WeaselBiscuit Strips BeaverTail and OtterCookie Down to Essentials

By c0a15726-c5b1-4b0d-85e6-fe15553df9e2

Published: 2026-09-17 · Archived: 2026-09-25 02:01:34 UTC

The OpenSourceMalware team spends a lot of time looking at malicious npm packages, and after a while you start seeing the same patterns over and over. So when something new and genuinely different shows up, it’s worth stopping for. That happened this week when we found 11 npm packages secretly hiding a brand new JavaScript stealer. It looks and feels like the DPRK’s BeaverTail and OtterCookie, but it isn’t quite either one. It’s smaller, lighter, and stripped down, with many of the heavier functions removed entirely.

We're calling it **WeaselBiscuit**. It isn’t an established family name. I made it up, and I think it’s dope.

Our automation flagged these packages as malicious and grouped them together because they share IOCs. That’s when we recognized an emerging Node.js infostealer architecture worth investigating, one that reads as a possible new or lightly documented DPRK-linked strain, or as a simplified branch or fork of the BeaverTail/OtterCookie ecosystem. You can see all the threat reports under the #weaselbiscuit tag.

Package	First seen
`@biz44/id10-client`	2026-09-12T13:51:46Z
`@biz44/id12-client`	2026-09-12T13:51:47Z
`@biz44/id44-client`	2026-09-12T08:03:37Z
`@biz44/id79-client`	2026-09-12T13:51:48Z
`@biz44/id95-client`	2026-09-12T13:51:46Z
`@biz44/id99-client`	2026-09-12T13:51:46Z

Package	First seen
`@biz44/process-runtime-utils`	2026-09-14T04:41:47Z
`@biz44/runtime-utils`	2026-09-14T03:39:58Z
`@railone/image-utils`	unknown
`@vibecheck-polid/process-runtime-utils`	unknown
`engin1`	2026-09-16T03:44:28Z (version 1.3.99 still live on NPM)
`id79-client`	2026-09-12T13:35:09Z
`laycot`	2026-09-16T13:51:46Z (version 1.3.10 still available on NPM)
`process-lhpm`	2026-09-15T00:44:34Z
`process-mite`	2026-09-16T23:57:52Z
`process-tailwind`	2026-09-15T00:20:01Z

We're tentatively attributing this malware to DPRK but it's important to be clear that it's an analytic hypothesis, not a confirmed attribution. The recovered code has meaningful overlap with DPRK-associated Contagious Interview tooling, but we haven’t recovered operator infrastructure, victimology, campaign metadata, signing material, or a unique-code comparison sufficient to name a new family or conclusively attribute it to DPRK.

What WeaselBiscuit does

- Lands via an npm `import` and auto-runs a detached background Node process

- Pulls its real payload from an Npoint URL and runs it in memory, never touching disk
- Beacons to a shared HTTP C2 at `103.170.217.184:8787`
- Profiles the host: hostname, user, OS, CPU/RAM, local and public IP, geolocation
- Steals Chrome extension storage on Windows, macOS, and Linux, which is exactly where wallet extensions keep their signing state
- Captures clipboard contents, on operator command
- Logs Windows keystrokes, on operator command
- Tags each install with a numeric campaign ID (`10` , `12` , `44` , `79` , `95` , `99`) for server-side sorting

More on exactly what gets uploaded, and what doesn't, in the Capabilities section below.

What it doesn't do, compared to BeaverTail and OtterCookie:

- No wallet-draining code, no hardcoded wallet extension ID list
- No Chrome password decryptor, no seed-phrase regex sweep
- No InvisibleFerret Python second stage
- No screenshot module
- No Socket.IO, WebSocket, or remote shell, just plain polling HTTP
- No persistence beyond the detached Node process and a `.pid` file

Why it's notable

The implant combines a package-borne Node.js loader, Npoint dead-drop resolution, Chrome extension-storage theft, clipboard collection, Windows keylogging, and a small custom HTTP control plane. Operationally, it's simpler than commonly documented OtterCookie deployments:

```
npm import
 -> detached Node loader
 -> Npoint-delivered JavaScript
 -> Npoint C2 configuration
 -> plain HTTP C2 polling and exfiltration
```

There's no Socket.IO, no remote shell, no screenshot capability, no Python InvisibleFerret handoff, no browser-password decryptor, no hardcoded wallet list, and no server-delivered follow-on payload in the recovered client. That absence could reflect an early-stage build, a deliberately reduced operational module, or a distinct actor copying familiar tradecraft.

Confirmed infection chain

1. An application imports `process-tailwind@1.1.99`.
2. `index.js` automatically calls `initialize()`.
3. `init.js` starts a detached `node loader.js` process and stores its PID in `<package-directory>/.pid`.
4. `loader.js` retrieves JSON from:

```
https://api.npoint.io/24c25d5f5fcbb0992a4f
```

5. The loader Base64-decodes the JSON `code` field and executes it through `new Function`.
6. The retrieved response is an exact match for the supplied second-stage artifact, SHA-256:

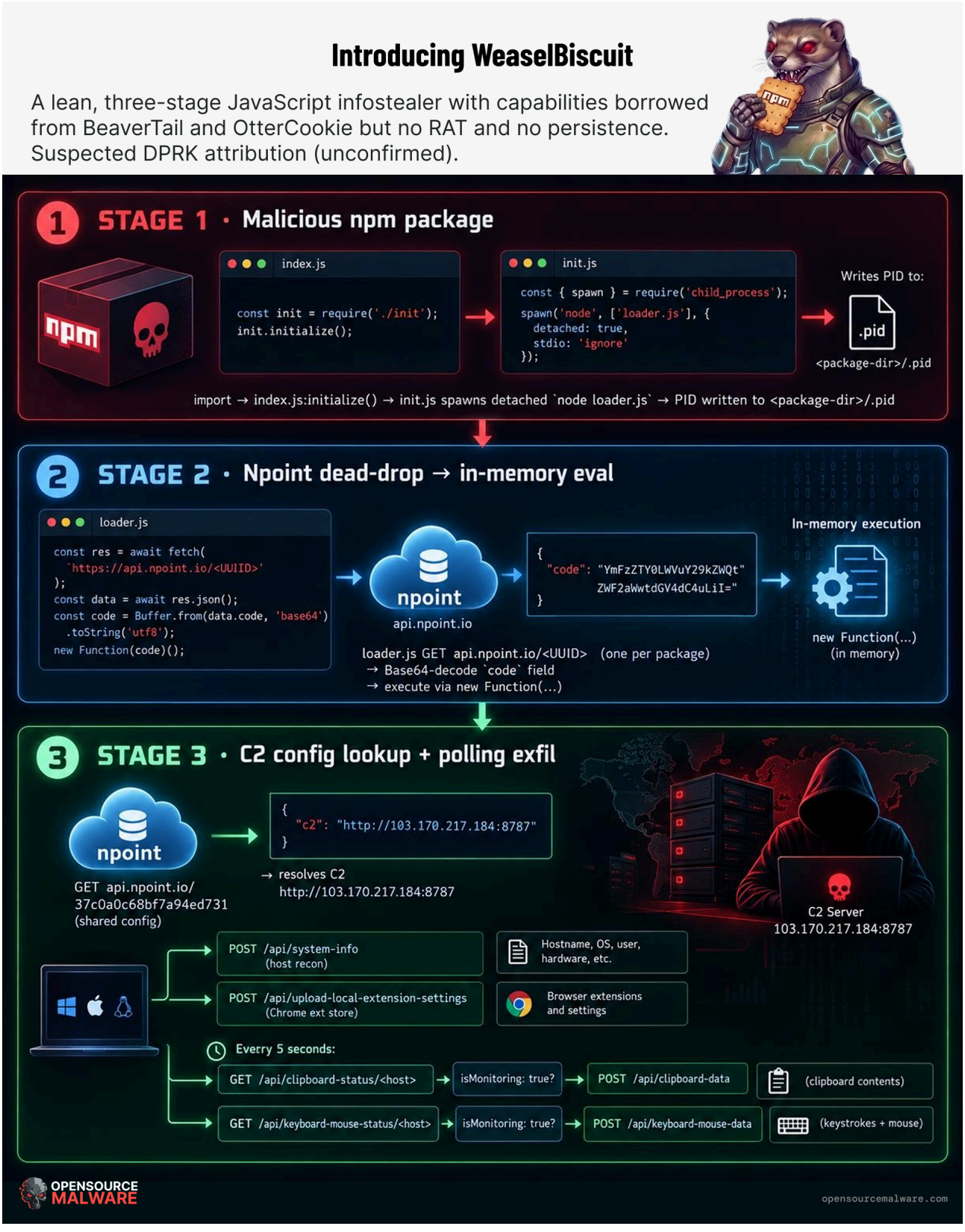
```
7b15605f23b131b3eeea57e031ae7cb32fc4b78c7bbb2025aa7a561ea5ae5159
```

7. The second stage obtains the C2 configuration from:

```
https://api.npoint.io/37c0a0c68bf7a94ed731
```

8. The configuration resolves the C2 to:

```
http://103.170.217.184:8787
```

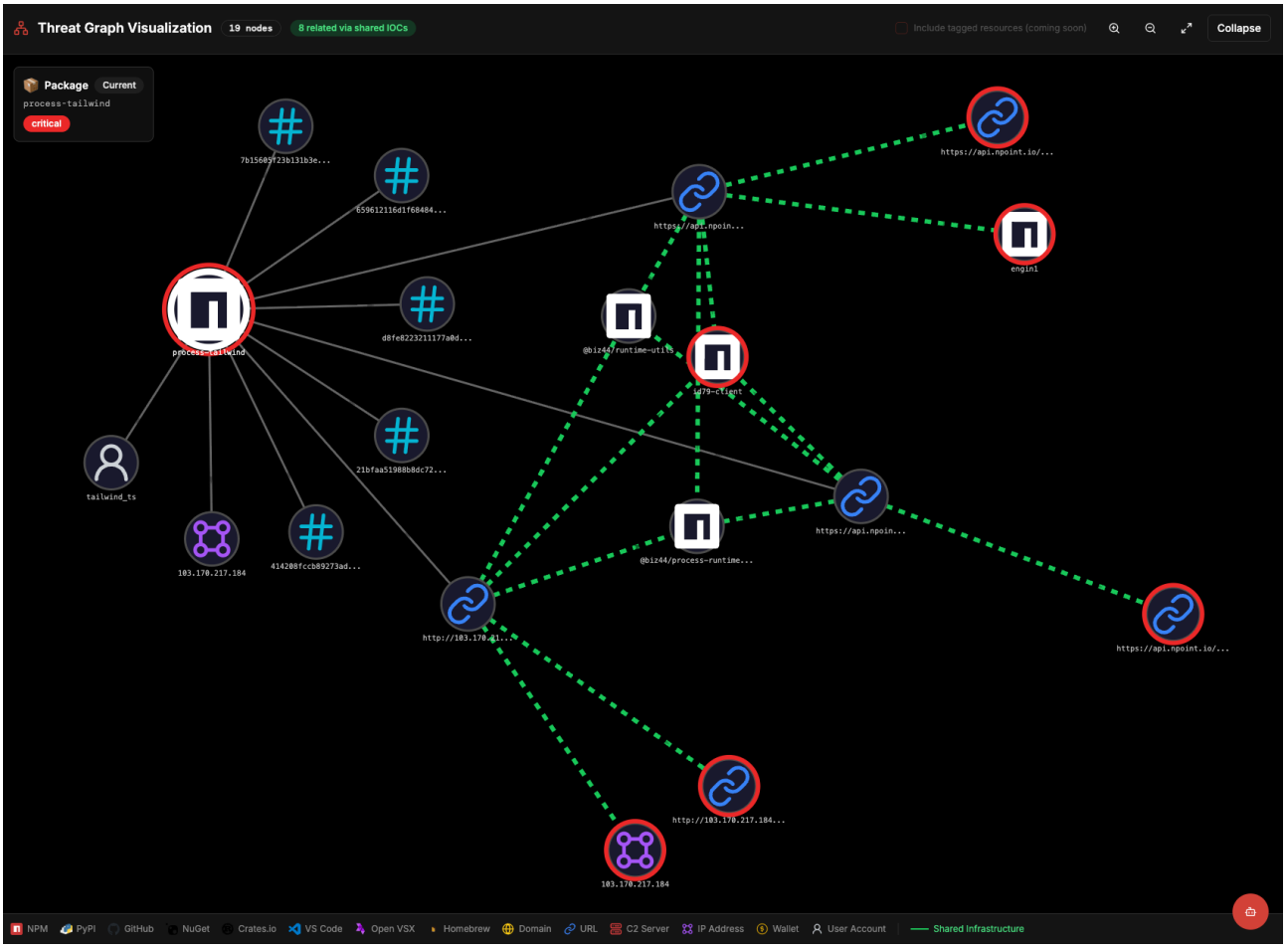


Campaign structure and identifiers

WeaselBiscuit isn’t an isolated sample. Static analysis and user-authorized inert retrievals identified a cluster of cloned npm loaders that pull near-identical WeaselBiscuit stages from distinct Npoint URLs. The stages differ

materially only in a numeric `identifier` field included in system-information and Chrome-extension uploads, which strongly suggests a server-side campaign, tenant, operator, or victim-group label.

That clustering is easier to see than to describe. The graph below maps the shared infrastructure our platform recovered across the cluster, with `process-tailwind` at the center, connected through shared file hashes, Npoint resolver URLs, and the `03.170.217.184` C2 to `id79-client`, `engin1`, and the `@biz44` packages covered in the tables below. It contains nineteen nodes, eight of them tied together by shared indicators rather than naming similarity alone. Every edge in this graph is a shared hash, resolver URL, or C2 address, not an assumption



First-stage resolver	Embedded stage identifier	Status
`https://api.npoint.io/24c25d5f5fcbb0992a4f`	`99`	Confirmed `process-tailwind` payload
`https://api.npoint.io/641d37178a880b1e8b8f`	`10`	Confirmed `swnwall` payload

First-stage resolver	Embedded stage identifier	Status
`https://api.npoint.io/33e8d008c334b060adad`	`79`	Confirmed WeaselBiscuit clone
`https://api.npoint.io/ddae72efbb6714fae922`	`12`	Confirmed WeaselBiscuit clone
`https://api.npoint.io/933a731a5e97f4b45249`	`95`	Confirmed WeaselBiscuit clone
`https://api.npoint.io/24c12c4b66a29747764f`	Unknown	Hard-coded by `id79-client`; returned HTTP 404 when retrieved

All five recovered stages use the same second Npoint configuration resolver, `https://api.npoint.io/37c0a0c68bf7a94ed731`, which supplied the same C2: `http://103.170.217.184:8787`.

The confirmed stage-level identifiers are `10`, `12`, `79`, `95`, and `99`. They're useful as campaign-hunting markers, but their precise meaning isn't yet known. The code doesn't label them as campaign IDs.

Package	Package label	First-stage resolver
`process-tailwind@1.1.99`	`ID-99 Client Module`	`24c25d5f5fcbb0992a4f`
`engin1@1.3.99`	`ID-99 Client Module`	`24c25d5f5fcbb0992a4f`
`swnwall@1.2.10`	`ID-10 Client Module`	`641d37178a880b1e8b8f`
`id79-client@1.1.79`	`ID-79 Client Module`	`24c12c4b66a29747764f`

All four package loaders share the same detached Node process, `.pid` marker, Npoint JSON retrieval, Base64 decoding, and dynamic `new Function` execution template. `process-tailwind` and `engin1` have byte-identical loader, init, and index source files. The `ID-79` label in `id79-client` is consistent with the separate recovered stage carrying `identifier: "79"`, but the package's current resolver response was unavailable, so that direct delivery link isn't proven.

Capabilities

The WeaselBiscuit infostealer uploads the following to the IP-hosted C2:

- Hostname, username, OS details, CPU, memory, home/temp paths, local interface IP/MAC addresses, public IP, and public-IP geolocation
- Every readable, nonempty file under Chrome profiles' `Local Extension Settings` directories on Windows, macOS, and Linux
- Changed clipboard contents, when the server enables monitoring
- Windows keyboard events, when the server enables monitoring

While this malware does not have the same crypto wallet stealer functions as its big siblings, the Chrome extension-storage capability is financially relevant: it can expose wallet-extension state or other extension-held sensitive data. It uploads **every readable, nonempty file** under the extension's `Local Extension Settings` directory — a raw LevelDB key/value store — wholesale.

That store legitimately contains a mix of:

- The victim's own account address(es) — typically a small number
- Wallet extensions like MetaMask and their internally cached token/contract address list (used for balance display, price feeds, swap routing)
- Address-book/contact entries (other people's addresses, not the victim's)
- Various other cached metadata

Meanwhile, clipboard and keylogging can capture credentials, API tokens, wallet addresses, or recovery phrases entered or copied during normal use. However, the code does **not** contain direct wallet draining, browser-password decryption, seed-phrase searching, or cryptocurrency transaction functionality.

C2 design

The C2 is a plain Express HTTP service, and its client-facing interface is small:

Route	Method	Function
`/api/system-info`	multipart POST	Host reconnaissance report
`/api/upload-local-extension-settings`	multipart POST	Chrome extension storage

Route	Method	Function
`/api/clipboard-status/<hostname>`	GET	Clipboard collection switch
`/api/clipboard-data`	JSON POST	Clipboard exfiltration
`/api/keyboard-mouse-status/<hostname>`	GET	Keylogger switch
`/api/keyboard-mouse-data`	JSON POST	Keystroke exfiltration

The client polls the two status routes every five seconds. Recorded responses for a synthetic host were `200 OK` Express JSON responses with `isMonitoring: false`. No payload, command, redirect, or next-stage URL was observed in those responses.

That's a real departure from documented OtterCookie variants, which use Socket.IO and can support broader command execution. This is a polling infostealer control plane, not a full interactive RAT, at least in the recovered code.

How it compares to BeaverTail and OtterCookie

The overlap supporting the DPRK hypothesis is behavioral, not dispositive:

- Node.js delivery and execution in a developer/package ecosystem
- Browser extension-storage theft with potential wallet relevance
- Native clipboard collection using `Get-Clipboard` and `pbpaste`
- Keylogging, host profiling, and HTTP exfiltration
- Use of lightweight external configuration to decouple the loader from C2 infrastructure

The divergences matter just as much:

- No Socket.IO, WebSocket, or remote-shell client
- No screenshots
- No explicit crypto-wallet extension IDs, browser credential databases, or broad file-targeting rules
- No InvisibleFerret/Python downloader

- No observed persistence beyond a detached process and a local PID marker

Cisco Talos has reported that the line between BeaverTail and OtterCookie has blurred in recent campaigns, including Node.js keylogging, clipboard monitoring, extension and wallet data targeting, and shifting C2 architectures. That makes a lineage relationship plausible, but it doesn't prove it. See BeaverTail and OtterCookie evolve with a new JavaScript module.

Capability comparison

Capability	BeaverTail	OtterCookie	WeaselBiscuit
Runtime	JavaScript (also ported to Qt/native)	Node.js	Node.js
Delivery	Fake-interview lure + malicious npm packages	Malicious npm packages	Malicious npm packages
Staging	Loaded directly by lure package	Loaded directly by lure package	Npoint dead-drop, Base64, in-memory `new Function`
C2 channel	HTTP POST to hardcoded C2	Socket.IO (bidirectional)	Polling HTTP (Express routes)
Host reconnaissance	Yes	Yes	Yes
Public IP + geolocation (nested `ipify` to `ip-api`)	Yes	Yes	Yes
Chrome extension storage theft	Yes	Yes	Yes
Hardcoded crypto-wallet extension ID list	Yes	Yes	**No**

Capability	**BeaverTail**	**OtterCookie**	**WeaselBiscuit**
Browser credential DB decryption	Yes	Partial (variant-dependent)	**No**
Seed-phrase / wallet-address regex sweep	Yes	Variant-dependent	**No**
File exfiltration (documents, keystores, profiles)	Yes	Yes	**No** (extension storage only)
Clipboard capture	Variant-dependent	Yes	Yes (operator-gated)
Keylogging	Not typical	Yes (recent variants)	Yes, Windows only (operator-gated)
Screenshot capture	No	Yes (recent variants)	**No**
Remote shell / arbitrary command execution	Via InvisibleFerret handoff	Yes (over Socket.IO)	**No**
InvisibleFerret / Python second stage	Yes	Yes (some variants)	**No**
Persistence	LaunchAgent / registry / startup entries	Detached process + variant-specific	Detached Node process + `.pid` marker only
Per-install campaign tagging in code	Not observed publicly	Not observed publicly	Yes, numeric identifier (`10` , `12` , `44` , `79` , `95` , `99`)

Confidence and gaps

Claim	Confidence	Basis / limitation
Package is a malicious staged Node.js loader	High	Auto-start, detached loader, remote Base64 code execution, and exact recovered payload match
Second stage is an infostealer	High	Static collection and upload logic
`103.170.217.184:8787` is the C2 used by this stage	High	Retrieved configuration and observed client routes
The implementation is a simplified or new branch	Moderate	Distinct architecture and reduced capability set, though it could also be a commodity copy
Linked to BeaverTail/OtterCookie lineage	Moderate	Npoint dead-drop pattern is near-identical to prior DPRK npm samples; nested `api.ipify.org` to `ip-api.com` lookup matches DPRK stealer convention; Express polling C2 reads as a lightweight port of OtterCookie's Socket.IO control plane
DPRK attribution	Low to moderate	Consistent DPRK tradecraft signals, but no exclusive infrastructure or shared code recovered
A wholly new malware family	Low	Requires code-cluster, infrastructure, and victimology comparison

Three of those signals are worth walking through in more detail.

Tradecraft signals worth flagging

Npoint.io as the dead drop. DPRK npm crews have been using `api.npoint.io` as a first-stage dead-drop for a long time, and WeaselBiscuit's usage is nearly identical to prior DPRK samples: hardcoded UUID, JSON blob containing a Base64 `code` field, in-process `new Function` execution. Same service, same shape, same execution pattern.

Nested public-IP and geolocation lookup. The second stage queries `api.ipify.org` for the public IP, then feeds that IP to `ip-api.com` for geolocation. That two-step nested lookup shows up in other DPRK-linked npm stealers too. It isn't a common commodity design.

Express HTTP C2 as a lightweight port of OtterCookie's Socket.IO plane. The C2 architecture itself is new: plain Express routes, polling-based, no bidirectional socket. But its shape, with per-host status endpoints gating live collection and separate exfiltration endpoints for clipboard and keyboard, reads as a stripped-down re-implementation of the same control model OtterCookie runs over Socket.IO.

Campaign markers resemble PolinRider. The numeric per-install identifiers (`10` , `12` , `44` , `79` , `95` , `99`) baked into both package names and stage uploads mirror the campaign-marker convention seen in the PolinRider cluster. The operator is tracking installs at the same granularity, using the same "ID in the package name" pattern.

Indicators of compromise

Type	Value
C2	`103.170.217.184:8787`
Npoint resolver	`https://api.npoint.io/24c25d5f5fcbb0992a4f` (identifier `99`)
Npoint resolver	`https://api.npoint.io/641d37178a880b1e8b8f` (identifier `10`)
Npoint resolver	`https://api.npoint.io/33e8d008c334b060adad` (identifier `79`)
Npoint resolver	`https://api.npoint.io/ddae72efbb6714fae922` (identifier `12`)
Npoint resolver	`https://api.npoint.io/933a731a5e97f4b45249` (identifier `95`)
Npoint resolver	`https://api.npoint.io/24c12c4b66a29747764f` (identifier unknown, hard-coded by `id79-client` , HTTP 404 on retrieval)

Type	Value
Npoint C2-config resolver	`https://api.npoint.io/37c0a0c68bf7a94ed731`
Second-stage SHA-256	`7b15605f23b131b3eeea57e031ae7cb32fc4b78c7bbb2025aa7a561ea5ae5159`
npm package	`@biz44/id10-client`
npm package	`@biz44/id12-client`
npm package	`@biz44/id44-client`
npm package	`@biz44/id79-client`
npm package	`@biz44/id95-client`
npm package	`@biz44/id99-client`
npm package	`@biz44/process-runtime-utils`
npm package	`@biz44/runtime-utils`
npm package	`engin1` (v1.3.99)
npm package	`id79-client`
npm package	`process-lhpm`

Type	Value
npm package	`process-tailwind` (v1.1.99)
npm package	`swnwall` (v1.2.10)

Recommended investigation priorities

1. Preserve the npm tarball, package publication metadata, maintainer account, dependency graph, download history, and source repository references.
2. Search public and internal telemetry for all six Npoint UUIDs, the C2 IP and port, the stage identifiers `10`, `12`, `79`, `95`, and `99`, and the API route strings.
3. Compare the loader and decoded stage against known BeaverTail/OtterCookie samples for shared functions, comments, string conventions, package names, and C2 patterns.
4. Identify the importing parent application. The package itself has no npm lifecycle hook, so execution requires an import or a manual start.
5. Review affected endpoints for `.pid` files, Node child processes, PowerShell processes, and `%TEMP%\kb-monitor\keyboard-monitor-*.ps1`.

Bottom line

Treat `process-tailwind` as a high-confidence malicious supply-chain package, and treat it as a potentially important data point in how DPRK-linked JavaScript infostealers are evolving. Treat the “new DPRK strain” label as a working investigative hypothesis, not a conclusion, until it’s backed by corroborating code, infrastructure, or campaign evidence.
