A few notes on the "Gathering weak npm credentials" article

This article was originally published in 2017 at https://gist.github.com/joepie91/828532657d23d512d76c1e68b101f436. Since then, npm has implemented 2FA support in the registry, and was acquired by Microsoft through Github.

Yesterday, an article was released that describes how one person could obtain access to enough packages on npm to affect 52% of the package installations in the Node.js ecosystem. Unfortunately, this has brought about some comments from readers that completely miss the mark, and that draw away attention from the real issue behind all this.

To be very clear: **This (security) issue was caused by 1) poor password management on the side of developers, 2) handing out unnecessary publish access to packages, and most of all 3) poor security on the side of the npm registry.**

With that being said, let's address some of the common claims. This is going to be slightly ranty, because to be honest I'm rather disappointed that otherwise competent infosec people distract from the underlying causes like this. All that's going to do is prevent this from getting fixed in *other* language package registries, which almost certainly suffer from the same issues.

"This is what you get when you use small dependencies, because there are such long dependency chains"

This is very unlikely to be a relevant factor here. Don't forget that a key part of the problem here is that publisher access is handed out unnecessarily; if the Node.js ecosystem were to consist of a few large dependencies (that everybody used) instead of many small ones (that are only used by those who actually need the entire dependency), you'd just end up with each large dependency being responsible for *a larger part of the 52%*.

There's a potential point of discussion in that a modular ecosystem means that more different groups of people are involved in the implementation of a given dependency, and that this could provide for a larger (human) attack surface; however, *this is a completely unexplored argument for which no data currently exists*, and this particular article does not provide sufficient evidence to show it to be true.

Perhaps not surprisingly, the "it's because of small dependencies" argument seems to come primarily from people who don't fully understand the Node.js dependency model and make a lot of (incorrect) assumptions about its consequences, and who appear to take every opportunity to blame things on "small dependencies" regardless of technical accuracy.

In short: No, this is not because of small dependencies. It would very likely happen with large dependencies as well.

"See, that's why you should always lock your dependency versions. This is why semantic versioning is bad."

Aside from semantic versioning being a practice that's separate from automatically updating based on a semver range, preventing automatic updates isn't going to prevent this issue either. The problem here is with *publish access to the modules*, which is a completely separate concern from "how the obtained access is misused".

In practice, most people who "lock dependency versions" seem to follow a practice of "automatically merge any update that doesn't break tests" - which really is no different from just letting semver ranges do their thing. Even if you *do* audit updates before you apply them (and let's be realistic, how many people *actually* do this for every update?), it would be trivial to subtly backdoor most of the affected packages due to their often aging and messy codebase, where one more bit of strange code doesn't really stand out.

The chances of locked dependencies preventing exploitation are close to zero. Even if you *do* audit your updates, it's relatively trivial for a competent developer to sneak by a backdoor. At the same time, "people not applying updates" is a far bigger security issue than audit-less dependency locking will solve.

All this applies to "vendoring in dependencies", too - vendoring in dependencies is no technically different from pinning a version/hash of a dependency.

In short: No, dependency locking will not prevent exploitation through this vector. Unless you have a strict auditing process (which you should, but many do not), you **should not** lock dependency versions.

"That's why you should be able to add a hash to your package.json, so that it verifies the integrity of the dependency.

This solves a completely different and almost unimportant problem. The only thing that a package hash will do, is assuring that everybody who installs the dependencies gets the exact same dependencies (for a locked set of versions). However, the npm registry *already does that* - it prevents republishing different code under an already-used version number, and even with publisher access you cannot bypass that.

Package hashes also give you absolutely zero assurances about future updates; *package hashes are not signatures*.

In short: This just doesn't even have anything to do with the credentials issue. It's totally unrelated.

"See? This is why Node.js is bad."

Unfortunately plenty of people are conveniently using this article as an excuse to complain about Node.js (because that's apparently the hip thing to do?), without bothering to understand what happened. Very simply put: **this issue is not in any way specific to Node.js.** The issue here is an issue of developers with poor password policies and poor registry access controls. It just so happens that the research was done on npm.

As far as I am aware, this kind of research has not been carried out for *any* other language package registries - but many other registries appear to be similarly poorly monitored and secured, and are very likely to be subject to the exact same attack.

If you're using this as an excuse to complain about Node.js, without bothering to understand the issue well enough to realize that it's a *language-independent issue*, then perhaps you should reconsider exactly how well-informed your point of view of Node.js (or other tools, for that matter) really is. Instead, you should take this as a lesson and *prevent this from happening in other language ecosystems*.

In short: This has absolutely nothing to do with Node.js specifically. That's just where the research happens to be done. Take the advice and start looking at other language package registries, to ensure they are not vulnerable to this either.

So then how should I fix this?

1. Demand from npm Inc. that they prioritize implementing 2FA immediately, actively monitor for incidents like this, and generally implement all the mitigations suggested in the article. It's really not reasonable how poorly monitored or secured the registry is, especially given that it's *operated by a commercial organization*, and it's been around for a *long* time.
2. If you have an npm account, follow the instructions here.
3. Carry out or encourage the same kind of research on the package registry for *your* favorite language. It's very likely that other package registries are similarly insecure and poorly monitored.

Unfortunately, as a mere consumer of packages, there's nothing you can do about this other than demanding that npm Inc. gets their registry security in order. This is fundamentally an infrastructure problem.

Revision #1
Created 11 December 2024 01:57:26 by joepie91
Updated 11 December 2024 18:44:19 by joepie91