

NEBULA

Seven Fake AI SDK Packages on npm Install a Windows RAT That Needs No DLL

Four npm burner accounts deliver a syscall-only Windows RAT via a fake NebulaAI SDK lure

CATEGORY**Adversary Intelligence****ECOSYSTEM****npm**

Contents

Executive Summary	3
Analysis	3
Malware Analysis	4
Who Is Being Targeted	7
How Far the Campaign Reaches	7
Indicators of Compromise	8
Impact	9
Recommendations	10
Conclusion	10
Appendix A: Package Inventory	11
Appendix B: Detection Rule	12

Executive Summary

In late September 2026, a single actor deployed a series of malicious npm packages posing as a "NebulaAI" software development kit to deliver a Windows remote-access trojan. CloudSEK tracks this activity under the moniker NEBULA. The operation spans seven identified packages distributed across four sequential burner accounts (`nebula11ms`, `nebula11ms2`, `nebula11ms3`, `nebula11ms4`). Among these, `api-nebula` and `llm-nebula` remain downloadable on the registry. Notably, `api-nebula` was active and unflagged for days prior to its formal categorization as MAL-2026-17531 on 5 October 2026, yet it continues to be installable. Each package couples a convincing AI client facade with a hidden, obfuscated installation hook that executes the primary malicious logic.

While individual feed entries document these drops in isolation detailing the install script, masqueraded console executable, and KNTRAT payload they miss the broader operational link: a single actor using a four-account burner sequence to distribute a fake AI SDK. The underlying RAT operates exclusively through direct NT and win32k syscalls, bypassing standard DLL imports and leaving an empty Import Address Table (IAT). Its feature set comprises hidden-desktop remote control (HVNC), Kernel Streaming for camera and microphone monitoring, and Winlogon Shell persistence. The source repository for the RAT remained private during the campaign, only going public after its conclusion (renamed from `kntrat-e` to `kntrat` on 6 October 2026). The package serves as the lure; the execution rests within the installation hook. The imported entry point, `nebula.js`, functions as a legitimate-looking client pointing to `api.nebulaai.dev`. The actual threat resides in `preinstall.cjs`, an obfuscated dropper triggered during npm installation. It writes an executable payload to `%LOCALAPPDATA%\Microsoft\Conhost\conhost.exe` masquerading as the legitimate Windows console host using two delivery methods: an embedded inline payload (base64 and zlib encoded) or an install-time network request. The deployed binary is a customized variant of KNTRAT, an open-source remote-access tool configured to communicate with `65.87.7.132` using the user-agent `kntrat/0xB15B00B6`. Static analysis of the package revealed these details, as the implant suppressed beaconing during a twelve-minute sandbox detonation, consistent with embedded anti-analysis provisions.

Immediate action

Security teams should immediately restrict the active packages and block infrastructure associated with the campaign. Specifically, prevent installs of `api-nebula` and `llm-nebula`, restrict network communication to `65.87.7.132`, and implement detection logic for the conhost drop location, custom user-agent, named-pipe constructions, and hidden-desktop artifacts.

Analysis

Designed to pass a cursory inspection, these packages feature a convincing entry point specified in `main`: `nebula.js`. This module acts as a plausible AI client library, complete with a `NebulaAIClient` class, streaming support, session handling, a default base URL pointing to `api.nebulaai.dev`, and the model designation `nebula-orbit-3`. A developer inspecting the package contents would see what appears to be a legitimate SDK, effectively diverting focus away from the critical malicious element: the installation hook.

To date, seven related package names have been identified. Six follow a clear naming scheme by appending terms like `ai`, `llm`, `api`, `sdk`, or `js` to the prefix `nebula`. Among these, `api-nebula` and `llm-nebula` remain active on npm under the publisher account `nebula11ms3`. Two others: `nebula-sdk` and `nebulajs-api` (published by `nebula11ms2`) were taken over by npm and replaced with security stubs, while `nebula-llm` and `nebulaai-sdk` (`nebula11ms`) were removed entirely. Legitimate packages on the registry incorporating "nebula" (such as particle engines or Qlik visualization

utilities) are unaffiliated with this threat group. The definitive link unifying the six packages is a byte-identical installation hook, confirming they originate from the same actor regardless of whether previous security advisories tracked them together.

Activity from the operator remains ongoing. On 2 October 2026, a seventh package named `testtestkjklfdnjdfn` was uploaded via a fourth account, `nebulallms4` (registered with `30tiffanye@uberip.com`, matching the disposable email pattern used by `nebulallms3`'s `66aundrea@uberip.com`). It was self-removed just 41 minutes after publication. Given its short lifespan and naming structure, it served as a test build: it introduced a re-packed `preinstall.cjs` file (sha256 `d8f5c42c...`, replacing the earlier `8bc90df9...`) while retaining the byte-identical decoy file `nebula.js` (sha256 `d6b33324...`). The consistent decoy file serves as the primary anchor connecting the campaign, while the updated packer demonstrates active iteration by the attacker.

Rather than using proprietary malware, the campaign delivers a customized build of KNTRAT, an open-source remote-access trojan. Internal strings in the recovered binary state its lineage and specific functional adjustments, revealing an unusual awareness of operational analysis: the payload directly addresses potential analysts. Explicit commentary inside the binary notes that process creation relies on `CreateProcessW` and references a source repository that was purportedly kept private until 1 October 2026. The repository was ultimately made public on 6 October 2026 five days later than claimed and after investigation had commenced following a rename from `kntrat-e` to `kntrat`. It describes the tool as a FOSS Remote Access Tool for x64 Windows XP – 11 (No dependencies, NT Syscalls only) and exhibits no forks or stars. Consequently, the campaign was launched using a RAT whose source code was intentionally hidden during active deployment and published only after the operation was underway.

A fake AI SDK that installs a Windows RAT

The package looks like a NebulaAI client; the malice is in the install hook.

1 The lure

`nebula.js` is a plausible "NebulaAI" SDK client (`api.nebulaai.dev`). It is the package main and the decoy.

2 The hook

```
preinstall: node
preinstall.cjs runs on npm
install. 187–238 KB,
obfuscated (a Function-VM
packer).
```

3 Drop

Either an inline payload (base64 → zlib → PE) or an install-time fetch, written to:
`%LOCALAPPDATA%\Microsoft\Conhost\conhost.exe`
a Microsoft-lookalike console-host path.

4 KNTRAT

A modified build of KNTRAT (open-source RAT). PE32+ x86-64.
UA `kntrat/0xB15B00B6`
C2 `65.87.7.132`
Under analysis it withheld its beacon (anti-analysis notice in the binary).

Figure 1. The lure and the payload. `nebula.js` is the believable SDK facade; `preinstall.cjs` is the dropper, which writes a modified KNTRAT build to a Microsoft-lookalike console-host path.

Malware Analysis

Our evaluation examines the distributed package following its logical sequence of execution: beginning with the deceptive facade designed to project legitimacy, progressing to the installation hook triggering the dropper, continuing to the extracted payload, and concluding with its dynamic behavior during detonation analysis. Due to heavy obfuscation, the dropper cannot be represented cleanly as line-numbered source code; instead, its packed state along with the strings retrieved from the embedded payload are presented as reconstructed technical evidence. Conversely, the legitimate-looking client code—being un-obfuscated plain text—is provided with its actual line numbering retained.

The facade and the hook

The package manifest names a believable entry point and, quietly, an install hook.

```
1 {
2   "name": "api-nebula",
3   "version": "1.0.0",
4   "description": "",
5   "main": "nebula.js",
6   "type": "module",
7   "scripts": {
8     "preinstall": "node preinstall.cjs",
9     "test": "echo \"Error: no test specified\" && exit 1"
10  },
```

Exhibit 1. api-nebula@1.0.0, package.json lines 1-10. File SHA-256 in Table 3.

Line 5 points `main` at `nebula.js`, the SDK facade a reader would inspect. Line 8 registers `preinstall`, which npm runs before the package is even placed, and before a developer who meant to read the code first has the chance. The facade itself is a working AI client.

```
1 export default class NebulaAIClient {
2   constructor({
3     apiKey,
4     baseUrl = "https://api.nebulaai.dev/v2",
5   }) {
```

Exhibit 2. api-nebula@1.0.0, nebula.js lines 1-5. The decoy client; api.nebulaai.dev does not resolve.

Line 4 fixes the decoy's base URL at `api.nebulaai.dev`, a domain that does not resolve. The class is otherwise a complete client, with sessions and a `nebula-orbit-3` model, which is the point: the believable half is the half a developer reads, and the dangerous half runs before they can.

The dropper

`preinstall.cjs` is 187 to 238 KB depending on the package and is obfuscated with a Function-constructor virtual machine and an XOR string cipher, so its strings are not greppable and it carries no plain line numbers to cite.

```
Function("pbeIUC", "var 00Z6vNI,t3qMhG,...; function aZkULCA(00Z6vNI){ ... XOR/hash VM ...
}");

// reconstructed: the same preinstall.cjs (sha256
// 8bc90df9b387849338d9c61a8d379cfbb0d70ea576c0e37e1fb07ba164921807)
// is byte-identical across nebula-sdk, nebula-llm and nebulaai-sdk.
```

Reconstructed from api-nebula@1.0.0 preinstall.cjs (packed; shown without line numbers).

Two delivery modes were recovered from it. One variant carries the payload inline: a large base64 string decodes to zlib-compressed data that decompresses to a Windows PE. The other fetches the payload at install time. Both write the result to the same Microsoft-lookalike path and launch it detached.

```
drop path : %LOCALAPPDATA%\Microsoft\Conhost\conhost.exe
inline    : base64 -> zlib (789c...) -> PE32+ (nebula-sdk@1.0.0)
recovered RAT PE : 257,536 bytes, PE32+ GUI x86-64
sha256    : 5222dd57b8859e791a16abcf7f616bbc59f8cf248798c7c8c2ba800f0b52cd8c
```

Reconstructed: the drop path and the inline decode chain, recovered by decoding preinstall.cjs from nebula-sdk@1.0.0.

The payload is a modified KNTRAT

The recovered PE identifies itself. Its strings name the RAT family, the modification, the command host and the source repository.

```
open source @ https://github.com/syskiel/kntrat-e (repo privated until October 1st, 2026)
You are analyzing a modified build of KNTRAT which has the difference of
    starting processes with CreateProcessW.
User-Agent: kntrat/0xB15B00B6
65.87.7.132
```

Reconstructed: strings recovered from the decoded payload PE (sha256 5222dd57...). The command host 65.87.7.132 is a raw IP; no domain is used.

These establish the payload without running it. It is a modified build of KNTRAT, an open-source remote-access trojan; it beacons to `65.87.7.132` with the user-agent `kntrat/0xB15B00B6`; and it carries an explicit notice that it expects to be analysed. The command host is a bare IP address, so there is no domain to seize, only an address to block. The embedded repository URL, `github.com/syskiel/kntrat-e`, returned 404 throughout the campaign window and at analysis time. On 6 October 2026 the repository was renamed to `github.com/syskiel/kntrat` and made public, describing itself as a FOSS Remote Access Tool for x64 Windows XP–11. It is a C++ project with zero stars and zero forks.

Under detonation, the implant withheld its beacon

The payload was detonated in an isolated Windows lab with a network sinkhole and no route to the real command host. Across a run of more than twelve minutes the capture showed only Windows operating-system traffic: no connection to `65.87.7.132`, no `kntrat` user-agent, no beacon of any kind. The reading is not that the implant is inert but that the NEBULA-modified build withholds its beacon under analysis, consistent with the anti-analysis notice in the PE and confirmed by the source code, which shows that the unmodified KNTRAT is a fully capable RAT. The clean-restore guarantee was verified on every run, and no traffic was relayed to operator infrastructure at any point. The command host and delivery rest on the recovered payload; the capabilities are confirmed from the published source.

KNTRAT capabilities from source

The KNTRAT repository became public on 6 October 2026 and its source confirms what the recovered binary claimed: a full-featured Windows remote-access trojan. The technical novelty is that the entire feature set operates via direct NT kernel syscalls and win32k system calls, bypassing `kernel32.dll`, `user32.dll`, `gdi32.dll` and every other user-mode wrapper. This is not the SysWhispers or Hell's Gate pattern of resolving a handful of syscall numbers for evasion; KNTRAT reimplements the Windows API surface in a header of over 100,000 lines covering Vista x64 through Windows 11 26H2. Because the PE's import table is empty, import-based antivirus and EDR detection is rendered ineffective.

Hidden desktop (HVNC)

The implant creates an invisible Windows desktop named `winsta0\kntrat` via a direct win32k syscall and spawns processes on it, assigned to a Job Object that auto-terminates on cleanup. Each window is captured individually with DWM compositing and composited bottom-to-top onto a framebuffer. Input is injected via `PostMessage` and

`SendMessage` to the hidden desktop's windows. The result is a fully interactive remote session browser, command shell, any GUI application that is completely invisible to the logged-in user.

Persistence via Winlogon Shell

The implant writes its executable path to the per-user Winlogon Shell registry value, comma-separated with `explorer.exe`, so that it launches at every logon alongside the Windows shell. The SID is resolved manually from the process token.

```
HKU\<SID>\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell  
explorer.exe,C:\...\conhost.exe
```

Reconstructed: the persistence registry key (from the KNTRAT source). The implant appends its path after the default shell.

Remote shell via named pipes

Rather than inheriting a socket, the shell module creates two named pipes — `\\Device\\NamedPipe\\kntrat<id>i` for stdin and `kntrat<id>o` for stdout — and spawns `cmd.exe` headless with the pipe handles inherited. These pipe names constitute a new forensic indicator: any pipe matching the `kntrat` prefix is the implant.

Camera and microphone via Kernel Streaming

The implant bypasses the entire COM, DirectShow and Media Foundation stack. A custom layer opens camera and audio filter devices directly through kernel streaming device interfaces, capturing frames via asynchronous reads. This technique is not documented in commodity remote-access tools, to CloudSEK's knowledge.

NTFS Extended Attributes for device identity

A 32-digit random UUID is stored as an NTFS Extended Attribute on the implant's own executable, invisible to most file browsers and forensic tools. The single-instance mutex uses `NtLockFile` on a boot-unique byte offset rather than a named kernel object, evading standard mutex enumeration.

Server and protocol

The C2 server (Python, websockets library) listens on port 443 for devices and port 1434 for operators. Communication is unencrypted WebSocket with a 10-byte binary header and 55 distinct packet types covering all capabilities. The server code is self-described as LLM-generated. The branding in the operator interface reads `made with <3 by github.com/syskiel`.

Who Is Being Targeted

There is no named victim organisation. The target is the developer who reaches for an AI SDK and installs one of these packages, on the strength of a believable NebulaAI facade, onto a Windows machine. The lure is specific to the current moment, when new AI client libraries appear constantly and a plausible one draws installs on name alone, but the victim selection is opportunistic: whoever installs it. The operator chose the AI-SDK disguise, not a particular company.

How Far the Campaign Reaches

Seven package names were recovered across four accounts. This is a floor: it is what the registry and our pivots still answer for, and names removed before either saw them would not appear. Two are live. `api-nebula` shipped in the wild

and went uncatalogued for days before it was filed as MAL-2026-17531 on 5 October 2026, and it remains installable even now the catalogues both lagged the live package and have not driven its removal.

The floor is partly corpus-checked. A YARA sweep of the code fingerprint across every npm package CloudSEK ingested from 25 to 27 September 2026, both dated and undated 162,471 packages over the three days, of which 162,464 archives were still fetchable and scanned recalled only the known members and raised no false positives on any day's full intake. No package beyond the known set carried the fingerprint across the campaign's active window. It remains a floor rather than a census for two reasons: seven archives (two on the 25th, five on the 27th) had already been taken down upstream with no copy held, so the sweep covers every archive we could still fetch rather than literally every name; and the days outside this window, together with the 2 October test build's day, are not swept.

One operator, six packages

A sequential four-account series (nebulallms / 2 / 3 / 4), all uberip.com burner emails. Two still live.

Package	Registry state	OSV	Account
api-nebula	● LIVE	MAL-2026-17531	nebulallms3
llm-nebula	● LIVE	MAL-2026-17230	nebulallms3
nebula-sdk	○ seized	MAL-2026-17219	nebulallms2
nebulajs-api	○ seized	MAL-2026-17220	nebulallms2
nebula-llm	○ removed	MAL-2026-17227	nebulallms
nebulaai-sdk	○ removed	MAL-2026-17228	nebulallms
testtestkjklfdnjdfn	○ test (self-unpub)	none	nebulallms4

Linked by a byte-identical preinstall.cjs (sha256 8bc90df9...) shared across the cluster. ● live ○ seized/removed

Figure 2. Seven packages across a sequential four-account series (nebulallms/2/3/4), all uberip.com burner emails, linked by a byte-identical install hook and decoy. Two remain installable.

The link between them is mechanical, not nominal. A byte-identical `preinstall.cjs` (sha256 `8bc90df9...`) appears across the cluster, so the packages are probably one operator's work even where no advisory connected them. Our own pipeline never ingested any of these packages; they were recovered by registry probing and corpus pull, which is itself the detection gap that let them sit at low priority.

The seven packages were published from a sequential four-account series: `nebulallms`, `nebulallms2`, `nebulallms3` and `nebulallms4`, each account registered against a distinct `uberip.com` disposable-mail address (`livia3769`, `kaylynresident`, `66aundrea`, `30tiffanie`) and publishing only one or two of the packages. The burner-per-account pattern defeats takedown-by-account and email-pivoting; the byte-identical decoy and preinstall are what link the accounts into one operator.

Indicators of Compromise

Network

Indicator	Type	Note
65.87.7.132	Command host	raw IP, KNTRAT C2; Internet Utilities NA LLC, privacy-masked
api.nebulaai.dev	Decoy URL	the SDK facade's base URL; does not resolve (not a C2)
port 1434	Operator port	KNTRAT operator WebSocket listener (from source)

Table 1. 65.87.7.132 is a bare-IP command host, so there is no domain to seize. api.nebulaai.dev is the decoy's URL, not attacker C2, and is safe to ignore. Port 1434 is the operator-facing listener identified from the KNTRAT source.

Content and behavioural signatures

String or structure	What it identifies
<code>preinstall.cjs sha256 8bc90df9...</code>	The byte-identical dropper linking the cluster
<code>Microsoft\Conhost\conhost.exe under %LOCALAPPDATA%</code>	The Microsoft-lookalike drop path
<code>User-Agent: kntrat/0xB15B00B6</code>	The KNTRAT beacon user-agent
<code>NebulaAIClient / api.nebulaai.dev / nebula-orbit-3</code>	The fake-SDK facade strings
<code>preinstall running preinstall.cjs on npm install</code>	A nebula-* package with a preinstall hook and main:nebula.js
<code>\\Device\\NamedPipe\\kntrat<id>i / kntrat<id>o</code>	Named pipes for the KNTRAT remote shell (from source)
<code>winsta0\\kntrat</code>	Hidden desktop name for HVNC (from source)
<code>HKU\\<SID>\\...\\Winlogon\\Shell</code>	Persistence: implant path appended after explorer.exe (from source)

Table 2. Content and behavioural signatures generalise across the cluster where a hash of one tarball does not. The byte-identical preinstall, the drop path and the named-pipe prefix are the strongest. Items marked "from source" were identified from the KNTRAT repository, made public 6 October 2026.

File hashes

KNTRAT payload PE	5222dd57b8859e791a16abcf7f616bbc59f8cf248798c7c8c2ba800f0b52cd8c
preinstall.cjs (byte-identical across the cluster)	8bc90df9b387849338d9c61a8d379cfbb0d70ea576c0e37e1fb07ba164921807
api-nebula@1.0.0 (tarball)	e99bab7b8bbbd7c821dae4ccfedfd1e608d65c466de9b08037c5ce5f56af3b1
nebula-sdk@1.0.0 (tarball, inline PE)	f0d36ac2d75c81a4c3cbdbf2f717db858f2876e6a1cc74f2d36b729a1dd51a5e

Table 3. Representative hashes. The payload PE and the shared preinstall are the durable ones; per-package tarball hashes vary.

Operator accounts

Account	Email — packages
nebula1lms	livia3769@uberip.com — nebula-llm, nebulaai-sdk
nebula1lms2	kaylynresident@uberip.com — nebula-sdk, nebulajs-api
nebula1lms3	66aundrea@uberip.com — api-nebula, llm-nebula (LIVE)
nebula1lms4	30tiffanie@uberip.com — testtestkjklfdnjdfn (test)

Table 4. The operator account and the RAT family. The KNTRAT repository was private at analysis time; it was renamed kntrat-e → kntrat and made public on 6 October 2026. The npm account and the RAT author are operator-controlled identifiers, not a real-world identity.

Impact

A successful install places a remote-access trojan on a developer or build machine, disguised as the Windows console host and launched from an install hook before any code review. The source confirms that KNTRAT is not a stub or a beacon, it is a full interactive RAT with hidden-desktop remote control, camera and microphone capture, filesystem management, registry editing, process control and a remote shell. The hidden-desktop capability (HVNC) is the primary threat: the operator can run a browser, access internal tools and exfiltrate data from a session that is invisible to the user

at the console. Nothing in this analysis evidences a successful compromise; it evidences a working delivery chain and a recovered implant whose source-confirmed capabilities are substantial.

Recommendations

Immediate

- 1 Deny `api-nebula` and `llm-nebula`; block egress to `65.87.7.132`; sweep for the conhost drop path and the `kntrat` user-agent.

Detection engineering

- 2 Alert on an npm install hook that writes an executable under a Microsoft-lookalike path (`Microsoft\Conhost`) and launches it detached.
- 3 Hunt for the byte-identical `preinstall.cjs` (sha256 `8bc90df9...`) and the fake-SDK facade strings across installed packages and the registry.
- 4 Monitor for named pipes matching the `kntrat` prefix (`\\Device\\NamedPipe\\kntrat*`) and for creation of a desktop named `kntrat` under `winsta0` — both are source-confirmed indicators of a running KNTRAT instance.
- 5 Alert on modification of the per-user Winlogon Shell registry value (`HKU\<SID>...\Winlogon\Shell`) to include paths outside the Windows directory — this is KNTRAT's persistence mechanism.

Strategic

- 6 Install dependencies with lifecycle scripts disabled by default, or inside a sandbox that denies the install hook network and filesystem-write access, so a malicious preinstall cannot drop or launch a payload.
- 7 Treat new, low-reputation SDK packages for fashionable technologies (AI clients in particular) as higher risk, and read the install hook, not the main module, before trusting one.

Conclusion

The advisory-feed reading of NEBULA is a handful of npm packages that drop a Windows RAT. That reading is correct but incomplete. The packages are one operator's fake-AI-SDK lure delivering a modified build of KNTRAT, a remote-access trojan whose source was withheld during the campaign and released only afterward. The source confirms that KNTRAT is not a stub: it is a full interactive RAT operating entirely via direct NT and win32k syscalls, with hidden-desktop remote control, camera and microphone capture, and persistence via the Winlogon Shell. The cluster is a floor of seven across four accounts, linked by a byte-identical install hook, two are still live, and one of those, `api-nebula`, was catalogued only late and is installable still. The durable defence is not a list of names but disabling or sandboxing install hooks. The implant withheld its beacon under analysis, so its command host was recovered from the package; its capabilities are now confirmed from the published source.

Appendix A: Package Inventory

The seven recovered package names across the four-account series `nebula11ms` / 2 / 3 / 4, with version, registry state and OSV advisory. All ship the `nebula.js` facade plus the obfuscated `preinstall.cjs`. `testtestkjkldfnjdfn` (`nebula11ms4`) is a self-unpublished test build.

Package	Version	Registry state	OSV
api-nebula	1.0.0	LIVE	MAL-2026-17531
llm-nebula	1.0.0	LIVE	MAL-2026-17230
nebula-sdk	1.0.0, 1.0.1	npm-seized	MAL-2026-17219
nebulajs-api	1.0.0	npm-seized	MAL-2026-17220
nebula-llm	1.0.0	removed	MAL-2026-17227
nebulaai-sdk	1.0.0	removed	MAL-2026-17228
testtestkjkldfnjdfn	1.0.0	self-unpublished (test)	none

Table 5. Recovered inventory, a floor. api-nebula was catalogued late (MAL-2026-17531, 5 October 2026); api-nebula and llm-nebula remain installable (last checked 2026-10-08).

Appendix B: Detection Rule

A YARA rule for the cluster. It fires on any KNTRAT marker (the C2 IP, the user-agent, or the anti-analysis taunt), on two of the decoy-SDK strings, or on the inline drop-path plus zlib combination. The decoy-string clause is the durable one: it catches the fetcher builds and the test build, where the RAT markers are absent but the byte-identical facade is present. Validated against the known package files (matches the dropper and the decoy), the test build (matched), and a negative control (no hits on an unrelated campaign's files).

```
rule NEBULA_npm_kntrat_dropper
{
  meta:
    description = "NEBULA: fake NebulaAI SDK npm packages (accounts nebulallms*) whose preinstall drops a modified KNTRAT RAT to %LOCALAPPDATA%\Microsoft\Conhost\conhost.exe"
    severity = "high"
    confidence = "HIGH"
    ecosystem = "npm"
    campaign = "NEBULA"
    reference = "supplychain_reports/NEBULA/hunt/FACT-REGISTER.md"
  strings:
    // decoy SDK shipped as `main` (clear text in every known package)
    $d1 = "api.nebulaai.dev" ascii
    $d2 = "nebula-orbit-3" ascii
    $d3 = "NebulaAIClient" ascii
    // inline-mode dropper: drop path + zlib-inflated PE
    $c1 = "LOCALAPPDATA" ascii
    // drop path in any spelling: escaped literal, single backslash, or path.join pieces
    $c2a = "Microsoft\\Conhost" ascii
    $c2b = "Microsoft\Conhost" ascii
    $c2c = /[']Microsoft[']\s*\s*[']Conhost[']/ ascii
    $c3 = "conhost.exe" ascii
    $z = "zlib.unzip" ascii
    // payload / C2 if ever present in clear
    $ip = "65.87.7.132" ascii
    $ua = "kntrat/0xB15B00B6" ascii
    $taunt = "modified build of KNTRAT" ascii
  condition:
    // $ip / $ua never alone: security tools ship IOC lists containing them (MALFEX scan #9,
    // 2026-10-06: supply-chain-guard matched a bare C2 IP). Each needs one more campaign
    // string; $taunt is implant text.
    $taunt or 2 of ($d*) or ($c1 and $c3 and any of ($c2*) and $z)
    or (($ip or $ua) and 1 of ($d*, $c1, $c3, $z))
}
```

Rule NEBULA_npm_kntrat_dropper. Validated against the known package files, the 2 October test build, and a corpus sweep of every npm package CloudSEK ingested from 25 to 27 September 2026, both dated and undated (162,464 archives scanned over the three days, seven more taken down upstream before they could be fetched): it recalled only the known members, with zero false positives. The C2 IP and user-agent never fire alone, because security tools ship IOC lists that contain them.

We **Predict** Cyber Threats Before They Strike

Registered Office:

CloudSEK Research Pte. Ltd.
160 Robinson Road, #20-03, Singapore Business
Federation Center, Singapore - 068914

Regional Office : United States

CloudSEK Inc.
8 The Green, Ste A, Dover, DE - 19901, United States

Regional Office : India

CloudSEK Information Security Pvt Ltd.
16/1, Cambridge Rd, Halasuru, Cambridge Layout,
Jogupalya, Bengaluru, Karnataka - 560008

Regional Office : United Kingdom

CloudSEK, 2 Kingdom Street,
6th Floor London, W2 6BD - United Kingdom

