



New Threats – New Defences



FREDRIK SKOGMAN @KOMMENDORKAPTEN
Staff Engineer
GitHub



Fredrik Skogman

Supply Chain Security engineering

- Build integrity and authenticity
- Tooling & standards
- Secure software delivery.

Active maintainer in various OSS projects such as Sigstore and TUF.



Software supply chain security in the AI era

A changing discipline

Like every software engineering discipline, supply chain security has changed in recent years.

Speed and scale

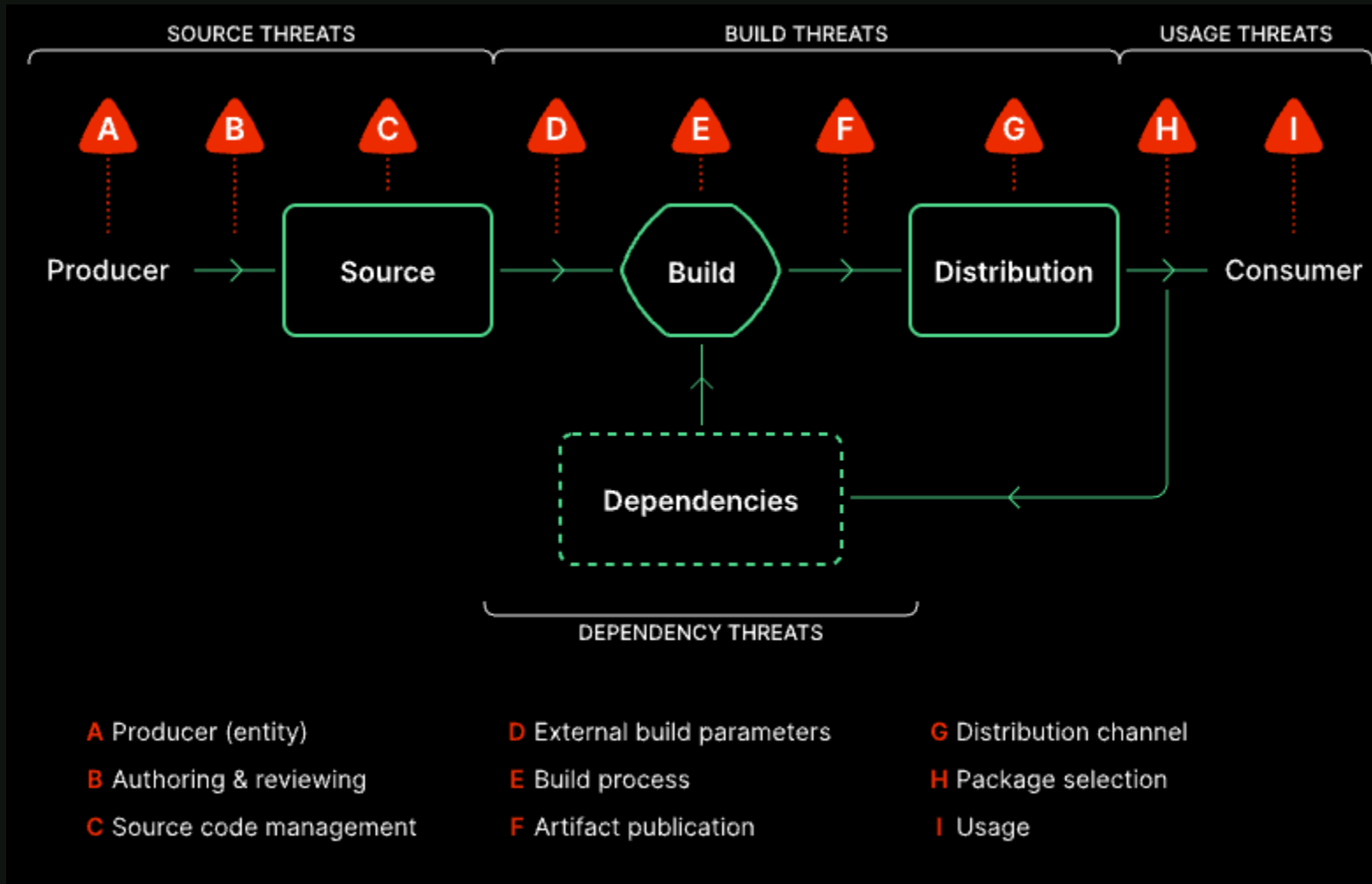
In the AI era, once mundane and laborious tasks can now be done fast and at scale.

A double-edged sword

AI agents boost developer efficiency, but are also being weaponized by malicious actors.



Overview of the Software Supply Chain



What attackers are after



Credential theft

Stealing secrets, tokens and keys from developers and build systems.

Persistent access in the victim's environment

Backdoors

Remote code execution (RCE)

Supply chain attacks exploit trust

Trusted maintainer

Taking over or impersonating a maintainer account to publish malicious releases.

Trusted dependency

Poisoning a package that downstream projects pull in automatically.

Trusted workflow/build

Tampering with CI workflows or build systems so official artifacts carry malicious code.

Trusted signature/provenance

Abusing signing keys or attestations so malicious artifacts look legitimate.



Patterns evolve

Package era

Compromise a publishing identity for mass distribution

CI era

Execute inside the trusted build system and obtain workload identity

AI era

More code, automation and dependencies (tooling), non-human actors increase both scale and ambiguity.



Axios (March 2026)

100M*

Weekly downloads

Compromised long-lived access token

Local build with injected malware.

Publish using stolen token.

Malicious versions **lacked SLSA Build provenance.**



SLSA Build Provenance

Build metadata

Records an artifact's origin, source code and builder.

GitHub Artifact Attestations

A turnkey solution for cryptographically signed build provenance.

Verification

Verify provenance with the GitHub CLI (gh) or Kubernetes admission controllers.

Open-source tooling

Many other open-source tools support build provenance.



Trusted Publishing

Federated trust

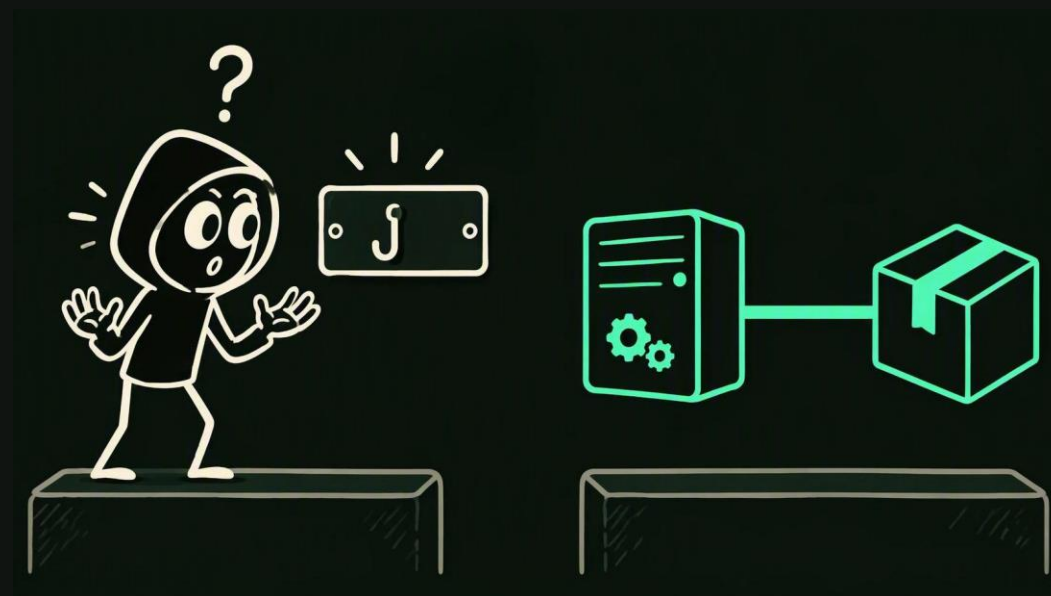
Establishes trust between the build system and the package registry.

Package registries

PyPI and npm

Combine with provenance

Pair trusted publishing with build provenance to give end users cryptographically verifiable attestations.



No long-lived token to steal

Summary

Trusted publishing

Protects the publishing credentials



Provenance

Identifies the builder and the source



What happens when malicious code executes inside the trusted workflow itself?

CI era attacks

Compromised build systems

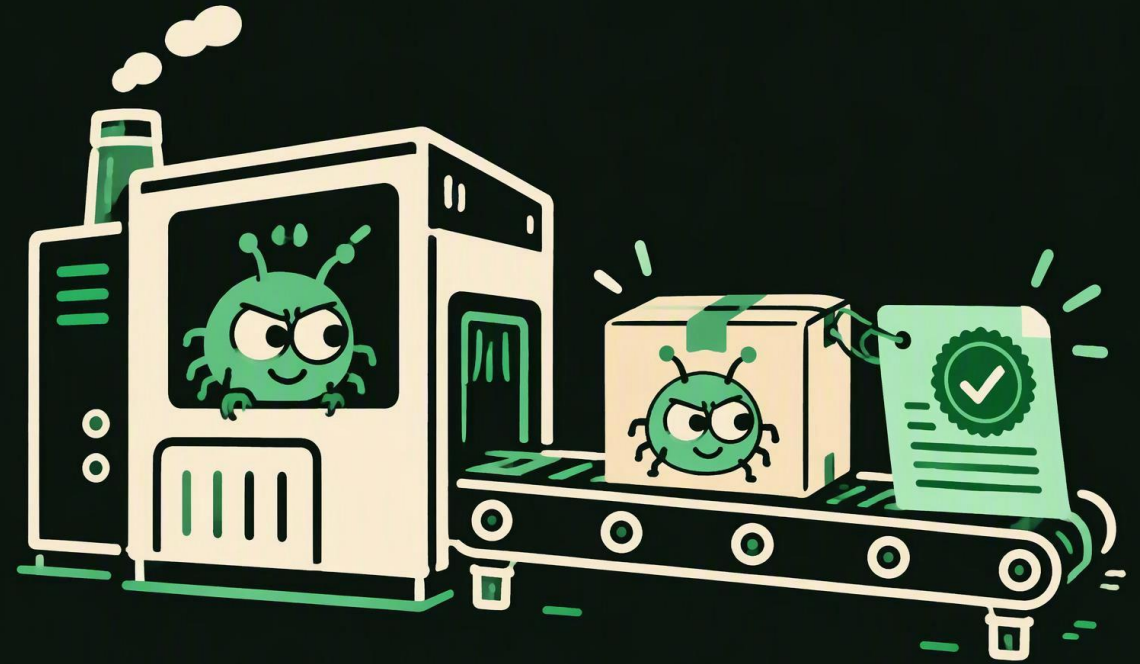
Attackers execute code inside the trusted build system.

npm lifecycle scripts

A path for malicious code to execute during the build.

Valid provenance, malicious artifacts

A compromised build may produce malicious artifacts with valid SLSA Build provenance.



Ultralytics

Poisoned cache

A “pwn request” led to cache poisoning.



Malicious release

The malicious release had build provenance.



Forensic evidence

Build provenance linked to the exact workflow run, aiding the investigation.



Shai-Hulud: waves 1 and 2

Secret theft

The first and second waves stole secrets (second wave expanded the scope of secrets stolen).

Package infection

Malicious install payload scripts infected packages that the environment had permission to publish.



Shai-Hulud: third wave

Trusted CI compromise

Lifecycle scripts executed inside trusted CI.

Workload identity

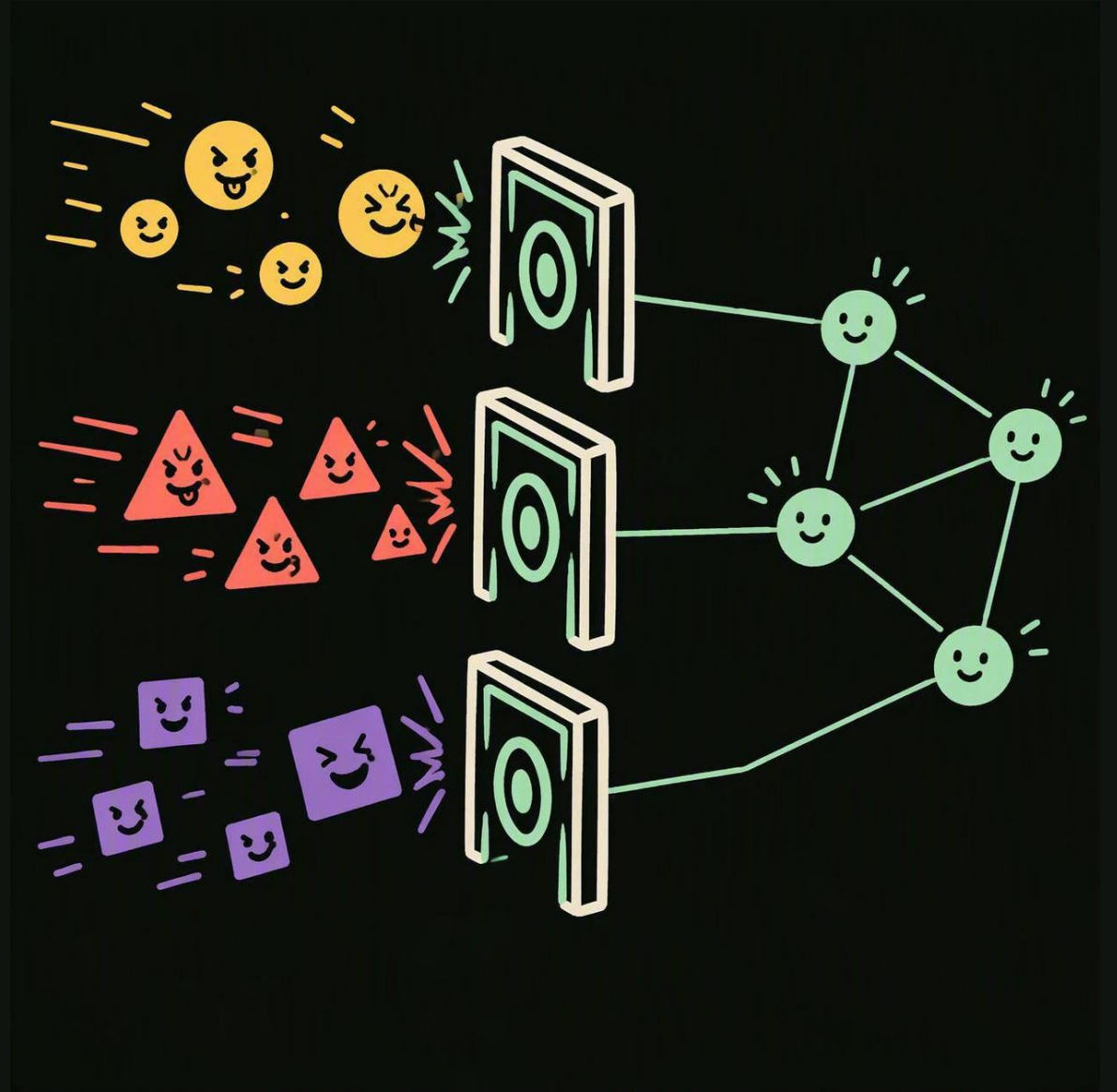
The malicious code obtained the workflow's workload identity.

Trusted publishing

It published malicious packages through trusted publishing with valid provenance.



Threat breakdown by trust boundary



Package

Trusted element

Publisher identity

Failure

Long-lived token theft

Controls

Trusted publishing (token theft)

Build provenance (build identity)

Remaining weakness

The workflow itself can be compromised



CI

Trusted element	Build workflow
Failure	Untrusted code reaches privileged workflow
Controls	Least privilege Workflow policies Cache controls Network egress control
Remaining weakness	Workflow may still process malicious source, dependencies and other inputs



Publication

Trusted element

CI identity

Failure

Compromised CI can publish immediately

Controls

Staged publishing

Protected GitHub Environments

Remaining weakness

Human approval can be a “rubber stamp”



Installation

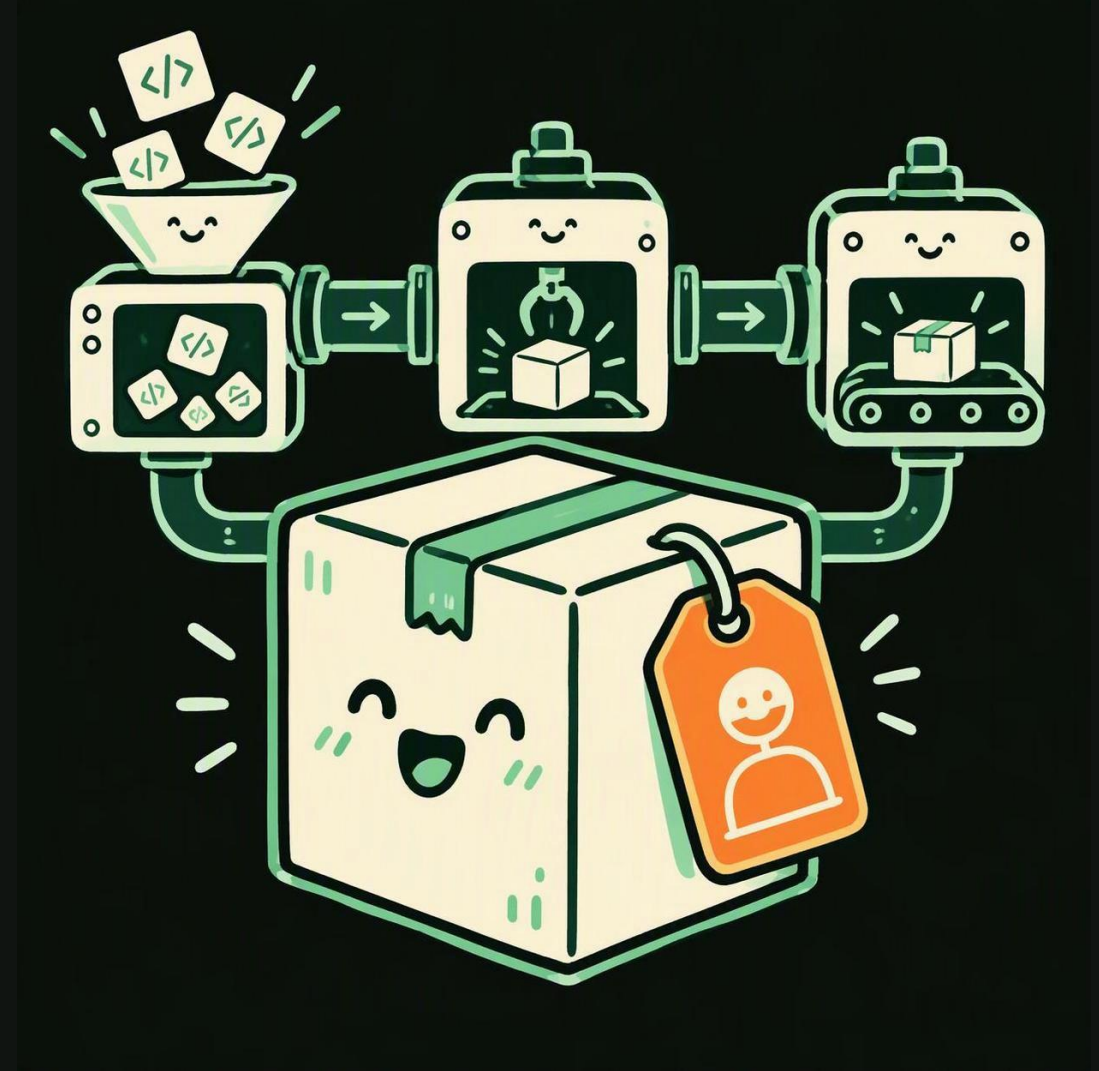
Trusted element	Dependency package
Failure	Lifecycle script executes automatically
Controls	Manual opt-in for install-time scripts
	Package firewalls (blocking proxies with policies)
	Cooldown
Remaining weakness	Approved scripts retain privileges



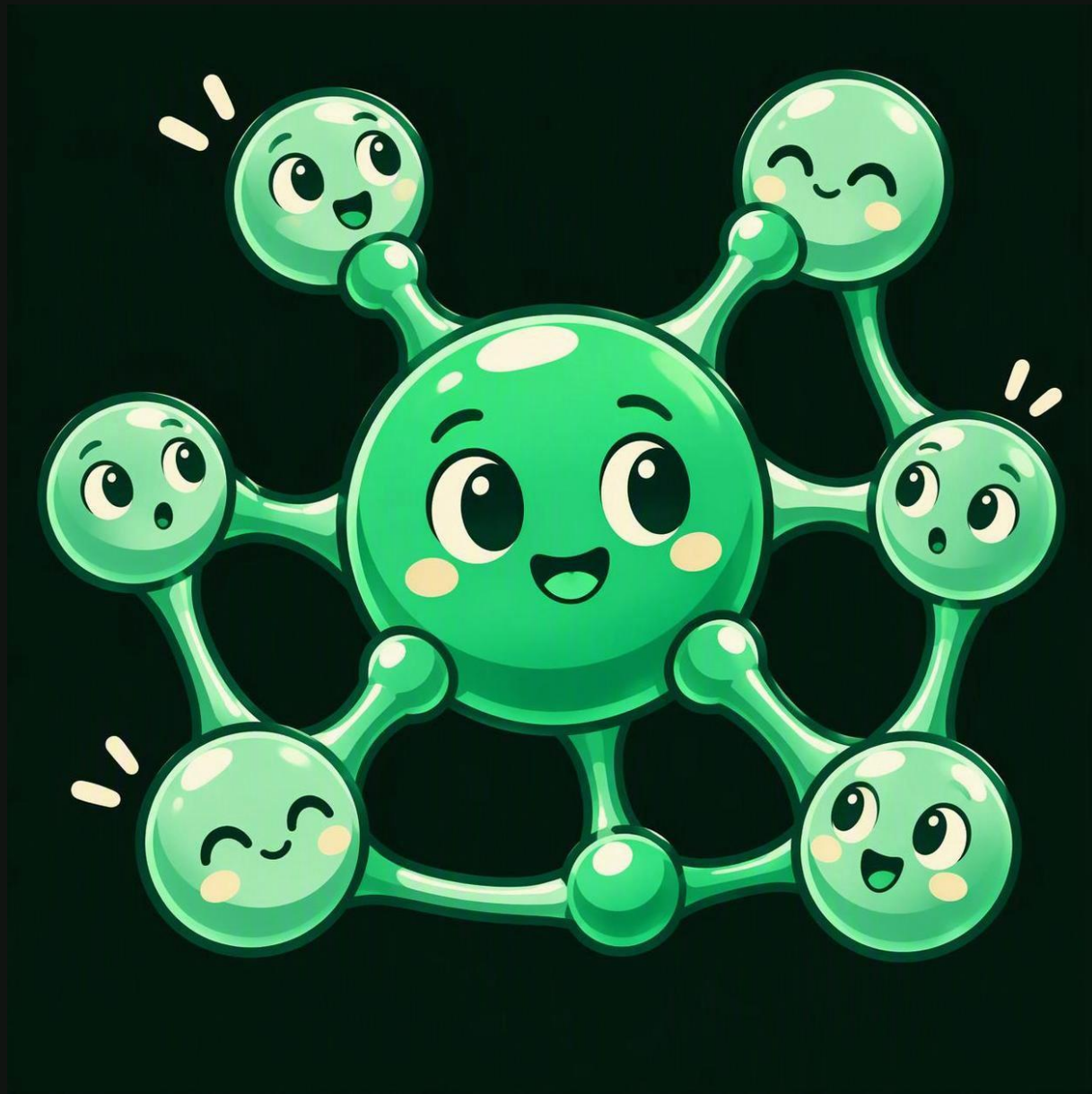
Summary

Package security asks who published.

CI security asks what workflow produced it.



AI-era



AI-era security must also ask:

Which **human or agent** initiated the work?

What **instructions and inputs** influenced it?

Which **tools and models** did it use?

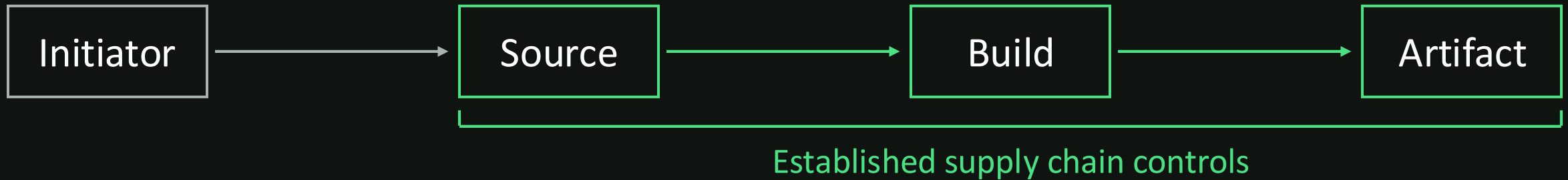
What **authority and network access** did it receive?

Who remains **accountable** for reviewing and releasing the result?

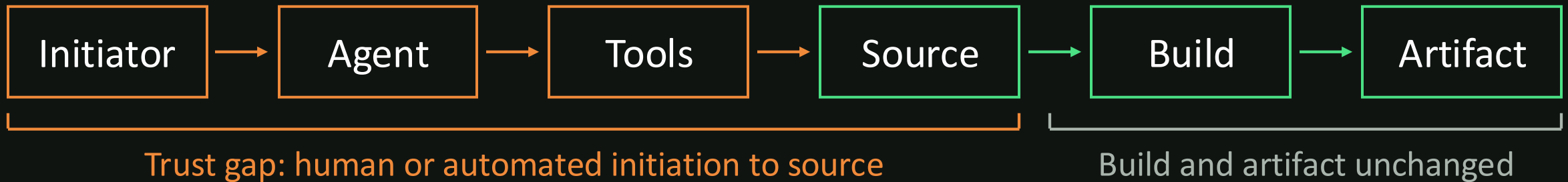


New blind spots

Traditional workflow



Agent-assisted workflow



Source code attribution

Agent-authored commits are signed

Agent-Logs-Url links the commit to the agent session logs

Git provides tamper-evident commit history

Push attribution records the authenticated identity that first pushed a commit

“We can increasingly answer who caused the change. We cannot yet prove that the agent’s instructions, tools, model and environment were trustworthy.”



Example threats

- **Indirect prompt injection** in issues, source files, documentation, dependency metadata and tool output
- **Hidden instructions** that make the agent act outside the initiator's original intent
- **Agent log or tool-output injection** that misleads later reviewers or agents
- **Broad credentials, network access and tool permissions** turn prompt injection into a supply chain compromise
- **Compromised model or tools**



Future controls should cover

- Delegating identity (human or agent)
- Verified model and tool provenance
- Portable, high-fidelity and tamper-evident session evidence
 - Policy decisions and tool-call audit events

Principles already available

- OpenSSF Model Signing
- Isolated agents and ephemeral environments
- Semantic firewalls
- Principle of least privilege
- Retain accountable human approval



Wrap up

- The AI era is here
- No industry standard framework yet
- Don't repeat the early CI mistake:
 - Trusting a highly extensible execution environment
- Frameworks like SLSA made builds **attestable**
 - Future frameworks must make delegated software creation **attestable, attributable and constrained**



Just one more thing

- Signed $\neq$ Secure
- Don't confuse attribution with trust





Thank you

