Abandoned Is a Range: Bounding the Supply-Chain Maintenance Gap

- Executive Summary
- Section 1 — Why this report exists
- Section 2 — Methodology
  - 2.1 Data sources
  - 2.2 Corpus expansion: full corpus, not top-N
  - 2.3 Classification: two rules, and why we report both
  - 2.4 Risk-weighted percentage
  - 2.5 Compromise history
  - 2.6 Hand-verification (the halt-press pass)
  - 2.7 What we could not do this quarter
- Section 3 — Headline findings
  - 3.1 The abandonment band, by ecosystem
  - 3.2 The watch list — out-of-band and compromised, by dependents
- Section 4 — The vulnerability feed is mostly malware now (by count), and it barely matters (by weight)
  - 4.1 Supporting analysis: maintainer turnover (and why it is not a headline)
- Section 5 — Implications
- Section 6 — Limitations
- Section 7 — Data, reproduction, and the road to trends

# Abandoned Is a Range: Bounding the Supply-Chain Maintenance Gap

*State of Supply Chain: Q3 2026*

*Joshua Marpet and Cairn Viktor, Value Chain Risk Institute*

## Executive Summary

Last quarter we asked a two-part question about the open-source packages the world builds on: *is the next CVE in this package likely, and is anyone going to fix it?* We measured the intersection of **out-of-band maintenance** (abandoned, archived, deprecated, or superseded) and **compromise history**, and reported a single risk-weighted percentage per ecosystem.

This quarter we did two things. First, we **expanded the corpus from the top ~10,000 packages per ecosystem to the entire published corpus — 13.1 million packages.** Second, we set out to produce the first quarter-over-quarter *trend*.

The trend attempt produced a more important finding than the trend itself: **the headline abandonment number is definitionally soft.** Depending on how you define "still maintained" — release activity alone, or release *and* repository-commit activity — the risk-weighted out-of-band rate for a given ecosystem moves by a factor of two to three. Not because the data is bad, but because "abandoned" is a judgment the data underdetermines. **The honest unit for this measurement is a band, not a number.** Any supply-chain report — including our own Q2 — that quotes a single abandonment percentage is quoting one end of a range and calling it the middle.

That is not a retreat. It is the finding. And underneath it, the verified watch list stands: across eight ecosystems, we name the load-bearing packages that are both out-of-band

**and** carry real, version-ranged security-advisory history — `log4j:log4j`, `pycrypto`, `commons-collections`, `swiftmailer`, `jwt-go`, `uri-js`, and their kin. Those names are not soft. They are specific, recognizable, and sitting under millions of dependency edges.

**What to do Monday:** pull the watch list (§3.2), check it against your own SBOM, and treat any match as a migration candidate. And when you read *any* report's abandonment statistic — ours included — read it as a band.

---

## Section 1 — Why this report exists

This is the second quarterly issue. Q2 established the method; Q3 stress-tests it at full scale and begins the longitudinal series. Our stated commitment last quarter was that the methodology would tighten each quarter and that we would name every framing decision that bears on credibility. This issue does both, and the tightening was substantial enough to be the story.

A note on honesty as method: three times while producing this issue, an early result looked exciting and turned out to be an artifact. A maintainer-turnover "takeover" signal dissolved into benign churn. An inflated npm number turned out to be stable-complete micro-packages miscounted as abandoned. A "malicious package" flag on a popular, legitimate package turned out to be a name collision with typosquat malware. Each is documented below rather than buried. A report that only survives if you don't check its work is not research.

---

## Section 2 — Methodology

## 2.1 Data sources

- **ecosyste.ms** package corpus, full published dump ( `packages-2026-02-05` , 13,108,568 packages across 89 registries; the eight ecosystems reported here are its Q2-comparable subset). This is a point-in-time snapshot dated 2026-02-05; see §6 on freshness.
- **OSV.dev**, complete mirror of all ecosystems (2026-07-17), for compromise history.
- Dependent-repository counts ( `dependent_repos_count` ) from the same ecosyste.ms corpus, for risk-weighting.

## 2.2 Corpus expansion: full corpus, not top-N

Q2 sampled the top ~10,000 packages per ecosystem. Q3 classifies the **entire** corpus. A finding falls out immediately: **it barely matters.** Because risk is weighted by dependent-repository count, and because the most-depended-upon packages are the same whether you take the top 10,000 or all 13 million, the full-corpus risk-weighted numbers land within about a point of a top-10k computation on the same rule. **The method is robust to corpus size.** This is worth stating because it means the expensive instinct — "get more packages" — buys less than the cheap instinct: get the *classification rule* right.

## 2.3 Classification: two rules, and why we report both

Q2's stated rule for "abandoned" was: no release in 730 days **and** no repository commit in 730 days. In implementation, maintenance state can be read two ways, and they diverge sharply:

- **Release-recency (upper bound):** a package with no release in 730 days is out-of-band. This *over-counts* — it flags **stable-complete** packages: code that is finished, not dead. `to-regex-range` , `fs.realpath` , `gopkg.in/yaml.v2` do not cut releases because they do not need to.
- **Release-and-commit (lower bound):** additionally require no repository commit ( `pushed_at` ) in 730 days. This *under-counts* — repository-level commit activity is not package-level. A dead `commons-foo` artifact maps to the still-active

`apache/commons` monorepo and is falsely rescued. Monorepo-heavy ecosystems (Maven, Go) are most affected.

The true abandonment rate sits **between** these bounds. We report both as a band. (A clean validation that the mechanism is real: conda-forge's release-recency rate is an absurd 99.6% — its release dates track recipe rebuilds, not activity — and the commit-aware rule correctly collapses it to 6.75%, rescuing `python`, `numpy`, `pandas`. Where a repository maps one-to-one to a package, the second signal works exactly as intended.)

Packages with a stale release but **no linked repository at all** (no commit signal to check) are placed in a separate **unverified** band rather than assumed dead — the honest "we cannot confirm" bucket.

## 2.4 Risk-weighted percentage

As in Q2: for each ecosystem, the fraction of total dependent-repo-count attributable to out-of-band packages, not the fraction of packages. Counting packages is the wrong unit; a million abandoned zero-dependency packages matter less than ten abandoned load-bearing ones.

## 2.5 Compromise history

For every out-of-band package, the OSV.dev all-time record is queried. As in Q2, this is *compromise history* — "has this package ever had a security event" — not "is the current version vulnerable." The intersection of out-of-band and historically-compromised, ranked by dependents, is the report's primary watch list (§3.2), hand-verified per §2.6.

## 2.6 Hand-verification (the halt-press pass)

Every package in the top-10 of each ecosystem's watch list was checked against its actual OSV records before publication. Result: **zero false memberships** — every named package is genuinely out-of-band and genuinely carries real, version-ranged advisories. **One false attribute:** `fsevents`, a legitimate and widely-used package, was auto-flagged "malicious" because a typosquat malware package (`MAL-2023-462`) reused its

name. `fsevents` has a real historical CVE and belongs on the list; the malware label was stripped. Consequence, adopted as a standing rule: **this report makes no per-package "malicious" claim about any high-dependency package** — name collisions make that flag unreliable at the popular end, where real malware (which carries near-zero dependents) never actually appears.

### 2.7 What we could not do this quarter

- **A clean quarter-over-quarter trend.** Because we changed the method (full corpus; explicit band), Q2's single numbers and Q3's bands are not directly comparable as a delta. Our first attempt at a trend revealed that the method must stabilize before quarter-over-quarter movement is meaningful. Q3 therefore establishes the *stable* method; Q4 is the first true delta. This is the honest cost of getting it right rather than getting it early.
- **A newer snapshot.** `packages-2026-02-05` is the latest ecosyste.ms *bulk dump* published. To keep the findings current, the named watch-list packages (§3.2) were **re-verified against the live ecosyste.ms API on 2026-07-17** — the named, load-bearing packages remain out-of-band as of that date (e.g., `uri-js`: last release 2021, 4.75M dependents; `log4j:log4j`; `swiftmailer`). The risk-weighted bands (§3.1) and the 13.1M-package corpus denominator remain the 2026-02-05 snapshot; only the load-bearing named set is refreshed to July. Fewer than a handful of named packages showed renewed activity since February, and those are the monorepo-attribution artifact described in §2.3, not genuine revivals. Securing a predictable dump cadence for the full corpus is an active priority; see §7.

---

## Section 3 — Headline findings

### 3.1 The abandonment band, by ecosystem

Risk-weighted out-of-band percentage, expressed as the honest band between the commit-aware lower bound and the release-recency upper bound. Full 13.1M-package corpus, point-in-time 2026-02-05.

| Ecosystem | Packages | Out-of-band % (lower–upper band) | Q2 single figure |
|---|---|---|---|
| npm | 5,325,740 | 35.7 – 52.4 | 30.9 (light-rigor) |
| Go | 2,033,883 | 33.2 – 57.9 | 53.1 |
| NuGet | 770,713 | 27.6 – 52.2 | 49.2 |
| Maven | 582,608 | 7.6 – 33.8 | 30.55 |
| PyPI | 783,722 | 10.6 – 24.5 | 23.4 |
| RubyGems | 202,186 | 19.7 – 38.5 | 37.26 |
| Cargo | 233,186 | 11.1 – 24.8 | 24.8 |
| Packagist | 481,047 | 15.3 – 19.2 | 20.34 |

Two readings. First, Q2's published figures sit at or near the **upper** (release-recency) end of every band — meaning Q2, despite describing a release-and-commit rule, behaved in practice like a release-only rule. Second, the bands are **wide** — Maven's spans 7.6 to 33.8, a factor of 4.4. That width is the finding: for Maven and Go especially, whether a third of the ecosystem's dependency-weight is "abandoned" depends entirely on whether you count an active parent monorepo as maintaining its dead child artifacts. Reasonable people define this differently; the number should therefore be reported as the range it actually is.

## 3.2 The watch list — out-of-band and compromised, by dependents

These are load-bearing packages that are both out-of-band **and** carry verified security-advisory history. Read alongside your own SBOM (§5.1); a match is a migration candidate. Characterization is honest: *superseded/EOL* (a named successor exists) is distinguished from *collapsed* (maintainer inactivity), because the fix differs even where the procurement implication — do not build new work on it — is identical.

- **npm:** `uri-js` (4.7M dependents; ReDoS), `set-value` (prototype pollution), `websocket-driver` / `websocket-extensions` (ReDoS), `deep-extend`, `sockjs`, `rc` (real 2021 malware-injection incident), `eslint-utils` (arbitrary code execution).
- **Go:** `gopkg.in/yaml.v2` (superseded by v3; DoS), `github.com/dgrijalva/jwt-go` (collapsed; auth bypass; migrate to `golang-jwt`), `github.com/aws/aws-sdk-go` (v1 superseded by v2), `gogo/protobuf`, `go-jose.v2`, `src-d/go-git.v4` (path-traversal RCE).
- **PyPI:** `pycrypto` (collapsed; 8 advisories incl. weak keys; migrate to `pycryptodome`), `py` (ReDoS), `typed-ast`, `codecov`, `python-apt`.
- **Maven:** `log4j:log4j` (1.x, EOL; deserialization), `commons-collections` (the deserialization CVE), `mysql:mysql-connector-java` (old coordinates; 9 advisories), `org.apache.derby:derby` (22 advisories), `dom4j`, `jackson-mapper-asl`.
- **RubyGems:** `paperclip` (superseded by ActiveStorage), `cocaine`, `websocket-extensions`, `rdiscount`.
- **Cargo:** `failure` (deprecated; RUSTSEC), `untrusted`, `adler`, `paste`, `bincode`.
- **Packagist:** `swiftmailer` (452K dependents; EOL, migrate to `symfony/mailer`), `facade/ignition` (CVE-2021-3129 RCE), `zendframework/*` family (abandoned; migrate to Laminas), `namshi/jose`.
- **NuGet:** `curl`, `starkbank-ecdsa`, `telerikmvcextensions` (removed).

Full CSVs at the published-data URL (§7).

---

# Section 4 — The vulnerability feed is mostly malware now (by count), and it barely matters (by weight)

An incidental but striking observation from the OSV mirror. The npm vulnerability feed contains **219,723** packages with advisories — of which **216,506 (98.5%) are malicious-package reports** (typosquats, dependency-confusion, account-takeover spam), not traditional CVEs. PyPI is similar (86%). Go, Maven, Cargo, and Packagist are almost purely traditional-CVE ecosystems (malware near zero).

The weight tells the opposite story. Those hundreds of thousands of malicious npm packages carry **near-zero dependents** — they are throwaway names nobody installs on purpose. Not one appears in the dependency-weighted watch list. **Malware dominates the feed by count and is absent from it by weight.** The practical implication for anyone consuming OSV data: raw advisory *counts* per ecosystem are now a measure of malware-spam volume, not of risk to real software. Weight, not count, is the only honest denominator — the same lesson §2.4 applies to abandonment, applied to compromise.

## 4.1 Supporting analysis: maintainer turnover (and why it is not a headline)

We also diffed maintainer sets across two vintages (June 2024 → February 2026) for 6.33 million packages trackable across both. **96.4% were stable.** Only 0.37% saw complete maintainer replacement — and inspection shows these are overwhelmingly benign (organizational restructuring, identity-record changes, legitimate handoffs), not takeovers. Turnover base rates vary wildly by ecosystem for structural reasons (rolling-release distributions churn maintainers by design), so any turnover-based risk signal must be normalized per-ecosystem before it means anything. Turnover is a *supporting* method inside the intersection above, not a standalone finding. We report it because a null result, honestly measured, is still a result.

## Section 5 — Implications

- **For procurement (§5.1):** the watch list is most useful checked against your own dependency tree, not read as a direct risk readout. A named package with hundreds of thousands of dependents is load-bearing by construction; you likely inherit it transitively even if your manifest never names it.
- **For regulators:** abandonment is a range, not a statistic. Any mandate or metric built on a single "percent abandoned" threshold is building on sand — the same ecosystem can be reported at 8% or 34% under equally defensible definitions. Require the band.
- **For registries:** a package-level "last meaningful maintenance" signal — distinct from repository-level commit activity — would collapse the width of these bands. The monorepo attribution problem is yours to fix at the source.
- **For the maintainer community:** stable-complete is not abandoned. The measurement problem this quarter is partly a *language* problem: we lack a machine-readable way for a maintainer to say "this is finished and correct," distinct from "this is neglected." That signal would be worth more than any scanner.

---

## Section 6 — Limitations

- Point-in-time snapshot (2026-02-05); not live.
- Compromise = history, not current-version vulnerability (no version-range intersection with installed versions).
- Repository-level commit data cannot reliably indicate package-level maintenance (the band's lower bound is a genuine floor, not a point estimate).
- Watch list verified to top-10 per ecosystem; the long tail is machine-classified, not hand-checked.

- OSSF Criticality Score — Q2's §4 time-series spine — was not re-incorporated this quarter; confirming whether that upstream pipeline has resumed publishing is outstanding and is itself a Q4 data point.

---

## Section 7 — Data, reproduction, and the road to trends

All per-ecosystem CSVs (maintenance bands, watch list, malware-flood counts, turnover) are published at [github.com/value-chain-risk-institute/state-of-supply-chain](github.com/value-chain-risk-institute/state-of-supply-chain) ( `data/2026-Q3/` ) under CC BY-SA 4.0 (share-alike inherited from ecosyste.ms source data), with attribution to ecosyste.ms and OSV.dev whose open data makes this possible. The full pipeline is documented in the SCSC Data Pipeline Runbook and reproduces from the named public dumps.

**On becoming trends research:** this issue is the start, and the operative word is *start*. A snapshot becomes a trend only with temporal depth — recovered history and a predictable future cadence. Two efforts follow directly: recovering additional historical package-corpus vintages, and securing a regular dump cadence with the upstream data providers. Each quarter published compounds the value of the last. Q3 establishes the stable method; Q4 is the first true year-over-year delta. The line, not the dot, is the argument.

---

*Value Chain Risk Institute — the neutral, open clearinghouse for supply-chain security posture. Methodology CC BY. Free methodology briefings via VCRI; commercial implementation via Cairn Risk Co. Nothing in this report is behind a paywall.*