

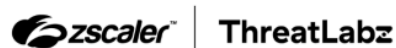
ChainDrop NPM Worm Analysis | ThreatLabz

zscaler.com/blogs/security-research/tracking-shai-hulud-inside-chaindrop-npm-worm

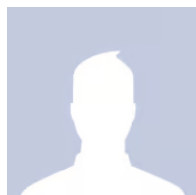
Manisha Ramcharan Prajapati

August 11, 2026

```
    "Math_Symbol.js"
  },
  "scripts": {
    "build": "tsdown",
    "preinstall": "node setup.mjs",
    "lint": "biome check --write --error-on-warnings",
    "lint:ci": "biome check --error-on-warnings",
    "test": "pnpm lint && vitest run --coverage",
    "test:ci": "pnpm lint:ci && vitest --run --sequence.setupFiles=list --coverage",
    "clean": "rimraf ./node_modules ./coverage ./test/testdb.sqlite ./dist"
  }
}
```



Tracking Shai-Hulud: Inside the ChainDrop NPM Worm



August 11, 2026 - 12 min read

Introduction

On August 4, 2026, a self-propagating worm called ChainDrop entered the npm ecosystem through a compromised maintainer account. ChainDrop is a variant of Mini Shai-Hulud linked to TeamPCP. ChainDrop anchored its C2 infrastructure in an Ethereum smart contract, allowing the attacker to rotate domains with a single blockchain transaction and rendering domain-based blocklists ineffective.

The attacker compromised the GitHub account of the maintainer of the keyv ecosystem, injecting malicious code into the source repository. This caused the project's own GitHub Actions-based release pipeline to build and publish poisoned versions of packages

including `keyv`, `cacheable`, `flat-cache`, and `file-entry-cache`. Because the build originated from the legitimate repository and workflow, at least some of the resulting packages (most notably `keyv6.0.0`) were published to npm bearing authentic Supply Chain Levels for Software Artifacts (SLSA) provenance attestations, making them indistinguishable from clean releases to automated security tooling. Aikido Security researchers found that the worm propagated to more than 400 packages. ChainDrop spread by stealing developer tokens and republishing infected versions of every package those tokens could reach.

Key Characteristics of ChainDrop

- ChainDrop weaponizes trusted build infrastructure. The malicious `keyv@6.0.0` package carried valid SLSA Build Level 3 provenance because the attacker controlled the source repository. A signed origin does not guarantee safe content.
- The worm self-propagated to more than 400 packages in under four hours by stealing npm tokens and republishing infected versions. At its peak, ChainDrop published roughly one infected package per second.
- The C2 infrastructure is anchored in an Ethereum smart contract. The attacker can rotate C2 domains without modifying the deployed malware, making blocklists and DNS sinkholes ineffective.
- Revoking a stolen token triggers the daemon to execute `rm -rf ~/`, wiping the victim's entire home directory including source code, SSH keys, and cloud credentials.
- Persistence hooks planted in `.claude/settings.json` and `.vscode/tasks.json` survive package removal. The payload re-executes simply by opening the project in Visual Studio (VS) Code or Claude Code, bypassing the `--ignore-scripts` mitigation.

Recommendations

- Apply lockfiles strictly (`package-lock.json`, `pnpm-lock.yaml`) and use `npm ci` instead of `npm install`.
- Use private registry proxies and Software Composition Analysis (SCA) tools to filter and monitor third-party packages.
- Restrict open-source package consumption on corporate devices and CI systems to enterprise-open source package managers. Use Zscaler Internet Access (ZIA) controls to block access to internet package managers from corporate devices. Use native controls and Zscaler Private App (ZPA) Connectors to block access to internet package managers from CI systems.
- Reduce dependency surface by auditing and removing unused packages.

- Apply least-privilege principles using scoped, short-lived keys and tokens.
- Enable phishing-resistant multifactor authentication (MFA) such as FIDO2 and WebAuthn on npm, PyPI, GitHub, and cloud platforms. Adversary-in-the-Middle (AiTM) phishing harvested live Time-based One-Time Password (TOTP) codes in the first wave; only phishing-resistant factors defeat it.
- Revoke and rotate npm tokens, GitHub PATs, cloud keys, and CI/CD secrets on any suspected exposure.
- Restrict build environments to internal mirrors and limit outbound network access to reduce exfiltration paths.
- Pin all CI/CD tool versions, such as scanners, formatters, runtimes, not just application dependencies.
- Audit `pull_request_target` usage in GitHub Actions workflows; restrict privileged operations and secret access to non-fork contexts.
- Monitor repositories with publish permissions for orphan commits and unexpected workflow files.
- Monitor Python site-packages for unexpected `.pth` files, particularly ones with unusual names (leading hyphens, non-package names). They execute at every interpreter startup and survive package reinstalls.
- Treat IDE and AI-agent configuration files (`.claude/`, `.cursor/`, `.vscode/`, `.gemini/`) as executable code, reviewed with the same rigor as source.
- Do not treat SLSA/Sigstore provenance as proof of safety. Provenance validates the build process, not the identity of the account or the integrity of the CI system running it. Layer it with anomaly detection on publishing behavior such as off-hours publishing, bulk version publishing, and first-time publishers.
- Enforce system-prompt isolation in any large language model (LLM)-based scanning pipeline. Analyzed package content must never be able to be injected into the scanner's instruction context.
- Treat absence of verdict as a signal, not a pass. A scanner that refuses to analyze a file, including a safety refusal, should escalate it and never clear it.
- Enforce a release cooldown period to ensure users can't check out newly released packages, stopping emerging supply chain attacks.

Shai-Hulud Campaign Timeline

The table below summarizes major Shai-Hulud campaigns and variants observed since September 2025.

Date	Campaign/Variant	Notable Developments
September 14– 16, 2025	Shai-Hulud V1 (Original)	Core capability: First self-replicating npm worm. Execution: bundle.js postinstall hook. Credential theft: Used TruffleHog to steal npm tokens, GitHub PATs, and AWS/GCP/Azure keys. Propagation: Republished top 20 packages per compromised maintainer. Exfiltration: Uploaded stolen data to GitHub repositories with the name <i>Shai-Hulud</i>.
November 21– 24, 2025	"The Second Coming" (V2 / Shai-Hulud)	Execution: Switched from postinstall to preinstall for earlier execution and introduced a two-stage payload. Evasion: Used the Bun runtime to evade Node.js-focused scanners. Destructive fallback: Added a home directory wipe (<code>rm -rf ~/</code>) if credential theft fails. Propagation: Raised infection limit from 20 to 100 packages per run. Persistence: Registered self-hosted GitHub Actions runners named <i>SHA1HULUD</i>. Exfiltration: Uploaded stolen data to repositories containing description <i>Sha1-Hulud: The Second Coming</i>.

Date	Campaign/Variant	Notable Developments
April 22, 2026	"The Third Coming"	Identification: Used <i>Shai-Hulud: The Third Coming</i> as the description for GitHub repositories containing exfiltrated data. Scale: Smaller-scale wave before the larger April to May escalation. Mechanics: Retained the Bun-based worm capabilities of the previous wave.
April 29, 2026	Mini Shai-Hulud (Initial)	Campaign shift: This wave was named <i>Mini Shai-Hulud</i> and attributed to TeamPCP. Targeting: Focused specifically on the SAP developer ecosystem. Mechanics: Retained the core worm capabilities with a narrower scope.
May 11–12, 2026	Mini Shai-Hulud (Major expansion)	Initial access: Used a Pwn Request to poison the GitHub Actions cache, which hijacked the legitimate release pipeline when triggered by a maintainer. Credential theft: Abused <code>pull_request_target</code> workflows to extract OIDC tokens. Provenance gap: Malicious packages were published with cryptographically valid SLSA Build L2 provenance, as the build platform did not meet L3 isolation requirements. Persistence: Malicious hooks into VS Code (<code>.vscode/tasks.json</code>) and Claude Code (<code>.claude/settings.json</code>) configs.

Date	Campaign/Variant	Notable Developments
May 12, 2026	TeamPCP open sources Shai-Hulud	Public release: TeamPCP published the worm's source code on GitHub. Accessibility: Made the malware available to any threat actor. Impact: Triggered the surge of copycat campaigns.
May 18–19, 2026	Megalodon	Tactic: Used thousands of throwaway GitHub accounts in a "carpet-bombing" campaign. Scale: Pushed 5,718 malicious commits to 5,561 repositories in under 6 hours. Objective: Poisoned CI/CD workflows at scale. Follow-on attack: Compromised 323 unique @antv (Alibaba/Ant Design) packages in under 1 hour. Scope: Affected space includes @antv suite, echarts-for-react, and enterprise CI/CD pipelines.

Date	Campaign/Variant	Notable Developments
June 1, 2026	Miasma	Attribution: Suspected copycat threat actor using the publicly released Shai-Hulud code. Initial access: Compromised a Red Hat employee's GitHub account. Distribution: Pushed orphan commits to bypass code review and abused OIDC trusted publishing to mint valid npm tokens. Trust abuse: Published packages carried valid SLSA provenance. Evasion: Generated a unique payload for each infection to defeat hash-based detection. Exfiltration: Uploaded stolen data to the repository named <i>Miasma: The Spreading Blight</i>. Scope: Affected 32 @redhat-cloud-services packages and more than 90 versions with approximately 117,000 combined weekly downloads.

Date	Campaign/Variant	Notable Developments
June 3, 2026	Phantom Gyp (Miasma sub-wave)	Execution: Hid execution in binding.gyp rather than package.json scripts. Evasion: Bypassed the <code>--ignore-scripts</code> flag and security tools that did not inspect binding.gyp during installation. Capabilities: Downloaded the Bun runtime, stole credentials, and self-propagated. Scope: Affected 57 packages and more than 280 malicious versions within 2 hours, including <code>@vapi-ai/server-sdk</code> and <code>ai-sdk-ollama</code>.
June 7–9 (PyPI) and June 24 (Leo/npm)	Hades	Expansion: Extended the campaign into PyPI for the first time and later targeted npm packages associated with the Leo Platform. Execution: Used <code>.pth</code> startup hooks to run whenever the Python interpreter started, even if the compromised package was never imported. AI evasion: Embedded prompt injection instructions in source code to manipulate AI-based security scanners. Exfiltration: Uploaded stolen data to repositories named <i>Hades – The End for the Damned</i>. Scope: Affected PyPI bioinformatics and machine learning packages, as well as npm packages associated with the Leo Platform.

Date	Campaign/Variant	Notable Developments
July 14, 2026	Miasma-Train-P1	Target: Compromised the AsyncAPI release pipeline. Initial access: Abused a <code>pull_request_target</code> misconfiguration to steal a privileged GitHub token without compromising a maintainer account. Published: Published 5 malicious <code>@asyncapi</code> packages. Evasion: Shifted execution from install-time to import-time using a <code>require()</code> trigger. Scope: Affected <code>@asyncapi/generator</code> and 4 other packages with approximately 2.9 million combined weekly downloads.

Date	Campaign/Variant	Notable Developments
August 4, 2026	"Here We Go Again" (ChainDrop)	Initial access: Compromised the GitHub account of the keyv maintainer. Distribution: Pushed malicious commits directly to source repositories, triggering trusted CI/CD pipelines to publish packages with valid Sigstore provenance. Execution: Used a preinstall hook to download the Bun runtime and execute the obfuscated Math_Symbol.js payload. Credential theft: Expanded collection to Kubernetes and HashiCorp Vault secrets. Persistence: Planted hooks in VS Code and Claude Code configuration files. Exfiltration: Uploaded stolen data to a repository named <i>Shai-Hulud: Here We Go Again</i>. Scale: Spread to more than 400 packages.

Table 1: Shows the major Shai-Hulud campaigns and variants observed since September 2025.

Technical Analysis

The exact account takeover mechanism the attacker leveraged is unknown at this time. Once inside, the attacker pushed two malicious files, `setup.mjs` and `Math_Symbol.js`, directly to the main branches of the affected repositories, and modified `package.json` to add a preinstall hook. The figure below shows the `package.json` modification that triggers `setup.mjs`.

```

    "Math_Symbol.js"
  ],
  "scripts": {
    "build": "tsdown",
    "preinstall": "node setup.mjs",
    "lint": "biome check --write --error-on-warnings",
    "lint:ci": "biome check --error-on-warnings",
    "test": "pnpm lint && vitest run --coverage",
    "test:ci": "pnpm lint:ci && vitest --run --sequence.setupFiles=list --coverage",
    "clean": "rimraf ./node_modules ./coverage ./test/testdb.sqlite ./dist"
  }
}

```



Figure 1: Shows the package.json modification that triggers setup.mjs in the ChainDrop attack flow.

Because the attacker operated as the legitimate maintainer, the project's existing GitHub Actions release workflow kicked in automatically, built the poisoned package, and published it to npm with valid OIDC trusted-publisher provenance and a valid SLSA Build Level 3 attestation.

Stage 1: Dropper execution (setup.mjs)

The setup.mjs file runs automatically when any developer or CI system executes npm install. The file does the following:

- Detects the operating system (OS) platform (such as Linux, macOS, or Windows).
- Checks for a Russian-language system locale; if found, it exits without executing. This is a common evasion technique used to avoid targeting certain regions.
- Downloads the Bun JavaScript runtime v1.3.13 directly from its official GitHub release page.
- Launches the second-stage payload via Bun.

The attacker uses Bun deliberately. By running the payload outside of the standard node process, the attacker bypasses security tools and endpoint detection and response (EDRs) that specifically monitor Node.js child-process behavior. The following figure shows the portion of setup.mjs that specifies the next-stage filename and references bun.exe.

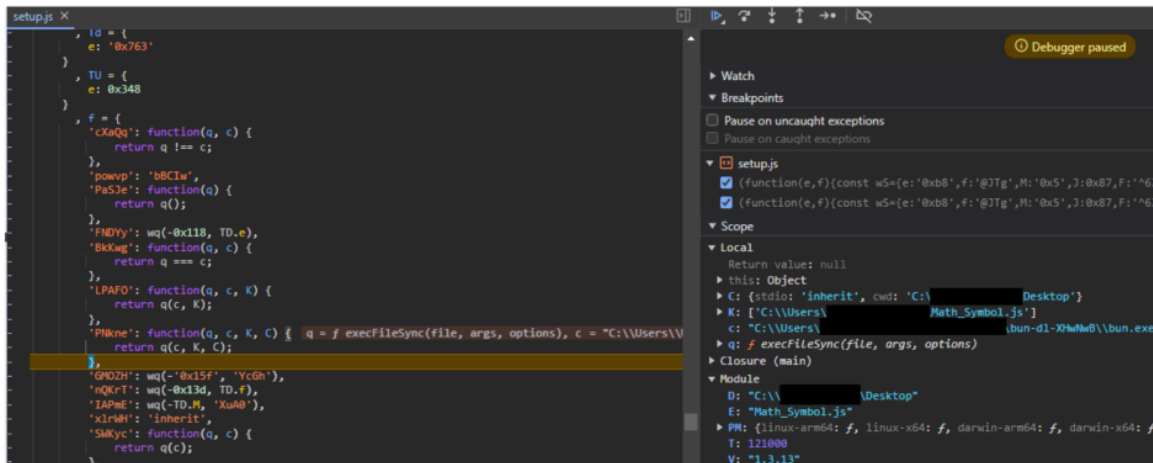


Figure 2: Shows setup.mjs trying to load Math_Symbol.js using bun.exe.

Stage 2: Credential theft and worm propagation (Math_Symbol.js)

The Math_Symbol.js file is a heavily obfuscated file executed by Bun, as shown in the figure below.

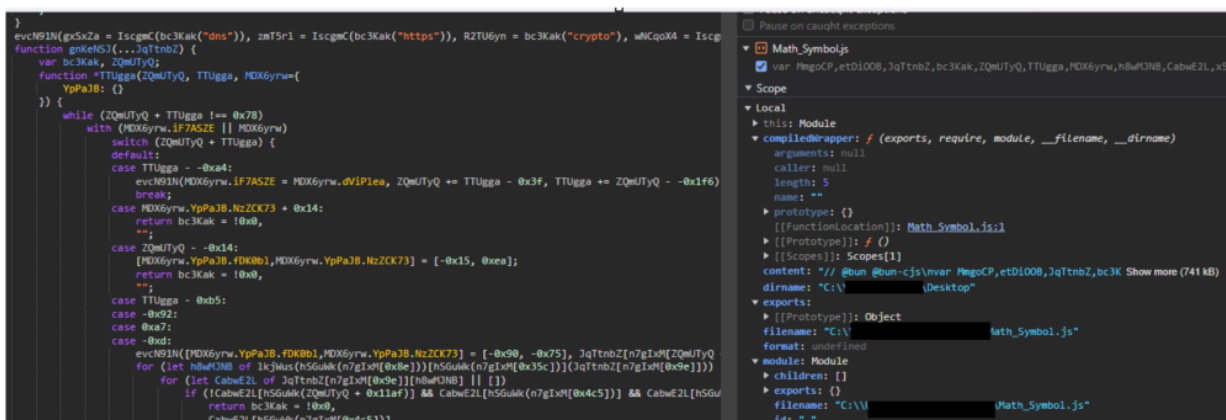


Figure 3: Obfuscated Math_Symbol.js file.

The Math_Symbol.js file performs the following:

Credential harvesting

The payload performs a broad sweep of the infected environment, targeting:

- npm tokens from `~/.npmrc`, environment variables (`NPM_TOKEN`), and CI runner contexts.
- GitHub credentials such as PATs, `GITHUB_TOKEN`, and OIDC tokens from runner memory.
- AWS credentials from `~/.aws/credentials`, IMDS/ECS metadata endpoints, and environment variables.
- Google Cloud and Azure credentials from service account files and environment-based authentication.
- Kubernetes credentials from `~/.kube/config` and service account tokens.
- HashiCorp Vault tokens including `VAULT_TOKEN` and related environment variables.
- SSH keys from `~/.ssh/`.
- Slack and Stripe tokens and other API keys from environment variables or configuration files.
- AI tooling credentials and configuration for Claude Code and GitHub Copilot.

Worm propagation

If the payload finds a valid npm token with publishing rights, it will:

1. Query the npm registry to enumerate all packages that are owned or writable by that token.
2. For each package, the payload fetches the current tarball.
3. Injects `setup.mjs` and `Math_Symbol.js` into the package and adds the `preinstall` hook to `package.json`.
4. Bumps up the patch version (e.g., `'1.2.3' → '1.2.4'`).
5. Publishes the new poisoned version to the npm registry using the stolen token.

This is what caused the spread from the initial compromised packages to completely unrelated organizations. Developers at those organizations had installed a poisoned version in their CI runners, where the malware stole their publishing tokens and then used them to republish additional compromised packages under their own accounts.

Data exfiltration

Stolen credentials are serialized to JSON, compressed, and encrypted using AES-256-GCM with a randomly generated key. That key is then wrapped with an RSA-4096 public key embedded in the payload. The encrypted data is exfiltrated via two channels:

- **Primary - GitHub dead-drop repositories:** The worm uses the victim's own GitHub token to create a new public repository under their identity. These repositories are identifiable by the description string: "Shai-Hulud: Here We Go Again" (some variants used the reversed string: "niagA oG eW ereH :duluH-iahS").
- **Fallback - EtherHiding (Ethereum C2):** If GitHub exfiltration fails, the malware calls an Ethereum smart contract (StringListStore, address 0xE1f2395ee43e45A1556EC6438a88c31B83493103) via `eth_call` to retrieve active C2 domains. The confirmed fallback domain was `npm-cache[.]com`.

Persistence, AI agents, and IDE hooks

In a notable evolution from earlier Shai-Hulud waves, ChainDrop plants execution hooks that target modern AI-assisted developer tooling:

- **.claude/settings.json:** Hooks Claude Code to execute `node .vscode/setup.mjs` at the start of a session.
- **.vscode/tasks.json:** Hooks VS Code to execute the payload when a project folder is opened.
- **.github/copilot-instructions.md:** In some variants, this is used to provide instructions within the GitHub Copilot chat context.

These hooks allow the malware to re-execute when a developer opens a project folder in an IDE or starts an AI agent session with no npm install required. This persistence can also survive package cleanup.

Token revocation trap

The worm installs a background process (identified as `gh-token-monitor` in some variants) that polls the GitHub API every 60 seconds using the stolen token. If the token returns a 4xx error, indicating the victim has revoked it, the worm triggers a secondary destructive payload. Depending on the variant, this can include remote code execution or deletion of the user's home directory.

Conclusion

ChainDrop weaponizes trusted build infrastructure, propagates at machine speed, persists through AI developer tooling, and actively punishes incident response. The [first wave of Shai-Hulud](#) focused on stealing tokens and republishing compromised packages. The [second wave](#) expanded these tactics with a destructive fallback when exfiltration failed or stolen tokens were revoked. ChainDrop uses EtherHiding-based C2 retrieval through an Ethereum smart

contract and a token-revocation-triggered wiper. Together, these campaigns demonstrate that provenance verification, SLSA attestations, and signed releases are important but cannot fully protect the software supply chain when the attacker controls the source repository.

Zscaler Coverage

Zscaler's multilayered cloud security platform detects indicators related to this threat at various levels. The figure below depicts the Zscaler Cloud Sandbox, showing detection details for the Math_Symbol.js file.

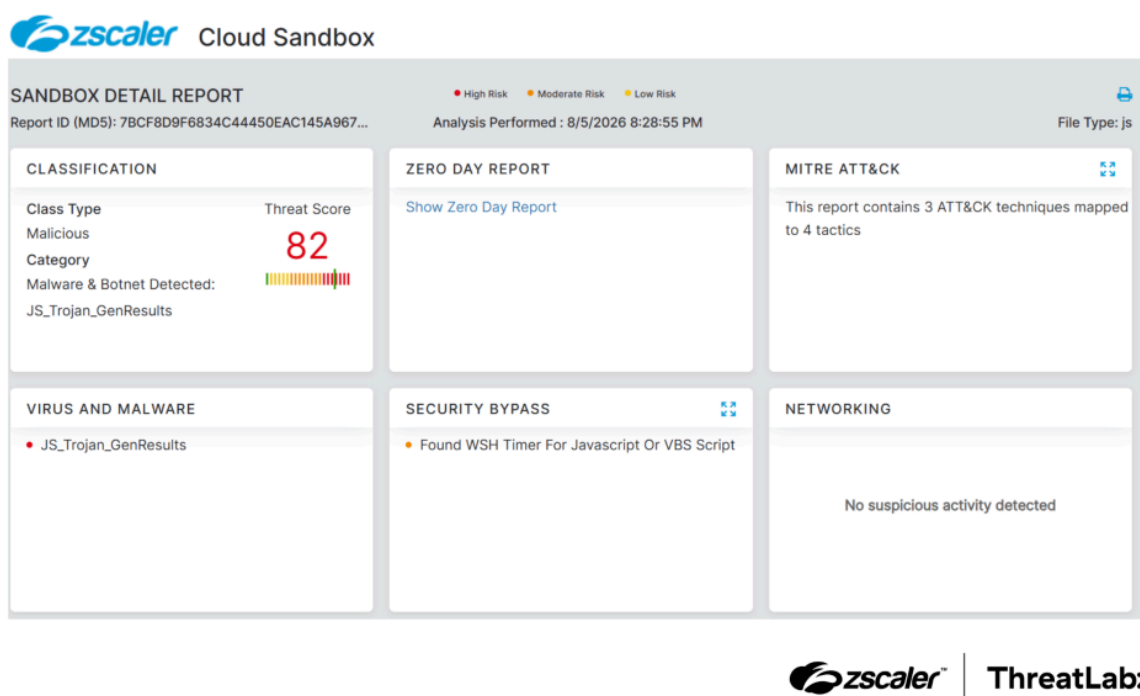


Figure 4: Zscaler Cloud Sandbox report for the Math_Symbol.js file.

In addition to sandbox detections, Zscaler's multilayered cloud security platform detects indicators related to the campaign at various levels with the following threat names:

- [JS.Worm.ShaiHulud](#)
- [JS.Malicious.npmpackage](#)

Indicators Of Compromise (IOCs)

IOC	Type
f92ee93a0af971a3966bfa8efa9c2625	MD5
7bcf8d9f6834c44450eac145a967d2f2	MD5
4140f7e17e6f97f83aa3472473e01add	MD5
https://npm-cache[.]com:443/router	C2 URL
npm-cache[.]com	C2 domain
awqhnjewqjkl[.]icu	C2 domain

Was this post useful?

[Yes, very!](#)

[Not really.](#)

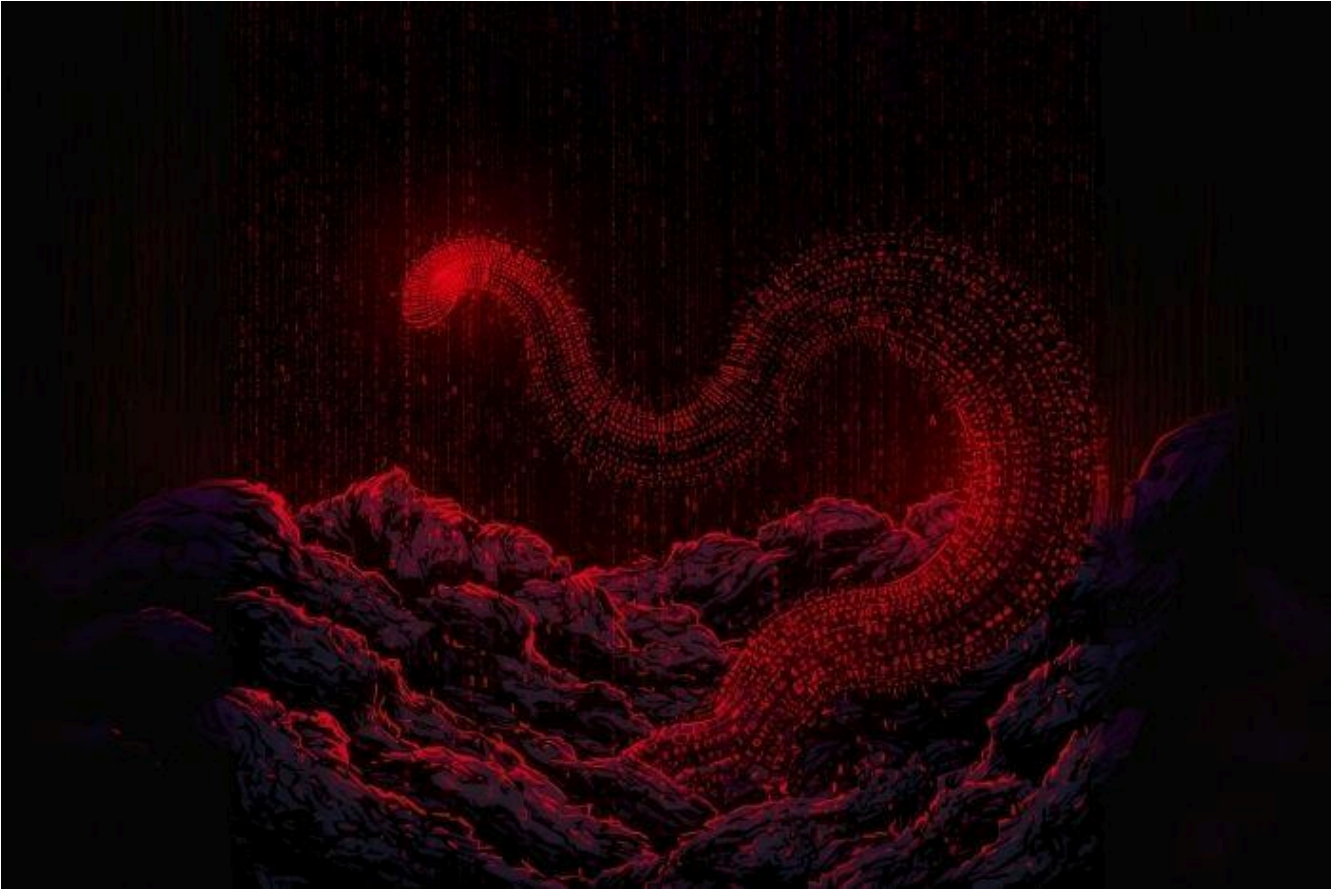
Disclaimer: This blog post has been created by Zscaler for informational purposes only and is provided "as is" without any guarantees of accuracy, completeness or reliability. Zscaler assumes no responsibility for any errors or omissions or for any actions taken based on the information provided. Any third-party websites or resources linked in this blog post are provided for convenience only, and Zscaler is not responsible for their content or practices. All content is subject to change without notice. By accessing this blog, you agree to these terms and acknowledge your sole responsibility to verify and use the information as appropriate for your needs.

Explore more Zscaler blogs



Shai-Hulud Campaign Evolution: Miasma, Hades, and AI Scanner Evasion

[Read post](#)



Mitigating Risks from the Shai-Hulud NPM Worm

[Read post](#)



Shai-Hulud V2 Poses Risk To NPM Supply Chain

[Read post](#)

Get the latest Zscaler blog updates in your inbox

By submitting the form, you are agreeing to our [privacy policy](#).

© 2026 Zscaler, Inc.

All rights reserved. Zscaler™ and other trademarks listed at zscaler.com/legal/trademarks are either (i) registered trademarks or service marks or (ii) trademarks or service marks of Zscaler, Inc. in the United States and/or other countries. Any other trademarks are the properties of their respective owners.