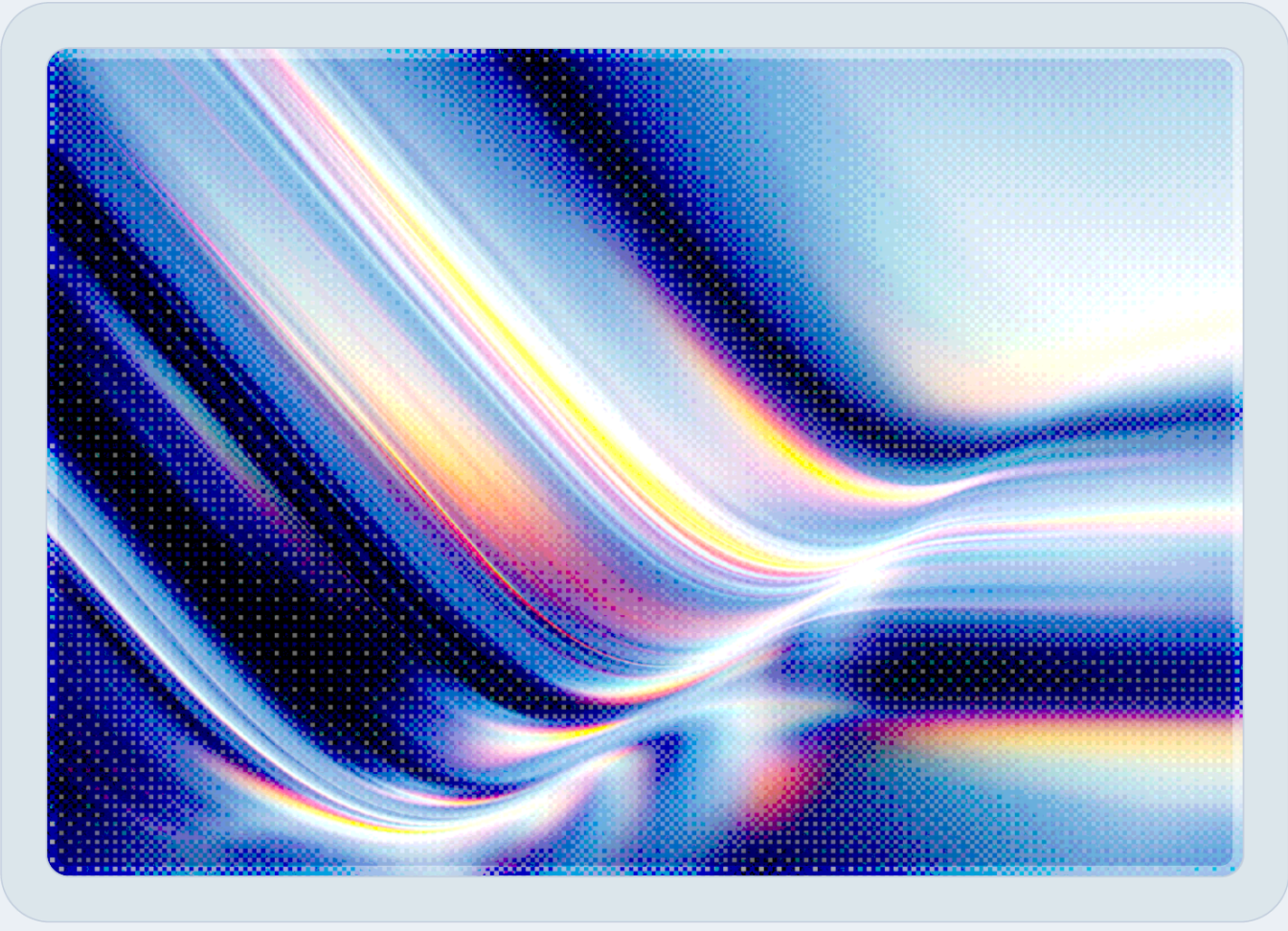
CSAI Foundation | Cloud Security Alliance

Miasma: Red Hat npm Supply Chain Worm

Multi-Cloud Credential Harvesting via GitHub Actions OIDC Abuse and Self-Propagating Malware

2026-06-03

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- On June 1, 2026, researchers at Wiz and multiple concurrent firms identified a supply chain compromise affecting at least 32 packages under the `@redhat-cloud-services` npm namespace, reaching an estimated 80,000 weekly downloads; Red Hat disclosed the incident as RHSB-2026-006 the same day [1][3].
- The attack was enabled by a single compromised employee GitHub account whose credentials had appeared in commercial infostealer logs as early as April 13, 2026—a gap of nearly seven weeks that illustrates the lag frequently observed between credential theft and weaponization in infostealer campaigns [5].
- Attackers abused GitHub Actions' OpenID Connect trusted-publishing mechanism to mint short-lived npm tokens and publish maliciously modified packages bearing forged SLSA provenance attestations, illustrating that supply chain integrity controls designed to eliminate long-lived secrets can themselves become an attack surface when the upstream identity provider is compromised [1][2].
- The malware payload, a new variant of the Mini Shai-Hulud family attributed to threat actor group TeamPCP—though copycat actors using the same publicly released codebase cannot be ruled out [1]—functions as a worm: it harvests credentials across GitHub, AWS, Azure, GCP, HashiCorp Vault, Kubernetes, and local developer environments, then uses stolen tokens to republish backdoored packages to packages the victim controls, extending the blast radius beyond the initial `@redhat-cloud-services` namespace [1][2].
- Organizations that consumed any `@redhat-cloud-services` package between approximately 10:53 UTC and the package removals on June 1 should treat all credentials present in that environment—npm tokens, cloud identity tokens, SSH keys, CI/CD secrets—as potentially compromised and rotate them immediately; Red Hat's advisory (RHSB-2026-006) states that no customer action is required based on current investigation findings, though independent security researchers recommend proactive rotation given the credential exposure scope [1][3].

Background

On the morning of June 1, 2026, automated supply chain security tooling and concurrent manual review by teams at Wiz, Aikido Security, StepSecurity, and others flagged anomalous preinstall scripts in multiple packages published under the `@redhat-cloud-services` npm scope [1][4][6]. The `@redhat-cloud-services` namespace hosts the frontend components, API clients, and platform tooling that power the Red Hat Hybrid Cloud Console and are consumed both internally and by the broader Red Hat developer ecosystem [3].

Investigation established that a Red Hat employee's GitHub account had been compromised and used to push orphan commits—commits that bypass the normal branch-and-review workflow—into three repositories in the RedHatInsights GitHub organization: `frontend-components`, `javascript-clients`, and `platform-frontend-ai-toolkit` [1]. The injected commits introduced malicious GitHub Actions workflow files that triggered on any branch push, requested an OpenID Connect identity token with the `id-token: write` permission, and used that token to authenticate directly to npm's trusted-publishing endpoint. This mechanism, designed to eliminate long-lived publish tokens as a security control, instead allowed attackers to mint valid, short-lived credentials on demand within the compromised CI/CD environment and publish backdoored package versions that carried legitimate SLSA provenance attestations—artifacts that attest to build provenance and are widely used as a trust signal, even though they verify the build process rather than the integrity of the content or the authorization of the publisher [2].

At minimum 32 packages were modified across two attack waves occurring at 10:53 UTC and 13:44 UTC on June 1, spanning over 90 version releases [2]. Affected packages include `@redhat-cloud-services/frontend-components` (versions 7.7.2–7.7.5), `@redhat-cloud-services/compliance-client` (versions 4.0.3–4.0.6), `@redhat-cloud-services/rbac-client` (versions 9.0.3–9.0.6), and 29 additional packages in the same namespace [1][2].

The campaign carries the name "Miasma: The Spreading Blight," a label the malware applied to the attacker-controlled GitHub repositories it created to exfiltrate stolen credentials. Researchers linked it to the Mini Shai-Hulud malware family, a credential-stealing worm codebase previously associated with threat actor group TeamPCP, though the use of publicly released malware code means copycat actors cannot be ruled out on the basis of code similarity alone [1].

Dark web intelligence firm CybelAngel subsequently reviewed its infostealer log archive and confirmed that the Red Hat employee's GitHub credentials—including an active session cookie capable of bypassing multi-factor authentication—had appeared in stealer logs on April 13 and again on May 15,

2026. This placed the initial compromise nearly seven weeks before the attack, consistent with the broader pattern in which credentials harvested by infostealer malware are sold and redistributed before eventual weaponization [5].

Security Analysis

The Payload: Four Layers Deep

The malicious package modification centers on a weaponized `preinstall` script added to `package.json` that runs automatically the moment a developer or CI system executes `npm install`, before any application code executes [2][4]. The entry point invokes a primary payload file (`_index.js`) that is 4.29 megabytes in size—orders of magnitude larger than the kilobyte-range size typical for utility scripts of this type—and conceals its logic under four sequential layers of obfuscation [2].

The outermost layer uses ROT-based character-code array decoding resolved through `eval()`. Once decoded, an AES-128-GCM decryption step unwraps the next stage, which also downloads the Bun JavaScript runtime from its official infrastructure—using a legitimate binary to host the subsequent execution. The third layer applies obfuscator.io string-array protection with rotated arrays and alias wrappers, and the innermost layer uses a custom PBKDF2-HMAC-SHA-256 cipher with 200,000 iterations to protect the most sensitive string constants. This cascade has the effect of falling outside the telemetry scope of Node.js-focused monitoring tools and evading endpoint detection rules written against common npm attack patterns, suggesting a deliberate detection-evasion design [2]. Before executing sensitive operations, the payload actively probes for the presence of CrowdStrike, SentinelOne, Carbon Black, and StepSecurity Harden-Runner; it adapts its behavior accordingly [6][7].

Notably, the Bun runtime is installed to a path beginning with `.claude/`—using a path that coincides with the directory associated with the Claude Code AI assistant, a choice likely intended to blend the second-stage payload into expected filesystem state [2].

Credential Targets and Exfiltration Mechanics

Once executing, the harvester sweeps credentials across a broad range of platforms and local sources. The following table summarizes the primary target categories and the collection method for each:

Target	Collection Method
GitHub tokens & Actions secrets	Token validation via GitHub API; `ACTIONS_RUNTIME_TOKEN`, `ACTIONS_ID_TOKEN_REQUEST_TOKEN` enumeration
npm tokens	Validated via `/-/whoami`; OIDC tokens exchanged for publish rights
AWS	IAM credentials via IMDS (169.254.169.254) and ECS metadata; Secrets Manager API access
Azure	IMDS OAuth2 tokens for management.azure.com, graph.microsoft.com, Key Vault endpoints
GCP	metadata.google.internal service-account tokens; Secret Manager; Resource Manager access
HashiCorp Vault	Probes localhost:8200 across multiple token paths
Kubernetes	Service account tokens from expected filesystem paths
Local secrets	SSH keys, `~/.npmrc`, CLI credentials, browser stored credentials, crypto wallet files
Anthropic API keys	Reads `~/.claude.json`; validates against a dormant (noop) channel

A notable capability—one that bypasses GitHub's log-level secret masking—is the payload's ability to scrape GitHub Actions Runner process memory. It locates the `Runner.Worker` process by scanning `/proc/<pid>/cmdline`, then extracts secrets directly from the process address space using `grep` patterns targeting the internal `"isSecret\":true"` serialization format. This technique bypasses GitHub's built-in secret-masking mechanism, which redacts secrets in log output but does not prevent a co-resident process from reading them from memory [2].

Stolen data is exfiltrated through three channels. The primary channel creates attacker-controlled public GitHub repositories—named to carry the "Miasma: The Spreading Blight" description—and commits stolen credentials as timestamped JSON files. A secondary channel injects a `.github/setup.js` payload into victim-controlled repositories via the Git Data API using commits spoofed with the `github-actions@github.com` author identity, establishing a persistence foothold in repositories the victim owns. A tertiary channel is a dormant HTTPS POST to

`api.anthropic.com:443/v1/api` with a `noop: true` flag, which appears designed for validating stolen Anthropic API keys when activated [2]. Across all channels, stolen tokens are double-Base64 encoded and transmitted with a spoofed `python-requests/2.31.0` user-agent string. Attacker infrastructure rotates across sixteen GitHub accounts per session [2].

The malware also plants a decoy honeytoken with the label "IfYouInvalidateThisTokenItWillNukeTheComputerOfTheOwner." Triggering this token invokes a wiper routine executing `rm -rf ~/` and targeting `~/Documents`, a destructive routine—likely intended to deter security teams from invalidating credentials as a defensive response [2].

The Worm: Self-Propagation at Ecosystem Scale

What distinguishes Miasma from a conventional credential stealer is its worm behavior: having harvested npm OIDC tokens, the payload enumerates all repositories and organizations accessible to the compromised account and uses those tokens to republish backdoored versions of packages the victim is authorized to publish [1][2]. Each newly infected package carries its own `preinstall` hook, seeding the payload into the supply chain of any downstream project that installs it. The stolen OIDC tokens—designed specifically to be short-lived and scoped—are used within their validity window to create long-term persistence through the infection of packages that persist indefinitely in the registry [1][2].

This mechanism also generates forged SLSA provenance attestations through Sigstore for the republished packages, because the OIDC tokens used were issued by a legitimate GitHub Actions workflow in a legitimate repository [2][4]. An organization relying on SLSA attestations as a trust signal would see a valid attestation on a malicious package, because the attestation accurately reflects that the package was built by a GitHub Actions workflow—it simply does not reflect that the workflow was compromised.

The combined effect is that the blast radius of a single compromised developer account extends beyond the 32 initial packages to encompass the full set of packages that account or its downstream victims can publish.

The Seven-Week Window

The most preventable aspect of the Miasma incident may be the intelligence gap between credential theft and exploitation. CybelAngel's post-incident review confirmed that the session cookie used to authenticate the attacker's actions had been harvested by infostealer malware and circulated in criminal

markets for nearly seven weeks before use [5]. Infostealer logs are sold, resold, and incorporated into criminal combo lists on timelines that, in this and similar documented cases, extend to weeks or months rather than hours [5]; a credential that appears in logs in April may not be weaponized until June.

This gap creates an actionable window for organizations that monitor dark web credential markets or subscribe to commercial infostealer intelligence feeds. An alert on April 13 would have provided seven weeks to rotate the compromised credentials, close the session, and harden the GitHub organization—time that remained unused, a gap consistent with the absence of infostealer intelligence monitoring, though precise organizational factors have not been publicly disclosed [5]. For organizations relying on MFA as their primary defense against account compromise, the session cookie aspect is particularly significant: an active session cookie bypasses MFA entirely, because the authentication challenge has already been satisfied [5].

Recommendations

Immediate Actions

Organizations that have consumed any `@redhat-cloud-services` package in CI/CD pipelines or developer workflows should treat the following as potentially compromised: npm publish tokens, GitHub personal access tokens and OAuth tokens, cloud provider credentials (AWS, Azure, GCP), Vault tokens, Kubernetes service account tokens, SSH keys, and any secrets injected as environment variables into affected build environments. Security researchers recommend rotating these credentials out of an abundance of caution; Red Hat's advisory (RHSB-2026-006) states that no actions from customers are required, noting that the affected packages are strictly limited to internal development, and confirms that compromised package versions have been removed from the registry [3].

To identify potential exposure, audit CI/CD pipeline logs for the presence of unexpected Bun runtime downloads or executions originating from Node processes, access to cloud metadata endpoints (169.254.169.254) from npm install sessions, outbound connections to github.com from package installation steps, or repository creation events in GitHub organizations tied to automated build accounts. Defender for Endpoint and Microsoft Defender XDR detections `Trojan:JS/ShaiWorm.DAW!MTB` and `Trojan:JS/ObfusNpmJs` cover this malware family [2]. Additionally, audit GitHub organizations for any repositories carrying the description "Miasma: The Spreading Blight" or containing unexplained `.github/setup.js` files in recent commits.

Short-Term Mitigations

Running `npm install` with the `--ignore-scripts` flag prevents preinstall and postinstall hooks from executing automatically, eliminating the primary delivery mechanism for this class of attack. Where `--ignore-scripts` cannot be applied universally due to legitimate dependency requirements, organizations should audit which dependencies in their tree actually require install scripts and allowlist only those, blocking execution for all others. Dependency allowlisting and software bill of materials (SBOM) generation, when combined with version pinning to known-good releases, reduce the surface area for future compromises of this type.

For GitHub organizations, the `id-token: write` permission should be scoped as narrowly as possible. Workflows that require OIDC token issuance should be restricted to specific branches and tag patterns rather than triggering on any push. Environments that publish to npm through trusted publishing should additionally require a manual approval step or a dedicated protected environment in GitHub Actions so that a compromised workflow cannot publish autonomously without human review.

Strategic Considerations

The Miasma incident illustrates a structural tension in modern supply chain security design. GitHub Actions trusted publishing and OIDC-based token exchange were adopted precisely to eliminate long-lived secrets as an attack surface, and they did eliminate long-lived publish tokens as a direct attack vector—though not the identity-compromise risk that enables token issuance. SLSA provenance attestations face the same structural constraint—they attest to build provenance accurately even when the workflow that produced them was malicious. Neither control is wrong, but neither is sufficient in isolation. Defense-in-depth requires monitoring the identity layer (detecting compromised accounts before they are weaponized), the behavior layer (detecting anomalous package publication events), and the network layer (detecting credential exfiltration from build environments).

Organizations should evaluate their investment in infostealer intelligence. The seven-week gap between credential theft and weaponization is not a guarantee but a pattern observed in this and similar documented incidents; commercial services that monitor dark web credential markets and alert on employee credentials appearing in infostealer logs provide actionable warning in a window during which reactive incident response—even when executed promptly—cannot recover credentials already circulating in criminal markets or revoke access that may have been exercised weeks earlier. This is particularly relevant for accounts with publish rights over widely-consumed packages, where the downstream blast radius is disproportionate to the access level of a single developer account.

CSA Resource Alignment

The Miasma incident connects directly to several active areas of CSA's research and frameworks.

CSA's AI Infrastructure Matrix for Cloud (AICM) addresses the dependency integrity concerns surfaced by this attack. The AICM's supply chain controls speak to the need for software composition analysis, SBOM generation, and integrity verification for AI system dependencies—controls that could support detection of anomalies such as a 4.29 MB preinstall script in an otherwise small utility package, if the controls are implemented with tooling that inspects install-time script execution in CI/CD pipelines. Organizations using AICM as a control baseline should treat this incident as a test case for validating whether their dependency controls address install-time script execution.

MAESTRO's threat model for agentic AI pipelines is directly applicable to the CI/CD context in which Miasma operated. Layer 2 (Data Operations) of the MAESTRO framework addresses the integrity of data and artifacts flowing through automated pipelines; an npm package with a weaponized preinstall script is precisely a Layer 2 integrity failure in an agentic build context. Layer 7 (Ecosystem) addresses third-party and dependency risk, which is the entry point for any supply chain compromise. Security teams applying MAESTRO to their AI-adjacent build infrastructure should map the specific controls in both layers against their npm dependency governance posture.

CSA's Zero Trust guidance reinforces the principle that identity-based access controls require continuous validation, not just point-in-time authentication. The session cookie bypass of MFA in this incident is a direct illustration of why Zero Trust architectures that enforce continuous re-authentication and session validity checks—rather than relying on initial authentication as a durable grant—reduce exposure to stolen-session attacks. The `id-token: write` permission abuse further argues for the Zero Trust principle of least privilege applied to CI/CD workflow permissions: granting publish rights only to the specific jobs, environments, and conditions that require them.

Organizations using AICM as their primary control baseline will find that AICM's DSP and STA domains (inherited from CCM) address secrets management and third-party software risk governance respectively—the two control areas most directly stressed by the Miasma incident. CSA's Cloud Controls Matrix (CCM) Domain DSP (Data Security and Privacy) and STA (Supply Chain Management and Transparency) are directly relevant to organizations conducting gap analyses in the wake of this incident. DSP controls address secrets management—specifically, how secrets are stored, rotated, and protected at rest and in transit across cloud environments. STA controls address third-party software risk governance, including the policies and tooling required to identify and respond to compromised dependencies. Organizations should assess their coverage against both domains.

The Miasma campaign also highlights the growing relevance of CSA's ongoing work on the AI Vulnerability Clearinghouse. The malware's credential sweep includes a dormant collection path for Anthropic API keys (via `~/.claude.json`), suggesting that AI developer tooling credentials may be entering the targeting scope of supply chain attacks, even if this specific channel was not yet activated at the time of analysis [2]. Security teams managing AI developer tooling should treat those credential stores as part of the software supply chain attack surface.

References

[1] Wiz Research. "Miasma: Supply Chain Attack Targeting RedHat npm Packages." Wiz Blog, June 1, 2026.

[2] Microsoft Threat Intelligence. "Preinstall to persistence: Inside the Red Hat npm Miasma credential-stealing campaign." Microsoft Security Blog, June 2, 2026.

[3] Red Hat Product Security. "RHSB-2026-006 Supply chain compromise of @redhat-cloud-services npm packages." Red Hat Customer Portal, June 1, 2026.

[4] Aikido Security. "Red Hat npm Packages Compromised to Spread a Credential-Stealing Worm." Aikido Blog, June 1, 2026.

[5] CybelAngel. "Miasma Supply Chain Attack: the Seven-Week Credential Trail." CybelAngel Blog, June 2, 2026.

[6] StepSecurity. "Multiple redhat-cloud-services npm Packages compromised." StepSecurity Blog, June 1, 2026.

[7] SafeDep. "Mini Shai-Hulud 'Miasma: The Spreading Blight' Hits @redhat-cloud-services: Multiple Packages at Risk." SafeDep Blog, June 1, 2026.