dependency enters, producing evidence as a matter of course.

About Cleanstart

CleanStart builds verified software supply chain infrastructure. CleanStart Images provides hardened, FIPS-capable container images, each carrying a complete attestation chain from signed source package to built image. Clean Libraries provides verified language dependencies with signed verdicts across eight ecosystems, every one attested and signed so you can anchor it into an attestation chain of your own. Both are deployable in managed, self-hosted, air-gapped and sovereign configurations.



Start Clean. Stay Secure

Verified container images and language dependencies,
with evidence you can carry anywhere.



www.cleanstart.com