Fixing Open Source at the Source: The Case for Continuously Verified Components Before January 2027

Hritik Kumar
Safeguard.sh Inc
hi@safeguard.sh

October 2026

Abstract

Modern software is assembled mostly from open source components that its builders did not write and often did not choose directly. When a vulnerability is disclosed in one of those components, the fix travels a long path: a volunteer maintainer prepares a patch, publishes a release, and thousands of downstream projects must notice and upgrade. Empirical studies of package ecosystems show that this path is slow and uneven, and incidents such as Log4Shell and the xz Utils backdoor show the cost when it fails. At the same time, regulation is moving from guidance to deadlines: the EU Cyber Resilience Act has required manufacturers to report actively exploited vulnerabilities since 11 September 2026, and most of its remaining obligations apply from 11 December 2027. This paper argues that downstream consumers should not have to wait on every upstream release, and describes a model for publishing continuously verified ("Gold") open source components. A component is Gold only while it carries no exploitable critical or high severity vulnerabilities, has a completed license review, has signed build provenance at SLSA Level 3 or higher, and has a public evidence record. We describe the lifecycle that produces such components, including upstream patching first and maintained forks only when upstream cannot move. We also set out its limitations, and propose a concrete objective: to help organisations remediate the open source they depend on before January 2027, ahead of the Act's main obligations.

1 Introduction

Open source components make up most of the code in typical applications, and most of those components arrive transitively. Studies of the npm ecosystem found that a small number of maintainers and packages sit on the dependency paths of a very large share of the ecosystem, so a single vulnerable or compromised package can reach a large part of it [18, 6]. Developers frequently do not update vulnerable dependencies promptly, even after advisories are published [9], and counting which vulnerable dependencies actually matter is itself non-trivial [14].

Two recent incidents illustrate the two failure modes. Log4Shell (CVE-2021-44228) was a flaw in a ubiquitous logging library; public guidance urged organisations to find and patch every affected system, a task that took many organisations months because they could not quickly enumerate where the library was embedded [1]. The xz Utils compromise (CVE-2024-3094) was a malicious change introduced into a trusted upstream project and caught shortly before wide distribution [2]. The first failure is *slow remediation of a known flaw*; the second is *loss of integrity at the source*.

Taxonomies of software supply chain attacks document many further variants of the second kind [10, 13].

This paper makes three contributions. First, we summarise why the remediation path for open source is structurally slow and why upcoming regulatory deadlines make that slowness costly (Sections 2 and 3). Second, we define a verification bar for open source components and a lifecycle for producing and maintaining components that meet it (Section 4). Third, we discuss limitations and propose a measurable objective for the period before January 2027 (Sections 5 and 6).

2 Background: why remediation is slow

A vulnerability in an open source dependency creates two gaps for a downstream team. The *upgrade gap* is the time between a fixed release being published and that release running in the team's production builds. The *fix gap* is the time before any fixed release exists. The upgrade gap is driven by dependency depth, version constraints that block upgrades, and review cost; studies of npm and Maven have repeatedly found that vulnerable versions persist long after fixes are available [6, 9]. The fix gap is driven by maintainer capacity, which is often a single volunteer for widely used packages [18].

Prioritisation helps but does not remove either gap. CISA's Known Exploited Vulnerabilities catalog identifies vulnerabilities with evidence of exploitation in the wild [3], and the Exploit Prediction Scoring System estimates the probability of exploitation [8]. Software bills of materials (SBOMs) let a team answer "do we ship this component?" quickly [12]. Signed provenance frameworks such as in-toto, Sigstore and SLSA let consumers check where an artifact came from and whether it was altered [17, 11, 15]. These are necessary building blocks, but each still leaves the consumer waiting on an upstream release when the fix gap is open.

3 Regulatory context

The EU Cyber Resilience Act (Regulation (EU) 2024/2847) sets cybersecurity requirements for products with digital elements placed on the EU market [4]. Its reporting obligations have applied since 11 September 2026: manufacturers must send an early warning of an actively exploited vulnerability within 24 hours and a notification within 72 hours, and most other obligations apply from 11 December 2027 [4, 5]. In the United States, Executive Order 14028 and the NIST Secure Software Development Framework set expectations for software supplied to federal agencies, including provenance and the management of third-party components [7, 16]. A manufacturer that cannot enumerate the open source components in its products, or cannot move them to safe versions quickly, will find these timelines difficult to meet.

4 Continuously verified components

4.1 The verification bar

We call a component version *Gold* only while all of the following hold:

1. **No exploitable critical or high severity vulnerabilities** known against the version.
2. **A completed license review**, so consumers know their obligations.
3. **Build provenance at SLSA Level 3 or higher**, with the artifact signed using Sigstore [15, 11].
4. **A public evidence record** per artifact that an auditor can inspect.

5. **Continuous re-verification**: the status is re-checked as new advisories appear, and a component that regresses loses its Gold status.

The last property is what distinguishes this bar from a one-time scan: the label describes the component's current state, not its state on the day it was built.

4.2 Lifecycle

The lifecycle that produces Gold components has six steps.

1. **Inventory.** Build an inventory of the open source a consumer depends on, from SBOMs and manifests, and identify which components have a Gold equivalent.
2. **Patch upstream first.** When a component is vulnerable, propose the fix to the upstream project, because a fix at the source benefits every consumer.
3. **Fork only when necessary.** If the issue is severe and upstream cannot or will not ship a fix, build a maintained fork, apply the fix, and track upstream so the fork can be retired once upstream catches up.
4. **Verify and sign.** Rebuild from source in a hardened pipeline, run the checks above, generate provenance, and sign the result.
5. **Publish with evidence.** Publish the artifact with its evidence record, through a registry that consumers can point their package managers at, so that hardened equivalents resolve without changing developer workflows.
6. **Re-verify continuously.** Re-evaluate published versions against new advisories, and revoke status on regression.

4.3 Consumer integration

Consumers can use such components in three ways that close the two gaps described in Section 2. A public lookup service answers "which version is safe?" for a package or vulnerability identifier. A continuous integration gate fails a build when a known exploited vulnerability is introduced, so removed versions cannot quietly return. A registry proxy resolves hardened equivalents in place of upstream artifacts. Safeguard operates these as Gold Open Source (a free public directory and lookup at `https://gold.safeguard.sh`), Gold Registry (hardened packages and container images), and a GitHub Action for CI gating; the fork-and-patch flow in step 3 is currently in early access.

5 Limitations and risks

- **Scope.** The model applies to open source that a team builds and ships. Vulnerabilities in vendor appliances, hosted services and operating systems a team does not build must still be fixed by their vendors.
- **Trust shifts, it does not disappear.** Consumers of rebuilt components must trust the rebuilder's pipeline. Signed provenance and public evidence make that trust checkable, but they do not remove the need for it [17, 11].
- **Fork maintenance.** Maintained forks carry long-term cost and can diverge from upstream behaviour. They should be a last resort, retired as soon as upstream ships an equivalent fix.
- **Severity is not exploitability.** "No exploitable critical or high vulnerabilities" depends on the quality of vulnerability data and of exploitability judgements; prioritisation signals such as KEV and EPSS help but are imperfect [3, 8].
- **Upstream relationships.** Forking popular projects can create friction with maintainers. Patching upstream first, with credit, is both the ethical and the practical default.

6 An objective for January 2027

We propose a concrete objective: *help organisations remediate the open source they depend on before January 2027*. That date leaves most of a year before the Cyber Resilience Act's main obligations apply in December 2027, and it follows the start of its 24-hour reporting regime [4, 5]. We state this as an objective, not as a result. Progress toward it can be measured with simple, reproducible indicators:

- the share of a product's direct and transitive dependencies covered by an SBOM;
- the number of components with known exploited vulnerabilities still shipping;
- the median upgrade gap, from a fixed release to production, for critical and high findings;
- the number of open fix gaps, where no upstream fix exists, and how each is being handled (upstream patch, maintained fork, or mitigation).

We intend to report against these indicators in future work.

7 Conclusion

Open source security today depends on every downstream team independently noticing and closing two gaps for every vulnerable component it ships. Regulation now puts a clock on that work. Continuously verified components, produced by patching upstream first and forking only when necessary, and published with signed provenance and public evidence, let consumers close those gaps without waiting on every upstream release. The approach has real limits, which we state plainly, but it offers a practical route to a cleaner open source supply chain before January 2027.

References

[1] Cybersecurity and Infrastructure Security Agency. *Apache Log4j Vulnerability Guidance*. `https://www.cisa.gov/news-events/news/apache-log4j-vulnerability-guidance`.

[2] Cybersecurity and Infrastructure Security Agency. *Reported Supply Chain Compromise Affecting XZ Utils Data Compression Library, CVE-2024-3094*. 29 March 2024.

[3] Cybersecurity and Infrastructure Security Agency. *Known Exploited Vulnerabilities Catalog*. `https://www.cisa.gov/known-exploited-vulnerabilities-catalog`.

[4] Regulation (EU) 2024/2847 of the European Parliament and of the Council on horizontal cybersecurity requirements for products with digital elements (Cyber Resilience Act). Official Journal of the European Union, 2024.

[5] National Cyber Security Centre, Ireland. *EU Cyber Resilience Act*. `https://ncsc.gov.ie/cra/`.

[6] A. Decan, T. Mens, and E. Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *Proc. 15th International Conference on Mining Software Repositories (MSR)*, 2018.

[7] Executive Order 14028, *Improving the Nation's Cybersecurity*. 12 May 2021.

[8] J. Jacobs, S. Romanosky, B. Edwards, I. Adjerid, and M. Roytman. Exploit Prediction Scoring System (EPSS). *Digital Threats: Research and Practice*, 2(3), 2021.

[9] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue. Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration. *Empirical Software Engineering*, 23(1), 2018.

[10] P. Ladisa, H. Plate, M. Martinez, and O. Barais. SoK: Taxonomy of attacks on open-source software supply chains. In *Proc. IEEE Symposium on Security and Privacy*, 2023.

[11] Z. Newman, J. S. Meyers, and S. Torres-Arias. Sigstore: Software signing for everybody. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2022.

[12] National Telecommunications and Information Administration. *The Minimum Elements for a Software Bill of Materials (SBOM).* July 2021.

[13] M. Ohm, H. Plate, A. Sykosch, and M. Meier. Backstabber's knife collection: A review of open source software supply chain attacks. In *Proc. DIMVA*, 2020.

[14] I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci. Vulnerable open source dependencies: Counting those that matter. In *Proc. ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2018.

[15] Open Source Security Foundation. *Supply-chain Levels for Software Artifacts (SLSA) Specification v1.0.* 2023. `https://slsa.dev`.

[16] M. Souppaya, K. Scarfone, and D. Dodson. *Secure Software Development Framework (SSDF) Version 1.1.* NIST SP 800-218, February 2022.

[17] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos. in-toto: Providing farm-to-table guarantees for bits and bytes. In *Proc. 28th USENIX Security Symposium*, 2019.

[18] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel. Small world with high risks: A study of security threats in the npm ecosystem. In *Proc. 28th USENIX Security Symposium*, 2019.