The design flaw of npm and how ‘everything’ broke it

ISAAC BOAZ, Western Washington University, USA

npm (Node Package Manager) is a popular package manager owned by GitHub that allows developers to share node packages and libraries [1]. This article will review and analyze the design flaw of npm and how it was broken by a single package, and will also review how GitHub responded to the incident and what actions they took in response.

CCS Concepts: • **Software and its engineering → Collaboration in software development**; • **Social and professional topics** → Intellectual property; *Systems analysis and design*; *Project management techniques*; Systems development; *Software management*; *Centralization / decentralization*; **Technology audits**.

Additional Key Words and Phrases: npm, github, design flaws, everything, left-pad

1 ARTICLE INTRODUCTION

Design flaws in software can sometimes lead to security breaches. More often than not, it is these types of design flaws that are the most popularized. However, sometimes a design flaw can lead to a catastrophic failure that was not even intended (or initiated) by a bad actor. With over 1.3 million packages as of April 14th, 2021 [2], npm is the world’s largest software registry [3, 4]. As a result, it has been heavily relied upon by developers for years.

The npm registry is a critical piece of infrastructure for the entire JavaScript community (and by extension, 99% of websites [5]). Developers expect that the npm registry will be reliable, secure, and complete. However, the left-pad incident in 2016 [6] and the everything package incident in 2023 [7] have shown that the npm registry is not as mature as developers had hoped.

2 HISTORICAL BACKGROUND

2.1 Introduction

One of the most infamous events in the history of npm was the left-pad incident. Azer Koçulu is a software developer who was working on a personal project called kik [8]. Coincidentally, Kik was also the name of a company in Ontario, Canada [8, 9]. The company reached out to Koçulu requesting that he change the name of his project. Koçulu refused, and the company proceeded to file a complaint with npm [8]. npm (following its Dispute Resolutinon Policy [10]) then decided to transfer the name kik to the company, which caused Koçulu to remove all of his packages from the npm registry, stating

"I want all my modules to be deleted including my account, along with this package. I don’t wanna be a part of NPM anymore. If you don’t do it, let me know how do it quickly. I think I have the right of deleting all my stuff from NPM." [11]

2.2 The left-pad package

As a result of Koçulu’s deciscion, all of his packages (including left-pad) were removed from the npm registry. Despite being only 17 lines of code [12], millions of packages relied on left-pad [13]. As a result, many packages that directly or indirectly relied on left-pad broke, including React, Babel, and Atom [6].

After an attempt to publish left-pad under a new version failed (due to some major dependencies relying on the old version specificially, which was no longer available), npm resulted to restoring the package from a backup [6]. The total downtime taken to restore the backup was 2.5 hours [6].

Author’s address: Isaac Boaz, boazi@wwu.edu, isaac.k.boaz@gmail.com, Western Washington University, Seattle, Washington, USA.

With the first ever unprecedented un-unpublishing of a package, npm was able to restore the left-pad package and the packages that relied on it [6].

2.3 Mitigation and Prevention

Once the left-pad incident was resolved, npm published a blog post detailing the incident and the steps they planned to take to prevent it from happening again [6].

'We will make it harder to un-publish a version of a package if doing so would break other packages. We are still fleshing out the technical details of how this will work. Like any registry change, we will of course take our time to consider and implement it with care.' [6]

npm ended up enacting two policies relating to unpublishing packages. The first policy applied on packages that were less than 72 hours old, allowing unpublishing as long as 'no other packages in the npm Public Registry depend on your package' [14].

The second policy applied to packages that were older than 72 hours, allowing unpublishing as long as three conditions were met [14]:

(1) no other packages in the npm Public Registry depend on it
(2) it had less than 300 downloads over the last week
(3) it has a single owner/maintainer

With these new policies in place, npm hoped to prevent a similar incident from happening again. However, these new policies lead towards a new unforseen design flaw.

3 THE EVERYTHING PACKAGE

3.1 Everything's Creation

In December 2023, a developer by the name PatrickJS decided to work on a package called 'everything'. He teamed up with Evan Boehs and a few other developers to work on creating an npm package that would declare every package in the npm registry as a dependency [15]. This invovled working around obscure npm limitations, such as the maximum number of dependencies being around 800 [7, 15], and the maxiumum package size being 10 MB [7, 15].

In order to work around these limitations, PatrickJS and Evan Boehs decided to chunk their dependencies into smaller packages @everything–registry/chunk–0 through to @everything–registry/chunk–4. At the time there was approximately 2.5 million packages in the npm registry [7], so sub-chunking was also required [7].

Unsurprisingly, npm also has a rate limit on how quickly packages can be published [16, 17]. Thus, PatrickJS and Evan Boehs used a GitHub Action workflow to publish the packages [7, 15].

3.2 The Aftermath

Upon publishing of the 'everything' package in Dec 2023 [18], PatrickJS and Evan Boehs had (unintentionally) disabled unpublishing every package ever on the npm registry [7]. This was an unforseen consequence of the new unpublishing policies that npm had put in place after the left-pad incident. Namely, *every package was now a dependency of the 'everything' package*, and thus, could not be unpublushed, regardless of age. everything itself could not be removed, as it had one dependency: everything–else [19].

This was also caused by the fact that the 'everything' package depended on a '*' version of all of its packages, which meant that it matched any version of each of its packages. Even if developers published a new version of their packages, the 'everything' package would actually just depend on both the old and new versions of the package [15].

3.3 Historic Related Incidents

npm was no a stranger to packages and projects similar to ‘everything’.

3.3.1 no-one-left-behind. was an npm package that was created in 2018 that depended on 1,000 npm packages [20]. npm managed to remove the package before it caused significant damage [21]. It was then re-uploaded under a different name, this time with 33,000 dependencies [22].

3.3.2 hoarders. was a package created by Josh Holbrook that was created near the conception of npm (2012) that also depended on 11,000 packages at its inception [23]. Isaac Schluter (the founder of npm) talked with the creator of the package and convinced him to unpublish it [24]. A few years later in Aug 19, 2021, Josh Holbrook found a way to work around having a heavy load on npm.

‘now, rather than installing the utilities as direct dependencies, hoarders installs them lazily, on-the-fly. this change is practically seamless - simply reach for the utility like you would normally and hoarders will do the rest! the only requirements are npm and an internet connection.’ [24]

3.4 Developer’s Response

PatrickJS and Evan Boehs weren’t aware of the consequences of their actions until they realized that they had broken the npm registry [7]. They reached out to npm and created an issue on the GitHub repository documenting and explaining the issue [25].

uncenter, a developer who assisted in the creation of the ‘everything’ package, wrote:

‘We immediately reached out to GitHub; Patrick used his network and contacts to speak to people at GitHub, and we sent multiple emails to the support and security teams on NPM. Unfortunately, these events transpired over the holidays and the NPM/GitHub teams were not responding (likely out of the office). We continued to get harsh and rude comments from random people with a little too much time on their hands... one person even wrote a 1400 word rant about the unpublishing issue, despite us repeatedly telling them we could do nothing further’ [15].

4 GITHUB’S RESPONSE

Due to this happening over the holidays, it was only until the end of day on Jan 2, 2024 that GitHub reached out to the developers and notified them that they were aware of the problem [15]. The next day (Jan 3rd), GitHub flagged the repository and began removing the packages on npm.

4.1 Review

On Jan 4th, GitHub sent a follow-up email to the developers classifying the project as against their terms of service, and that they would be removing the repository [15]. GitHub removed the repository and packages from npm.

5 DESIGN FLAW ANALYSIS

The ‘everything’ incident exposed a fundamental flaw: npm’s design allows developers to create packages that depend on every other package in the registry, or a large number of packages.

5.1 Proposed Solutions

There have been a few proposed solutions to the design flaw of npm. It appears that GitHub does not have any immediate plans to change their policies judging by their response to the ‘everything’ package incident [15].

5.1.1 uncenter. proposes ‘NPM should either a) prevent folks from publishing packages with star versions in the package.json entirely, or b) don’t consider a dependent of a package if it uses a star version when tallying how many packages depend on a package for unpublishing.’ [15].

5.1.2 PatrickJS. proposes 'A) allow folks to unpublish when the packages that depend on them use a "star" version, B) not permit star versions in published packages going forward' [25]

At its core, this is a fundamental design flaw of npm. The npm registry is designed to be a public registry, and as such, it is difficult to prevent developers from creating packages that depend on every other package in the registry. This is a problem that is difficult to solve without fundamentally changing the design of the npm registry.

5.2 Other Potential Solutions

5.2.1 Rate Limiting. One potential solution to this problem is to rate limit the number of packages that a single package can depend on. This would prevent developers from creating packages that depend on every other package in the registry.

Realistically, an initial package should not depend on more than a few hundred other packages (especially not directly). This would be a good indicator that the package is not well-designed, and would allow the npm team to review the package before allowing it to be published.

The idea of rate-limiting the growth of a package's dependencies would require careful consideration and implementation, as it could potentially break existing packages that rely on a large number of dependencies.

5.2.2 Dependency Analysis. Another potential solution is to analyze the dependencies of a package before allowing it to be published. If a package depends on every other package in the registry (or a significant portion of it), it could be flagged for review by the npm team.

Static code analysis could additionally be used to detect dependencies that are not actually used by the package, and flag the package for review.

5.2.3 Unpublishing Policies. The unpublishing policies that npm put in place after the left-pad incident were designed to prevent a similar incident from happening again. However, they ended up creating a new design flaw that was exploited by the 'everything' package. npm could consider revising these policies to prevent a similar incident from happening in the future.

While there may not be a perfect policy that works for every package, npm could consider implementing a review process for new packages that are published to the registry from new developer (accounts), or from developers that have not published a package in a long time.

5.2.4 Review Process. npm could also consider implementing a review process for new packages that are published to the registry and contain a large number of dependencies. This would allow the npm team to catch packages like 'everything' before they are published to the registry.

Such a review process would require some form of automation and user reporting system. Considering npm is free to use [26], it may not be feasible to have sufficient staffing for this process.

5.2.5 Accept the Flaw. The route that GitHub seems to take (for now at least) is to accept the flaw and mark packages like 'everything' as against their terms of service [15]. While this may have worked in the case of the 'everything' package, it is not a long-term solution to the problem. GitHub will most likely have to automate some form of detection and removal of packages like 'everything' in the future.

6 CONCLUSION

The 'everything' package incident was a catastrophic failure that was not intended by the developers of the package. It was the result of a design flaw in the npm registry. The incident highlights the need for better security and reliability in the npm registry, and the need for better policies to prevent similar incidents from happening in the future.

7 ACKNOWLEDGEMENTS

I would like to thank Theo Browne for introducing this recent incident to me by the form of a YouTube video [7]. I would also like to thank the developers of the ‘everything’ package for sharing their story and providing valuable insights into the incident.

REFERENCES

[1] npm. npm, January 2023. URL https://www.npmjs.com/.
[2] Ahmad Nassri. So long, and thanks for all the packages!, April 2020. URL https://blog.npmjs.org/post/615388323067854848/so-long-and-thanks-for-all-the-packages.
[3] Luke Karrys and Edward Thomson. About npm, October 2023. URL https://docs.npmjs.com/about-npm.
[4] w3schools. What is npm?, January 2023. URL https://www.w3schools.com/whatis/whatis_npm.asp.
[5] w3techs. Usage statistics of javascript as client-side programming language on websites, February 2024. URL https://w3techs.com/technologies/details/cp-javascript.
[6] Isaac Schlueter. Kik, left-pad, and npm, March 2016. URL https://blog.npmjs.org/post/141577284765/kik-left-pad-and-npm.
[7] Theo Browne. How one command broke npm, January 2024. URL https://youtu.be/IzqtWTMFv9Y.
[8] Keith Collins. How one programmer broke the internet by deleting a tiny piece of code, March 2016. URL https://qz.com/646467/how-one-programmer-broke-the-internet-by-deleting-a-tiny-piece-of-code/.
[9] Crunchbase. Kik - crunchbase company profile & funding, February 2024. URL https://www.crunchbase.com/organization/kik-interactive.
[10] npm. Dispute policy, December 2023. URL https://docs.npmjs.com/policies/disputes.
[11] Mike Roberts. A discussion about the breaking of the internet, March 2016. URL https://web.archive.org/web/20160324000044/https://medium.com/@mproberts/a-discussion-about-the-breaking-of-the-internet-3d4d2a83aa4d#.rv5x9r23t.
[12] Azer Koçulu. left-pad, August 2014. URL https://github.com/left-pad/left-pad/blob/192444bbe670b3e429164ba0a5808e05f7ca5af4/index.js.
[13] Azer Koçulu et al. left-pad, May 2019. URL https://github.com/left-pad/left-pad/network/dependents.
[14] npm. npm unpublish policy, January 2023. URL https://docs.npmjs.com/policies/unpublish.
[15] uncenter. The package that broke npm (accidentally), January 2024. URL https://uncenter.dev/posts/npm-install-everything/.
[16] Ceej Silverio. Api rate limiting rolling out, August 2017. URL https://blog.npmjs.org/post/164799520460/api-rate-limiting-rolling-out.html.
[17] Myles Borins. Rate limit documentation for npm registry api, March 2022. URL https://github.com/npm/feedback/discussions/658.
[18] PatrickJS. everything - npm, December 2023. URL https://www.npmjs.com/package/everything.
[19] Alex Gee. everything-else, June 2015. URL https://www.npmjs.com/package/everything-else.
[20] Pierre Krafft. no-one-left-behind, February 2018. URL https://web.archive.org/web/20210512200558/https://www.npmjs.com/package/no-one-left-behind.
[21] npm. no-one-left-behind, January 2023. URL https://www.npmjs.com/package/no-one-left-behind.
[22] Feross Aboukhadijeh. When "everything" becomes too much: The npm package chaos of 2024, January 2024. URL https://socket.dev/blog/when-everything-becomes-too-much.
[23] Josh Holbrook. hoarders/package.json at f0172e83540c868621d1f67832493ca4f7add3f2 · jfhbrook/hoarders, June 2022. URL https://github.com/jfhbrook/hoarders/blob/f0172e83540c868621d1f67832493ca4f7add3f2/package.json.
[24] Josh Holbrook. jfhbrook/hoarders: node.js’s most complete "utility grab-bag". dedicated to substack., March 2022. URL https://github.com/jfhbrook/hoarders.
[25] Ax Sharma. ’everything’ blocks devs from removing their own npm packages, January 2024. URL https://www.bleepingcomputer.com/news/security/everything-blocks-devs-from-removing-their-own-npm-packages/.
[26] npm. npm terms of service, December 2023. URL https://docs.npmjs.com/policies/terms.
[27] Azer Koçulu. I’ve just liberated my modules, March 2016. URL https://web.archive.org/web/20160323000000/https://medium.com/@azerbike/i-ve-just-liberated-my-modules-9045c06be67c#.5g9x9r23t.
[28] PatrickJS. npm i everything, January 2024. URL https://everything.npm.lol/.

A APPENDIX: READING RESOURCES

- Azer Koçulu himself wrote a blog post about the incident, which can be found at [27].
- The everything team created a website advertising the package, which can be found at [28].