Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Dormant at the core: Inactivity, dependence and exit among the most downloaded npm packages

Revanth Krishna Maddula

revanth802@gmail.com

Independent Researcher https://orcid.org/0009-0006-0596-6180

Research Article

Keywords: open source software; software maintenance; software dependencies; npm; JavaScript; software ecosystems; software sustainability

Posted Date: November 5th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11614271/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Dormant at the core: Inactivity, dependence and exit among the most downloaded npm packages

Revanth Krishna Maddula[1*]

1 Independent Researcher, San Jose, California, United States of America

* revanth802@gmail.com

Abstract

Modern software is assembled from open-source packages maintained largely by volunteers, and a package that stops being maintained can leave thousands of dependent projects without bug or security fixes. We studied the 15,868 packages that receive at least one million weekly downloads on npm, the package registry of the JavaScript ecosystem, using the complete release and dependency history in the public registry up to October 2026. Of these packages, 34.7% had not published a stable release in the preceding 24 months (27.0% to 44.1% for thresholds of 36 and 12 months), and a further 4.6% had been formally deprecated by their maintainers. Dormancy was common at every level of popularity, including 26% of the 100 most downloaded packages. Dormant packages were not peripheral: 26.5% of actively maintained packages depended directly on at least one dormant package, and single dormant packages were reachable through dependency chains from more than a thousand other high-download packages. Only 3.2% of dormant packages stated in their repository README that they were no longer maintained. Two years after a reference date, dependents had removed 27.0% of their dependencies on packages that were dormant at that date, compared with 22.5% of dependencies on active packages (adjusted odds ratio 2.15); removal rose to 65.4% when the dormant package was later deprecated through npm's deprecation mechanism (odds ratio 6.50), whereas a README notice showed no detectable effect. Fifteen percent of dormant packages were later revived with a new release. Few dormant packages carried known vulnerabilities in their latest version (0.47%, against 0.06% of active packages), but those that did could no longer be fixed upstream. Dormancy is widespread and mostly silent among the most used packages; explicit deprecation through the registry is the signal that dependents act on.

Introduction

Almost all modern software depends on open-source packages, and in many ecosystems on hundreds or thousands of them. The npm registry, which serves the JavaScript and TypeScript ecosystem, is the largest such registry, and its most popular packages are downloaded billions of times per week. Much of this infrastructure is maintained by a small number of volunteers [1], and maintainers stop working on packages for many reasons: they change jobs, lose interest, burn out, or consider a package finished [2, 3]. When a widely used package stops being maintained, its dependents lose access to bug fixes, compatibility updates and security patches, and must eventually decide whether to keep the dependency, fork it or replace it [4, 5].
Previous work has studied why open-source projects are abandoned [2], how often projects are revived by new contributors [3], which ecosystem factors sustain activity [6], how developers experience and navigate dependency abandonment [4], and how npm dependents respond to abandonment [5]. Miller et al. found that 15% of widely used npm packages were abandoned between 2015 and 2020, and that only 18% of exposed active projects removed the abandoned dependency [5]. Their abandonment signals came from GitHub (repository archiving, README notices and contributor activity); they explicitly left aside npm's own deprecation mechanism, revivals and indirect exposure, and called for further study of these [5].
We complement that work with a registry-centred view of the most downloaded npm packages, as of 2026. Using only the public npm registry, we measure how many of the 15,868 packages with at least one million weekly downloads have stopped releasing, how much of the active ecosystem depends on them directly and transitively, how dependents respond when a dependency goes dormant, and whether dormant packages carry vulnerabilities that can no longer be fixed. We ask four research questions:

- **RQ1:** How common is dormancy among the most downloaded npm packages, and is it announced?
- **RQ2:** How much of the actively maintained ecosystem depends on dormant packages?
- **RQ3:** How do dependents respond when a dependency goes dormant, and how often are dormant packages revived?
- **RQ4:** Do dormant packages carry known, unfixed vulnerabilities?

Our contributions are: an ecosystem-wide measurement of dormancy among npm's most downloaded packages in 2026, with sensitivity to the threshold used; a measurement of direct and transitive dependence on dormant packages within this core; evidence that npm's deprecation mechanism, not README notices, is the signal associated with dependents removing a dormant dependency; a measurement of revival; and a replication package.

Related work

Abandonment and sustainability of open-source projects. Coelho and Valente studied why popular GitHub projects fail, finding reasons such as being superseded by competitors, obsolescence and lack of maintainer time [2]. Avelino et al. studied projects whose core developers left and found that a minority survived through new contributors [3]. Valiev et al. showed that a project's position in the PyPI dependency network is associated with whether it remains active [6]. Eghbal described how widely used open-source infrastructure depends on unpaid maintainers [1].
Dependency abandonment and deprecation in npm. Miller et al. interviewed developers about how they detect and respond to abandoned dependencies [4] and then measured abandonment and response across widely used npm packages, finding that explicit end-of-life notices were associated with faster removal [5]. Cogo et al. characterized npm's deprecation mechanism and how packages and releases are deprecated [7]. Mujahid et al. proposed a method to recommend alternatives for declining npm packages [8].
Dependency risk. Zimmermann et al. showed that a small number of npm maintainer accounts can reach a large part of the ecosystem and that unmaintained packages are a security risk [9]. Decan et al. studied how vulnerabilities propagate through npm dependencies [10], and Zerouali et al. measured technical lag in npm dependencies [11].

Materials and methods

Packages

npm considers a package high-impact if it has at least one million weekly downloads or at least 500 dependents. We used the `npm-high-impact` dataset [12] (version 1.13.0, 8 June 2026), which lists 15,916 packages qualifying by downloads, ordered by downloads, and 1,422 qualifying only by their number of dependents. We studied the packages qualifying by downloads. We excluded the dependents-only part of the list because it contains packages created in bulk to inflate dependent counts: 264 of these 1,422 packages were first published after mid-2023, many with machine-generated names, and four were flagged by npm as malicious code. On 31 October 2026 we retrieved the full registry metadata of each package. Five packages could not be retrieved and 43 had no stable release, leaving 15,868 packages.

Releases, dormancy and deprecation

For each package we recorded every version's publication time, its runtime dependencies (the `dependencies`, `optionalDependencies` and `peerDependencies` fields), its size and whether it was deprecated. We considered only stable releases, i.e. versions of the form `major.minor.patch` without pre-release labels. We classified each package at the end of the study period (30 October 2026) as *deprecated* if its latest version carried an npm deprecation message, as *dormant* if it was not deprecated and had published no stable release in the preceding 24 months, and as *active* otherwise. We repeated the classification with thresholds of 12 and

36 months. We call a package *small* if its latest version is under 10 kB unpacked or has at most five files, since small single-purpose utilities may stop releasing because they are finished rather than abandoned.

Maintenance notices

For dormant and deprecated packages hosted on GitHub, we retrieved the README of the repository's default branch (`main` or `master`) and searched its first 3,000 characters for explicit end-of-maintenance statements, such as "no longer maintained", "unmaintained", "this repository has been archived" or "looking for maintainers". A first, broader pattern produced false positives (for example, "drop support for unmaintained versions of Node"), so we restricted the search to unambiguous phrases; in a random sample of 40 matches of the final pattern, all matched phrases were end-of-maintenance statements. This measure misses notices placed elsewhere, such as repository archiving without a README change.

Dependency graph

We built a directed graph of runtime dependencies among the studied packages, using each package's latest version. We counted each package's direct dependents among the studied packages, its direct dependents that are themselves active, and its transitive dependents, i.e. the studied packages that reach it through any chain of dependencies. This graph covers only dependencies within the most downloaded packages; it does not count the millions of applications and less popular packages that also depend on them.

Response of dependents and revival

To observe how dependents respond, we fixed a reference date 24 months before the end of the study period (30 October 2024). A package was *dormant at the reference date* if it had published no stable release in the 24 months before that date, and *active at the reference date* if it had published one in the preceding 12 months. For every studied package that published at least one release after the reference date, we took the dependencies of its last release before the reference date and recorded whether each dependency on a dormant or active package was still present in its latest release. We compared removal between dormant and active dependencies with a logistic regression that controls for the number of releases the dependent published after the reference date (more releases give more opportunities to change dependencies) and for whether the dependency is small, with standard errors clustered by dependent. Among dormant dependencies, we examined whether removal was associated with the dependency being deprecated through npm by the end of the study period, carrying a README notice, or being revived with a new release after the reference date.

Vulnerabilities

We queried npm's bulk security advisory endpoint, which returns GitHub Advisory Database entries [13], for the latest version of every studied package, and recorded advisories that affect that version. For an active package such an advisory may be fixed in a coming release; for a dormant package it is unlikely to be fixed by its maintainers.

Use of AI tools

Data-collection and analysis scripts were written with the assistance of an AI model (Claude, Anthropic). The author designed the study, reviewed all scripts and outputs, and takes responsibility for the results.

Results

RQ1: Dormancy is common and mostly silent

At the end of October 2026, 9,629 of the 15,868 packages (60.7%) were active, 5,514 (34.7%) were dormant and 725 (4.6%) were deprecated (Table 1). The share of dormant packages depends on the threshold: 44.1% had not released for 12 months and 27.0% for 36 months. Fig 1 shows that the time since the last release is

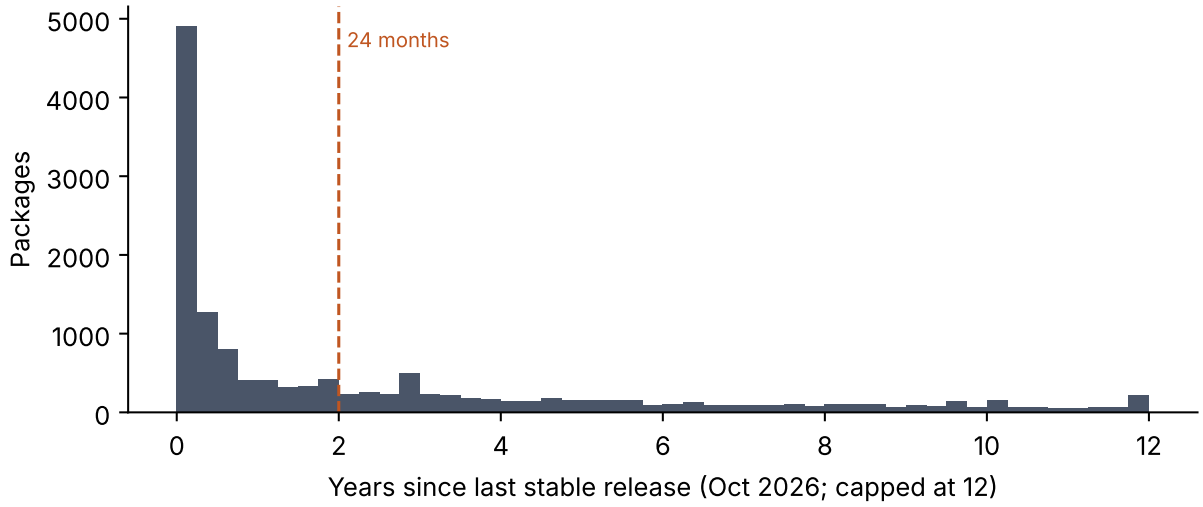


Figure 1. Time since the last stable release of active and dormant packages, October 2026. Deprecated packages are not shown. Values above 12 years are shown at 12.

Table 1. Status of the most downloaded npm packages in October 2026, by download rank. Dormant: no stable release in 24 months and not deprecated.

Download rank	Packages	Active	Dormant	Deprecated
1–100	100	74.0%	26.0%	0.0%
101–1,000	900	69.4%	29.7%	0.9%
1,001–5,000	3,991	61.5%	35.3%	3.3%
5,001–15,916	10,877	59.5%	35.1%	5.4%
All (24-month threshold)	15,868	60.7%	34.7%	4.6%
All (12-month threshold)	15,868	51.3%	44.1%	4.6%
All (36-month threshold)	15,868	68.5%	27.0%	4.6%

spread over many years rather than concentrated just beyond the threshold: the median dormant package last released 5.1 years earlier, and 1,504 dormant packages had not released for more than eight years.
Dormancy occurred at every level of popularity (Table 1). Among the 100 most downloaded packages, 26 were dormant, and dormancy rose to about 35% beyond the 1,000 most downloaded. Some dormant packages are small utilities that may simply be finished: 32.7% of dormant packages were small, compared with 21.9% of active ones. Most dormant packages, however, were not small.
Dormancy was rarely announced. Of the 5,209 dormant packages whose repository README we could retrieve, 166 (3.2%) stated that the project was no longer maintained, archived or looking for maintainers. Even among deprecated packages the README rarely said so (68 of 559, 12.2%), although the deprecation message itself is shown to everyone who installs the package. Of the 725 deprecation messages, 46.3% named an alternative, such as a renamed or successor package.

RQ2: The active ecosystem depends on dormant packages

Dormant packages were embedded in the dependency graph of the most downloaded packages. Of the 5,514 dormant packages, 2,132 (38.7%) were direct dependencies of at least one active studied package. Of the 37,452 direct dependencies that active studied packages declared on other studied packages, 5,873 (15.7%) pointed to dormant packages and 143 (0.4%) to deprecated ones. As a result, 2,549 of the 9,629 active packages (26.5%) depended directly on at least one dormant package.
Through transitive dependencies, single dormant packages reached a large part of the studied set (Table 2).

Table 2. Dormant packages with the largest transitive reach among the 15,868 studied packages. Direct: studied packages that depend on it directly (active ones in parentheses). Transitive: studied packages that reach it through any dependency chain.

Package	Download rank	Last release	Direct (active)	Transitive
`ms`	8	Dec 2020	27 (24)	2,135
`picocolors`	66	Oct 2024	76 (69)	1,594
`escalade`	107	Aug 2024	6 (5)	1,401
`es-errors`	123	Feb 2024	45 (38)	1,310
`function-bind`	105	Oct 2023	6 (5)	1,309
`graceful-fs`	108	Mar 2023	73 (49)	1,245
`@jridgewell/resolve-uri`	130	Feb 2024	1 (1)	1,158
`safe-buffer`	37	May 2020	43 (19)	1,108
`is-extglob`	128	Dec 2016	2 (0)	1,070
`is-glob`	97	Sep 2021	20 (15)	1,069

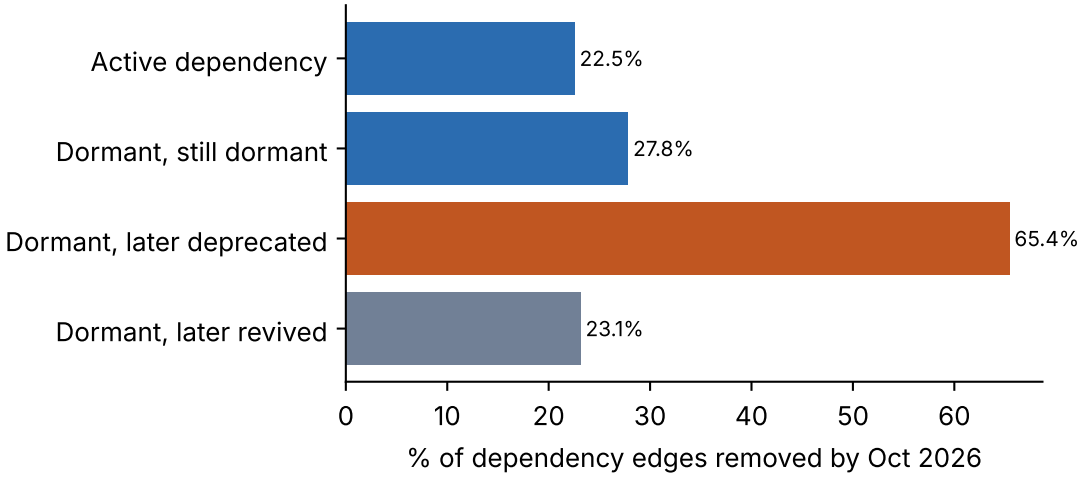


Figure 2. Share of dependencies removed between October 2024 and October 2026, by the state of the dependency in October 2024 and what happened to it afterwards. Dependencies declared by studied packages that released at least once after October 2024.

Eleven dormant packages were each reachable from more than a thousand of the 15,868 studied packages. Many of these are small, stable utilities, such as `ms` (time-string parsing), `safe-buffer` or `is-glob`, for which the absence of releases may reflect completion rather than neglect; others, such as `graceful-fs`, are substantial modules on which much of the Node.js tooling depends.

RQ3: Dependents respond slowly unless the package is deprecated

At the reference date of 30 October 2024, 4,396 studied packages were dormant. Studied packages that released again afterwards had 5,699 dependencies on these dormant packages (from 2,424 dependents) and 28,998 dependencies on active packages (from 6,265 dependents). By October 2026, dependents had removed 27.0% of their dependencies on dormant packages and 22.5% of those on active packages (Fig 2). Controlling for the number of releases each dependent published and for package size, a dependency on a dormant package was more likely to be removed (odds ratio 2.15, 95% CI 1.92–2.42; Table 3). Most dependencies on dormant packages, however, remained in place two years later.

What happened to the dormant package mattered. When a package dormant at the reference date was deprecated through npm by the end of the study period, dependents had removed 65.4% of their

Table 3. Logistic regressions for whether a dependency present in October 2024 was removed by October 2026. Standard errors clustered by dependent package.

Predictor	Odds ratio	95% CI	p
Model 1: all dependencies on dormant or active packages (n = 34,697)			
Dependency dormant in Oct 2024	2.15	1.92–2.42	<.001
Releases of dependent after Oct 2024 ($\log_2$)	1.47	1.42–1.53	<.001
Dependency is small	1.03	0.92–1.16	.582
Model 2: dependencies on packages dormant in Oct 2024 (n = 5,699)			
Deprecated through npm by Oct 2026	6.50	4.07–10.36	<.001
README end-of-maintenance notice	0.86	0.41–1.83	.698
Revived with a release after Oct 2024	0.78	0.67–0.90	<.001
Releases of dependent after Oct 2024 ($\log_2$)	1.31	1.25–1.38	<.001
Dependency is small	0.78	0.66–0.91	.002

Table 4. Known advisories affecting the latest version, October 2026.

Status	Packages	With advisory	High or critical
Active	9,629	6 (0.06%)	3
Dormant	5,514	26 (0.47%)	17
Deprecated	725	10 (1.38%)	6

dependencies on it, compared with 27.8% for packages that stayed dormant without deprecation (odds ratio 6.50, 95% CI 4.07–10.36). A README notice of end of maintenance showed no detectable association with removal (odds ratio 0.86, 95% CI 0.41–1.83), although only 45 dependencies concerned such packages. Dependencies on small packages were less often removed (odds ratio 0.78), consistent with finished utilities being kept deliberately.

Dormancy was not always permanent. Of the 4,396 packages dormant at the reference date, 658 (15.0%) published a new stable release in the following two years, and 277 (6.3%) were deprecated. Dependents of revived packages removed them less often (23.1%) than dependents of packages that stayed dormant.

RQ4: Few, but unfixable, vulnerabilities

Known vulnerabilities in the latest version were rare overall but more common among dormant and deprecated packages (Table 4): 26 dormant packages (0.47%), 17 of them with high or critical severity, against 6 active packages (0.06%). Because dormant packages are unlikely to release a fix, dependents can only remove, replace or patch them themselves. Most vulnerable dormant packages had few dependents among the studied packages; together they were reachable from 459 studied packages, mostly through `deepmerge`, whose advisory is of low severity. Our advisory data cover only published advisories in the latest version, so unreported problems in dormant code are not counted.

Discussion

Dormancy is normal, and mostly silent. One in three of npm's most downloaded packages had not released in two years, including a quarter of the top 100. Only about one in thirty dormant packages said so in its README. For dependents, dormancy is therefore something to infer from release dates, not something they are told. Our estimate is higher than the 15% abandonment Miller et al. reported for 2015–2020 [5], mainly because our definition is broader: we count every package that stopped releasing, including finished utilities, whereas they required evidence of prior sustained activity and its end. The two measures answer different questions: theirs, how often maintainers walk away from active projects; ours, how much of the code people install is no longer being updated.

Finished or abandoned? Many of the most depended-on dormant packages are small utilities that do one

thing and may never need another release. For them, dormancy is not a defect. The problem is that dependents cannot tell a finished package from an abandoned one. A simple, explicit statement in the registry, such as a "complete" or "maintenance-only" status, would let maintainers distinguish the two, and let dependents and tools treat them differently.

Deprecation is the signal that works. Dependents removed deprecated dormant packages at more than twice the rate of silently dormant ones, while README notices showed no detectable effect. This complements Miller et al., who found that explicit end-of-life notices on GitHub were associated with faster removal [5] but did not study npm's deprecation mechanism. A plausible explanation is that npm deprecation is shown to every developer at install time, whereas a README notice is seen only by those who visit the repository. For maintainers who stop maintaining a package, the most effective step is therefore to deprecate it in the registry, ideally naming an alternative, as fewer than half of deprecation messages currently do.

Implications. For maintainers: deprecate packages you no longer maintain and name an alternative; if a package is finished rather than abandoned, say so. For dependents and tool builders: track the time since a dependency's last release and its transitive reach, and prioritize replacing dormant dependencies that are large, central or affected by advisories. For the registry: surface dormancy and deprecation prominently, and consider a way for maintainers to mark packages as complete.

Limitations

We infer dormancy from the absence of stable releases; a package may still be maintained in its repository without releasing, and a finished package is not abandoned. We report results at three thresholds and separate small packages to address this. Our README check covers only the default branch's README and misses other notices, such as repository archiving. Our dependency graph includes only dependencies among the most downloaded packages and so understates exposure; it also uses declared dependencies, not those actually installed. The response analysis is observational and cannot establish that dormancy or deprecation caused removal. The studied list reflects popularity in June 2026, which favors packages that are popular now. Our results concern npm and may differ in other ecosystems.

Conclusion

Among npm's most downloaded packages, one in three has not released for two years, few say so, and a quarter of actively maintained packages depend directly on at least one of them. Dependents rarely remove such dependencies unless the package is deprecated through the registry, which is the one signal that reliably reaches them. Making dormancy visible, and making deprecation the norm when maintenance ends, would help the ecosystem respond before a dormant package becomes a problem.

Supporting information

S1 Data and code. Package-level dataset, dependency-response dataset, README and advisory results, and all collection and analysis scripts.

Acknowledgments

None.

References

[1] Eghbal N. Roads and bridges: The unseen labor behind our digital infrastructure. Ford Foundation; 2016.

[2] Coelho J, Valente MT. Why modern open source projects fail. In: Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE). ACM; 2017. p. 186–196.

[3] Avelino G, Constantinou E, Valente MT, Serebrenik A. On the abandonment and survival of open source projects: An empirical investigation. In: Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE; 2019. p. 1–12.

[4] Miller C, Kästner C, Vasilescu B. "We feel like we're winging it": A study on navigating open-source dependency abandonment. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE). ACM; 2023.

[5] Miller C, Jahanshahi M, Mockus A, Vasilescu B, Kästner C. Understanding the response to open-source dependency abandonment in the npm ecosystem. In: Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE). 2025.

[6] Valiev M, Vasilescu B, Herbsleb J. Ecosystem-level determinants of sustained activity in open-source projects: A case study of the PyPI ecosystem. In: Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE). ACM; 2018. p. 644–655.

[7] Côgo FR, Oliva GA, Hassan AE. Deprecation of packages and releases in software ecosystems: A case study on npm. IEEE Transactions on Software Engineering. 2022;48(7):2208–2223.

[8] Mujahid S, Costa DE, Abdalkareem R, Shihab E. Where to go now? Finding alternatives for declining packages in the npm ecosystem. In: Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2023.

[9] Zimmermann M, Staicu CA, Tenny C, Pradel M. Small world with high risks: A study of security threats in the npm ecosystem. In: Proceedings of the 28th USENIX Security Symposium. 2019. p. 995–1010.

[10] Decan A, Mens T, Constantinou E. On the impact of security vulnerabilities in the npm package dependency network. In: Proceedings of the 15th International Conference on Mining Software Repositories (MSR). ACM; 2018. p. 181–191.

[11] Zerouali A, Constantinou E, Mens T, Robles G, González-Barahona J. An empirical analysis of technical lag in npm package dependencies. In: New Opportunities for Software Reuse (ICSR 2018). Lecture Notes in Computer Science, vol. 10826. Springer; 2018. p. 95–110.

[12] Wormer T. npm-high-impact: The high-impact (popular) packages of npm. Version 1.13.0; 2026. Available from: `https://github.com/wooorm/npm-high-impact`.

[13] GitHub. GitHub Advisory Database. Available from: `https://github.com/advisories`. Accessed 31 October 2026.