RESEARCH ARTICLE

Journal of Sociotechnical Inquiry
Volume 1, Issue 1, 2026, pp. 13–22

Fixed Before Filed: Security Advisories, Fix Releases and Maintainer Labour in Eight Open-Source Package Ecosystems

Milica Popović[1,*]

[1]Department of Software Engineering, University of Kragujevac, Serbia

ABSTRACT

Open-source package ecosystems are judged by the number of security advisories they attract and by the severity scores those advisories carry. This article asks what that accounting leaves out: the work of producing a fix and recording it for automated tools. It analyses all 34,849 GitHub-reviewed advisories (44,909 advisory-package pairs) in eight ecosystems from the OSV.dev export, and a random sample of 1,800 npm and PyPI pairs linked to registry release dates, download counts and OpenSSF Scorecard signals. Reviewed advisories disclosed between January and late October 2026 numbered 10,937, four times the 2,740 of the same period in 2025, and 84.7% of 2026 advisories came from maintainers' own repository advisories. In the sample, 92.1% of datable fixes were released before or within a day of disclosure. A fix was missing for 12.2% of pairs overall and for 16.9% of critical ones. Severity barely predicted a missing fix. What predicted it was the absence of a maintainer in the process: pairs not published as maintainer repository advisories were far more likely to list no fix (22.4% versus 4.8%, adjusted), as were packages with no release in the year before disclosure. Of unfixed npm and PyPI pairs, 47.5% belonged to packages that were dormant both before and after disclosure. The advisory system thus records labour it does not count: fixes arrive mostly through maintainers who are already active, and where no maintainer is present the system produces alerts without remedies.

Keywords: open-source software, software supply chain, security advisories, maintainer labour, Dependabot, vulnerability disclosure

1. Introduction

Modern software is assembled from packages that strangers publish to public registries such as npm for JavaScript and the Python Package Index (PyPI). When a vulnerability is found in one of these packages, a chain of technical systems turns it into action. An advisory database records the flaw, a severity score ranks it, and tools such as GitHub's Dependabot alert the projects that depend on the package and propose an upgrade (GitHub, 2026b). That upgrade is only possible if someone has released a fixed version. The fix is the one link in the chain that no database or bot can produce on its own.

Public discussion of supply-chain security tends to use the outputs of this chain as its measures. Ecosystems are compared by advisory counts, and individual flaws by Common Vulnerability Scoring System (CVSS) scores, a 0 to 10 rating of technical severity. These measures are useful, but they describe the vulnerability, not the work that resolves it. Studies of open-source work have long shown that much of this work is unpaid or loosely paid, unevenly spread and easy to overlook (Crowston et al., 2012; Geiger et al., 2021). Severity scores and counts say nothing about whether a maintainer existed to write the fix, publish the release and fill in the advisory fields that downstream tools read.

This article treats the advisory record itself as evidence of that work. Each GitHub-reviewed advisory lists the affected package, a severity label, and, when known, the first fixed version. Each package registry records when every version was released. Joining the two shows when a fix existed relative to public disclosure, which advisories lack a fix, and what kind of package sits behind the gap. The analysis asks three research questions.

RQ1. How have the volume, origin and severity of reviewed advisories changed across eight package ecosystems, and how often do advisories list a fixed version?

RQ2. When does a fixed release exist relative to an advisory's first public disclosure, and how long were affected versions on the registry before the fix?

RQ3. Which advisory and package characteristics are associated with an advisory that lists no fix, and what does registry activity show about those packages?

The findings point to a gap between how the system is evaluated and how it works. Severity, the measure the system foregrounds, hardly predicts whether a fix exists. Whether a maintainer took part in the advisory, and whether the package was being released at all, predicts it strongly. Most fixes are already out by the time an advisory appears, which means the visible event (the advisory and the alert) follows invisible work done earlier. Where no one is doing that work, the database still produces an advisory and the tooling still raises alerts, but there is nothing to upgrade to.

The article makes three contributions. It gives an up-to-date account of the GitHub-reviewed advisory record across eight ecosystems, including a sharp rise in maintainer-filed advisories in 2026. It joins advisory records to registry release histories to show what the common timing measures capture and what they miss. It identifies the packages behind missing fixes and separates two kinds of absence: packages that no one maintains and packages whose advisory records no one updated. Section 2

reviews research on open-source labour, vulnerability timing and automated dependency tools. Section 3 describes the data and measures, Section 4 reports the findings by research question, and Sections 5 to 7 discuss the results, their limits and their implications.

2. Related work and framing

2.1 Open-source labour and its uneven distribution

Free and open-source software (FLOSS) is built by a mix of volunteers and paid staff. A review of FLOSS research described a field in which participation is highly skewed, with a small core doing most of the work (Crowston et al., 2012). Economic accounts explained volunteer effort through signalling and career concerns (Lerner & Tirole, 2002). Later evidence showed that a large share of commits is made during working hours, which suggests that corporate employment funds much of the activity (Riehle et al., 2014). The labour is still unevenly distributed. Interviews with maintainers of widely used projects describe growing demands for support, triage and security response that scale with use rather than with resources (Geiger et al., 2021). Donations reach few projects and are modest where they arrive (Overney et al., 2020).

Projects also stop. Developers of failed projects most often cited lack of time or interest, or the arrival of a competitor (Coelho & Valente, 2017). When core developers leave, many projects are not taken over by anyone (Avelino et al., 2019). In npm, unmaintained packages and single-maintainer accounts form weak links in the supply chain (Zahan et al., 2022). Maintainers interviewed about security describe informal trust processes and few written policies (Wermke et al., 2022). These studies suggest that the capacity to fix a vulnerability depends on the presence of active people, which a severity score does not measure.

2.2 Vulnerabilities, fixes and the timing of disclosure

Empirical software engineering has measured how vulnerabilities move through package networks. In npm, vulnerabilities often took a long time to be discovered and to be fixed, and fixes propagated slowly through dependent packages (Decan et al., 2018). Python packages show similar patterns, with vulnerabilities that went undiscovered for years (Alfadel et al., 2023). Even after a fix is released, it reaches users with delay, because the release is often bundled with other changes and dependents do not upgrade (Chinthanet et al., 2021). Many projects keep outdated dependencies even after advisories are published (Kula et al., 2018), a delay that has been formalised as technical lag (Zerouali et al., 2018).

Timing studies also show that disclosure and fixing are tangled. Disclosure of a vulnerability speeds up vendors' patch releases (Arora et al., 2010). Analyses of security patches in open-source code found that vulnerabilities often lived in code for a long time before being fixed (Li & Paxson, 2017). Security fixes in open-source packages are often released quietly, with the advisory following the release (Imtiaz et al., 2023). Counts of vulnerable dependencies can also overstate exposure, because many reported cases do not matter in practice (Pashchenko et al., 2018). Each of these studies treats timestamps as measures of delay. This article follows that practice but also reads the timestamps as traces of who did the work and when.

2.3 Automation and the advisory as infrastructure

Automated tools now sit between advisories and users. Dependabot opens pull requests that upgrade a vulnerable dependency to the minimum version that includes the patch (GitHub, 2026b). Studies found that many such pull requests are merged quickly but many are ignored, and that developers report noise and fatigue (Alfadel et al., 2021; He et al., 2023). These tools read structured fields: the package name, the affected range and the first patched version. Dependabot raises alerts only for advisories that GitHub has reviewed (GitHub, 2026a). The fields that make automation possible must therefore be written by someone, either a maintainer filing a repository security advisory or a curator reviewing an imported record.

Science and technology studies offer terms for this arrangement. Classification systems make some work visible and leave other work silent (Bowker & Star, 1999). Invisible work is work that is needed but not recognised in the formal account of a system (Star & Strauss, 1999). Repair and maintenance are the ongoing work that keeps technical systems usable after their design is finished (Jackson, 2014). Commensuration, the conversion of different qualities into a common metric, makes things comparable but strips away context (Espeland & Stevens, 1998). CVSS is such a metric. Its scores do not track exploitation well (Allodi & Massacci, 2014), and its use as a risk measure has been criticised by people involved in its design (Spring et al., 2021). The analysis below asks how much of fix availability the metric captures, and how much depends instead on the presence of maintainers.

3. Data and methods

3.1 The advisory data

The study uses the GitHub Advisory Database as distributed through OSV.dev, an open vulnerability service that publishes records in the Open Source Vulnerability (OSV) format. Bulk exports for eight ecosystems were downloaded on 23 October 2026: npm, PyPI, Maven (Java), Packagist (PHP), Go, NuGet (.NET), crates.io (Rust) and RubyGems. Only records with a GitHub advisory identifier (GHSA) and the flag github_reviewed were kept, because these are the records that drive Dependabot alerts (GitHub, 2026a). Withdrawn advisories and malware reports were removed. Malware was identified by the CWE-506 weakness code or by the words malicious or malware in the summary, because malware advisories describe hostile packages, not flaws awaiting a fix. OSV records sometimes split one package's version ranges into several entries, so entries were merged by advisory and package.

The unit for most analyses is the advisory-package pair, because a fix is released per package. The final set contains 34,849 advisories, 44,909 pairs and 12,769 distinct packages. For each pair the analysis records the GitHub severity label (low, moderate, high or critical), the CVSS version 3 base score computed from the vector where present (80.6% of advisories), and whether any affected range lists a fixed version. It also records whether the advisory references a repository security advisory URL on GitHub. Such a reference means the advisory began as a maintainer-filed repository security advisory, the route by which a project's own maintainers disclose a flaw. This study calls these maintainer repository advisories and calls all

others the other route, which mainly means records imported from the National Vulnerability Database (NVD) and then reviewed by GitHub curators.

3.2 Measuring time

Timestamps in advisory data measure the paperwork, not the flaw. The GHSA publication date is when GitHub published the record, which can be years after the vulnerability became public. In 2022 GitHub published 8,586 reviewed advisories, many of them older vulnerabilities, and 15.4% of all advisories were published by GitHub more than a year after their NVD entry. To reduce this backfill bias, the study defines first public disclosure as the earlier of the GHSA publication date and the NVD publication date. This is still an upper bound on when the flaw became known, because issue trackers, commit messages and mailing lists can reveal a flaw earlier.

The fix date is the registry release date of the earliest-released version listed as fixed. The fix lag is the fix date minus the first public disclosure date, so negative values mean the fix was out before the advisory. The exposure window is the time from the release of the first affected version (or the package's first release when all earlier versions are affected) to the fix date. These measures have a known blind spot. A version on a newer release line that was never affected is not listed as fixed, so when an advisory lists only backported patch versions, the measured fix date can fall later than the first safe release.

3.3 The npm and PyPI sample

Registry release histories were fetched for a sample, because querying every package in eight registries was not feasible. The sampling frame was all reviewed pairs in npm (6,052) and PyPI (5,778) with first public disclosure between 1 January 2019 and 30 June 2026. The end date leaves at least four months of follow-up. A simple random sample of 900 pairs was drawn from each ecosystem with a fixed seed, giving 1,800 pairs in 982 distinct packages. For each package, the full release history was taken from the PyPI JSON API and the npm registry. Monthly download counts came from the npm downloads API and from pypistats.org. Where the advisory named a GitHub repository, star counts and the OpenSSF Scorecard results were taken from the deps.dev API. Scorecard is an automated audit of repository practices, and its Maintained check scores recent commit and issue activity from 0 to 10. All requests used a descriptive user agent and pauses between calls. Download counts, stars and Scorecard values are snapshots from October 2026, not values at the time of disclosure, and are treated as rough signals of present popularity and activity. Registry release dates, by contrast, are historical: the analysis counts releases in the 365 days before disclosure and any releases after it.

3.4 Analysis

RQ1 is answered with counts and shares across the full set, with 95% Wilson intervals for proportions. A logistic regression models the probability that a pair lists no fixed version, with severity, ecosystem, disclosure year (2017 to 2026), maintainer repository advisory and presence of a CVE identifier as predictors. Standard errors are clustered by package, because many packages carry several advisories (Zeileis, 2004). Average predicted probabilities translate odds ratios into percentages. For RQ2, Kaplan-Meier estimates describe the cumulative share of sampled pairs with a fixed release by time since disclosure (Kaplan & Meier, 1958). Fixes released before disclosure count as available at time zero, and pairs with no fixed version are censored at 23 October 2026. Pairs whose listed fix could not be matched to a registry release (81 of 1,553 fixed pairs) are excluded from timing analyses. For RQ3, logistic regressions on the sample add package age, any release in the 365 days before disclosure, and log downloads, with standard errors clustered by package. To describe unfixed pairs, each was assigned to one of three registry states. Released after disclosure means at least one new version appeared after the disclosure date. Active before but silent after means a release in the prior year and none since. Dormant on both sides means no release in the prior year and none since. These states describe activity, not intent, and a release after disclosure may or may not contain a fix. All models are associational and do not identify causes. The parsing and fetching code is in Python and the analysis is in R; all numbers in this article come from these scripts.

4. Findings

4.1 RQ1: Volume, origin and fix availability

Figure 1 shows reviewed advisories by year of first public disclosure. Annual counts rose from 741 in 2017 to between 3,211 and 3,475 in each year from 2022 to 2025. Then the series broke. Between 1 January and 23 October 2026, 10,937 reviewed advisories were first disclosed, compared with 2,740 in the same part of 2025, an increase by a factor of four. The rise was steepest from March to July 2026, when monthly counts ranged from 1,319 to 1,651, against a 2025 monthly range of 203 to 419. It touched every ecosystem; npm alone had 3,219 advisories in 2026 and Go had 2,070.

The surge was carried by maintainers. The share of advisories that began as maintainer repository advisories rose from 0.3% in 2018 to 18.4% in 2020, 53.7% in 2025 and 84.7% in 2026. The work was spread across many projects rather than a few: advisories with a GitHub repository came from 2,405 distinct repositories in 2026, up from 1,453 in 2025, and the ten busiest repositories accounted for 15.7% of 2026 advisories. One project published 598 advisories in 2026. The data do not say why disclosures multiplied, and the article does not speculate on the cause. They do show that the new volume reached the database through maintainers filing advisories themselves. The route matters for timing. Maintainer repository advisories were marked as reviewed at publication in 92.6% of cases, while 39.4% of other-route advisories were reviewed more than 30 days after GitHub first published them, with a median delay of 3.8 days.

Table 1 describes the full set by ecosystem. Maven, npm, Packagist and PyPI each carry more than 6,000 advisories. Shares rated high or critical range from 36.4% in NuGet to 56.0% in npm. Overall, 87.8% of pairs list a fixed version, but this ranges from 83.9% in Maven to 98.4% in NuGet. Advisories concentrate on few packages: 63.4% of the 12,769 packages have only one advisory, while the top 1% of packages account for 31.6% of pairs.

The choice of date changes the history the database appears to tell. By GitHub publication date, 2022 is a peak year with 8,586 reviewed advisories, because GitHub reviewed and published a large backlog of older NVD records that year. By first public disclosure, 2022 holds 3,463 advisories, close to its neighbours. A count of advisories by publication year would therefore record curation work at GitHub as if it were a change in the security of the ecosystems. The same caution applies to severity. The median CVSS 3 score was 6.5 across all advisories but 7.5 in npm and 5.9 in crates.io, and the share rated high or critical differs by 20 percentage points across ecosystems. These differences mix the vulnerability classes common in each language with the scoring habits of those who file and review advisories.

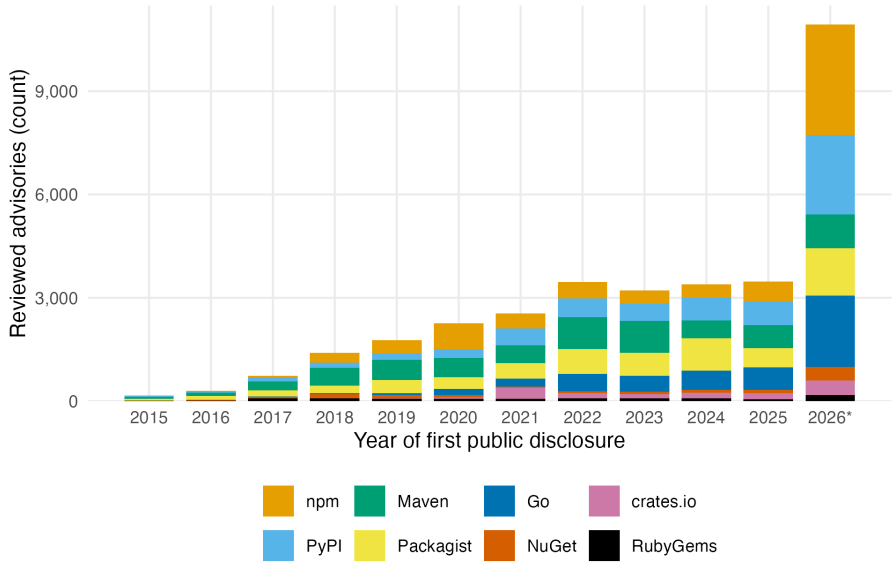


Figure 1. GitHub-reviewed advisories by year of first public disclosure and ecosystem, 2015–2026. First public disclosure is the earlier of the GHSA and NVD publication dates. *2026 covers 1 January to 23 October. Withdrawn and malware advisories are excluded; 1,193 advisories disclosed before 2015 are not shown.

Table 1. GitHub-reviewed advisories by ecosystem, all disclosure years.

Ecosystem	Advisories	Pairs	Packages	High or critical (%)	Median CVSS 3	Maintainer route (%)	Fixed version listed (%)
Maven	7,052	8,583	3,267	48.2	7.1	21.0	83.9
npm	7,018	7,626	3,176	56.0	7.5	63.9	84.3
Packagist	6,399	7,181	1,075	39.0	6.5	38.3	85.3
PyPI	6,359	7,455	1,511	48.7	6.6	58.6	90.1
Go	4,771	5,685	1,469	48.0	6.7	66.5	87.1
crates.io	1,472	2,028	1,238	39.7	5.9	63.8	88.6
NuGet	1,135	5,227	613	36.4	6.2	84.0	98.4
RubyGems	1,083	1,124	420	45.0	7.4	40.2	93.7
All	34,849	44,909	12,769	46.3	6.5	52.8	87.8

If severity reflected the attention a flaw receives, critical advisories would be the most likely to list a fix. The reverse holds in raw shares. Pairs without a fixed version made up 10.3% of low, 11.2% of moderate, 12.5% of high and 16.9% of critical pairs. Figure 2 shows that this pattern is not uniform across ecosystems. In npm, 22.4% of critical pairs and 18.6% of high pairs listed no fix, against 10.3% of moderate pairs; in Maven, low-severity pairs had the highest share without a fix (24.7%).

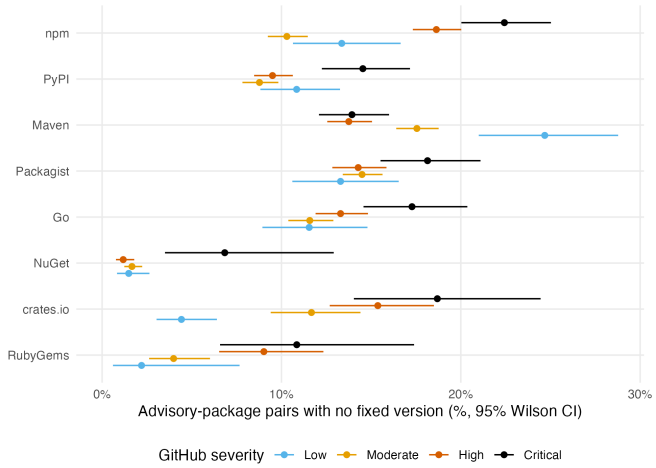


Figure 2. Share of advisory-package pairs with no fixed version, by ecosystem and GitHub severity label, with 95% Wilson confidence intervals. All disclosure years; withdrawn and malware advisories excluded.

The logistic model in Table 2 (top panel) adjusts for ecosystem, disclosure year and CVE status. After adjustment, severity has little association with a missing fix. Relative to low severity, the odds ratio for critical is 1.12 (95% CI 0.91–1.38), and the average predicted share without a fix moves only from 13.2% if every pair were low severity to 14.4% if every pair were critical. The maintainer route has a much larger association. The odds ratio is 0.17 (0.13–0.22), and the average predicted share without a fix is 4.8% for maintainer repository advisories against 22.4% for the other route. Ecosystem also matters: NuGet pairs had much lower odds of lacking a fix than npm pairs (OR 0.12, 0.08–0.18).

The year pattern fits the same account. The share of pairs without a fix stayed between 14.3% and 17.7% for disclosures from 2018 to 2025, except for 11.9% in 2021. It fell to 7.8% for 2026 disclosures, the year in which maintainers filed most advisories. The fall does not mean that the ecosystems became better at fixing. It means that a larger share of advisories arrived through the route that almost always includes a patched version.

4.2 RQ2: When fixes exist

In the sample, 1,553 of 1,800 pairs listed a fixed version, and 1,472 of those could be matched to a registry release date. Figure 3 shows the distribution of the fix lag. Most fixes came first. For 91.6% of datable pairs the fixed release preceded first public disclosure, and 92.1% were out before or within one day of it. The median fix lag was minus 4.0 days (minus 5.9 in npm and minus 2.3 in PyPI). For 39.7% of pairs the fix was released more than a week before disclosure, and for 5.4% more than a year before. Only 5.2% of fixes appeared more than 30 days after disclosure.

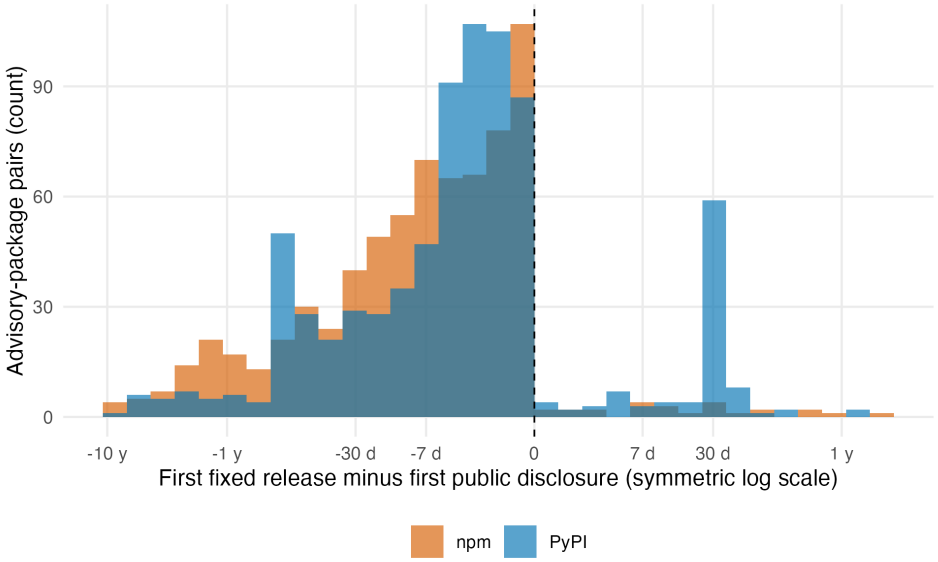


Figure 3. Distribution of the fix lag for sampled npm and PyPI pairs with a datable fixed release (n = 1,472). Negative values mean the first fixed version was released before first public disclosure. The horizontal axis uses a symmetric log scale; the dashed line marks disclosure.

The two routes produce different timing. For maintainer repository advisories the median lag was minus 2.5 days, consistent with a maintainer releasing a fix and publishing the advisory shortly afterwards. For the other route the median lag was minus 18.8 days, and 60.9% of fixes predated disclosure by more than a week. Here the advisory documents a fix that already existed, often long after the fact. The spike near plus 30 days in PyPI comes mainly from TensorFlow advisories that list backported patch versions released after disclosure. TensorFlow and its two alternative distributions account for 22.2% of PyPI pairs in the frame. Excluding them, 91.3% of PyPI fixes were out before or within a day of disclosure and 4.1% came more than 30 days after.

Severity did not change this timing much. Without TensorFlow, the median fix lag was minus 2.4 days for low, minus 4.7 for moderate, minus 5.3 for high and minus 5.3 for critical pairs. Between 88.1% and 95.4% of fixes at each level were out before or within a day of disclosure. When a fix exists, then, it arrives on roughly the same schedule whatever the score. The difference that severity labels fail to capture is whether a fix exists at all.

The exposure window was long. The median time from the first affected release to the first fixed release was 708 days

(interquartile range 181 to 2,036 days), or 843 days in npm and 592 in PyPI. For the median pair, the entire window fell before disclosure. Most of the time a vulnerable version sat on the registry, then, no advisory existed and no alert could fire.

Figure 4 turns these timings into Kaplan-Meier curves that include unfixed pairs. At disclosure, a fixed release existed for 79.8% of npm pairs and 77.1% of PyPI pairs. One year later the shares were 82.6% and 88.4%; the late PyPI rise again comes mostly from TensorFlow backports, and PyPI without TensorFlow reached 84.4%. The curves are flat after the first month. If a fix does not exist soon after disclosure, it rarely arrives later. By route, a fix existed at disclosure for 90.6% of maintainer repository advisories and for 59.0% of other-route pairs, and one year later the shares were 97.2% and 66.9%. Severity again offers little guidance: 29.2% of critical pairs still had no fix at disclosure and 24.8% still had none a year later, the highest of the four levels.

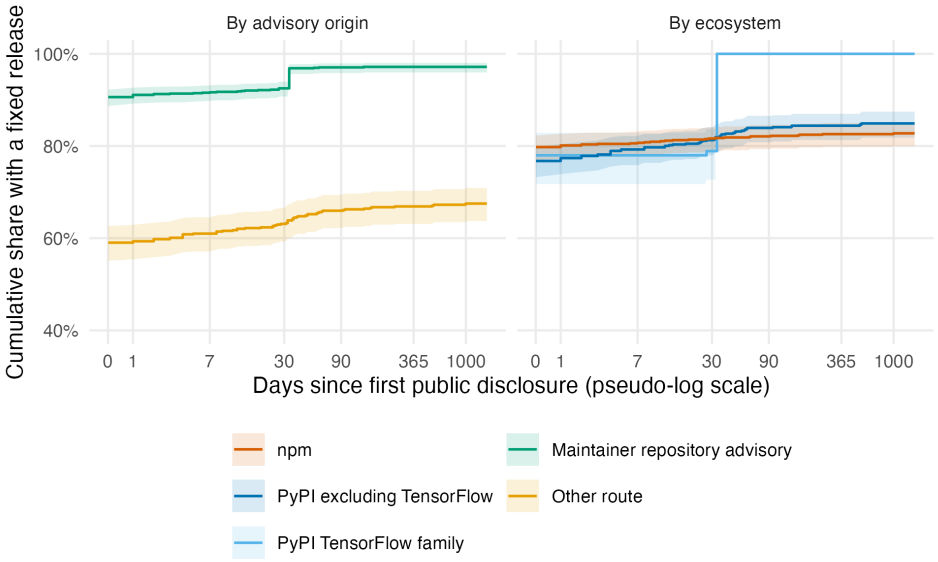


Figure 4. Kaplan-Meier estimates of the cumulative share of sampled pairs with a fixed release, by days since first public disclosure, with 95% confidence bands. Fixes released before disclosure count at day 0; pairs without a fixed version are censored on 23 October 2026. Left: by advisory origin. Right: by ecosystem, with the three TensorFlow distributions on PyPI shown separately. The horizontal axis uses a pseudo-log scale.

4.3 RQ3: Who stands behind a missing fix

The sample allows a closer look at the 247 pairs that list no fix. Compared with fixed pairs, they belong to quieter packages. Only 44.0% had any release in the 365 days before disclosure, against 96.1% of fixed pairs. The median time since the last release before disclosure was 612 days for unfixed pairs and 2.0 days for fixed pairs. Median monthly downloads in October 2026 were 4,103 for unfixed pairs and 1,082,573 for fixed pairs. Among pairs with a Scorecard result, the mean Maintained score was 4.7 for unfixed and 8.3 for fixed pairs, and 50.4% of unfixed pairs scored zero.

Table 2 (lower panels) reports the sample models. In model M1, a release in the year before disclosure is associated with much lower odds of a missing fix (OR 0.11, 0.06–0.21), as is the maintainer route (OR 0.12, 0.07–0.21). Each tenfold increase in monthly downloads is associated with lower odds (OR 0.75, 0.66–0.86). Severity (high or critical against low or moderate: OR 0.88, 0.61–1.28) and package age show no clear association. Model M3, restricted to 1,046 pairs with a Scorecard result, gives a similar picture: each point on the Maintained check lowers the odds of a missing fix (OR 0.93, 0.88–0.98). Dropping the TensorFlow distributions leaves the estimates almost unchanged (maintainer route OR 0.14; prior-year release OR 0.11).

Table 2. Logistic regression estimates of the odds that an advisory-package pair lists no fixed version.

Predictor	Full set (n = 43,046)	Sample M1 (n = 1,784)	Sample M3 (n = 1,046)
Severity: moderate (ref. low)	0.83 (0.70–0.99)		
Severity: high (ref. low)	0.99 (0.81–1.20)		
Severity: critical (ref. low)	1.12 (0.91–1.38)		
Severity: high or critical (ref. low or moderate)		0.88 (0.61–1.28)	1.10 (0.68–1.78)
Maintainer repository advisory	0.17 (0.13–0.22)	0.12 (0.07–0.21)	0.14 (0.08–0.27)
Has CVE identifier	1.27 (0.97–1.67)		
PyPI (ref. npm)	0.55 (0.42–0.72)	1.01 (0.62–1.62)	0.68 (0.40–1.17)
Any release in 365 days before disclosure		0.11 (0.06–0.21)	
log(1 + package age in years)		0.84 (0.59–1.20)	1.44 (0.86–2.41)

log10(1 + monthly downloads)	0.75 (0.66–0.86)	0.64 (0.55–0.74)
Scorecard Maintained score (0–10)		0.93 (0.88–0.98)

Registry histories show what lies behind the missing fixes, and they do not all mean the same thing. Of the 244 unfixed pairs whose package was still on the registry, 47.5% belonged to packages that were dormant on both sides, with no release in the year before disclosure and none since. Another 9.4% had released in the prior year but have released nothing since disclosure. The remaining 43.0% released at least one new version after disclosure, a median of 23 days later, yet the advisory still lists no fix. These cases differ by ecosystem: 62.4% of unfixed npm pairs were dormant on both sides, while 65.3% of unfixed PyPI pairs had a later release. A later release does not prove a fix, but it shows that a person was active and that the advisory record was not updated to say whether the new version is safe. Unfixed packages are not all obscure: 42.6% of unfixed sample pairs had at least 10,000 monthly downloads.

5. Discussion

5.1 The visible advisory follows invisible work

The advisory is the public face of supply-chain security, and it is the object that dashboards count. In npm and PyPI, the fix behind it was almost always written earlier. Nine in ten datable fixes were out before or within a day of disclosure, and for maintainer repository advisories the gap was a few days. This is the pattern of coordinated disclosure: a maintainer receives a report, writes and tests a patch, cuts a release, and only then publishes the advisory that sets the tooling in motion. In the terms of Star and Strauss (1999), the patch and release are the invisible work, and the advisory and alert are the visible arena. The 2026 surge makes the point at scale. Maintainers filed 84.7% of that year's advisories through their own repositories, so a fourfold rise in advisories also implies a large rise in reports triaged, patches written and advisory forms completed by maintainers. The data cannot show how many hours this took or who paid for them. Interview studies describe exactly this kind of growing, uncompensated demand on maintainers (Geiger et al., 2021).

The same timestamps also complicate the common reading of fix lag as maintainer slowness. Measured from disclosure, most lags are negative. Measured from the first affected release, the median exposure window was almost two years, nearly all of it before anyone filed an advisory. The long delay lies in discovery, which is consistent with earlier studies of npm and Python packages (Decan et al., 2018; Alfadel et al., 2023). Once a flaw is known and a maintainer is present, the fix usually comes quickly.

The download gradient adds a further layer. Packages with more users were much less likely to lack a fix, even after controlling for recent releases and the filing route. One reading is that heavily used packages attract paid maintainers or company sponsorship, as research on paid work in open source suggests (Riehle et al., 2014). Another is that heavy use brings more reporters and more pressure to respond. The data cannot separate these readings. Both mean that the capacity to fix is distributed by popularity, not by the severity of the flaw. Read together, the models imply that a critical flaw in a little-used, inactive package is more likely to stay unfixed than a moderate flaw in a busy one.

5.2 Severity measures the flaw, not the capacity to fix it

The journal's question is where a system is evaluated on one set of measures and experienced on another. Here the evaluation measures are advisory counts and severity scores. The experience that matters to a downstream project is whether there is a version to upgrade to. The analysis shows these two diverge. Critical advisories were, if anything, more likely to lack a fix in raw shares, and severity had no clear association with a missing fix after adjustment, in the full set or in the sample. What did predict a missing fix were signs of absent labour: no maintainer repository advisory, no release in the prior year, few downloads and a low Maintained score. CVSS commensurates flaws into a single number (Espeland & Stevens, 1998), and like other criticisms of the score (Allodi & Massacci, 2014; Spring et al., 2021), these results suggest that the number omits something users need. In this case, it is whether anyone is in a position to act.

5.3 What the tooling does when no one is there

Dependabot proposes an upgrade to the minimum patched version (GitHub, 2026b). When the advisory lists no patched version, the alert still fires, but no automated remedy follows, and the burden moves downstream to every project that depends on the package. Across the full set, 12.2% of pairs are in this state, and the sample suggests that about half of them belong to packages that no one has released for at least a year. For these, the advisory system works as designed: it records, classifies and alerts. But the repair work that Jackson (2014) describes as the ongoing condition of technical systems has no one to perform it. Earlier research has documented that unmaintained packages are a weak point in npm (Zahan et al., 2022) and that alert fatigue reduces the effect of automated pull requests (He et al., 2023). The present data link the two: unfixed advisories concentrate in inactive packages, and these produce alerts without remedies.

The other large group is different. In 43.0% of unfixed sample pairs, the package released a new version after disclosure, typically within a month. Either the release did not fix the flaw, or the fix happened and the structured fields were never updated. The data cannot tell these apart, but both outcomes point to curation work that the system needs and does not assign. Updating a fixed-version field is small work, yet without it Dependabot cannot act. For imported advisories the field is filled by GitHub curators and community contributors rather than the maintainer, and the review delay for this route (more than 30 days for 39.4% of other-route advisories) shows that curation has its own queue. Bowker and Star (1999) argued that classification systems carry the choices of those who maintain them. In the advisory database, the field that decides whether automation can help is the one most dependent on someone

remembering to fill it.

5.4 Different ecosystems, different gaps

The npm and PyPI results differ in a way that matters for any remedy. In npm, most unfixed pairs (62.4%) sat in packages that had been dormant for at least a year before disclosure and stayed dormant. Many of these are small packages with one maintainer, the kind that earlier work identified as weak links (Zahan et al., 2022). No reminder to update a field will help such a package, because there is no one to receive it. In PyPI, most unfixed pairs (65.3%) belonged to packages that released again after disclosure. There, the gap lies at least partly in the record: someone was present, but the advisory does not say whether the new release closed the flaw. NuGet stands out in the other direction, with 98.4% of pairs listing a fix and 84.0% filed by the maintainer route. The data do not identify who maintains these packages, so the reasons for this pattern remain open. The general point is that a single metric, the share of advisories with a fix, combines at least two kinds of missing work: missing maintenance and missing curation.

5.5 Implications

Three practical implications follow, each tied to a finding. First, reports that rank ecosystems by advisory counts should note that counts now largely reflect how many maintainers file advisories, since 84.7% of 2026 advisories came by that route. A rising count can mean more disclosure work, not less secure code. Second, the database and tooling could record maintenance status alongside severity. A flag for packages with no release in the past year would tell a user, at the moment of the alert, that no fix is likely to come. Third, support for maintainers, whether through funding programmes or employer time (Riehle et al., 2014; Overney et al., 2020), would be better targeted at the security response work that precedes most advisories than at the advisories themselves.

5.6 What advisory timestamps measure

The findings also carry a lesson for research that uses advisory data. Every date in an advisory record is produced by an organisational process: a maintainer's decision to publish, a curator's review, an import from NVD. Publication dates by GitHub measure curation capacity, as the 2022 backlog shows. Fix lags measured from disclosure are mostly negative because coordinated disclosure puts the release first, so a short or negative lag says more about disclosure practice than about the speed of repair. Exposure windows measured from the first affected release mostly capture how long discovery took. Studies that compare ecosystems or years on any one of these quantities should state which process the timestamp records. Otherwise a change in who files advisories, such as the 2026 shift towards maintainer filing, will look like a change in security.

6. Limitations

Several limits apply. First, the timestamps measure records, not events. The first public disclosure date is an upper bound, and the fix date is the release of the earliest listed fixed version, which can miss a newer unaffected release line, as the TensorFlow cases show. Second, the maintainer route is inferred from a reference URL, which can miss maintainers who disclose through other channels. It is also partly circular as an explanation for fix listing, since a maintainer filing an advisory usually knows the patched version. The association is therefore best read as evidence about who supplies the structured fields, not as a causal effect of the filing route. Third, download counts, stars and Scorecard values are present-day snapshots, so for older advisories they may reflect what happened after disclosure; registry release counts avoid this problem. Fourth, the sample covers only npm and PyPI, and its random design means that projects with many advisories, such as TensorFlow, carry weight in proportion to their advisories; a sensitivity check without TensorFlow left the conclusions unchanged. Fifth, the study cannot see the people behind the releases. It cannot tell paid from volunteer work, or hours spent, and its claims about labour rest on traces of activity rather than on accounts from maintainers. Finally, the 2026 surge is recent and its causes are not examined here.

7. Conclusion

This study read the GitHub Advisory Database together with npm and PyPI release histories to ask what advisory counts and severity scores leave out. For RQ1, reviewed advisories rose fourfold in 2026, and maintainers filed most of them. Twelve percent of advisory-package pairs listed no fix, and severity hardly predicted which ones. For RQ2, nine in ten datable fixes were released before or within a day of disclosure, while vulnerable versions had typically been on the registry for about two years. For RQ3, missing fixes clustered in packages without a maintainer route, without recent releases and with few users, and nearly half of unfixed npm and PyPI pairs sat in packages dormant on both sides of disclosure. The software supply chain is evaluated through the advisory and its score, but it is kept working by maintainers whose effort appears in the data only as a release date. Where that effort is absent, the database and its tools continue to record and alert, and the cost of repair passes to everyone downstream.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship and/or publication of this article.

Funding

This research received no external funding.

Ethics statement

This study is a secondary analysis of publicly available data: advisory records, package registry metadata and public repository metrics. Package and repository names are public identifiers of software projects, and no personal data about individual developers were collected or analysed. No human participants were contacted, and no ethics approval was required.

Data availability statement

All data are public. GitHub Advisory Database records were obtained from the OSV.dev bulk exports for npm, PyPI, Maven, Packagist, Go, NuGet, crates.io and RubyGems

(https://osv-vulnerabilities.storage.googleapis.com/<ecosystem>/all.zip; source database https://github.com/github/advisory-database), accessed 23 October 2026. Package release histories came from the PyPI JSON API (https://pypi.org/pypi/<package>/json) and the npm registry (https://registry.npmjs.org/), download counts from the npm downloads API (https://api.npmjs.org/downloads/) and pypistats (https://pypistats.org/api/), and repository stars and OpenSSF Scorecard results from the deps.dev API (https://api.deps.dev/v3/projects/), all accessed in October 2026. The analysis code will be deposited in a public repository on acceptance; it is withheld during review only to keep review anonymous.

References

Alfadel, M., Costa, D. E., & Shihab, E. (2023). Empirical analysis of security vulnerabilities in Python packages. Empirical Software Engineering, 28(3), Article 59. https://doi.org/10.1007/s10664-022-10278-4

Alfadel, M., Costa, D. E., Shihab, E., & Mkhallalati, M. (2021). On the use of Dependabot security pull requests. In 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR). IEEE. https://doi.org/10.1109/MSR52588.2021.00037

Allodi, L., & Massacci, F. (2014). Comparing vulnerability severity and exploits using case-control studies. ACM Transactions on Information and System Security, 17(1), 1–20. https://doi.org/10.1145/2630069

Arora, A., Krishnan, R., Telang, R., & Yang, Y. (2010). An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure. Information Systems Research, 21(1), 115–132. https://doi.org/10.1287/isre.1080.0226

Avelino, G., Constantinou, E., Valente, M. T., & Serebrenik, A. (2019). On the abandonment and survival of open source projects: An empirical investigation. In 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE. https://doi.org/10.1109/ESEM.2019.8870181

Bowker, G. C., & Star, S. L. (1999). Sorting things out: Classification and its consequences. MIT Press. https://doi.org/10.7551/mitpress/6352.001.0001

Chinthanet, B., Kula, R. G., McIntosh, S., Ishio, T., Ihara, A., & Matsumoto, K. (2021). Lags in the release, adoption, and propagation of npm vulnerability fixes. Empirical Software Engineering, 26(3), Article 47. https://doi.org/10.1007/s10664-021-09951-x

Coelho, J., & Valente, M. T. (2017). Why modern open source projects fail. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM. https://doi.org/10.1145/3106237.3106246

Crowston, K., Wei, K., Howison, J., & Wiggins, A. (2012). Free/Libre open-source software development: What we know and what we do not know. ACM Computing Surveys, 44(2), 1–35. https://doi.org/10.1145/2089125.2089127

Decan, A., Mens, T., & Constantinou, E. (2018). On the impact of security vulnerabilities in the npm package dependency network. In Proceedings of the 15th International Conference on Mining Software Repositories. ACM. https://doi.org/10.1145/3196398.3196401

Espeland, W. N., & Stevens, M. L. (1998). Commensuration as a social process. Annual Review of Sociology, 24(1), 313–343. https://doi.org/10.1146/annurev.soc.24.1.313

Geiger, R. S., Howard, D., & Irani, L. (2021). The labor of maintaining and scaling free and open-source software projects. Proceedings of the ACM on Human-Computer Interaction, 5(CSCW1), 1–28. https://doi.org/10.1145/3449249

GitHub. (2026a). About the GitHub Advisory Database. GitHub Docs. Retrieved October 23, 2026, from https://docs.github.com/en/code-security/security-advisories/working-with-global-security-advisories-from-the-github-advisory-database/about-the-github-advisory-database

GitHub. (2026b). About Dependabot security updates. GitHub Docs. Retrieved October 23, 2026, from https://docs.github.com/en/code-security/dependabot/dependabot-security-updates/about-dependabot-security-updates

He, R., He, H., Zhang, Y., & Zhou, M. (2023). Automating dependency updates in practice: An exploratory study on GitHub Dependabot. IEEE Transactions on Software Engineering, 49(8), 4004–4022. https://doi.org/10.1109/TSE.2023.3278129

Imtiaz, N., Khanom, A., & Williams, L. (2023). Open or sneaky? Fast or slow? Light or heavy?: Investigating security releases of open source packages. IEEE Transactions on Software Engineering, 49(4), 1540–1560. https://doi.org/10.1109/TSE.2022.3181010

Jackson, S. J. (2014). Rethinking repair. In T. Gillespie, P. J. Boczkowski, & K. A. Foot (Eds.), Media technologies: Essays on communication, materiality, and society (pp. 221–240). MIT Press. https://doi.org/10.7551/mitpress/9780262525374.003.0011

Kaplan, E. L., & Meier, P. (1958). Nonparametric estimation from incomplete observations. Journal of the American Statistical Association, 53(282), 457–481. https://doi.org/10.1080/01621459.1958.10501452

Kula, R. G., German, D. M., Ouni, A., Ishio, T., & Inoue, K. (2018). Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration. Empirical Software Engineering, 23(1), 384–417. https://doi.org/10.1007/s10664-017-9521-5

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. The Journal of Industrial Economics, 50(2), 197–234. https://doi.org/10.1111/1467-6451.00174

Li, F., & Paxson, V. (2017). A large-scale empirical study of security patches. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM. https://doi.org/10.1145/3133956.3134072

Overney, C., Meinicke, J., Kästner, C., & Vasilescu, B. (2020). How to not get rich: An empirical study of donations in open source. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. ACM. https://doi.org/10.1145/3377811.3380410

Pashchenko, I., Plate, H., Ponta, S. E., Sabetta, A., & Massacci, F. (2018). Vulnerable open source dependencies: Counting those that matter. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM. https://doi.org/10.1145/3239235.3268920

Riehle, D., Riemer, P., Kolassa, C., & Schmidt, M. (2014). Paid vs. volunteer work in open source. In 2014 47th Hawaii International Conference on System Sciences. IEEE. https://doi.org/10.1109/HICSS.2014.407

Spring, J., Hatleback, E., Householder, A., Manion, A., & Shick, D. (2021). Time to change the CVSS? IEEE Security & Privacy, 19(2), 74–78. https://doi.org/10.1109/MSEC.2020.3044475

Star, S. L., & Strauss, A. (1999). Layers of silence, arenas of voice: The ecology of visible and invisible work. Computer Supported Cooperative Work (CSCW), 8(1–2), 9–30. https://doi.org/10.1023/A:1008651105359

Wermke, D., Wöhler, N., Klemmer, J. H., Fourné, M., Acar, Y., & Fahl, S. (2022). Committed to trust: A qualitative study on security & trust in open source software projects. In 2022 IEEE Symposium on Security and Privacy (SP). IEEE. https://doi.org/10.1109/SP46214.2022.9833686

Zahan, N., Zimmermann, T., Godefroid, P., Murphy, B., Maddila, C., & Williams, L. (2022). What are weak links in the npm supply chain? In Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice. ACM. https://doi.org/10.1145/3510457.3513044

Zeileis, A. (2004). Econometric computing with HC and HAC covariance matrix estimators. Journal of Statistical Software, 11(10). https://doi.org/10.18637/jss.v011.i10

Zerouali, A., Constantinou, E., Mens, T., Robles, G., & González-Barahona, J. (2018). An empirical analysis of technical lag in npm package dependencies. In New opportunities for software reuse (pp. 95–110). Springer. https://doi.org/10.1007/978-3-319-90421-4_6

Corresponding author: Milica Popović, University of Kragujevac. Email: milicapopovic.dev@gmail.com

Received: 2 June 2026. Accepted: 2 August 2026. Published: 16 August 2026.

© 2026 The Author(s). This is an open access article distributed under the terms of the CC BY 4.0 licence (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution and reproduction in any medium, provided the original work is properly cited.