A census of 4,296,340 npm packages: 84.9% list one maintainer and 1.74% run code when you install them

Every name in the replication index fetched over 48 hours at 25 requests a second, counted in one streaming pass, with a second pass that separates the install hooks that fire from the one that does not

Dmitriy Semenkevich · ORCID 0009-0009-3013-4978

21 August 2026 · dimhold.by

ABSTRACT

Shares quoted about the npm registry usually come from a sample or from the popular end of it. This is the whole registry on a fixed date. The replication index listed 4,305,887 names on 19 August 2026 and the registry returned data for 4,296,340 of them; the rest are names with no package behind them any more. Every share is over the packages that answered. 84.9% list exactly one maintainer, 4.0% list none, 65.8% have not been published to in 2 years and 32.8% have not in 5. 37.4% carry exactly one version and 38.5% declare no dependencies at all. The newest version of some package was published by 1,056,758 distinct accounts, of which the 2 largest are the registry itself and a CI identity. A second pass then separates the lifecycle scripts that run for somebody installing a published tarball from the one that does not. Counting all 4 gives 273,278 packages and 6.36%, which is the figure a grep over manifests returns. Counting only the 3 that actually fire gives 74,664 packages and 1.74%. The gap is entirely the prepare script, declared by 198,614 packages that never run it on an installer's machine. The error is in the definition, so a larger sample would have agreed more closely and stayed just as wrong.

1. The question

Numbers about npm circulate without a denominator. A share is quoted for the popular packages or for a convenience sample or for whatever a search returned that week and then it is read as a statement about the registry. The registry is a specific finite thing and it can be counted.

So this counts it. Every name in the replication index, fetched from the registry itself, on a date that is written down. The point of the exercise is not any single share but the denominator underneath all of them.

One result came out of that and was not expected. Counting all 4 npm lifecycle scripts gives 6.36% of the registry. Counting only the 3 that run on the machine of somebody installing a published tarball gives 1.74%. The gap is a factor of 3.66 and it is a definition rather than a sample, so no sample size closes it. Which of the 2 figures circulates more is not something this study established and no claim about that is made here: the only practitioner measurement section 6 found already counts the 3 correct hooks.

2. Method

2.1 The frame

The population is every name `replicate.npmjs.com` lists, paged in full before the crawl starts and written to a file. That file held 4,305,887 names on 19 August 2026.

The registry answered with a package for 4,296,340 of them. 9,546 names are in the index with nothing behind them any more, which is HTTP 404. Exactly 1 name never produced an answer after 5 attempts and because the crawler counted the failure without recording which name it was, that package is unmeasured and unnamed.

Every share below is over the 4,296,340 packages that returned data. Both denominators are printed in the artifact so the other one can be used.

2.2 The crawl

One request per name, 25 requests a second, holding to that rate on purpose because the registry is a public service and this was a long visit. The run started at 2026-08-19T09:15:07Z and finished at 2026-08-21T09:15:31Z, averaging 24.9 packages a second and writing 20.65 GB of slim records. The registry asked the crawler to slow down 5 times and it did. Byte figures in this paper are decimal: 20.65 GB is 20,652,163,999 bytes over 10 to the 9th, not a power of 2 quantity.

The crawler is resumable, which matters at this length: the output file is the state, so killing the process costs nothing but the request in flight. Ages are measured against the moment the crawl finished.

Counting is a separate step, one streaming pass over the crawl output, 278 seconds, which writes raw counters and nothing else. Percentages live in a third file that divides those counters. That split exists so a share can never drift away from the count it came from.

2.3 Definitions that decide the numbers

- **Newest version** is the version with the latest date in the `time` map, not `dist-tags.latest`. On a package that has not moved in 5 years they agree. On a live package publishing prereleases they do not and this counts the prerelease.
- **Maintainers** is the length of the `maintainers` array on the newest version. That array is the registry's answer to who is allowed to publish.
- **Dependencies** means `dependencies` only. No `devDependencies`, no `peerDependencies`, no `optionalDependencies`.
- **Publisher** is the account that published the newest version.
- A package with no versions at all is counted in the denominator and skipped everywhere else. There are 1,933 of those, 0.04%.

That skip splits the denominators of the 3 means quoted below. The version count is accumulated before it, so 16.5 versions per package divides by all 4,296,340. The dependency and maintainer sums are accumulated after it, so 4.57 and 1.46 divide 4,294,407 packages' worth of data by 4,296,340. The difference is 0.045% and it does not reach the second decimal of either mean, which is why the figures stand as printed rather than being recomputed.

2.4 The second pass over install hooks

The crawler stored the presence of a lifecycle script as a single bit across all 4 of `preinstall`, `install`, `postinstall` and `prepare`. That bit answers the wrong question, so every package carrying it in any version was fetched again by name and its newest version read for which of the 4 it declares. 323,536 candidates came back on 21

August 2026 with no errors. That is the number every split below is taken over and it is the `candidatesScanned` field of `hooks-full.json`. The pass's own state file is not published, so how many names it asked for is not checkable from the artifact and is not quoted.

The reason for the second pass is a semantic claim that should not be taken from documentation, so it was run instead. A probe package declaring all 4 hooks, each printing a line, on node 22.17.0 and npm 10.9.2:

HOW IT IS INSTALLED	PREINSTALL	INSTALL	POSTINSTALL	PREPARE
tarball from the registry	yes	yes	yes	**no**
`npm install` inside the package itself	yes	yes	yes	yes
as a git dependency	yes	yes	yes	yes, first

`--foreground-scripts` is required to see any of this. Without it npm swallows hook output and the run looks like nothing fired.

Row 1 is an ordinary install. Rows 2 and 3 are building the package rather than consuming it. So `prepare` does not run on the machine of somebody installing a published package and counting it as though it did is what inflates the usual figure.

3. Results

84.9% of npm packages list exactly one maintainer, 65.8% have not been published to in 2 years and 1.74% run code when they are installed rather than the 6.36% a grep over manifests reports.

3.1 Maintainers on the newest version

MAINTAINERS	SHARE	PACKAGES
0	4.0%	173,979
1	**84.9%**	3,647,096
2	4.3%	184,906
3	1.9%	82,315
4	1.4%	61,197
5	0.8%	35,894
6 or more	2.5%	109,020

The mean is 1.46 and the tail is long: exactly 1 package lists 830 maintainers.

The 4.0% with no maintainer at all is a stranger category than it sounds. 135,533 of those 173,979 packages, 77.9%, had their newest version published by the `npm` account. That is the registry doing housekeeping, replacing a removed package with a `0.0.1-security` stub under its own name and an empty maintainer list. Reading that bucket as 173,979 abandoned packages would be wrong.

3.2 Time since the last publish

LAST PUBLISH	SHARE	PACKAGES
under a month	6.2%	267,991
1 to 6 months	11.2%	480,028
6 to 12 months	7.6%	325,961
1 to 2 years	9.2%	394,839
2 to 5 years	32.9%	1,414,844
over 5 years	**32.8%**	1,410,744

2,825,588 packages, 65.8%, have not been published to in 2 years or more. Packages touched in the last month are 6.2% of the registry.

3.3 Versions and dependencies

37.4% of packages carry exactly one version, 1,608,047 of them, published once and never updated. The mean is 16.5 versions per package and it describes nothing, because the distribution is split between packages published once with nothing in them and a much smaller set with real release histories: 100,954 packages have over 100 versions.

38.5% declare no dependencies at all, 1,653,429 packages. The mean is 4.57 direct dependencies. 66.6% of the registry declares no more than 2.

DIRECT DEPENDENCIES	SHARE	PACKAGES
none	**38.5%**	1,653,429
1 to 2	28.1%	1,209,287
3 to 5	15.9%	681,039
6 to 10	9.5%	407,816
11 to 25	5.7%	246,255
over 25	2.2%	96,581

3.4 Who publishes

1,056,758 distinct accounts published the newest version of at least one package. The 2 largest are not people.

PACKAGES	ACCOUNT
150,784	`npm`
147,917	`GitHub Actions`
30,365	`terryfei`
11,403	`types`
10,923	`rajhsinggg`

3.5 Install hooks: 74,664 packages, 1.74%

74,664 packages, 1.74% of the registry, run code on the machine of whoever installs them. That is `preinstall`, `install` or `postinstall` declared in the newest version.

	PACKAGES	% OF REGISTRY
runs code on install	**74,664**	**1.74**
postinstall	49,813	1.16
install	17,692	0.41
preinstall	9,432	0.22
declares `prepare` only	198,614	4.62
had a hook once, not in the newest version	50,258	1.17

The 3 hook rows sum to more than 74,664 because a package can declare several.

Counting all 4 lifecycle scripts gives 273,278 packages and 6.36%. That is what a grep over manifests returns and it is 3.66 times the number of packages that will actually execute anything. The whole difference is `prepare`, declared and nothing else by 198,614 packages, which is 61.4% of the 323,536 candidates the first pass flagged.

`prepare` is not dead code. It fires when the package is built from source and it fires when the package arrives as a git, file or link dependency, which row 3 of the probe table shows.

So those 198,614 are packages whose `prepare` does not run for somebody pulling a published tarball, the ordinary case, rather than packages whose `prepare` never runs.

A fourth counter sits in the same file and is outside the table because it answers a different question. 101,811 packages, 2.37% of the registry, declare `preinstall`, `install` or `postinstall` in any version, not only in the newest one. For a reader who cares what an old pin can still execute, that is the honest figure and it is 1.4 times the 74,664 above.

3.6 A sample of 100,000 was wrong by at most 0.16 points

Before the census a uniform sample of 100,000 names was drawn with seed 20260819 and crawled; 99,751 of them answered. Both files were aggregated by the same script against the same reference date, so what separates the 2 columns is sampling error alone.

METRIC	FULL CENSUS	SAMPLE OF 100,000	DIFFERENCE
exactly one maintainer	84.89%	84.73%	-0.16 pp
no maintainer listed	4.05%	4.10%	+0.05 pp
2 or more maintainers	11.02%	11.13%	+0.11 pp
last publish 2 years ago or more	65.77%	65.68%	-0.09 pp
last publish 5 years ago or more	32.84%	32.91%	+0.08 pp
declares any of the 4 hooks	6.38%	6.29%	-0.09 pp
no versions at all	0.04%	0.04%	-0.00 pp

Line 6 of that table is the crawl's single bit, counted in the same pass as everything else in it. The second pass of section 2.4 counts the same definition over the same registry and gets 6.36% rather than 6.38%, because it re-read every candidate 2 days later and packages publish in between. That is 820 packages out of 4,296,340. Every split by which script comes from the second pass, because the crawl bit cannot be split.

The worst error across every metric is 0.16 percentage points. A uniform sample of 99,751 described a registry of 4,296,340 to within a fifth of a point. That is the textbook result and

it is worth having in hand rather than assuming, because it says the 48 hour crawl was not needed to obtain these shares, only to prove them.

4. The error is in the definition, where sample size has no purchase

Line 6 of the table above is the one to read twice. The sample said 6.29% of packages declare an install hook, the census said 6.38% and the agreement between them is real. The number is still the wrong answer to the question it gets asked. Split by which script, over the second pass that is the only measurement able to split it, 6.36% of the registry is 1.74% that runs for an installer plus 4.62% that declares `prepare` and nothing else. A larger sample would have agreed more closely and stayed exactly as wrong.

Those 2 shares for one definition, 6.38% from the crawl and 6.36% from the second pass, are quoted separately on purpose. Subtracting the `prepare` share of one measurement from the total of the other would produce a corrected figure that belongs to neither, so the correction is taken inside the second pass and the crawl figure is used only where it is compared with the sample.

That is the general shape worth carrying away from a census: sampling error is the failure mode people check for and a definition that quietly answers a neighbouring question is the failure mode that survives every check for it.

The corrected figure changes the argument in both directions. The attack surface that executes code at install time is roughly a quarter of what counting all 4 lifecycle scripts gives, so `--ignore-scripts` breaks fewer projects than a manifest grep implies. It also becomes concrete: 74,664 names fit in a file that a person can read, which an ecosystem sized abstraction does not. npm v12 draws its default allowlist boundary around exactly these 3 scripts, according to a Semgrep post of 11 June 2026, so the definition used here is the one that decides who has to write an allowlist entry.

The maintainer figure needs the same care applied to it. **A maintainer here is a publishing account, not a person.** The array is an access control list. An account can be a bot, a CI identity, a shared company login or a human and the top of the publisher table settles the point: the first 2 rows are the registry itself and `GitHub Actions`. So

84.9% means one account holds the publish right on 5 packages out of 6 and whether a person stands behind that account is a question this data cannot answer.

5. Threats to validity

This is the registry as a warehouse, not as what people install. Every package counts once whether millions of projects install it or nobody ever has. A census weighted by downloads would produce different numbers and would answer a different question. Quoting one of these shares as though it described the packages in a working `node_modules` would be wrong.

The frame is the replication index, not every name npm has ever held. Ecosyste.ms, the source section 6 cites first, reported 5,797,555 npm packages when this paper was written. The index this census walked listed 4,305,887 names on 19 August 2026 and 4,296,340 of them answered, so the 2 frames are about 1,500,000 apart. This study did not reconcile them name by name and offers no account of the gap, so no share here should be read as a share of whatever Ecosyste.ms counts. What every percentage in section 3 divides by is written down exactly. That is the most the artifact supports.

The snapshot is smeared across 48 hours. A package fetched in the first hour and a package fetched in the last were read 2 days apart. At this scale that moves nothing visible and it is still true.

9,546 names returned 404 and sit outside every share. Folding them into the denominator would move any share by at most 0.22 percentage points.

`time` **is the registry's word.** Publish dates are not independently verified here and a republished or transferred package can carry dates that do not mean what they look like.

Install hooks are a declaration, not a behaviour. Nothing was executed and no script body was stored, so this data cannot tell a native build from anything else. It also counts direct installs; how far these 74,664 packages reach through transitive dependencies is a separate measurement. And the lifecycle set has changed across major versions of npm, so 10.9.2 is part of the result.

Scoped and unscoped names are counted the same way. No attempt was made to group packages by organisation, which would lower the share with one maintainer by an amount this data cannot estimate.

The crawler introduced itself under the wrong name. Its user agent string says `github.com/dimhold/dep-weight`, which is where this work started and which holds a different study. The script is published unedited, user agent included, because changing it afterwards would make the file stop being the thing that ran.

6. Prior work

Registry metadata is well covered and this census is not first to it. The 2 continuously updated datasets are larger than anything a single crawl produces.

- Ecosyste.ms aggregates package registry, repository and advisory metadata across ecosystems continuously. At the time of checking it reported 14,400,000 packages across 109 sources and 2,180,000 maintainers, of which 5,797,555 packages and 1,240,824 maintainers are npm. That npm figure is 1,500,000 above the frame counted here and section 5 says what follows from it.
- deps.dev, Google's Open Source Insights, indexes 7 package managers including npm and exposes dependency graphs through an API and a public BigQuery dataset.
- `npm-follower`: A Complete Dataset Tracking the NPM Ecosystem (ESEC/FSE 2023) went further than metadata and tracked package contents. Its repository's last commit is dated 16 December 2023, so that ground is currently open.
- Small World with High Risks: A Study of Security Threats in the npm Ecosystem (USENIX Security 2019) is the reference work on maintainer concentration. It finds that a very small number of maintainer accounts could inject code into the majority of the ecosystem and that lack of maintenance leaves packages depending on vulnerable code for years. The registry held over 800,000 packages when it was written.
- What are Weak Links in the npm Supply Chain? (arXiv:2112.10165, 2021) studies 1,630,000 npm packages and names the presence of install scripts as one of 6 weak link signals. The abstract does not report what share of packages carry them and does not separate `prepare` from the scripts that fire for an installer.

- A practitioner measurement published on 10 August 2026, How many npm packages actually run code when you npm install, reports 3.0% over 658 packages assembled from registry search queries. It says plainly that its frame is a search result rather than a census and it does not analyse `prepare` separately.

2 companion measurements of ours sit on the other side of the same registry and are named because their numbers contradict these. A disk walk of installed `node_modules` trees (10.5281/zenodo.22128826) finds 50.0% of 1,598 installed names listing exactly 1 maintainer and 44.7% silent for 2 years, against the 84.9% and 65.8% here. A resolution of 42 frozen manifests against registry metadata (10.5281/zenodo.22128854) sits with the disk walk on that axis. The reason is section 5's first threat: this is the registry as a warehouse, where most entries are a single publish nobody installed, while both companions count only what a real manifest resolves to. Any share taken over the whole registry reads far more abandoned than the same share taken over a working tree. Neither is the wrong answer to its own question.

What was not found stated anywhere is the separation itself: 6.36% of the registry declaring one of the 4 lifecycle scripts against 1.74% declaring one that runs for an installer, counted over every package rather than over a sample. What this repository holds beyond that is a dated frozen snapshot of the whole registry with the crawler, the counting scripts and a verification slice, where every number quoted is recomputed from raw counters by a check that fails if the text and the data disagree.

Searched arXiv, Semantic Scholar and GitHub on 29 August 2026. Semantic Scholar rate limited most queries and general web search was unavailable that day, so the open web outside those sources is not claimed as checked.

The novelty check for this measurement was written on 27 August 2026, after the counting rather than before it. That order is wrong and is recorded as such; from 27 August 2026 the check is required beside the disproof condition, before the first count.

7. Availability

The crawler, the counting scripts, the aggregates and the hook pass are in the repository. The archived release carries the DOI above.

The 20.65 GB crawl output is deliberately not published. It is derived data that anybody can regenerate from the registry and a copy in git would be a stale mirror of a public service. In its place there is a slice of 5,000 raw crawl records drawn by reservoir sampling with seed 20260821.

`npm run verify` does 3 things in seconds and without the network. It checks the crawler's own state file against the aggregate, so a truncated or swapped file shows up. It re-derives the headline figures from the raw counters and then checks that each derived string appears somewhere in the README, which catches a number that has drifted but not one that is quoted in a wrong sentence. Then it re-runs the counting script, in the reader's clone, over the published slice and requires it to reproduce the published slice aggregate exactly.

That last check is worth being precise about. It proves the aggregates came out of the code beside them, on real crawl records that can be read. It does not prove the full totals. Nothing short of crawling the registry again proves the full totals, which is why the crawler is published byte for byte as it ran.

The hook pass has one gap of the same kind. Its aggregate and a slice of every 160th candidate ship. Its 727 MB of raw records and its state file do not, so the count of names it requested cannot be recovered from the repository.

8. References

- Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, Michael Pradel. Small world with high risks: a study of security threats in the npm ecosystem. USENIX Security, 2019. arXiv:1902.09217. https://arxiv.org/abs/1902.09217
- Nusrat Zahan, Thomas Zimmermann, Patrice Godefroid, Brendan Murphy, Chandra Maddila, Laurie Williams. What are weak links in the npm supply chain? arXiv, 2021.

arXiv:2112.10165. https://arxiv.org/abs/2112.10165

- Donald Pinckney, Federico Cassano, Arjun Guha, Jonathan Bell. `npm-follower`: a complete dataset tracking the npm ecosystem. ESEC/FSE, 2023. arXiv:2308.12545. https://arxiv.org/abs/2308.12545
- Myzura. How many npm packages actually run code when you npm install. dev.to, 10 August 2026. https://dev.to/myzura/how-many-npm-packages-actually-run-code-when-you-npm-install-i-measured-a-sample-45e4
- Semgrep. RIP npm postinstall scripts: the npm v12 default change. Semgrep blog, 11 June 2026. https://semgrep.dev/blog/2026/rip-npm-postinstall-scripts-npm-v12-default-change/
- Dmitriy Semenkevich. Duplicate copies hold 14.0 to 17.2% of node_modules. 2026. DOI 10.5281/zenodo.22128826. https://doi.org/10.5281/zenodo.22128826
- Dmitriy Semenkevich. Dependency trees did not grow: 25 of 39 frozen manifests held flat or shrank. 2026. DOI 10.5281/zenodo.22128854. https://doi.org/10.5281/zenodo.22128854
- Google Open Source Insights. deps.dev. https://deps.dev/
- Ecosyste.ms. Open source ecosystem metadata. https://ecosyste.ms/

CITE THIS

```
Semenkevich, D. (2026). A census of 4,296,340 npm packages: 84.9% list one maintainer and 1.74% run code when you install them. dimhold.by. https://doi.org/10.5281/zenodo.22128843
```

npm · package registry · software ecosystems · software supply chain · install scripts · maintainer concentration · mining software repositories · census