"Indistinguishable From an Attack"

Indistinguishable From an Attack

An autonomous bounty bot opened a pull request on my project that quietly repointed a core dependency to a fork it controlled. It did not even work. That is the part that should worry you.

On 21 June 2026, a pull request landed on Cupertino, my open-source Apple-documentation MCP server. The title was reassuringly boring: "Fix for issue #13." The body was one line: "Implemented automated fix." Three files changed, a hundred-odd lines, green-ish checks. The kind of contribution a maintainer waves through between meetings.

Then I read the lockfile.

One line in a lockfile

Buried in `Packages/Package.resolved`, the PR changed a single dependency's location:

```
- "location": "https://github.com/mihaelamj/SwiftMCPCore.git"
+ "location": "https://github.com/KartavyaDikshit/SwiftMCPCore.git"
```

`SwiftMCPCore` is the wire-protocol core my server is built on. The PR repointed it from my repository to a fork owned by the person who opened the PR. The new code in the same PR called an API (`variables`) that does not exist in the real `SwiftMCPCore`, only in that fork. The fork had been created 33 minutes before the PR was opened.

That one line is the entire anatomy of a software supply-chain compromise. If I merge it, or if my CI

so much as builds the branch, the build tool fetches and runs code from a repository I do not control, and bakes it into the binary that ships to every user through Homebrew. The payload today is harmless. The repository is not mine. Whoever owns it can change it tomorrow. That is the whole move: get a trusted project to depend on something you control, dressed up as a helpful fix.

The twist: it would not have worked

Here is what turns this from a scary anecdote into something useful. The PR changed the lockfile but not the manifest. Swift Package Manager trusts `Package.swift`, not `Package.resolved`, for where a dependency actually comes from, and `Package.swift` still pointed at my canonical repository. So on a real build the fork URL gets ignored or overwritten, and the new code, which depends on the fork, fails to compile. No build check ever ran green, because it cannot.

A competent attacker edits the manifest too, so the swap actually takes effect. This one did not. And that is exactly what told me what I was really looking at.

The real story: a bounty bot

This was not a hand-crafted attack on my project, and I do not have to guess, because the author published the tool. Among his own repositories is one called `bounty-hunter`, described in his own words as an "Autonomous GitHub open-source bounty hunter bot." He created it at midnight UTC, finished building it within the hour, and then let it run.

The timeline of my incident reads like a machine working a list:

Time (UTC, 21 June)	Event
00:00	`bounty-hunter` bot repository created
00:53	bot finished and pushed
04:21	bot forks `cupertino`
04:27	bot forks `SwiftMCPCore` (it needs a type the real package lacks)
05:00	bot opens my PR #1294
all day	36 "Fix for issue #N" PRs across 34 repositories

The targets give away the goal, and it is farming, not sabotage. One target was a crypto-bounty repo that literally pays per merged contribution, hit twice. Another was a well-known honeypot whose own description invites newcomers to "make pull requests and build their reputation." The rest were high-visibility AI and developer-tool projects where good-first-issues live, including, with some irony, a Python supply-chain auditing tool.

To make my build compile inside its sandbox, the bot needed a type the upstream package did not have. So it forked the package, added the type, and its local resolve wrote the fork URL into my lockfile, which it then committed. The dependency repoint was a byproduct of a machine trying to make a build pass on its way to a merged PR. On the other 33 repositories, the same bot never touches dependencies at all.

The author looks like an early-career developer: a five-year-old account, a portfolio of student machine-learning and web projects, based in a German university town. This is not a nation-state. It is one person, a sixteen-kilobyte Python script, and a language model.

Why the harmless explanation is the alarming one

Intent barely matters. Whether this bot is farming bounties, padding a contribution graph, or demonstrating a product, the artifact it produced is indistinguishable from a supply-chain attack and one manifest edit away from a working one. And it operates at volume: one bot, 34 repositories, in a single day.

The defense open source has always leaned on is a human maintainer reading the diff. That defense does not scale to this throughput. Review fatigue is the attack surface now. The weaponized version writes itself: same pipeline, swap the benign type for a payload, fix the manifest so it resolves, fan out to a few hundred repositories, and wait for one tired maintainer to merge a green-looking fix at 1am. The barrier to producing this used to be skill and intent. Now it is a weekend project.

What actually protects you

Read the dependency files first. On any PR that touches `Package.resolved`, `package-lock.json`, `go.sum`, or `Cargo.lock`, that diff is the review, not the feature code.

A dependency URL that points at a contributor's fork is an automatic reject, no matter what the fork contains today.

Do not let CI build untrusted external PRs without a human gate. Building is execution.

Keep the manifest and lockfile consistent, pinned to a canonical source and a known revision, and reject any PR where the two disagree.

How I handled it

I declined and closed the PR with a public explanation, because the reasons are a heads-up for every other maintainer this bot visited. The feature it reached for is actually worth having: typed resource-template variables. The right way to get it is to propose the type to the canonical package, where it can be reviewed, tagged, and pinned, then build on top of that. The idea was fine. Depending on a fork I do not control is the disqualifier.

If you maintain anything popular enough to have open issues, a bot is already reading them. Read your lockfiles.