Volume 3 · Issue 1 · 2026 · pp. 22–30

Where Is the Fix? A Cross-Ecosystem Census of Machine-Actionable Remediation Endpoints in OSV

Fabrizio Pastore
ORCID: 0000-0003-3541-3641
SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg

Norrathep Rattanavipanon
ORCID: 0000-0003-1192-5079
Department of Computer Science, Faculty of Science, Prince of Songkla University, Thailand

Correspondence: Fabrizio Pastore · contact details held in the journal record

Publication history. Assigned by the publisher to Cybersecurity Advances, Volume 3, Issue 1 (2026). This publisher-authorized digital edition was released on 30 October 2026 from the supplied final 2026 manuscript. Review-stage notices were removed and journal typography, pagination, metadata, and accessibility were normalized; substantive scholarly content was preserved.

© 2026 the author(s). Published by AlphaTech Scholarly Press under the Creative Commons Attribution 4.0 International License (CC BY 4.0).
No DOI is displayed unless and until registration is independently verified.

Background: Open Source Vulnerability (OSV) records can describe affected versions with fixed, last_affected, limit, and enumerated-version representations. Only an explicit fixed event directly identifies a version that contains a fix.

Objective: We measured the availability and concentration of remediation endpoints across six package ecosystems in an OSV snapshot.

Note on shared data acquisition: this study and its companion papers in this issue (e002, e004) share the same byte-identical OSV archive snapshot, acquired and frozen once for reproducibility across all three analyses. Readers should consult the shared-acquisition disclosure in the companion papers for the full protocol.

Methods: Archives for PyPI, npm, Maven, Go, crates.io, and RubyGems were frozen by SHA-256. A protocol-locked, test-driven pipeline deduplicated records and classified package entries, ranges, and terminal events. The primary estimand was the proportion of ranged affected entries containing at least one fixed event. Prespecified sensitivities included withdrawn records, exact-version enumeration, SEMVER-only ranges, pre-2026 records, and exclusion of the largest no-fix source family.

Results: Among 274,705 active ranged entries, 51,540 (18.76%) exposed an explicit fix. Prevalence ranged from 3.74% for npm to 95.51% for RubyGems. Of 223,165 entries without a fix, 207,804 had an open range without enumerated affected versions. However, 214,278 no-fix entries (96.02%) belonged to the MAL identifier family, malicious-package advisories whose operational response is removal rather than upgrade to a repaired release. Excluding that family increased explicit-fix prevalence to 85.29% (51,508/60,395). Results were deterministic across two executions.

Conclusions: Corpus-wide endpoint prevalence is dominated by advisory semantics and should not be interpreted as a simple data-quality score. Consumers should distinguish malicious-package removal from repairable-vulnerability upgrading, preserve enumeration and terminal-event provenance, and fail safely when no compatible fixed version is represented.

Keywords: OSV; vulnerability remediation; affected versions; open-source security; vulnerability databases; package ecosystems

1. Introduction

Machine-readable vulnerability records support two related but different questions: whether an installed version is affected, and what change can remediate that condition. A database can answer the first question by listing affected versions or by describing a version range. The second question requires stronger evidence. For upgrade-oriented automation, the most direct representation is the first version known to contain the fix. If that endpoint is absent, a tool may still identify affected software, but it cannot safely infer a repaired release from the range alone.

The Open Source Vulnerability (OSV) schema provides several endpoint forms. A fixed event identifies a version containing a fix. A last_affected event states the highest version known to be affected at publication time. A finite limit constrains a range but need not identify a fixed version. In the absence of a limit, the specification assumes an unbounded * limit. OSV strongly recommends fixed over last_affected where possible because the latter can create false-negative risk [1]. The schema also permits explicit enumeration of affected versions, which can support matching even when no fixed endpoint is present.

These distinctions matter increasingly as scanners move from detection toward guided remediation. OSV-Scanner, for example, provides an experimental workflow that proposes dependency changes [11]. Such workflows must combine advisory data with package-manager semantics, dependency constraints, and project-specific compatibility. An explicit fix does not make an upgrade safe, but its absence removes one important machine-readable input.

Prior research has shown that affected-version metadata is difficult to curate. Dong et al. detected large-scale disagreement between standardized National Vulnerability Database fields and underlying reports [4]. Ren et al. similarly studied inconsistent vulnerable-version descriptions across databases [5]. Other work reconstructs affected releases from source-code history, patches, developer logs, or package-registry versions [6–9]. CVEfixes provides a large public mapping between vulnerabilities and fix commits for empirical research [10]. These studies address correctness, inconsistency, or reconstruction. They do not provide a cross-ecosystem census of which OSV package entries expose the schema's explicit remediation endpoint.

We therefore ask how explicit fixes are represented across six official OSV ecosystem archives and what alternative evidence is present when they are absent. The contribution is descriptive rather than causal. We measure a frozen representation, preserve its source composition, and show why corpus-wide prevalence can be misleading when records with different response semantics are pooled. The study publishes its frozen protocol, test-first implementation, source checksums, row-level inventories, aggregate tables, figures, and deterministic semantic-result hash.

2. Background and related work

2.1 OSV range semantics

An OSV affected entry associates a package with ranges and/or enumerated versions [1]. SEMVER ranges use semantic-version precedence; ECOSYSTEM ranges rely on ecosystem-specific ordering; and GIT ranges use commit ancestry. Every usable range must contain at least one introduced event. Range events form a timeline of status changes.

The endpoint events are not interchangeable. fixed identifies a version containing the repair. last_affected is a publication-time ceiling in the absence of a known fixed version. A finite limit narrows reachability or ordering but does not assert that the limit is a repaired release. If no explicit limit exists, infinity is implied. The schema disallows mixing fixed and last_affected in one event sequence and recommends fixed because it more precisely supports remediation [1].

Enumeration adds a separate evidence channel. An explicit affected-version list may allow a consumer to match installed releases without evaluating an ecosystem-specific range. It still does not necessarily identify the earliest repaired version. We therefore measure enumeration as an alternative representation, not as a substitute for a fixed event.

2.2 Affected-version evidence

Vulnerability databases aggregate information produced under different disclosure and curation practices. Dong et al. analyzed 78,296 CVE identifiers and found substantial disagreement between reports and normalized NVD information [4]. Ren et al. developed a system for detecting vulnerable-version inconsistencies across reports and databases [5]. These results caution against treating any one structured field as ground truth.

A complementary research line attempts to reconstruct affected versions. Nguyen et al. scanned historical source versions for code evidence associated with vulnerability fixes [6]. He et al. combined patches and developer logs with version-tree analysis [7]. Wu et al. used package-registry versions and selected vulnerable and patched code elements to identify affected library releases [8]. Chen et al. later compared affected-version identification techniques at broader scale [9]. These approaches evaluate or generate version boundaries; the current study instead audits whether a frozen production data representation exposes a particular boundary type.

Fix evidence has also been organized for downstream research. CVEfixes links vulnerability records to source-code fixes [10]. Such datasets are valuable for code-level analysis, but a fix commit is not automatically equivalent to a released package version. Upgrade automation still needs ecosystem-specific release evidence.

2.3 Why source semantics matter

Not every advisory describes a defect followed by a repaired release. Malicious-package advisories often describe packages that should be removed entirely. An open range beginning at 0 can be semantically appropriate when all releases are considered malicious and no trustworthy fixed release exists. Pooling these records with conventional vulnerabilities can make endpoint absence look like missing remediation metadata when it actually reflects a different response model.

The protocol therefore prespecified source-family concentration analysis and exclusion of the largest no-fix family as a sensitivity, without assuming in advance which family would dominate. This design preserves the corpus-wide result while preventing it from becoming an unsupported quality ranking.

3. Methods

3.1 Frozen protocol and corpus

The protocol was frozen and hashed before any Manuscript 4 endpoint outcome was computed. Its SHA-256 is 26ce1c7611c4dc7f7fc9b7f148241fa6662d47c8b6665fe15ad1ccfb6cc05021. The analysis reused byte-identical official OSV archives acquired on 1 October 2026 for a preceding study. Reuse was explicit in the protocol and avoided treating a later moving download as the same snapshot.

The six archives were PyPI, npm, Maven, Go, crates.io, and RubyGems [2]. For each, the pipeline verified its recorded URL, acquisition time, HTTP validators, byte size, SHA-256, and ZIP CRC. It parsed every JSON member. The archives contained 25,995, 229,753, 7,160, 9,421, 2,870, and 5,395 records, respectively. No member was malformed, and no non-JSON member was present.

Records were grouped by OSV identifier. Canonically identical payloads were collapsed. IDs with divergent payload hashes were to be excluded from the primary analysis and represented deterministically in a sensitivity; none occurred. The pipeline observed 280,270 unique IDs, 324 exact cross-archive duplicates, and 280,247 unique in-scope records before excluding withdrawals. The primary active-record set contained 278,522 records.

3.2 Units and inclusion criteria

The primary unit was an affected-package entry. An entry was in scope when it identified one of the six target ecosystems, supplied a non-empty package name, and included a non-empty range or versions list. Package-less Git entries were excluded because interpreting them requires repository-graph analysis and they do not identify a package-ecosystem release.

The primary denominator contained entries with at least one usable range: a string range type and non-empty event list. Enumeration-only entries were reported separately. Withdrawn records were excluded from the primary analysis and included in a prespecified sensitivity.

Each valid event object had exactly one recognized event key with a string value. For each range, the pipeline recorded the presence of introduced, fixed, last_affected, finite limit, infinite limit, and an open endpoint. An endpoint was open when no valid fixed, last_affected, or finite limit existed; this implements the schema's implicit infinite limit. The implementation did not compare arbitrary ecosystem version strings or infer event-order correctness.

3.3 Outcomes

The primary outcome was has_any_fixed: at least one usable range in the affected entry contained a valid fixed event. Secondary entry-level indicators captured enumerated versions, non-fix terminal evidence (last_affected or finite limit), and an open range without enumeration. These secondary indicators can overlap and are not additive categories.

At record level, the study classified active records as: all ranged entries having a fix; some having a fix and some lacking one; no ranged entry having a fix; or enumeration-only. Repeated identical affected entries within one record were deduplicated by canonical content.

Source family was the identifier prefix before the first hyphen. Package identities used the ecosystem plus Unicode-NFKC, trimmed, case-folded package names. Among no-fix entries, concentration was measured by source family and package using shares and the Herfindahl-Hirschman index.

3.4 Sensitivity analyses

Six analyses were prespecified: inclusion of withdrawn records; deterministic inclusion of divergent-ID representatives; exclusion of entries with exact version enumeration; exclusion of the largest no-fix source family; restriction to SEMVER-only entries; and restriction to records published before 2026. The corpus is a finite snapshot, so we report counts, proportions, percentage-point contrasts, and descriptive Wilson intervals rather than null-hypothesis tests.

3.5 Implementation and reproducibility

Python 3.13 code was developed test-first. Fifteen tests cover canonical hashing, Unicode package normalization, exact and divergent duplicates, event validity, implicit open endpoints, finite and infinite limits, mixed terminal semantics, enumeration-only entries, record classes, Wilson intervals, concentration, semantic hashing, source-family extraction, and safe ZIP parsing. The full suite passed.

All outputs were deterministically sorted. A canonical projection of headline counts, denominators, ecosystem results, sensitivities, and source hashes produced semantic SHA-256 0fb7de6c29f8dd5656119bc7971ba1bf4634d2bbe28f6b99c9034ccb43ec3446. A complete second execution reproduced the hash exactly.

4. Results

4.1 Corpus-wide endpoint availability

The active corpus contained 298,598 in-scope affected entries. Of these, 274,705 had at least one usable range and formed the primary denominator. Explicit fixes occurred in 51,540 entries, for a prevalence of 18.76% (descriptive Wilson interval, 18.62%–18.91%). The remaining 223,165 entries lacked an explicit fixed endpoint.

Alternative representations were uneven. Among no-fix ranged entries, 13,103 had affected-version enumeration, 5,346 had at least one non-fix terminal event, and 207,804 had an open range without enumeration. These indicators overlap: an entry can have enumeration and a non-fix terminal, for example. They should not be summed as mutually exclusive causes.

At record level, 33,092 active records (11.88%) had explicit fixes for every ranged in-scope entry. Another 1,487 (0.53%) mixed entries with and without fixes. A total of 220,154 (79.04%) had no fixed endpoint in any ranged entry, while 23,789 (8.54%) were enumeration-only.

4.2 Ecosystem differences

Endpoint prevalence differed sharply across ecosystems (Table 1 and Figure 1). RubyGems had the highest explicit-fix prevalence at 95.51% (1,597/1,672), followed by Maven at 85.62% (10,882/12,710), crates.io at 80.06% (2,838/3,545), Go at 77.84% (11,103/14,264), and PyPI at 68.00% (16,994/24,991). npm was the outlier at 3.74% (8,126/217,523).

Table 1. Explicit fixed events by ecosystem.

Ecosystem	Ranged entries	With fixed	Without fixed	Fixed prevalence
PyPI	24,991	16,994	7,997	68.00%
npm	217,523	8,126	209,397	3.74%
Maven	12,710	10,882	1,828	85.62%
Go	14,264	11,103	3,161	77.84%
crates.io	3,545	2,838	707	80.06%
RubyGems	1,672	1,597	75	95.51%

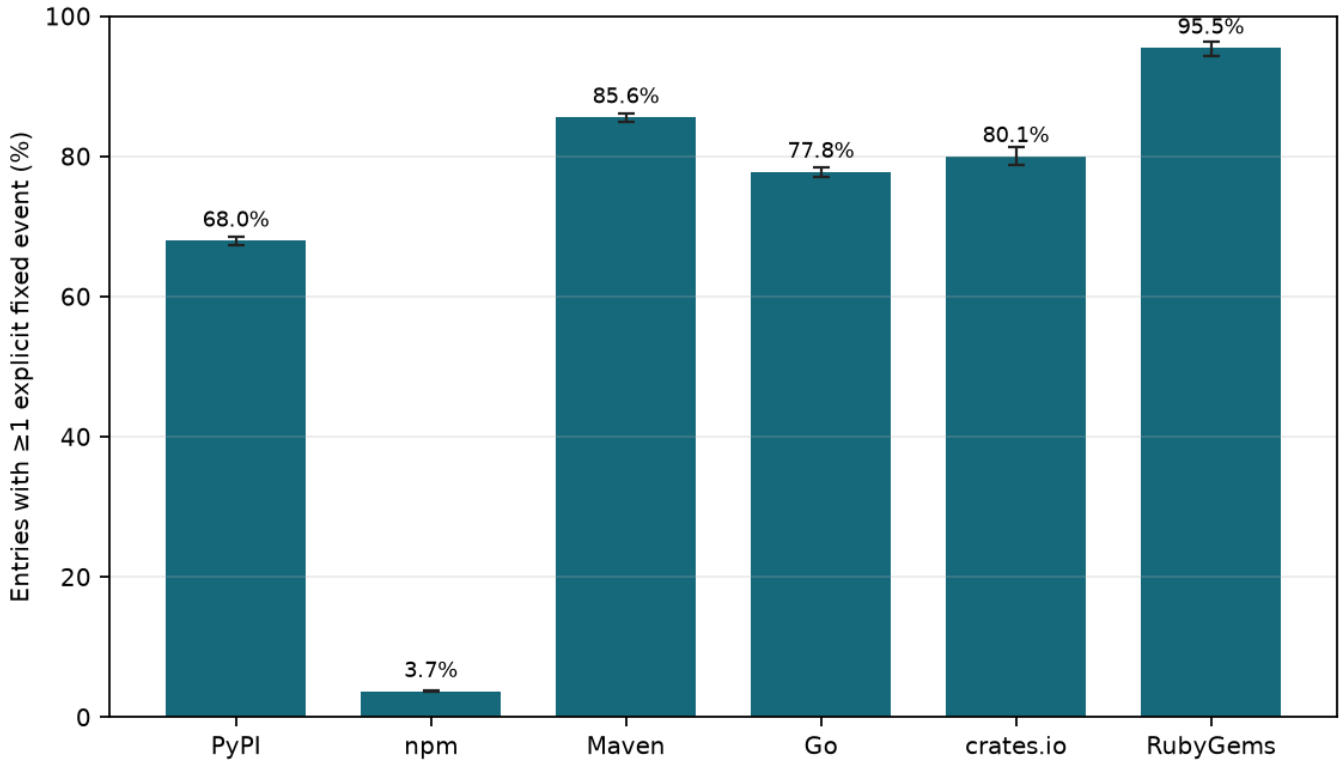

Figure 1. Explicit fixed-event prevalence by ecosystem; bars show observed proportions and descriptive Wilson intervals.

The form of no-fix evidence also varied (Figure 2). Maven's no-fix entries usually retained finite or last_affected boundaries and often enumerated versions; only eight had open ranges without enumeration. In contrast, 198,460 npm entries combined no explicit fix with an open range and no enumerated affected versions. Go, crates.io, and PyPI showed smaller but nontrivial open-range counts.

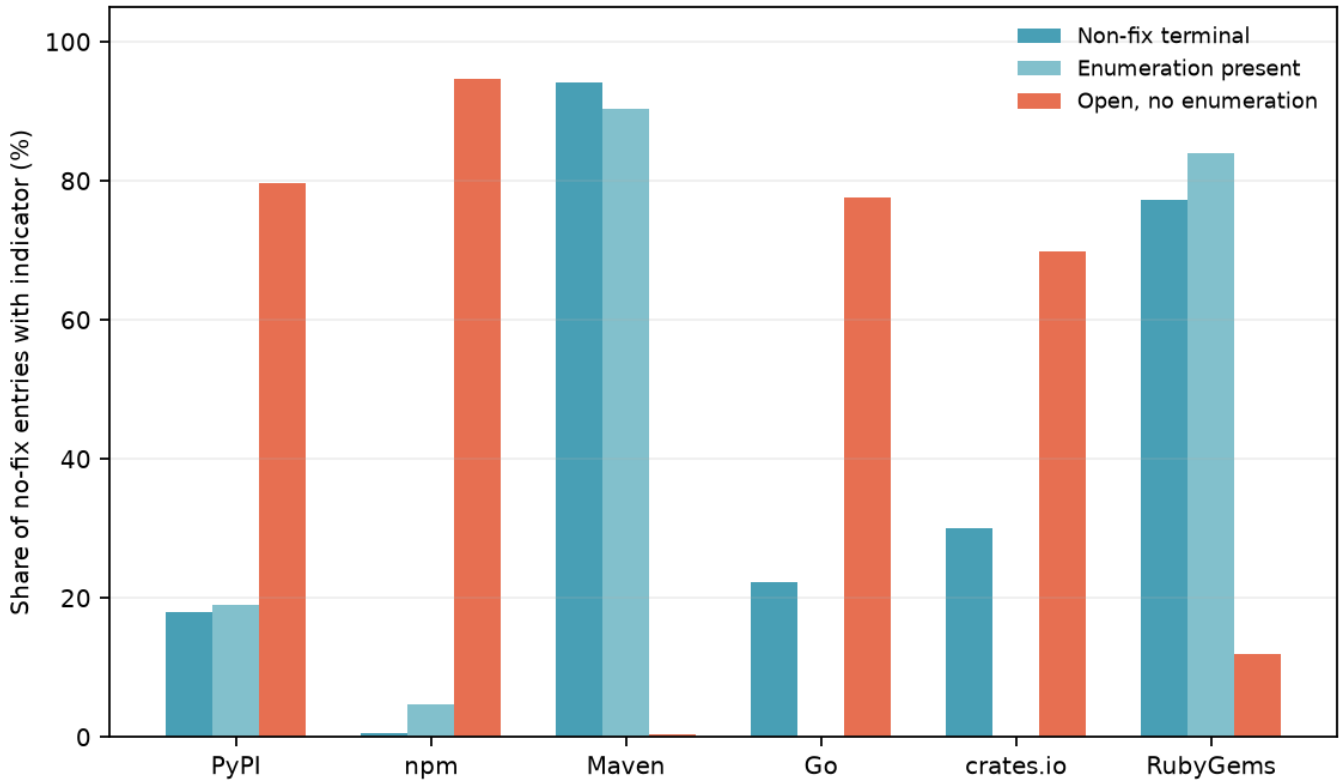

Figure 2. Prevalence of three overlapping representation indicators among no-fix entries by ecosystem. Indicators are not additive.

4.3 Source-family concentration

The aggregate result was dominated by one identifier family. MAL accounted for 214,278 of 223,165 no-fix entries (96.02%). GHSA contributed 5,282 (2.37%), GO 2,317 (1.04%), PYSEC 828 (0.37%), and RUSTSEC 460 (0.21%). Source-family HHI was 0.923, confirming extreme concentration (Figure 3).

Inspection of representative MAL records showed malicious npm package advisories with SEMVER ranges introduced at 0 and no fixed endpoint. For these records, absence of a repaired version is not surprising: removal and dependency eradication, not upgrade to a trusted release, is the relevant response. This observation changes the interpretation of the pooled 18.76% estimate without changing its arithmetic.

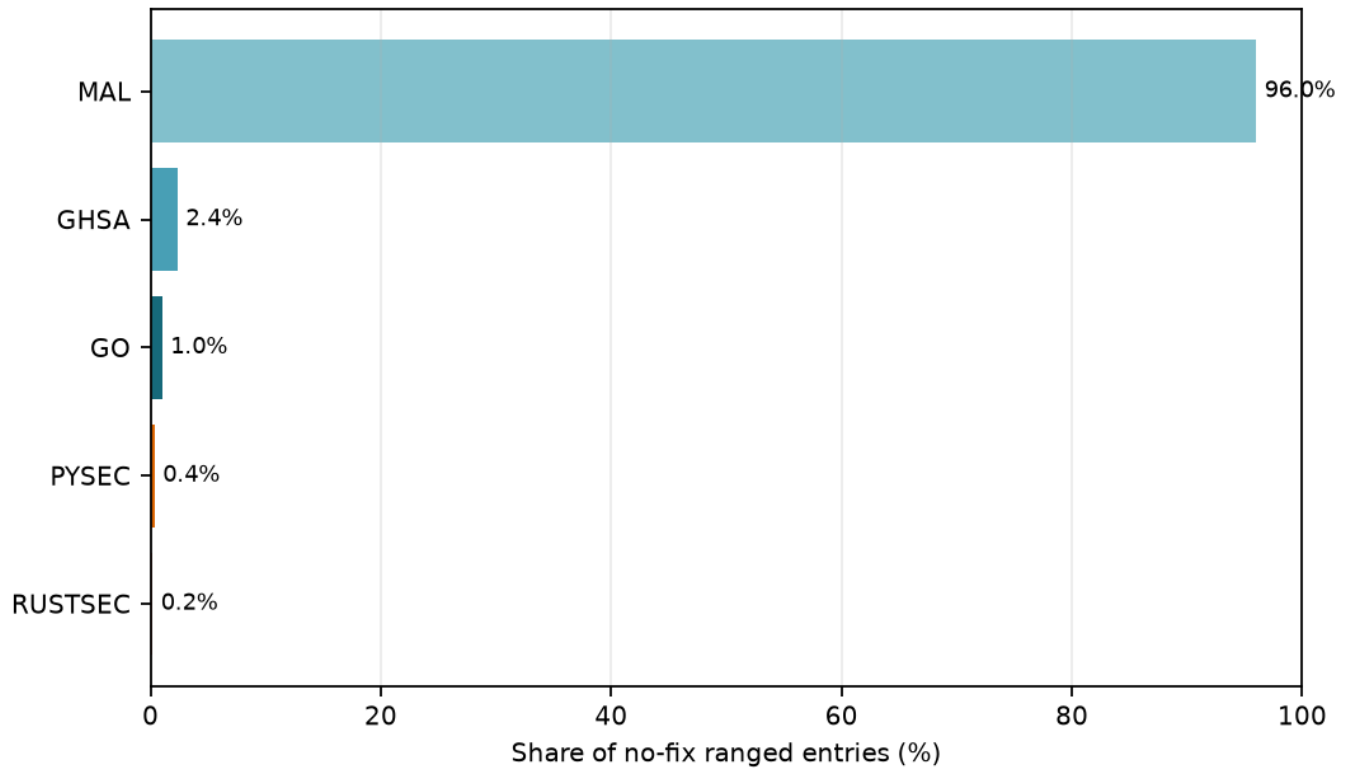

Figure 3. Source-family concentration among ranged entries without an explicit fix.

Excluding the largest family removed 214,310 ranged entries, including 32 with fixed events. Among the remaining 60,395 entries, 51,508 had an explicit fix, giving 85.29% prevalence. This is a prespecified source-composition sensitivity, not a corrected replacement for the primary result.

4.4 Sensitivity analyses

Including withdrawn records changed prevalence from 18.76% to 18.97% (52,410/276,347). No divergent IDs occurred, so deterministic divergent-record inclusion produced the primary result exactly. Excluding entries with any affected-version enumeration reduced prevalence to 10.07% (23,520/233,582), showing that entries relying most heavily on ranges were also less likely to name a fix. SEMVER-only prevalence was 9.35% (21,986/235,237). Restricting publication to years before 2026 yielded 13.68% (33,845/247,379).

Table 2. Prespecified sensitivity analyses.

Analysis	Ranged entries	With fixed	Fixed prevalence
Primary, active records	274,705	51,540	18.76%
Include withdrawn records	276,347	52,410	18.97%

Exclude entries with enumeration	233,582	23,520	10.07%
Exclude largest no-fix source family	60,395	51,508	85.29%
SEMVER-only	235,237	21,986	9.35%
Published before 2026	247,379	33,845	13.68%

Figure 4 places record classes and sensitivity estimates side by side. The shift after excluding MAL is much larger than shifts caused by withdrawal status or publication year, indicating that advisory purpose is the main organizing factor in this snapshot.

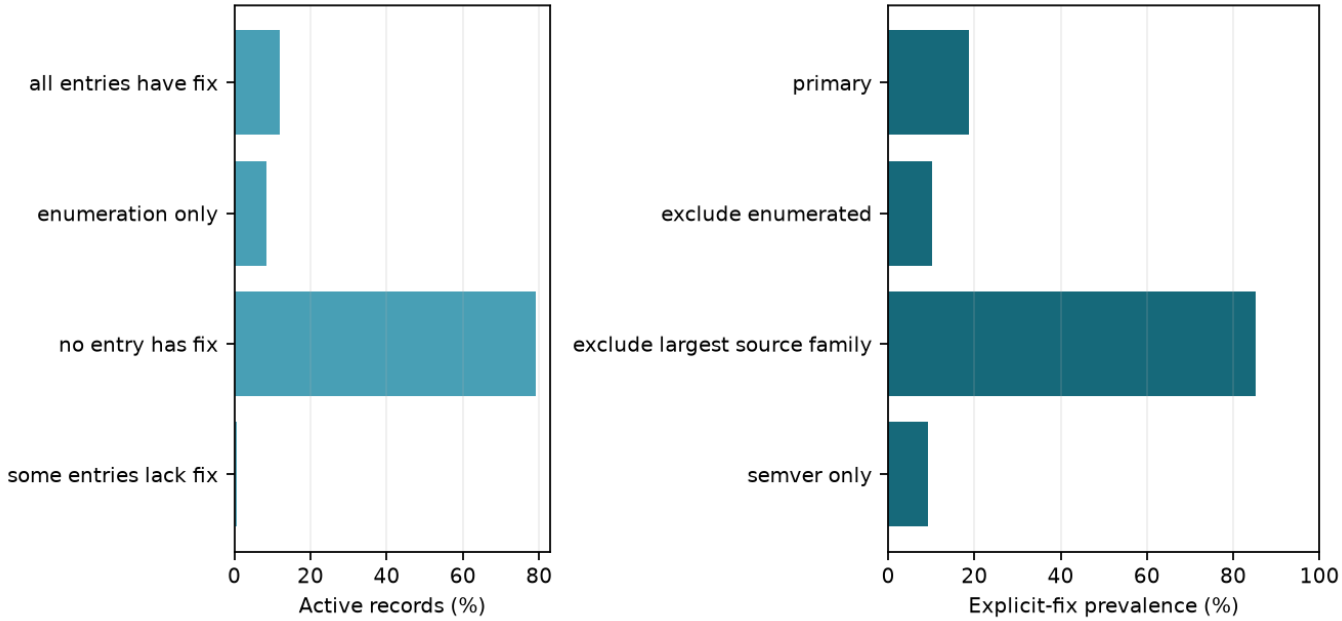

Figure 4. Record-level classes and selected explicit-fix prevalence sensitivities.

5. Discussion

5.1 Principal interpretation

The main result is not that OSV lacks remediation data in four out of five entries. The defensible result is that only 18.76% of ranged entries in the unstratified six-ecosystem snapshot exposed an explicit fixed event, and that this estimate was overwhelmingly shaped by malicious-package advisories. Once the largest source family was excluded under a prespecified sensitivity, explicit-fix prevalence was 85.29%.

This distinction illustrates a general measurement problem in security data. A field can be absent because evidence is incomplete, because the vulnerability remains unfixed, because affected versions are represented another way, or because the record describes an object that should be removed rather than repaired. A single completeness score hides those states. Source-aware reporting is therefore not optional when heterogeneous advisory classes are combined.

The ecosystem results reinforce this point. npm's 3.74% pooled prevalence reflects the volume of malicious-package records in its archive. It should not be read as evidence that npm's conventional vulnerability advisories almost never identify fixes. Conversely, high prevalence in RubyGems or Maven does not establish that their fixed versions are correct or compatible. The study audits representation, not truth.

5.2 Implications for consumers

Consumers should preserve event type and advisory source rather than reducing all endpoints to a generic upper bound. fixed, last_affected, and limit encode different claims. Treating last_affected as a known repaired version can create unsafe upgrade advice. Treating an open malicious-package range as a missing ordinary fix can produce meaningless recommendations.

Remediation engines should first classify response semantics. For a malicious package, the default action may be removal, replacement, credential review, and incident-response analysis. For a conventional defect with a fixed event, a tool may propose the first compatible fixed release, subject to dependency solving and policy. For a no-fix conventional record, automation should fail safely, state that no fixed release is represented, and avoid inventing one from the next observed version.

Exact version enumeration remains useful for exposure matching but is not remediation evidence by itself. The sensitivity excluding enumeration showed that range-dependent entries had even lower explicit-fix prevalence. Tools that operate without registry lookups or project context should disclose this limitation prominently.

5.3 Implications for data producers

For repairable vulnerabilities, explicit fixed events improve the path from detection to action. Where the first fixed release is not yet known, last_affected may accurately represent current knowledge, but records should be updated when a fixed release becomes available. Where all releases are malicious, a source-specific marker or response classification can help consumers avoid interpreting an intentionally open range as a conventional vulnerability awaiting a patch.

The OSV quality guidance emphasizes machine-readable answers at scale [3]. Endpoint completeness is one part of that objective, but quality assessment should be conditional on advisory purpose. A useful future extension would define source-aware expectations: fix-version evidence for repairable software defects, removal semantics for malicious packages, and explicit unresolved status for vulnerabilities without released repairs.

5.4 Limitations

First, the study is a snapshot. OSV archives change continuously, and records may have been updated after 1 October 2026. Exact hashes support reproduction of the frozen state but not claims about the current database.

Second, the analysis does not verify whether any fixed version actually contains the repair, whether the affected range is complete, or whether the package can be upgraded without breaking changes. Code-history methods studied in prior work [6–9] address parts of that harder validation problem.

Third, source family is inferred from the identifier prefix. MAL concentration is clear in this corpus, but prefix-based grouping is not a universal ontology. Representative inspection supports the malicious-package interpretation; a complete content-level taxonomy was outside the frozen protocol.

Fourth, entry-level outcomes weight records with multiple packages or entries more heavily than record-level outcomes. We report both levels, but neither estimates the number of affected deployments or users.

Fifth, open-without-enumeration is a representational condition, not proof that all future versions are vulnerable. It can reflect an unfixed issue, malicious-package semantics, publication timing, or incomplete evidence. Finally, descriptive Wilson intervals are included for scale; the archives are a finite census snapshot rather than a random sample.

6. Conclusion

In a frozen six-ecosystem OSV snapshot, 51,540 of 274,705 active ranged affected entries exposed an explicit fixed version. The low pooled prevalence of 18.76% was driven primarily by MAL malicious-package advisories. Excluding that prespecified largest source family raised prevalence to 85.29%. The contrast shows why remediation metadata must be interpreted through advisory purpose rather than scored as an undifferentiated completeness field.

Machine-actionable remediation requires more than a range boundary. Consumers should distinguish fix, ceiling, limit, enumeration, and open-range evidence; separate remove-and-investigate advisories from upgrade-oriented records; and refuse to invent a repaired version when none is represented. Data producers can improve automation by stating response semantics explicitly and updating conventional records when fixed releases become known.

Declarations

Funding

This research received no external funding.

Competing interests

The authors declare no competing interests.

Ethics

The study used public machine-readable vulnerability records and involved no human participants, private personal data, live-system exploitation, or interaction with vulnerable systems. Institutional review-board approval was not required.

Data and code availability

A blinded reproducibility supplement accompanies the submission. It contains the frozen protocol, source manifest and hashes, analysis code, tests, dependency lock, aggregate outputs, figures, and rerun instructions. Raw OSV archives are omitted from the compact journal supplement because of file-size constraints; their exact public URLs and SHA-256 hashes are provided.

Author Contributions

Fabrizio Pastore: conceptualization, methodology, investigation, writing – original draft. Norrathep Rattanavipanon: validation, remediation-endpoint analysis, writing – review & editing.

AI assistance declaration. AI-based tools were used in a limited supporting capacity (literature search, figure preparation, and language editing); all research content, analysis, and conclusions were produced, reviewed, and verified by the authors, who take full responsibility for the work.

References

1. OpenSSF. *Open Source Vulnerability format*. OSV Schema, checkout v1.9.1-1-g65280fa, commit 65280faa676c7a69bb9208d876b2419f88dd7e03. https://ossf.github.io/osv-schema/
2. OSV.dev. *Data sources*. https://google.github.io/osv.dev/data/
3. OSV.dev. *Properties of a High Quality OSV Record*. https://google.github.io/osv.dev/data_quality.html
4. Dong Y, Guo W, Chen Y, Xing X, Zhang Y, Wang G. Towards the Detection of Inconsistencies in Public Security Vulnerability Reports. In: *28th USENIX Security Symposium*. 2019. https://www.usenix.org/conference/usenixsecurity19/presentation/dong
5. Ren H, Li X, Lei L, et al. Detecting Inconsistent Vulnerable Software Version in Security Vulnerability Reports. *Communications in Computer and Information Science*. 2022:78–99. https://doi.org/10.1007/978-981-19-0523-0_6
6. Nguyen VH, Dashevskyi S, Massacci F. An automatic method for assessing the versions affected by a vulnerability. *Empirical Software Engineering*. 2016;21(6):2268–2297. https://doi.org/10.1007/s10664-015-9408-2
7. He Y, Wang Y, Zhu S, et al. Automatically Identifying CVE Affected Versions With Patches and Developer Logs. *IEEE Transactions on Dependable and Secure Computing*. 2024;21(2):905–919. https://doi.org/10.1109/TDSC.2023.3264567
8. Wu S, Wang R, Huang K, et al. Vision: Identifying Affected Library Versions for Open Source Software Vulnerabilities. In: *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 2024:1447–1459. https://doi.org/10.1145/3691620.3695516
9. Chen X, Liu C, Cao J, et al. Vulnerability-Affected Versions Identification: How Far Are We? In: *2025 40th IEEE/ACM International Conference on Automated Software Engineering*. 2025:2970–2982. https://doi.org/10.1109/ASE63991.2025.00244
10. Bhandari G, Naseer A, Moonen L. CVEfixes: automated collection of vulnerabilities and their fixes from open-source software. In: *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*. 2021:30–39. https://doi.org/10.1145/3475960.3475985
11. OSV-Scanner. *Guided Remediation*. https://google.github.io/osv-scanner/experimental/guided-remediation/