GitHub

Operationalizing SSDF Dependency Management and Vulnerability Response at Ecosystem Scale



Ankit Kumar Honey

Senior Engineering Manager, GitHub
Supply Chain Security | Dependabot Team

NIST SSCA Forum | September 22, 2026

WHO THIS IS FOR

What you should walk away with

01

You build or operate software

A context-based triage pattern that cuts dependency alert noise without ignoring real risk, and what it takes to run it as policy instead of heroics.

02

You do security research or ops

Why pre-advisory malicious packages break CVE-anchored defense, and what registry-level behavioral detection and quarantine require in practice.

03

You work in policy or law

Three specific, evidence-backed gaps where guidance around the Secure Software Development Framework has not yet caught up with field practice.

One argument for all three: dependency security at ecosystem scale must be behavioral, enforced before install, risk-install, risk-prioritized, and able to operate behind the air gap.

Agenda

01

The new attack surface

02

The view from 30 million repositories

03

Malware moves faster than CVEs (RV.1)

04

Not every critical is critical (RV.2)

05

Automation behind the air gap (PW.4)

06

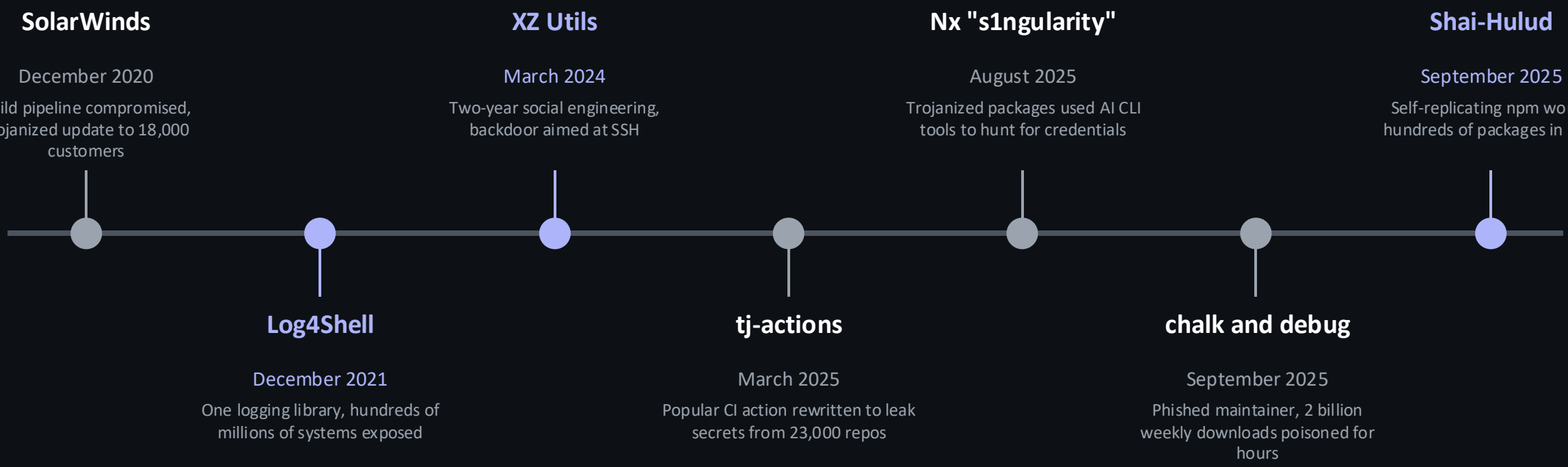
Gaps where field experience can inform guidance



Modern software is assembled, not written. written.

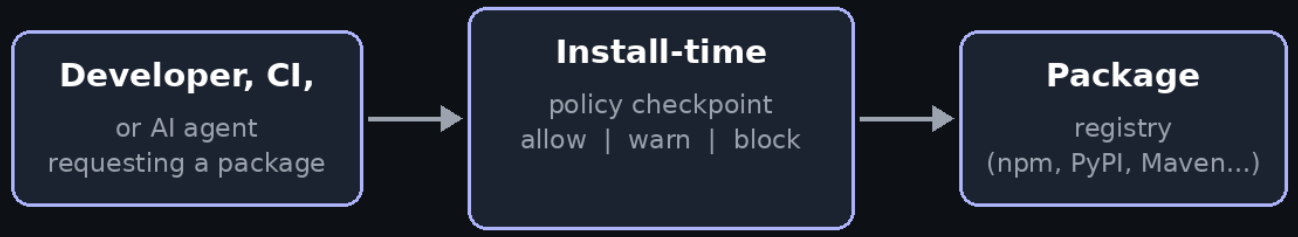
Most of the code in a typical application was downloaded from an open source registry, source registry, not authored in house. That is the supply chain we all now depend on. depend on.

Six years of attacks, each one further upstream



Highlighted: the three we go deep on. The trend: from exploiting bugs to exploiting trust, and from targeting humans to targeting automation.

Detection is the alarm. Enforcement is the lock.



Every package is judged before any code runs.

Signals the checkpoint uses

- Advisories and malware verdicts (known bad)
- Provenance and publisher identity (who built it)
- Name and behavior anomalies (typosquats, new install scripts)
- Organization policy and reviewed exceptions

Detection tells you a package is bad. Enforcement stops it from being installed.

Why detection is not enough

- The worm did its damage during install. An alert that arrives afterwards is a cleanup notice.
- AI coding agents now install packages on their own, at machine speed, with nobody reading alerts.
- Any design that scans after publish or alerts after install leaves a window. The window is the attack.

The prevention pattern

- A policy checkpoint between the installer and the registry. Every package is judged before code runs.
- Explainable decisions and reviewable exceptions, so blocking does not become a new kind of noise.

OUR VANTAGE POINT

Dependabot at GitHub

Ecosystem = the whole open source supply chain, 30+ public registries. Scale = every declared open source dependency across GitHub, public and private, not one application or one enterprise.

Repositories monitored

30M+

Automated dependency security across public public and private code on GitHub

Package ecosystems

34+

npm, PyPI, Maven, NuGet, Go, RubyGems, RubyGems, Cargo and more

Update jobs per month

40M+

Backed by the GitHub Advisory Database and its 28,000+ reviewed advisories

The Secure Software Development Framework (SSDF)

RV.1

Identify vulnerabilities

SSDF SAYS

Know which third-party code you use, and find out fast when it has a known flaw.

IN THIS TALK

Section 01. What happens when malware arrives with no advisory to find.

RV.2

Assess and remediate

SSDF SAYS

Rank each flaw by real risk, then fix the ones that matter first.

IN THIS TALK

Section 02. Why not every critical is critical.

PW.4

Reuse secure software

SSDF SAYS

Keep the components you reuse patched and current wherever they run.

IN THIS TALK

Section 03. Keeping that promise behind an air gap.

RV = Respond to Vulnerabilities | PW = Produce Well-Secured Software (SSDF practice families)

01

Malware does not wait for a CVE

SECTION 1 | RV.1

The CVE lifecycle assumes time: discover, disclose, assign an identifier, publish, patch.

Day 0

A malicious package does its damage the moment it is installed. A defense that waits for an advisory is a defense that arrives after the compromise.

Defending at registry speed

01

Behavioral heuristics

Instead of looking up known identifiers, it analyzes package behavior at the registry level.

02

Quarantine and cooldown

Suspect packages are held out of automated updates before any advisory exists. A cooldown delays adoption of brand-new versions, since most malicious releases are pulled within days.

03

A distinct alert class

Malware alerts are separated from CVE alerts, so the urgent signal is not buried in routine noise and false positives stay reviewable.

Detection buys time. Prevention needs an enforcement point before install, above all where AI agents install packages on their own.

02

Not every critical is critical

SECTION 2 | RV.2

Industry research consistently finds that only a small share of published CVEs are ever exploited in the wild.

~5%

Treating every Critical and High alert as equally urgent creates fatigue, and fatigue slows down the fixes that actually matter.

Sources: Jacobs, Romanosky, Adjerid, Baker, "Improving vulnerability remediation through better exploit prediction," Journal of Cybersecurity, 2020 (5.5% observed exploited) | Kenna Security + Cyentia Institute, Security + Cyentia Institute, Prioritization to Prediction Vol. 1 (under 2% actively exploited) | FIRST EPSS data (2.5 -3% of CVEs show exploitation activity in any 30-day window)

Context decides what is urgent

01

Same CVE, different risk

A vulnerability in a runtime production dependency is not the same as that vulnerability isolated in a build-time test tool.

02

Policy as code

Auto-triage rules let teams encode remediation governance once, then deprioritize low-risk findings safely and consistently.

03

The payoff

Less noise for developers, and measurably faster remediation on the critical-path runtime flaws that deserve the attention.

03

Automation behind the air gap

SECTION 3 | PW.4

Dependency automation assumes the internet

01

The assumption

Public registries, advisory feeds, metadata ingestion, and cloud compute are always reachable.

02

The reality

Federal, defense, and sovereign environments are network-isolated by design.

03

The pattern

Dependabot runs on GitHub Actions, including self-hosted runners inside your own network, against private registries. Patches are proposed and validated inside the boundary.

04

The payoff

Continuous patch validation with no external network exposure.

Where guidance could grow

RV.1

Pre-advisory malware

Response guidance anchors on CVE and NVD. Malicious packages need a playbook that starts before an identifier exists: behavioral detection, a cooldown before adopting new versions, and an enforcement point at install time.

RV.2

Programmatic governance

"Fix the riskiest first" is the right instruction. What is missing is worked examples of the rules, so teams can copy them instead of each writing their own.

PW.4

Isolated environments

Guidance largely assumes connectivity. Secure reuse deserves a reference architecture for air-gapped operation.

GitHub

Thank you!

Ankit Kumar Honey

Senior Engineering Manager, Dependabot, GitHub

linkedin.com/in/ankit-kumar-honey

Email: honeyankit@github.com

Connect / share feedback

