Shai-Hulud

Ransomwared CTI report

Erik Westhovens

18-09-2025

Ransomwared
CTI Report

Executive Summary

Shai-Hulud is an active, self-replicating malware campaign observed in mid-September 2025 targeting the npm JavaScript ecosystem. The malware trojanizes package releases, harvests developer and CI tokens, publishes stolen secrets publicly, and uses harvested npm/GitHub tokens to publish additional malicious releases – producing rapid, worm-like propagation. Early research indicates between ~150 and 400+ affected packages and several hundred infected versions as discovery progressed. This report summarizes impact, scope, TTPs, IOCs, timeline, recommended response steps, and longer-term structural mitigations.

Technical severity	🔴 **High**
Worm-like spread, credential theft, CI/CD compromise	

Business impact	🟠 **Medium-High**
Wide disruption & trust damage; remediation costly	

Likelihood	🔴 **High**
Automated propagation increases spread probability	

Overall risk	🔴 **Critical**
Requires immediate action and structural hardening	

Technical Overview

The Shai-Hulud malware is a novel self-replicating supply-chain worm first observed in September 2025 within the npm JavaScript ecosystem. It spreads by trojanizing package versions, executing malicious postinstall scripts during npm install, and harvesting developer/CI/CD/cloud credentials. Stolen tokens are both exfiltrated and used to automatically publish additional malicious packages under victim accounts. Within 48 hours, more than 150–400 packages were trojanized across multiple namespaces, including widely used open-source components.

Business Impact

Organizations consuming compromised packages are at risk of:

- Supply-chain contamination, as downstream software inherits malicious code.
- Credential leakage, with stolen GitHub/npm/cloud tokens made public in attacker-created repositories.
- Operational disruption, requiring mass token rotation, artifact rebuilds, and rapid security communications.
- Reputational risk, as even major vendors (e.g., CrowdStrike-related packages) were briefly impacted, amplifying trust concerns in the ecosystem.

Risk Outlook

Shai-Hulud demonstrates that worm-like propagation is feasible in modern software registries, magnifying supply-chain risk.

- Short-term risk: High likelihood of additional infected packages and compromised developer tokens.
- Medium-term risk: Potential pivot to cloud environments and CI/CD infrastructure via exposed credentials.

Long-term risk: Increased probability of copycat worms targeting other package ecosystems (PyPI, RubyGems, etc.).

The Shai-Hulud incident demonstrates how quickly a single compromised package can escalate into a large-scale supply-chain crisis. Its worm-like propagation exposed weaknesses in developer endpoint security, token management, and registry trust. While the immediate priority is containment and credential rotation, the broader lesson is the urgent need for structural improvements in software supply-chain governance. Organizations should view this not as an isolated incident, but as a preview of more sophisticated, automated threats targeting open-source ecosystems.

Threat Actor Overview

Attribution

At present, no specific threat actor has been formally attributed to the Shai-Hulud campaign. The behavior observed points less to a long-term, targeted espionage group and more toward an opportunistic actor or small team experimenting with worm-like propagation in the npm ecosystem. Unlike traditional supply-chain compromises, which typically aim for stealth and persistence, Shai-Hulud's methods favored rapid replication and visibility, suggesting the actor prioritized scale and disruption over discretion.

Some industry researchers have speculated that this could be the work of a financially motivated adversary testing new attack techniques. By harvesting and leaking developer, GitHub, and cloud credentials, the attacker not only exposed sensitive secrets but also created opportunities for follow-on exploitation (e.g., cloud resource hijacking, data theft, or resale of tokens on underground forums). Others point to the possibility of a proof-of-concept "chaos actor" — either hacktivist-driven or researcher-turned-malicious — given the overt style of publishing stolen data in public GitHub repositories named Shai-Hulud.

Motivation

The exact motivation remains unclear, but several plausible drivers exist:

- Credential theft and monetization: Stolen npm, GitHub, and cloud tokens are valuable for lateral attacks, cloud exploitation, and underground resale.
- Demonstration of capability: The worm-like propagation itself may have been a deliberate show of force to highlight weaknesses in open-source ecosystems.
- Disruption and reputational damage: By briefly compromising packages from major vendors, the campaign undermined trust in open-source supply chains.
- Exploration of automation: The high level of automation (auto-publishing trojanized updates, creating public repos, scanning secrets) suggests the actor was testing large-scale propagation frameworks that could be adapted to future operations.

Tactics and Style

The threat actor's style is notable for being aggressive and noisy:
Public exposure of stolen secrets instead of covert exfiltration.
Rapid version bumping and mass publishing, rather than stealthy backdoors.
Heavy reliance on automation, with minimal manual targeting or customization.
This makes the campaign distinct from APT groups, which usually seek stealth and persistence. Instead, Shai-Hulud aligns closer to opportunistic malware operators or disruptive "supply-chain worms" designed to stress-test global software dependencies.
Threat Actor Assessment
Skill level: Moderate to high. The actor demonstrated technical skill in manipulating npm publishing, GitHub workflows, and token theft, but did not use advanced custom malware beyond malicious JavaScript payloads.

Resources:
Limited. No evidence of a large infrastructure footprint or extended command-and-control; reliance on public services (GitHub repos, webhook endpoints).
Intent: Likely a mix of monetization, disruption, and experimentation.
Threat posture: High risk to open-source ecosystems due to worm behavior; unclear if the actor intends to escalate beyond credential harvesting.

Attack lifecycle & TTPs

High-level lifecycle (stage-by-stage)

1. Initial compromise — malicious package release
The campaign begins with one or more trojanized npm package releases (first public sighting tied to `@ctrl/tinycolor@4.1.1`), where the attacker supplies a release containing a bundled `bundle.js` payload. These malicious releases are published to the public npm registry and are pulled by developer workstations or CI during normal installs.

2. Execution on developer/CI host
When an infected package is installed, its `postinstall` (or other install-time) JavaScript executes on the host. The payload is large and capable: it executes node scripts, spawns child processes, and installs/uses legitimate secret-scanning tools (e.g., TruffleHog) to search for credentials and tokens on the host and in cloned repositories.

3. Credential & artifact harvesting
The malware enumerates common local token locations, environment variables, git credential files, and CI configuration to harvest npm publish tokens, GitHub personal/access tokens, and cloud credentials (AWS/Azure/GCP). It may also query cloud metadata endpoints to discover instance/role credentials when running in cloud-based CI/build agents.

4. Exfiltration & public disclosure
Harvested secrets are exfiltrated outward. In multiple observed cases, the actor published harvested data into public GitHub repositories named `Shai-Hulud` (or posted to attacker-controlled webhook endpoints), thereby making tokens and secrets publicly visible and reusable by the attacker or any third party who finds them.

5. Automated propagation (worm behavior)
Crucially, stolen npm/GitHub tokens are used programmatically to pack, republish, or modify other packages maintained by the victim account. The malware contains functions (observed as `NpmModule.updatePackage`-style routines) that inject the `bundle.js` payload into other packages, rebuild the package, increment version numbers, and publish new trojanized releases — enabling rapid, automated lateral spread across maintainers' namespaces. This behavior is what makes the incident "wormable".

6. Persistence & CI footholds
The campaign has been observed creating or abusing GitHub Actions/workflows to maintain access or to automate further propagation and exfiltration. Attackers also made unexpected repository visibility changes (private → public) to expose harvested data. These workflow artefacts and visibility changes act as additional persistence and distribution mechanisms.

7. Escalation (cloud / environment takeover)
If cloud or CI tokens permit, attackers can enumerate cloud resources, access storage/credentials, or pivot to production build pipelines. The combination of stolen cloud credentials and CI access creates a high-impact escalation path from a single compromised developer machine to broad cloud or production compromise.

Key TTPs mapped to MITRE ATT&CK (select, high-value mappings)

- **T1195 — Supply Chain Compromise:** malicious packages published to npm. JFrog
- **T1059.007 — Command and Scripting Interpreter: JavaScript:** malicious JS executed during package installation. ReversingLabs
- **T1552 / T1081 — Credentials in Files / Credentials from Web Browsers / Files:** harvesting tokens from local files, environment variables, and CI configs. StepSecurity
- **T1041 / T1537 — Exfiltration Over Web Services / Exfiltration to Cloud Storage:** publishing harvested secrets to public GitHub repos and webhook endpoints. Krebs on Security
- **T1609 — Container and Repository Artifacts (Persistence):** trojanized package versions persist in the npm registry and malicious GitHub Actions/workflows. Sonatype
- **T1078.004 — Valid Accounts: Cloud Accounts:** misuse of harvested CI/cloud credentials to access cloud resources (metadata endpoint abuse). Arctic Wolf

Notable behavior patterns & IOCs (operational)

- **Malicious package versions**: e.g., `@ctrl/tinycolor@4.1.1/4.1.2` were early, confirmed trojanized releases. Vendor scanners later enumerated 150–200+ packages and several hundred infected versions. JFrog+1
- **Large JavaScript payload** (bundled `bundle.js`, >1–3 MB) that calls secret-scanning tools and performs network exfiltration. Infosecurity Magazine
- **Public repository creation**: GitHub repos named `Shai-Hulud` containing JSON/text dumps of harvested secrets. Krebs on Security
- **Automated `npm publish` spikes**: sudden version bumps and multiple publishes across packages owned by the same publisher in rapid succession. JFrog
- **Malicious/unauthorized GitHub Actions**: presence of unexpected workflow YAMLs used for persistence/exfiltration. getsafety.com

Detection opportunities (practical detections)

- **Registry/Publishing anomalies:** alert on unusual mass `npm publish` events from a single account, especially across unrelated packages and with fast succession. Integrate with publisher account monitoring. JFrog
- **Endpoint behavioral detections:** detect `node` processes invoked from package install contexts spawning network connections, or running `trufflehog`/secret-scanning processes unexpectedly. Monitor process trees and command-lines during `npm install`. Semgrep
- **GitHub API & workflow monitoring:** alert on creation of new public repos (especially named `Shai-Hulud` pattern), unauthorized workflow files, or tokens used from unusual IPs/locations. Krebs on Security+1
- **Cloud metadata queries:** detect unexpected outbound requests to 169.254.169.254 or other cloud metadata endpoints from developer/CI hosts. Arctic Wolf
- **Secret & token scanning:** run proactive secret scanning across org repos and CI configurations; any token flagged as present in a compromised environment should be rotated immediately. StepSecurity

Recommended immediate mitigations for each lifecycle stage

- **Prevent installation of flagged packages:** block or cache suspect package versions at your internal npm proxy/mirror; whitelist vetted packages. Sonatype
- **Harden install-time execution:** adopt policies that restrict or sandbox package install-time scripts (e.g., `npm ci --ignore-scripts` where feasible for build agents).
- **Rotate and short-lived tokens:** treat all developer and CI tokens as compromised if a host ran an infected package; revoke and reissue from a vault with least privilege. StepSecurity
- **Hunt & isolate:** collect forensic artifacts from any host that installed a flagged package; look for evidence of GitHub API usage and unexpected publish events. ReversingLabs

Sources / corroboration

The lifecycle and TTP descriptions above are drawn from multiple vendor analyses and community reporting, including JFrog's malware scanner findings, Palo Alto Unit 42, ReversingLabs, Sonatype, and investigative reporting by KrebsOnSecurity. These sources provide technical validation for the stages, behaviors, and IOCs described.

Impact Assessment

Technical Impact

1. **Developer and CI/CD system compromise**
   - Any developer or build system that executed an infected package immediately ran malicious postinstall scripts.
   - These scripts scanned local environments for credentials, including npm tokens, GitHub PATs, SSH keys, and cloud provider tokens.
   - In cloud-based CI/CD environments, the malware queried **metadata service endpoints** (e.g., AWS, Azure, GCP), enabling theft of temporary credentials and role-based access keys.
2. **Credential exposure and exfiltration**
   - Stolen tokens were not only exfiltrated but also **published openly** in GitHub repositories named *Shai-Hulud*, exposing sensitive credentials to the entire internet.
   - This public exposure means secrets could be reused not just by the original attacker but by **any third party** monitoring GitHub activity.
3. **Automated supply-chain contamination**
   - With valid npm publish tokens, the malware incremented versions and re-published trojanized packages, contaminating multiple projects in parallel.
   - This worm-like spread significantly increases the **attack surface**, as even cautious organizations could inherit malicious code simply by updating dependencies.
4. **Potential for escalation**
   - Access to CI/CD tokens and cloud credentials creates a pivot point for deeper compromise, including deployment of malicious code into production pipelines, theft of proprietary software, or abuse of cloud infrastructure.

Business and Organizational Impact

1. **Operational disruption**
   - Emergency response required **immediate token rotation**, CI/CD audits, forensic investigations, and artifact rebuilds.
   - For organizations with extensive npm dependency trees, this can stall build pipelines and slow down product delivery.
2. **Reputational risk**
   - Prominent vendor namespaces (e.g., CrowdStrike-related packages) were briefly compromised, amplifying the visibility of the attack and raising doubts about open-source trustworthiness.
   - Customers may question the security posture of vendors who rely on npm or fail to communicate promptly.
3. **Financial cost**
   - Expenses arise from incident response, remediation, developer downtime, and possible customer compensation.
   - If stolen tokens enabled cloud misuse, organizations could face **direct infrastructure costs** (e.g., cryptomining, service abuse).
4. **Legal and compliance exposure**
   - Public release of secrets may constitute a **data breach**, with regulatory implications (GDPR, HIPAA, PCI-DSS, etc.) depending on the nature of the exposed credentials and data.

Risk Assessment Conclusion

- **Technical severity:** High — worm-like spread, credential theft, and potential cloud/CI pivot.
- **Business impact:** Medium-High — reputational damage and operational costs, potentially escalating to regulatory impact.
- **Likelihood:** High — the campaign relied on automation and widely used ecosystems, ensuring rapid spread.
- **Overall Risk:** 🔴 **Critical** — organizations must treat this as a major supply-chain incident requiring both immediate containment and structural security improvements.

Sector-Specific Threat Analysis

The Shai-Hulud campaign represents a significant **supply-chain compromise** in the npm ecosystem. Because npm packages are widely used across industries, the impact of this incident is not uniform — some sectors face far higher exposure than others.

1. Technology & SaaS Providers

Exposure: Extremely High

- Technology companies and SaaS vendors heavily rely on open-source JavaScript libraries in their web applications and internal tooling.
- Many use automated CI/CD pipelines with npm dependencies integrated directly into production builds.
- Compromise here can lead to **malicious code execution in customer-facing applications**, theft of proprietary source code, and disruption of software delivery pipelines.

Implications:

- High risk of customer data leakage.
- Erosion of trust in software-as-a-service products if supply-chain contamination is confirmed.

2. Financial Services (Banking, FinTech, Insurance)

Exposure: High

- FinTech companies often leverage npm libraries for payment portals, web dashboards, and APIs.
- Stolen credentials (GitHub, cloud, CI/CD) could expose **sensitive financial transaction data** or allow unauthorized modification of financial applications.
- Even short-lived disruption in build or deployment systems can lead to **operational outages** with customer impact.

Implications:

- Increased risk of regulatory scrutiny (PCI DSS, SOX).
- High reputational damage potential if compromised software reaches customers.

3. Critical Infrastructure & Energy

Exposure: Medium-High

- While operational technology (OT) systems may not rely directly on npm, supporting IT systems, vendor dashboards, and monitoring software often do.
- If infected, these systems could provide **indirect entry points** into critical environments.

Implications:

- Risk of attacker footholds in supporting IT networks.
- Potential cascade effects if compromised software manages industrial processes or SCADA dashboards.

4. Healthcare & Life Sciences

Exposure: Medium

- Healthcare platforms and patient-facing portals frequently use open-source components for web interfaces.
- A compromise here risks **sensitive patient data exposure** and violation of HIPAA/GDPR regulations.

Implications:

- Strong compliance impact (reportable data breach).
- Patient safety concerns if compromised applications are tied to medical workflows.

5. Government & Public Sector

Exposure: Medium

- Many government portals and public-service apps rely on common npm packages.
- A successful compromise could **erode public trust** in digital services and potentially expose sensitive citizen information.

Implications:

- Potential for nation-state exploitation of harvested government developer tokens.
- Heightened political and diplomatic consequences if attackers weaponize public-sector apps.

6. Small and Medium Enterprises (SMEs)

Exposure: Variable

- SMEs often lack strict supply-chain controls and may unknowingly integrate infected packages.
- With limited security budgets, SMEs are less likely to have **dedicated incident response capacity**, making them vulnerable to prolonged compromise.

Implications:

- High relative business disruption, even if technical scope is smaller.
- Financial losses from downtime, remediation costs, and reputational damage.

Cross-Sector Observations

- **Universal exposure**: Any organization using npm is at some level of risk.
- **Greatest risk**: Sectors with heavy CI/CD automation and strong dependency on open-source libraries (Technology, SaaS, FinTech).
- **Secondary risk**: Sectors where regulatory requirements amplify the consequences of credential leakage (Healthcare, Finance, Public Sector).

Strategic Outlook

Shai-Hulud underlines a systemic problem: **ecosystem-level weaknesses** in open-source package registries can ripple across industries simultaneously. Sectors with heavy digital dependence must urgently adopt **structural defenses** — internal package mirrors, artifact signing, continuous dependency scanning, and enforced credential hygiene — to limit blast radius in future supply-chain worm events.

References & Links

- JFrog Security Research — *“Shai-Hulud npm supply chain attack – new compromised packages detected”* (164 malicious packages, 338 infected versions)
  https://jfrog.com/blog/shai-hulud-npm-supply-chain-attack-new-compromised-packages-detected/ JFrog
- KrebsOnSecurity — *“Self-Replicating Worm Hits 180+ Software Packages”* (naming Shai-Hulud, CrowdStrike-namespace briefly impacted)
  https://krebsonsecurity.com/2025/09/self-replicating-worm-hits-180-software-packages/ Krebs on Security
- Wiz Research — *“Shai-Hulud: Ongoing Package Supply Chain Worm Delivering Data-Stealing Malware”*
  https://www.wiz.io/blog/shai-hulud-npm-supply-chain-attack wiz.io
- Arctic Wolf Labs — *“Wormable Malware Causing Supply Chain Compromise of npm Code Packages”*
  https://arcticwolf.com/resources/blog-uk/wormable-malware-cause-supply-chain-compromise-of-npm-code-packages/ Arctic Wolf
- Sonatype Security Research Team — *“Ongoing npm Software Supply Chain Attack Exposes New Risks”* (tracking 180+ compromised packages)
  https://www.sonatype.com/blog/ongoing-npm-software-supply-chain-attack-exposes-new-risks Sonatype
- Checkmarx Zero Research — *“NPM Hit By Shai-Hulud, The Self-Replicating Supply Chain Attack”*
  https://checkmarx.com/zero-post/npm-hit-by-shai-hulud-the-self-replicating-supply-chain-attack/ Checkmarx
- Infosecurity-Magazine — *“Shai-Hulud Worm Prowls npm to Steal Hundreds of Secrets”*
  https://www.infosecurity-magazine.com/news/supply-chain-worm-hundreds-npm/ Infosecurity Magazine
- Palo Alto Networks Unit 42 — *“‘Shai-Hulud’ Worm Compromises npm Ecosystem in Supply Chain Attack”*
  https://unit42.paloaltonetworks.com/npm-supply-chain-attack/ Unit 42

MITRE ATT&CK Mapping

Initial Access

- **T1195 – Supply Chain Compromise**
  Malicious npm packages published and consumed during normal dependency installs.
  → Entry point for Shai-Hulud into developer and CI/CD environments.

Execution

- **T1059.007 – Command and Scripting Interpreter: JavaScript**
  Malicious JavaScript (bundle.js, postinstall hooks) executed on developer/build systems.
  → Triggered automatically by `npm install`.

Persistence

- **T1609 – Container and Repository Artifacts**
  Trojanized versions persisted in the npm registry; malicious GitHub Actions/workflows added for persistence.
  → Ensures continued re-infection even after clean builds.

Privilege Escalation / Discovery

- **T1082 – System Information Discovery**
  Collecting host and environment details on developer machines/CI nodes.
- **T1057 – Process Discovery**
  Enumerating running processes to identify tools/agents present.
- **T1087 – Account Discovery**
  Harvesting user accounts and tokens from configuration files.

Credential Access

- **T1552 – Unsecured Credentials**
  Harvesting secrets from environment variables, .npmrc, and CI configs.
- **T1081 – Credentials in Files**
  Scanning local files for GitHub, npm, and cloud tokens.
- **T1528 – Steal Application Access Tokens**
  Exfiltration of npm and GitHub tokens from developer endpoints.

Lateral Movement / Propagation

- **T1078 – Valid Accounts (Cloud/Developer)**
  Using stolen npm publish tokens and GitHub tokens to move laterally into other packages/repos.
- **T1021.004 – Remote Services: SSH** (possible)
  If SSH keys were harvested, could allow lateral pivot to other hosts.

Exfiltration

- **T1041 – Exfiltration Over C2 Channel**
  Sending secrets to attacker-controlled webhook endpoints.

- **T1567.002 – Exfiltration to Cloud Storage**
  Publishing harvested secrets into public GitHub repositories (named "Shai-Hulud").

Impact

- **T1485 – Data Destruction / Data Exposure**
  Private repos made public, sensitive data leaked to GitHub.
- **T1499 – Endpoint Denial / Resource Hijacking** (potential)
  Cloud tokens could be misused for cryptomining or service abuse.
- **T1589 – Gather Victim Identity Information**
  Collecting developer identities and tokens to propagate further.

Summary Mapping Table

ATT&CK Phase	Technique ID	Technique Name	Shai-Hulud Usage
Initial Access	T1195	Supply Chain Compromise	Malicious npm packages
Execution	T1059.007	Command/Scripting: JavaScript	Postinstall execution
Persistence	T1609	Container/Repository Artifacts	Trojanized packages, GitHub workflows
Discovery	T1082, T1057	System & Process Discovery	Host/CI enumeration
Credential Access	T1552, T1081, T1528	Unsecured Credentials, Credentials in Files, Steal Tokens	Harvesting npm/GitHub/cloud tokens
Lateral Movement	T1078	Valid Accounts (Cloud/Dev)	Using stolen tokens to republish packages
Exfiltration	T1041, T1567.002	Exfiltration over C2, to Cloud Storage	Webhook + GitHub repo leaks
Impact	T1485, T1499, T1589	Data Exposure/Resource Hijacking/Identity Gathering	Publishing secrets, potential cloud misuse

Known IOCs

Type	Indicator	Notes / Source
Packages & Versions (npm)	`@ctrl/tinycolor@4.1.1`, `@ctrl/tinycolor@4.1.2`	Among earliest identified compromised versions. Sonatype+4wiz.io+4JFrog+4
	`@ctrl/deluge@7.2.2`, `@ctrl/deluge@7.2.1`	Affected versions per Wiz/StepSecurity/Snyk. Sonatype+3wiz.io+3Snyk+3
	`@ctrl/golang-template@1.4.3`, `@ctrl/golang-template@1.4.2`	Same. StepSecurity+3wiz.io+3Snyk+3
	`@ctrl/magnet-link@4.0.4`, `@ctrl/magnet-link@4.0.3`	
	`@ctrl/ngx-codemirror@7.0.2`, `@ctrl/ngx-codemirror@7.0.1`	
	`@ctrl/ngx-csv@6.0.2`, `@ctrl/ngx-csv@6.0.1`	
	`@ctrl/ngx-emoji-mart@9.2.2`, `@ctrl/ngx-emoji-mart@9.2.1`	
	`@ctrl/ngx-rightclick@4.0.2`, `@ctrl/ngx-rightclick@4.0.1`	
	`@ctrl/qbittorrent@9.7.2`, `@ctrl/qbittorrent@9.7.1`	
	`@ctrl/torrent-file@4.1.2`, `@ctrl/torrent-file@4.1.1`	
	`@ctrl/transmission@7.3.1`	Single version listed. wiz.io+1
	`@ctrl/ts-base32@4.0.2`, `@ctrl/ts-base32@4.0.1`	
	`encounter-playground@0.0.5`	
	`ngx-bootstrap` — versions `20.0.4`, `20.0.5`, `20.0.6`, `19.0.3`, `18.1.4`	These ngx-bootstrap versions had malicious postinstall + bundle.js. Snyk+1
	`ng2-file-upload` (versions impacted, but details thinner)	As per Snyk; tagged as "looking affected." Snyk+1
	`@nativescript-community/*` namespace (multiple packages & versions)	Many in this namespace affected — e.g. material-bottomsheet, ui-collectionview, text, etc. Snyk+1
	CrowdStrike-namespace packages: e.g. `@crowdstrike/commitlint`, `@crowdstrike/foundry-js`, `@crowdstrike/logscale-file-editor`, etc.	Reported for compromise in Snyk's findings. Snyk

Hashes / Files / Scripts	Indicator	Notes
SHA-256 of a malicious JS file: **`46faab8ab153fae6e80e7cca38eab363075bb524edd79e42269217a083628f09`**	Identified in some of the `bundle.js` files across packages. wiz.io	
`bundle.js` file in package root, large (~3-3.7MB, obfuscated, Webpack-bundled)	Common payload file dropped/used in postinstall hooks. StepSecurity+2wiz.io+2	

Script names on disk: `/tmp/processor.sh`	Payload component involved in creating malicious branches/actions. wiz.io	
Script name `/tmp/migrate-repos.sh`	Used in migration of private repos to public ones. wiz.io	

Endpoints / Webhooks / Repos	Indicator	Notes
`webhook[.]site/bb8ca5f6-4175-45d2-b042-fc9ebb8170b7`	Used to exfiltrate secrets via GitHub actions/workflows. wiz.io+2Sonatype+2	
Public GitHub repos named `Shai-Hulud`	Contain JSON dumps of harvested credentials and environment data. wiz.io+2Sysdig+2	
GitHub Actions workflow file(s): patterns like `.github/workflows/shai-hulud-workflow.yml`	Used for exfiltration + workflow injection into repositories. wiz.io+1	

Behavioral / Environmental Indicators	Indicator	Notes
Postinstall hook in `package.json` invoking `node bundle.js`	Signature of compromised package. Sysdig+2Snyk+2	
Running of TruffleHog (or similar secret scanning) locally within install scripts	Used to find local secrets. wiz.io+1	
Querying cloud provider metadata endpoints (AWS / GCP / Azure) from developer/CI host	To harvest cloud-access credentials. Sysdig+1	
Creation of new branches named `shai-hulud` in victim repositories	To insert malicious workflows or trigger exfiltration. wiz.io	
Migration of private repos to public with `-migration` suffix and description "Shai-Hulud Migration"	Observed in some compromised accounts. wiz.io	
PublicEvent following CreateEvent (repo visibility change) in GitHub audit logs	Sign of repo being made public after creation. wiz.io	

This is a compiled list; treat it as evolving, not exhaustive.