

# Please don't include minified builds in your npm packages!

This article was originally published at

<https://gist.github.com/joepie91/04cc8329df231ea3e262dffe3d41f848>.

There's quite a few libraries on npm that not only include the regular build in their package, but also a minified build. While this may seem like a helpful addition to make the package more complete, it actually poses a real problem: it becomes very difficult to audit these libraries.

## The problem

You've probably seen incidents like the [event-stream incident](#), where a library was compromised in some way by an attacker. This sort of thing, also known as a "supply-chain attack", is starting to become more and more common - and it's something that developers need to protect themselves against.

One effective way to do so, is by auditing dependencies. Having at least a cursory look through every dependency in your dependency tree, to ensure that there's nothing sketchy in there. While it isn't going to be 100% perfect, it will detect most of these attacks - and not only is briefly reviewing dependencies *still* faster than reinventing your own wheels, it'll also give you more insight into how your application actually works under the hood.

But, there's a problem: a lot of packages include almost-duplicate builds, sometimes even minified ones. It's becoming increasingly common to see a separate CommonJS and ESM build, but in many cases there's a *minified* build included too. And those are basically impossible to audit! Even with a code beautifier, it's very difficult to understand what's really going on. But you can't ignore them either, because if they are a part of the package, then other code can require them. So you *have* to audit them.

There's a workaround for this, in the form of "reproducing" the build; taking the original (Git) repository for the package which only contains the original code and not the minified code, checking out the intended version, and then just running a build that *creates* the minified version,

which you can then compare to the one on npm. If they match, then you can assume that you only need to audit the original source in the Git repo.

Or well, that *would* be the case, if it weren't possible for the *build tools* to introduce malicious code as well. Argh! Now you need to audit *all of the build tools being used* as well, at the specific versions that are being used by each dependency. Basically, you're now auditing hundreds of build stacks. This is a massive waste of time for every developer who wants to make sure there's nothing sketchy in their dependencies!

All the while these minified builds don't really solve a problem. Which brings me to...

## Why it's unnecessary to include minified builds

As a library author, you are going to be dealing with roughly two developer demographics:

1. Those who just want a file they can include as a `<script>` tag, so that they can use your library in their (often legacy) module-less code.
2. Those with a more modern development stack, including a package manager (npm) and often also build tooling.

For the first demographic, it makes a lot of sense to provide a pre-minified build, as they are going to directly include it in their site, and it should ideally be small. But, here's the rub: those are *also* the developers who probably aren't using (or don't want to use) a package manager like npm! There's not really a reason why their minified pre-build should exist *on npm*, specifically - you might just as well offer it as a separate download.

For the second demographic, a pre-minified build isn't really useful at all. They probably already have their own development stack that does minification (of their own code *and* dependencies), and so they simply won't be using your minified build.

In short: there's not really a point to having a minified build *in your npm package*.

## The solution

Simply put: don't include minified files in your npm package - distribute them separately, instead. In most cases, you can just put it on your project's website, or even in the (Git) repository.

If you really do have some specific reason to need to distribute them through npm, at least put them in a *separate package* (eg. `yourpackage-minified`), so that only those who actually *use* the minified version need to add it to their dependency folder.

Ideally, try to only have a single copy of your code in your package at all - so also no separate CommonJS and ESM builds, for example. CommonJS works basically everywhere, and there's [basically no reason to use ESM anyway](#), so this should be fine for most projects.

If you really *must* include an ESM version of your code, you should at least [use a wrapping approach](#) instead of duplicating the code (note that this can be a breaking change!). But if you can, please leave it out to make it easier for developers to understand what they are installing into their project!

Anyone should be able to audit and review their dependencies, not just large companies with deep pockets; and not including unnecessarily duplicated or obfuscated code into your packages will help a long way towards that. Thanks!

---

Revision #2

Created 11 December 2024 12:46:16 by joepie91

Updated 11 December 2024 18:44:19 by joepie91