

NPM Isn't Prepared For North Korean PolinRider Attack

 opensourcemalware.com/blog/polinrider-npm-case-study-dprk-attack

c0a15726-c5b1-4b0d-85e6-fe15553df9e2

August 25, 2026



MALWARE

AUG 24, 2026

npm is wholly unprepared for the army of compromised npm package maintainers that PolinRider has weaponized to deploy DPRK malware into the JavaScript ecosystem

GitHub

North Korea

npm

PolinRider



Throughout the Game of Thrones series, the evil Night King built an army of undead zombies that eventually attacked the kingdoms in Westeros. I think of this a lot when I watch North Korean threat actors compromising legitimate GitHub users to take control of their accounts. In the Game of Thrones universe, the army was truly dead, and brought back to life with magic. In the real world, thousands of GitHub users don't realize their computers have been compromised and that DPRK threat actors are weaponizing their machines to deploy malware into the software supply chain.



PolinRider is supply-chain campaign that hides obfuscated JavaScript inside compromised developers' `.vscode/tasks.json` files, fake `.woff2` fonts, and legitimate config files like `tailwind.config.js`, `postcss.config.mjs`, `eslint.config.mjs`, `App.js` and `babel.config.cjs`. Our [July hunt](#) put the confirmed footprint at **4,367 repositories across 2,152 owners (a 6.5x increase since March)**, but as we've said from the beginning, the real number is certainly much higher. We suspect that its in the tens of thousands, and we've had other threat intel vendors tell us its in the hundreds of thousands. Because PolinRider's purpose is persistence and spread, it's both incredibly hard for developers to "disinfect" their machines and it's also highly contagious. This enabled DPRK to build a huge "human bot network" of compromised, legitimate GitHub users. And unsurprisingly, these victims also maintain or contribute to npm packages. DPRK is now weaponizing those accounts to deploy the latest, and best, DPRK malware into npm.

The npm registry is wholly unprepared for the army of PolinRider-compromised maintainers. That's because npm's (and GitHub's) response model — a security advisory for a specific package at a specific version, followed by a takedown of that exact tarball — was designed for a world of one-off malicious packages. It's not designed for a world where the *author* of thousands of legitimate packages has been silently compromised, and where the compromise keeps reproducing itself in every new release the author publishes.

This post is a case study of one such developer: DiogoAngelim, and their npm packages `fetch-page-assets` and `html-to-gutenberg`. It's a five-month story with one advisory, two visible cleanup attempts by the maintainer, and — as of last Sunday — a fresh DPRK payload sitting live in the latest release on npm. It is not remarkable in what it shows about the attacker. It is remarkable in what it shows about how badly the current system scales. npm is unprepared for the coming storm. *Winter is coming.*

Overview of the case study

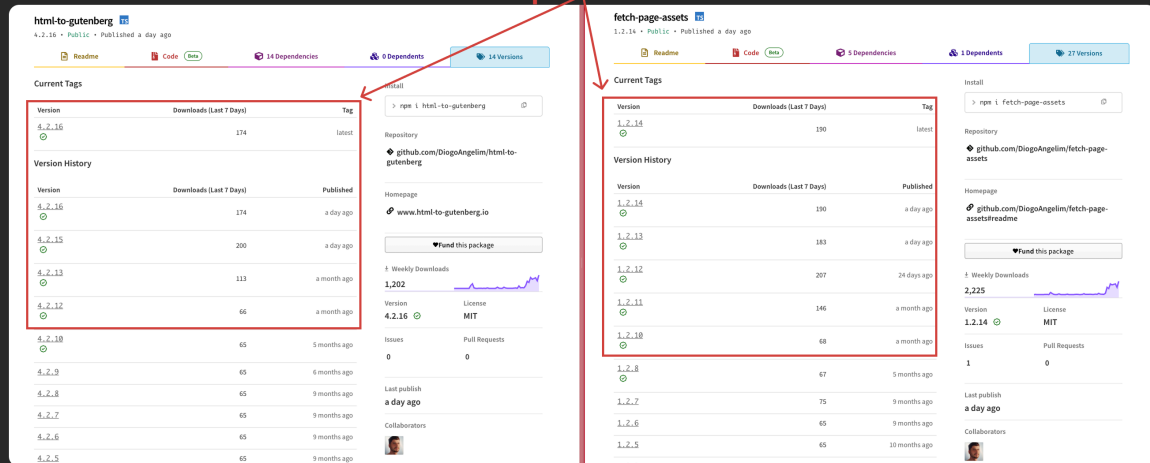
The maintainer (Diogo Angelim's handles are GitHub = `digelo` and npm = `digelim`) is a legitimate developer who's account was created in February 2016. It has eight public repositories, a small but genuine follower graph, and a decade of sporadic real-world contributions. They provide useful packages, such as `fetch-page-assets`, a Node.js utility for extracting CSS, JavaScript, fonts, and images from a URL. It's the sort of tool a working web developer writes for themselves and casually publishes.

Their GitHub account and local build environment were compromised in March (the malware auto-execution trigger has been in the repo continuously since then). They noticed *something* was wrong and attempted to clean it up, but like many PolinRider victims, removed the visible symptom but missed mechanism. Diogo is a victim.

In June, JFrog's security research team published [a blog](#) about two compromised npm packages maintained by `digelim`: `fetch-page-assets` v1.2.9 and `html-to-gutenberg` v4.2.11. We saw their research, and recognized key indicators that proved this was not a simple "account takeover", but rather PolinRider in action. The maintainer's GitHub account was infected, and it spread to their npm packages. You can see that research in [PolinRider Caused Dozens of npm, Go, PHP Compromises](#)

The malicious versions were pulled down by npm. But that's not the end of the story. We've discovered that each subsequent release of these packages are *still* shipping DPRK malware without the maintainer's knowledge.

Malicious! Compromised with PolinRider



npm's response model handles one version at a time, and there are more than 2,100 known victim maintainers capable of doing exactly what happened here, getting flagged, only having the visible part cleaned up, and shipping something worse under the operators' control again.

This case also fits the pattern documented across PolinRider: the victim pool is individuals, not organizations. It has always been a campaign that farms the accounts of working programmers, not one that stands up fake maintainers from scratch. The fetch-page-assets maintainer is one of thousands, and everything that follows in this post is about a victim whose infection was mishandled.

The compromise is account-wide, not just package-wide

fetch-page-assets and html-to-gutenberg aren't DiogoAngelim's only affected repository. Server-side GitHub Events data (push timestamps that GitHub sets on ingest, which the attacker cannot backdate) shows that every one of their seven non-archived public repositories was infected in two coordinated force-push bursts, one on 2026-07-31 and one on 2026-08-23. The July burst hit seven repositories in 56 seconds:

UTC

Repository

Actor

16:01:20 algai-landing DiogoAngelim

16:01:34 DiogoAngelim (profile repo) DiogoAngelim

UTC	Repository	Actor
16:01:42	fetch-page-assets	DiogoAngelim
16:01:50	html-to-gutenberg	DiogoAngelim
16:01:54	html-to-jsx	DiogoAngelim
16:01:57	html-to-jsx-demo	DiogoAngelim
16:02:11	signal	DiogoAngelim

The August burst repointed the same seven repositories in **30 seconds**, from 06:06:11 to 06:06:40 UTC. Every push in both bursts is attributed by GitHub's server to the DiogoAngelim identity, meaning either the account's credentials (PAT, OAuth grant, SSH key, or the CI's GITHUB_TOKEN) are directly in the operator's possession, or the maintainer's local dev environment has an agent capable of driving `git push` as them on command. Either way, the operator has push access to everything the maintainer owns, and demonstrates the capacity to weaponize all of it inside a minute.

Several of the pushed commits carry backdated author/committer dates, one on `html-to-jsx-demo` claims 2025-11-06 while the actual push arrived 2026-07-31 (~267 days of backdating). This is the same commit-metadata forgery pattern we've documented across PolinRider more broadly (see [Go pseudo-version timestamp forgery](#)); the GitHub server-side push timestamp is the only reliable anchor.

The launcher blob is identical across all seven infected repositories: Git blob 5e226620d2e360205cc8634e3c581a008d382561, 799 bytes, the same `.vscode/tasks.json` we describe below. That's the same payload propagated to every one of the maintainer's projects in a single coordinated action. A raw-content check on 2026-08-24 returned HTTP 200 for `.vscode/tasks.json` on `main` in all seven repositories. All are still infected as of this writing.

`fetch-page-assets` is the only one of the seven that publishes to npm on this cadence, which is why it's the focus of this post. But the takeaway is that the maintainer's identity itself is the compromised asset, not any single one of their projects. And this is what an already-caught,

already-flagged, “resolved” PolinRider victim looks like two months after the takedown: still pushing malware from their own account, into every repo they own, on demand.

The timeline of a partially-effective cleanup

All timestamps here are drawn from unspoofable server-side data, npm registry publish times, GitHub Actions triggering-actor logs, GitHub API commit metadata, and file hashes computed from the actual published tarballs. Where npm has pulled a version, the tarball was recovered from jsdelivr’s CDN mirror, which caches independently of the npm registry.

2026-03-29, initial compromise

At **04:28:41 UTC**, a `github-actions[bot]` commit (`a14ae12`) on the maintainer’s own `sync-npm.yml` release workflow adds 23 files to the repo, including the full `.vscode/` directory and a `public/fonts/` tree, during what should have been a two-file version-bump commit. The maintainer’s workflow only stages `package.json` and `package-lock.json`. Everything else was written to disk by something running earlier in the same CI job, almost certainly a compromised dependency’s lifecycle script executing during `npm ci / npm run build / npm test`.

The two files that matter:

- `.vscode/tasks.json`, a hidden VS Code task labeled “eslint-check,” set to “runOn”: “folderOpen”, whose actual command is `node ./public/fonts/fa-solid-400.woff2`.
- `public/fonts/fa-solid-400.woff2`, not a font. Plaintext JavaScript with a `.woff2` extension. This is the [NullReceiver](#) loader.

npm ignores excluded `.vscode/` and `public/fonts/` from the `v1.2.8` tarball by default, so this release didn’t reach installers. But the malicious files were now sitting on main, waiting for the next release cycle.

2026-05-25, first shipment to npm consumers (v1.2.9)

Two months later, at **19:19:35 UTC**, the release workflow ran again. This time the tarball actually included the `.vscode/` and `public/fonts/` trees. Every user who ran `npm install fetch-page-assets@1.2.9` between 2026-05-25 and 2026-06-24 received the DPRK loader.

2026-06-24, GitHub catches it once

At **01:44:29 UTC**, GitHub Security Advisory Database publishes `GHSA-vxq2-vhm7-7mhq`, a critical-severity malware advisory for `fetch-page-assets`. npm pulls `v1.2.9` from the registry.

The advisory names exactly one affected version: = 1.2.9.

This is the moment the defender-side clock ran out. Any tool consulting the GHSA feed (Dependabot, npm audit, Snyk, Socket, GitHub's own dependency graph) would from that point onward report `fetch-page-assets` as safe, provided consumers avoided the exact string 1.2.9. There is no evidence the maintainer was ever contacted. Their build pipeline was never inspected. The malicious `.vscode/tasks.json` was never removed from the repo. The only public signal was a version gap in npm's version listing that most maintainers would never think to look for.

2026-07-15, the maintainer tries to clean up (v1.2.10, v1.2.11)

Three weeks after the advisory, two releases went out on the same day. Something changed in the repository between v1.2.9 and v1.2.10, and it's the most telling change in this whole story: **the fake `public/fonts/fa-solid-400.woff2` payload disappeared from the shipped tarball.** All of the other font files went with it, **the entire `public/fonts/` tree was cleared.**

But the trigger, `.vscode/tasks.json`, with its `runOn: folderOpen` auto-execution task still explicitly invoking `node ./public/fonts/fa-solid-400.woff2`, was **left in place, unchanged, byte-for-byte the same as in v1.2.9.** The same file with the same SHA-256 (`f5c6be4753d6613c97f1b10c4d93a5d97a8f4fb21eb13da0ed04b23a8a61c2f6`) ships in every release from v1.2.9 through v1.2.14.

This is a maintainer looking at the symptom without seeing the mechanism. Fake fonts got flagged? Delete the fake fonts. The VS Code task that was calling the fake fonts? That's a config file, that's normal, that's not malware. But the auto-execution scaffolding is exactly what the attacker needed to keep running any subsequent payload, and it stayed.

Beneath that, the actual root cause, that something on the maintainer's build environment was re-injecting these files during CI, was never addressed. There is no way to know whether the maintainer was told the underlying vector, whether they noticed the missing v1.2.9 on npm at all, or whether they just cleaned what looked wrong and moved on. What we can say from the evidence is: they made an attempt, and it was insufficient.

2026-07-31, the attacker responds account-wide (v1.2.12)

Sixteen days after the failed cleanup, the operators pushed back, not just against `fetch-page-assets`, but against **all seven of the maintainer's non-archived repositories in a coordinated 56-second burst** beginning at 16:01:20 UTC (see the account-wide table above). `fetch-page-assets` was hit at 16:01:42, midway through the burst; the release

automation then packaged the infected tree and published v1.2.12 to npm 40 seconds later. Commit 7bd8eb2 (2026-07-31 at 16:02:20 UTC) restored the fake public/fonts/fa-solid-400.woff2 payload to the tarball, and, more importantly, added a **second, independently obfuscated copy of the same NullReceiver loader**, this time appended directly to babel.config.cjs. It sits after the closing `};` of the legitimate 5-line Babel config, with roughly 1,400 characters of whitespace between the two, so that in a text editor without word-wrap the file looks unchanged:

```
module.exports = {
  presets: [
    ['@babel/preset-env', { targets: { node: 'current' } }],
  ],
};
global.i="A8-3292-1";const 0xb40cd9=0x4963;(function(_0x28261c,_0x1c6ad4){const 0x1f55f0=0x4963,_0x424395=_0x28261c();while(![]){try{...
```

Same C2 wallet, same Ethereum RPC list, same XOR keys, same `spawn('node').unref()` persistence, the operators literally used a different JavaScript obfuscator on the same loader source and glued it to a config file that no one was looking at, precisely because the previous injection point had drawn attention.

This second injection point is worse than the first for two reasons. First, `babel.config.cjs` is loaded automatically by Babel and Jest as part of every `npm test` and `npm run build` in every project that installs `fetch-page-assets` as a dependency. It doesn't need a human to open a folder in VS Code, a CI runner is enough. Second, it's inside a legitimate-looking config file, not a suspiciously-named font file, so it's much less likely to trigger a manual review.

2026-08-23, payload refresh, current latest (v1.2.13, v1.2.14)

Sunday of last week, the operators repeated the account-wide burst, this time even faster, **30 seconds to repoint all seven repositories** (06:06:11 to 06:06:40 UTC). The `html-to-gutenberg` push in the same burst dropped a different 31,303-byte payload variant (Git blob 0272b9f2b1a0dda93baebf3bae505e22959a5464), showing the operators are actively iterating on their loader, not just replaying the same file.

`fetch-page-assets` was hit at 06:06:20; the release automation packaged the infected tree and published v1.2.13 to npm 26 seconds later (06:06:46), then v1.2.14 followed 108 seconds after that. v1.2.13 refreshed the `babel.config.cjs` payload with a new campaign marker (A8-3292-2 instead of A8-3292-1). v1.2.14 is now the latest dist-tag on npm.

Anyone running `npm install fetch-page-assets` today gets both loaders. Anyone running `npm test` in a project that depends on it triggers the babel one automatically.

The malware that shipped in each release

Here is the verified file inventory across the last five `fetch-page-assets` releases, taken from actual downloaded tarballs. Every one of these releases is malicious, but only one has been pulled.

Version	Released	<code>.vscode/tasks.json</code> (auto-run trigger)	<code>fa-solid-400.woff2</code> (font-file payload)	<code>babel.config.cjs</code> (config-file payload)	On npm Today
1.2.9	2026-05-25	shipped	shipped	N/A	Pulled 2026-06-24
1.2.10	2026-07-15	shipped (unchanged)	removed by maintainer	N/A	Live
1.2.11	2026-07-15	shipped (unchanged)	removed by maintainer	N/A	Live
1.2.12	2026-07-31	shipped (unchanged)	re-added by attacker	new payload added	Live
1.2.13	2026-08-23	shipped (unchanged)	shipped	payload refreshed	Live
1.2.14	2026-08-23	shipped (unchanged)	shipped	shipped	latest

Why this is an existential problem for npm

So, why didn't npm identify these as malicious?

In July, npm [announced](#) that they're scanning all packages for malware at publish time. Considering that fetch-page-assets version 1.2.9 was flagged as malicious and removed from npm back in May, you might think that this would highlight this package as one to watch. But you would be wrong. Every version published after that was malicious and no one identified it as bad.

NPM didn't catch this and the only reason I did was because I was forward hunting for PolinRider in NPM.

Somehow, this package slipped through the cracks with a live DPRK payload in it for weeks. This is particularly frustrating as GitHub could have checked the npm user's GitHub account. If they had, they would have seen that the author was obviously compromised, with many GitHub repos infected with malware, including the repo that the npm package is built from.

Let's zoom out.

PolinRider is confirmed at **4,367 repositories across 2,152 unique owners**, a number that has roughly doubled every three months since we first published on it in March. That's [2,152 maintainers](#), the majority of them individual developers, whose accounts and build environments are silently compromised. Most of these developers have no idea. Most of them also publish to npm.

The fetch-page-assets timeline shows what happens when the current registry response model meets a compromised legitimate maintainer at scale. Three systemic failures stack on top of each other:

1. **GitHub advisories are pinned to exact versions.** GHSA-vxq2-vhm7-7mhq says = 1.2.9. Not >= 1.2.9. Not "all versions from this maintainer pending review." Not "affected pattern." Every downstream scanner respects that exact boundary, so when the byte-identical trigger ships as v1.2.10, no advisory covers it. The registry pulled the file; nothing pulled the pattern.
2. **There is no feedback loop to the maintainer that says your build environment is compromised.** As best we can tell from public data, DiogoAngelim was never told what specifically triggered the takedown, or what an appropriate remediation would look like. So the cleanup they attempted, removing the visible fake fonts, leaving the invisible auto-execution trigger, was a reasonable-looking response to no information. And when the attacker responded to that partial cleanup with a fresh injection into a different file, there was no advisory system prepared to notice.

3. **The pattern is a known, published PolinRider indicator.** `.vscode/tasks.json` with `runOn: folderOpen` invoking a `.woff2` file is the same technique we've been documenting for five months. It sits in the default branch of `DiogoAngelim/fetch-page-assets`, unchanged and unflagged, right now, as you read this. GitHub's code-scanning could detect it. GitHub's advisory system could detect it. They don't talk to each other about this class of finding.

Multiply those three failures by 2,152 compromised maintainers and the shape of the problem becomes clear. This is not a case of one badly-handled incident. This is a system that is architecturally unprepared for the scale at which DPRK is operating. Every PolinRider-compromised maintainer is capable of pushing multiple new releases per month, and, as DiogoAngelim demonstrates, of being weaponized across all of their repositories inside a single 30-second window on the attacker's schedule. If each incident triggers a single-version, single-package advisory per detection round, and only a fraction of releases get detected at all, the attacker's publish rate will outpace the defender's response rate indefinitely.

npm's implicit threat model is "some packages are malicious, we take those down." PolinRider's threat model is "the authors of legitimate packages are compromised, and they don't know it, and we can keep pushing malware through them until someone forces us out." The registry is not currently designed to force anyone out.

What Game of Thrones teaches us about scale and constancy of attacks

When the Night King's army finally invaded Westeros, nothing could stand in their way. They were too numerous. When they got to the fort city of Winterfell and threw themselves against the walls, eventually some got through. The sheer numbers made it a statistical certainty that they would get in.

The same thing is happening with npm. With thousands of infected GitHub users, some n percentage of them will find their way into the npm registry. [We're already seeing this with Go and PHP](#), as more than a hundred PolinRider packages have found their way into the Go and Packagist ecosystems.



More npm packages with PolinRider are appearing

While `fetch-page-assets` and `html-to-gutenberg` were the first npm packages we found, they aren't the only ones:

Last week, the `@lambda-platform/lambda-vue` package version [3.3.24](#) was pushed to npm with a PolinRider payload appended to `tailwind.config.js`. What makes this worse, is that the GitHub repository owned by the lambda-platform GitHub organization that owns this npm package hasn't been cleaned, and still includes the malware. The malicious package has been removed from npm, but the source code that builds it is still infected. The next time this npm maintainer pushes the `@lambda-platform/lambda-vue` package, there is a pretty good chance that it will have the latest DPRK payload in it.

Yesterday, the `denislistiadi` npm user was compromised by PolinRider and the threat actor published 3 malicious versions:

- [@syncraft-labs/core](#), version 0.4.1
- [@syncraft-labs/react](#), version 0.4.1
- [@syncraft-labs/vue](#), version 0.4.1

If I'm right, this is just the start. PolinRider is coming.

Why aren't the authors catching these infections?

When OpenSourceMalware finds a PolinRider payload in a public repository, we do our best to alert the maintainers via their public email (if there is one) and/or LinkedIn. In parallel we will often create a public Issue on the GitHub repository explaining our findings. When the victim reads one of our disclosures, they will often respond back explaining they had no idea they were compromised. The victims are often incredulous and/or don't believe it until we show them GitHub artifacts.

Security: Repository and published npm package contain malware (NullReceiver DPRK C2 technique) #1

 Open

 6mile opened 12h ago ...

Security Disclosure: This repository and its published npm package have been compromised

DiogoAngelim/fetch-page-assets, and the npm package **fetch-page-assets** it publishes, contain a malware loader using **NullReceiver**, a DPRK blockchain C2 technique documented by the OpenSourceMalware.com research team:
<https://opensourcemalware.com/blog/nullreceiver-dprk-c2-technique>

This is not spam. This disclosure is from the [OpenSourceMalware.com](https://opensourcemalware.com) research team.

This is currently live on npm

`npm install fetch-page-assets` at the current `latest` dist-tag (**1.2.14**, and **1.2.13**) installs a malicious `babel.config.cjs`. Since Babel/Jest `require()` this file directly, the payload executes automatically on `npm test` / `npm run build` in any project that depends on this package — no manual step needed.

Compromised files

`babel.config.cjs` — the legitimate 5-line config has ~30KB of obfuscated JavaScript appended after the closing `};` on the same line:

```
global.i="A8-3292-2";const _0xb40cd9=_0x4963; ... (javascript-obfuscator.io-style string-array obfuscation)
```

Executes the moment the file is loaded by Babel/Jest/Node — no user interaction required.

`.vscode/tasks.json` — a second, independent copy of the same loader disguised as an "eslint-check" task, set to `"runOn":`

```
"folderOpen":
```

What needs to change

We're not going to solve npm's supply-chain problem in a blog post, but a few concrete things would help immediately.

Maintainer-level advisories. When a package version is flagged as malware, the maintainer account itself needs a review, not just that version. Any repository or package with the same known-bad indicators (in this case, the exact `.vscode/tasks.json` SHA-256 that's been

sitting in `fetch-page-assets` for five months) needs to be surfaced to that maintainer as part of the same incident. The advisory shouldn't close until the mechanism is gone, not just the payload.

Direct notification. A compromised legitimate maintainer will almost always try to help fix the problem if they know it exists. The current model where a version silently disappears from the registry, with no notification and no explanation, actively wastes the best remediation resource available.

Pattern-based detection at the registry. The specific technique used against `fetch-page-assets`, a hidden VS Code task set to `folderOpen` invoking a non-code file, is a published PolinRider signature. Any registry release matching a known-published indicator should at least generate an internal review flag, if not an automatic quarantine. Waiting for an external advisory to name each version one-by-one is not going to scale to thousands of maintainers.

Cross-system correlation. GitHub owns both the source repository and the advisory database. It also owns npm. The fact that `.vscode/tasks.json` with a matching trigger has been visible in the default branch of the same repo the whole time, and the two systems haven't correlated that, is the most fixable failure in this whole story.

Indicators of compromise (IOCs)

Here are the IOCs specific to the case study we discussed in this blog.

Malicious npm package versions

<code>npm/fetch-page-assets@1.2.9</code>	(pulled 2026-06-24 – GHSA-vxq2-vhm7-7mhq)
<code>npm/fetch-page-assets@1.2.10</code>	(currently live on npm – unflagged)
<code>npm/fetch-page-assets@1.2.11</code>	(currently live on npm – unflagged)
<code>npm/fetch-page-assets@1.2.12</code>	(currently live on npm – unflagged, adds <code>babel.config.cjs</code> payload)
<code>npm/fetch-page-assets@1.2.13</code>	(currently live on npm – unflagged, refreshes payload)
<code>npm/fetch-page-assets@1.2.14</code>	(currently live on npm – unflagged, latest dist-tag)

Compromised GitHub account (all 7 non-archived public repos infected)

<code>github.com/DiogoAngelim</code>	(profile repo)
<code>github.com/DiogoAngelim/algai-landing</code>	
<code>github.com/DiogoAngelim/fetch-page-assets</code>	(also on npm)
<code>github.com/DiogoAngelim/html-to-gutenberg</code>	
<code>github.com/DiogoAngelim/html-to-jsx</code>	
<code>github.com/DiogoAngelim/html-to-jsx-demo</code>	
<code>github.com/DiogoAngelim/signal</code>	

All seven confirmed still infected on main as of 2026-08-24. The archived rd-app repository was not touched in either burst and is not confirmed infected.

Account-wide force-push bursts (unspoofable GitHub server timestamps)

2026-07-31 16:01:20 → 16:02:16 UTC (56 seconds, 7 repos + branch propagations, actor: DiogoAngelim)

2026-08-23 06:06:11 → 06:06:40 UTC (30 seconds, 7 repos, actor: DiogoAngelim)

Malicious commits on main

a14ae12 2026-03-29T04:28:41Z release v1.2.8 (initial injection via CI)
88b79e9 2026-05-25T19:19:35Z release v1.2.9 (first shipped-to-npm release)
7bd8eb2 2026-07-31T16:02:20Z release v1.2.12 (adds babel.config.cjs payload, marker A8-3292-1)
9775d8d 2026-08-23T06:06:46Z release v1.2.13 (refreshes babel.config.cjs payload, marker A8-3292-2)

PolinRider / NullReceiver C2 anchor (Ethereum wallet)

0xa322e5f3d311d3080e6f0121063e9adc2490ef1a

Matches the wallet published in the [OpenSourceMalware NullReceiver research](#).

File SHA-256 hashes (as they ship on npm today)

f5c6be4753d6613c97f1b10c4d93a5d97a8f4fb21eb13da0ed04b23a8a61c2f6 .vscode/tasks.json (all 7 repos)
9fbb31129c04e8eb1a50519fc864c74d1b20d57c07d4099348f8fbc9a1a1eae6 public/fonts/fa-solid-400.woff2 (fetch-page-assets)

Git blob identifiers (for cross-repo tracking)

5e226620d2e360205cc8634e3c581a008d382561 .vscode/tasks.json launcher (all 7 repos, 799 bytes)
6df5b2c9c172d3b822494c3ef9083dbfc135ecf6 fa-solid-400.woff2 payload (July burst, 9,129 bytes)
0272b9f2b1a0dda93baebf3bae505e22959a5464 variant payload (html-to-gutenberg, August burst, 31,303 bytes)

Payload campaign markers (grep-friendly)

global.i="A8-*#" (fake-font variant, all releases)
global.i="A8-3292-1" (babel.config.cjs, v1.2.12)
global.i="A8-3292-2" (babel.config.cjs, v1.2.13 and v1.2.14)

Public Ethereum RPC endpoints queried by the loader

`https://1rpc.io/eth`
`https://eth.drpc.org`
`https://ethereum-rpc.publicnode.com`
`https://eth-mainnet.public.blastapi.io`
`https://eth.blockscout.com/api`

C2 URI paths (against attacker-resolved IP)

`http://<resolved-ip>:443/0x/cls`
`http://<resolved-ip>:443/0x/ls`

Related OpenSourceMalware research

- [PolinRider Attack](#), original March writeup
- [PolinRider Rides Again](#), April expansion
- [PolinRider Passes 4,300 Repositories](#), July scale update
- [NullReceiver: The DPRK C2 Technique](#), the loader's blockchain-based C2 mechanism

Conclusion

The fetch-page-assets compromise is not novel. The NullReceiver loader was already documented. The PolinRider technique of hiding JavaScript in `.vscode/tasks.json`, in fake `.woff2` fonts, and in legitimate config files was already documented. There is no zero-day here. What this case documents is what happens when a well-understood attack pattern meets a registry response model that assumes bad packages rather than compromised authors, and when scale outstrips remediation capacity.

DiogoAngelim tried to clean up. It wasn't enough. Two months later, the attackers escalated by moving into a config file, and npm's latest release has been shipping DPRK malware to every downstream project that runs `npm test` or `npm run build` since Sunday. There are 2,151 other compromised maintainers on GitHub that we already know about, and every one of them is one release away from the same story.

npm cannot scale to PolinRider one advisory at a time. The problem is architectural, and it needs an architectural response.

If you use `fetch-page-assets`, directly or transitively, pin your dependency to `<= 1.2.8`, or remove it. If you have `.vscode/tasks.json` files in projects you didn't personally author every

line of, check them for `run0n: folder0pen`. And if you have any outbound network logs showing a node process talking to `1rpc.io`, `eth.drpc.org`, `ethereum-rpc.publicnode.com`, or `blockscout.com`, treat that host as compromised.

If you encounter similar packages or suspicious activity, please report them to OpenSourceMalware.com.