INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-11 19:24 UTC

# Miasma Supply Chain Worm Toolkit Leak Fuels Hades Campaign Against Open-Source Registries and CI/CD Pipelines

**THREAT CAMPAIGN** | **CRITICAL** | CVSS 9.0

| | |
|---|---|
| **SCC Item ID** | SCC-CAM-2026-0443 |
| **Type** | Threat Campaign |
| **Severity** | CRITICAL |
| **CVSS Base Score** | 9.0 |
| **Affected Products** | PyPI, npm, RubyGems (public package registries), JFrog Artifactory, GitHub repositories (73+ Microsoft repos reported), GitHub Actions workflows, AI coding tool configurations |
| **Published** | 2026-06-11 |
| **Discovery Source** | Gemini |

## Executive Summary

A credential-stealing attack framework called Miasma was leaked publicly and weaponized into the 'Hades Campaign,' a multi-stage supply chain attack targeting open-source package registries (PyPI, npm, RubyGems), CI/CD pipelines, and developer environments. Over 304 software components and 73 Microsoft GitHub repositories are reported affected, with attackers injecting malicious code into widely used packages and stealing secrets including API keys, tokens, and credentials from pipeline configurations. Organizations that consume open-source packages or run automated build pipelines face immediate risk of software supply chain compromise, backdoored software delivery, and downstream credential theft at scale.

## Technical Analysis

The Hades Campaign weaponizes the leaked Miasma toolkit to execute a modular, multi-stage supply chain attack. Initial access relies on compromised developer credentials (CWE-522: Insufficiently Protected Credentials; CWE-798: Hard-coded Credentials) to inject malicious code into packages hosted on PyPI, npm, and RubyGems. The toolkit includes embedded malicious payloads in package components (CWE-506: Embedded Malicious Code; CWE-494: Download of Code Without Integrity Check) and abuses inclusion of functionality from untrusted sources (CWE-829). MITRE ATT&CK techniques observed: T1195.001 (Compromise Software Dependencies and Development Tools), T1195.002 (Compromise Software Supply Chain), T1552.001 (Credentials In Files), T1552.004 (Private Keys), T1528 (Steal Application Access Token),

T1567 (Exfiltration Over Web Service), T1059 (Command and Scripting Interpreter), T1543 (Create or Modify System Process). The toolkit establishes persistence in CI/CD environments and exfiltrates secrets from pipeline configuration files, environment variables, and AI coding tool configurations. No CVE ID is assigned. CVSS score is pending NVD publication; qualitative_rating reflects editorial severity assessment based on attack scope and business impact. Attribution to a named threat actor remains unconfirmed. SOURCE CONFIDENCE NOTE: Primary sourcing is a June 2026 Rescana ThreatsDay Bulletin (Tier 3); specific technical claims including the 304-component and 73-repository figures should be treated as medium confidence pending corroboration from CISA, NVD, or OSV.

## Action Checklist

1. Step 1: Containment. Audit all CI/CD pipeline configurations immediately for unauthorized changes to workflow files, build scripts, and dependency manifests. Rotate all secrets, tokens, and API keys stored in GitHub Actions secrets, environment variables, or pipeline configuration files. Enforce CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts) to limit which identities can push to registries or modify pipeline definitions.
2. Step 2: Detection. Query SCM audit logs for unexpected commits to package manifest files (package.json, setup.py, Gemfile), changes to GitHub Actions workflow YAML files, and anomalous pushes to PyPI, npm, or RubyGems from CI service accounts. Review AU-2 (Event Logging) coverage to confirm pipeline activity, credential access events, and outbound data transfers are captured. Apply D3-LAM (Local Account Monitoring) to detect unauthorized use of developer or service accounts within pipeline contexts. Look for outbound exfiltration patterns consistent with T1567 (web service-based data exfiltration).
3. Step 3: Eradication. Pin all package dependencies to known-good, integrity-verified versions and validate checksums against upstream registry manifests. Enable CIS 7.1 (Establish and Maintain a Vulnerability Management Process) review for all third-party dependencies in affected registries. Implement D3-FMBV (File Magic Byte Verification) or equivalent integrity controls on downloaded packages. Remove any unauthorized GitHub Actions workflow modifications and re-verify repository branch protection rules per NIST AC-3 (Access Enforcement).
4. Step 4: Recovery. Validate all pipeline outputs from the suspected compromise window by re-running builds from clean, pinned dependency states. Confirm AU-9 (Protection of Audit Information) is enforced so pipeline logs have not been tampered. Monitor post-remediation builds for anomalous outbound network connections, unexpected process spawns (T1059), and new persistence mechanisms (T1543). Re-verify that credential rotation completed across all affected systems including AI coding tool configuration stores.
5. Step 5: Post-Incident. Conduct a control gap review against NIST AC-6 (Least Privilege) for all CI/CD service account permissions. Implement software composition analysis (SCA) tooling as a mandatory pipeline gate aligned with CIS 2.1 (Establish and Maintain a Software Inventory) and CIS 2.2 (Ensure Authorized Software is Currently Supported). Establish SBOM generation for all internal builds. Apply D3-CRO (Credential Rotation) as a recurring control, not a one-time response. Review AI coding tool configurations as an emerging attack surface and assess what secrets or tokens those tools have access to.

## IR / Forensic Enrichment

| | |
|---|---|
| **Triage Priority** | IMMEDIATE |
| **Escalation Criteria** | Escalate immediately to CISO, Legal, and external IR if any of the following are confirmed: internal packages consumed by production systems were built from a compromised pipeline during the exposure window; secrets or tokens exfiltrated by the Miasma toolkit have been used to authenticate to production infrastructure; affected repositories contain code deployed to customer-facing systems that may trigger breach notification obligations under applicable data protection regulations; or the organization's JFrog Artifactory instance distributed tainted packages downstream to external customers or partners. |
| **Recovery Notes** | Before returning any CI/CD pipeline to production service, all builds from the suspected compromise window must be re-executed from scratch using clean, integrity-verified dependency states on freshly provisioned runner infrastructure — do not reuse self-hosted runners that executed builds during the exposure period without a full OS reimaging, as Miasma variants establish persistence in runner home directories and shell profile files. Post-recovery monitoring should remain elevated for a minimum of 30 days, with particular attention to any outbound HTTPS connections from runner hosts to non-registry destinations, new GitHub Actions workflow YAML commits, and any PyPI/npm/RubyGems publish events originating from CI service accounts. SBOM validation should be run against all artifacts promoted during the recovery window to confirm no residual tainted dependencies were introduced during the rebuild process. |
| **Forensic Artifacts** | Tampered GitHub Actions workflow YAML files (.github/workflows/*.yml) in their modified state — these contain the injected Miasma loader stanzas, which typically appear as obfuscated `run:` steps invoking curl or python -c with base64-encoded payloads; preserve full file content and git blame output before any revert. \| GitHub organization audit log (JSON export, full retention window) — specifically entries for `git.push` events on manifest files (package.json, setup.py, Gemfile, *.lock), `secret.access` events on GitHub Actions secrets, and `integration.create` events for OAuth app authorizations made by CI service accounts during the compromise window. \| Self-hosted runner host memory image and Sysmon logs — Miasma installs persistence in runner user home directories (~/.bashrc, ~/.profile, or Windows runner service user AppData); a memory image captures any in-memory credential stores or active C2 connections before runner process termination, and Sysmon Event ID 1/3 logs document the process tree and network destinations of malicious build steps. \| JFrog Artifactory access and download logs for the compromise window — identifies which internal consumers (services, developers, downstream pipelines) pulled tainted package versions from the internal mirror, establishing the full blast radius of the supply chain compromise beyond the initially identified PyPI/npm/RubyGems packages. \| AI coding tool configuration files and OAuth token stores on developer workstations (~/.config/github-copilot/, ~/.cursor/, or equivalent) — the Hades Campaign specifically targets developer environment secrets; these files may contain access tokens with repository read/write scope that were exfiltrated by Miasma and must be audited and rotated as part of the credential compromise assessment. |

### Per-Action IR Details

**Step 1: Containment — Audit all CI/CD pipeline configurations immediately for unauthorized changes to workflow files, build scripts, and dependency manifests. Rotate all secrets, tokens, and API keys stored in GitHub Actions secrets, environment variables, or pipeline configuration files. Enforce CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts) to limit which identities can push to registries or modify pipeline definitions.**

**NIST Phase:** Containment

**Reference:** NIST 800-61r3 §3.3 — Containment Strategy

**Controls:** CIS 5.4 (IG1/IG2/IG3) — Restrict Administrator Privileges to Dedicated Administrator Accounts, NIST AC-6 — Least Privilege, NIST AC-2 — Account Management

**Compensating:** Use the GitHub CLI (`gh api /repos/{owner}/{repo}/actions/secrets`) to enumerate all repository and organization-level secrets. Diff current `.github/workflows/*.yml` files against last-known-good commits using `git log --diff-filter=M --name-only` on each affected repo. For PyPI/npm/RubyGems publisher tokens, enumerate active API tokens via registry web UIs and revoke any not tied to a named, documented service account. A 2-person team can script this across repos with a bash loop using the GitHub REST API.

**Evidence:** Before rotating any secrets or revoking tokens, capture: (1) current GitHub Actions workflow YAML file hashes (`sha256sum .github/workflows/*.yml`) to establish tampered vs. clean baselines; (2) GitHub audit log export via `gh api /orgs/{org}/audit-log?include=all` covering the suspected compromise window, preserving CI service account push events and secret access events; (3) pipeline execution logs for all runs in the compromise window from GitHub Actions UI or via `gh run list --limit 100`; (4) environment variable dumps from any self-hosted runner hosts (volatile — gone after process termination or runner restart). Capture these before any credential rotation, as rotation will destroy the evidentiary linkage between the stolen token and its usage timeline.

**Step 2: Detection — Query SCM audit logs for unexpected commits to package manifest files (package.json, setup.py, Gemfile), changes to GitHub Actions workflow YAML files, and anomalous pushes to PyPI, npm, or RubyGems from CI service accounts. Review AU-2 (Event Logging) coverage to confirm pipeline activity, credential access events, and outbound data transfers are captured. Apply D3-LAM (Local Account Monitoring) to detect unauthorized use of developer or service accounts within pipeline contexts. Look for outbound exfiltration patterns consistent with T1567 (web service-based data exfiltration).**

**NIST Phase:** Detection Analysis

**Reference:** NIST 800-61r3 §3.2 — Detection and Analysis

**Controls:** NIST AU-2 — Event Logging, NIST AU-6 — Audit Record Review, Analysis, And Reporting, NIST AU-12 — Audit Record Generation, CIS 8.2 (IG1/IG2/IG3) — Collect Audit Logs

**Compensating:** Without a SIEM, run the following targeted queries: (1) GitHub audit log search via `gh api "/orgs/{org}/audit-log?phrase=action:git.push+actor:{ci-service-account}"` filtered to the compromise window; (2) on self-hosted runners, query Sysmon Event ID 3 (Network Connection) for outbound connections to non-registry destinations from the runner process (`GitHub.Runner.Worker.exe` or `Runner.Worker` on Linux); (3) use `git log --all --full-history -- 'package.json' 'setup.py' 'Gemfile' '*.lock'` across cloned repo mirrors to surface manifest changes; (4) deploy the free Sigma rule `proc_creation_win_susp_curl_download.yml` on runner hosts to catch Miasma-style exfiltration over HTTPS to attacker-controlled endpoints. Cross-reference PyPI upload logs (`pypi.org/account/token/`) and npm publish history (`npm profile ls --token`) for service account activity outside change windows.

**Evidence:** Before any containment action that would alter log state, preserve: (1) GitHub organization audit log in its entirety for the 90-day retention window (export to JSON immediately — GitHub's audit log is a rolling window); (2) Sysmon Event ID 1 (Process Creation) and Event ID 3 (Network Connection) from self-hosted runner hosts, specifically filtering on runner worker child processes that spawned interpreters (`python.exe`, `node.exe`, `ruby.exe`) or outbound curl/wget calls; (3) DNS query logs from runner hosts for the compromise window — Miasma-style worms frequently use DNS-over-HTTPS or low-TTL C2 domains; (4) PyPI/npm/RubyGems publish audit trails downloadable from each registry's account settings dashboard; (5) any `.env` files, `*.tfvars`, or `config.yml` files present on runner disk that may contain cleartext secrets the worm targeted.

**Step 3: Eradication — Pin all package dependencies to known-good, integrity-verified versions and validate checksums against upstream registry manifests. Enable CIS 7.1 (Establish and Maintain a Vulnerability Management Process) review for all third-party dependencies in affected registries. Implement D3-FMBV (File Magic Byte Verification) or equivalent integrity controls on downloaded packages. Remove any unauthorized GitHub Actions workflow modifications and re-verify repository branch protection rules per NIST AC-3 (Access Enforcement).**

**NIST Phase:** Eradication

**Reference:** NIST 800-61r3 §3.4 — Eradication

**Controls:** NIST AC-3 — Access Enforcement, CIS 7.1 (IG1/IG2/IG3) — Establish and Maintain a Vulnerability Management Process, CIS 7.2 (IG1/IG2/IG3) — Establish and Maintain a Remediation Process, CIS 2.1 (IG1/IG2/IG3) — Establish and Maintain a Software Inventory, CIS 2.2 (IG1/IG2/IG3) — Ensure Authorized Software is Currently Supported

**Compensating:** Run `pip-audit` (free, PyPA-maintained) against all `requirements.txt` and `setup.py` files to identify compromised or tampered package versions. For npm, run `npm audit --audit-level=critical` and cross-reference with the affected package list from the Hades Campaign IOC feed. For RubyGems, use `bundle-audit check --update`. Validate package integrity by comparing `sha256` hashes of downloaded `.whl`, `.tgz`, and `.gem` archives against the corresponding hashes published on each registry's package detail page. Revert tampered workflow YAML files using `git revert ` and immediately re-enable branch protection rules requiring signed commits (`git config --global commit.gpgsign true`) and required PR reviews for the `.github/workflows/` path via GitHub branch protection settings.

**Evidence:** Before removing tampered workflow files or reverting package manifests, capture: (1) full content of all modified `.github/workflows/*.yml` files in their tampered state — these contain the injected Miasma loader stanzas and are the primary forensic artifact; (2) checksums and metadata (`pip show `, `npm view `, `gem info `) of all installed dependency versions on runner hosts and in JFrog Artifactory caches, as these document which malicious versions were pulled; (3) JFrog Artifactory download/access logs confirming which internal consumers pulled tainted packages, establishing blast-radius scope; (4) any artifacts written to runner workspaces or `/tmp` directories by malicious build steps — Miasma variants have been observed staging exfiltration payloads as build artifacts before transmission. This capture must occur before eradication steps overwrite or delete tampered files.

**Step 4: Recovery — Validate all pipeline outputs from the suspected compromise window by re-running builds from clean, pinned dependency states. Confirm AU-9 (Protection of Audit Information) is enforced so pipeline logs have not been tampered. Monitor post-remediation builds for anomalous outbound network connections, unexpected process spawns (T1059), and new persistence mechanisms (T1543). Re-verify that credential rotation completed across all affected systems including AI coding tool configuration stores.**

**NIST Phase:** Recovery

**Reference:** NIST 800-61r3 §3.5 — Recovery

**Controls:** NIST AU-9 — Protection Of Audit Information, NIST AU-11 — Audit Record Retention, CIS 4.6 (IG1/IG2/IG3) — Securely Manage Enterprise Assets and Software

**Compensating:** Re-run all builds that executed during the compromise window in an isolated, ephemeral environment (a fresh GitHub Actions runner or a clean Docker container with no network access to production systems) using dependency lockfiles pinned to pre-compromise hashes. Use `cosign verify` (Sigstore, free) to validate artifact signatures on rebuilt packages before promoting to JFrog Artifactory. To monitor for post-remediation re-compromise, deploy Sysmon on self-hosted runners with EventID 1 filtering for child processes of `Runner.Worker` that invoke `python`, `node`, `ruby`, `curl`, or `wget` outside expected build steps. For AI coding tool configurations (e.g., GitHub Copilot, Cursor, Codeium), manually audit `~/.config/` and `~/.local/share/` directories on developer workstations for stored OAuth tokens and rotate any that were active during the compromise window.

**Evidence:** Before promoting any rebuilt artifacts to production or restoring pipeline service accounts to full access: (1) verify pipeline log integrity by comparing SHA-256 hashes of stored GitHub Actions run logs against immutable copies captured during detection_analysis phase; (2) collect post-remediation network baseline from runner hosts using `ss -tnp` or `netstat -ano` to establish expected outbound connection patterns; (3) document the completion timestamp and scope of all credential rotations across GitHub secrets, JFrog Artifactory API keys, PyPI tokens, npm publish tokens, and AI tool OAuth tokens — this record is required for any downstream regulatory notification assessment; (4) confirm Sysmon logging is active and shipping before declaring recovery complete, as absence of monitoring is operationally equivalent to blind recovery.

**Step 5: Post-Incident — Conduct a control gap review against NIST AC-6 (Least Privilege) for all CI/CD service account permissions. Implement software composition analysis (SCA) tooling as a mandatory pipeline gate aligned with CIS 2.1 (Establish and Maintain a Software Inventory) and CIS 2.2 (Ensure Authorized Software is Currently Supported). Establish SBOM generation for all internal builds. Apply D3-CRO (Credential Rotation) as a recurring control, not a one-time response. Review AI coding tool configurations as an emerging attack**

**surface and assess what secrets or tokens those tools have access to.**

**NIST Phase:** Post Incident

**Reference:** NIST 800-61r3 §4 — Post-Incident Activity

**Controls:** NIST AC-6 — Least Privilege, NIST AC-2 — Account Management, CIS 2.1 (IG1/IG2/IG3) — Establish and Maintain a Software Inventory, CIS 2.2 (IG1/IG2/IG3) — Ensure Authorized Software is Currently Supported, CIS 7.1 (IG1/IG2/IG3) — Establish and Maintain a Vulnerability Management Process, CIS 7.2 (IG1/IG2/IG3) — Establish and Maintain a Remediation Process

**Compensating:** For free SCA gating, integrate `pip-audit` (Python), `npm audit` (Node), and `bundle-audit` (Ruby) as required CI steps that fail the build on high/critical findings — achievable with a single GitHub Actions workflow step addition. Generate CycloneDX-format SBOMs using `cyclonedx-bom` (Python), `cyclonedx-npm` (Node), or `cyclonedx-ruby` (Ruby) — all free and open-source — and store outputs as signed build artifacts. For recurring credential rotation, implement a GitHub Actions scheduled workflow (cron: `0 0 1 * *`) that generates alerts when service account tokens exceed a configurable age threshold, using the GitHub API to list token creation dates. For AI tool attack surface review, enumerate all OAuth app authorizations at `github.com/settings/applications` for each developer account and revoke any AI coding tool authorizations that have `repo` or `workflow` scope beyond what is operationally required.

**Evidence:** Post-incident documentation to preserve for lessons-learned and potential regulatory reporting: (1) complete timeline of the compromise window derived from GitHub audit logs, registry publish timestamps, and runner Sysmon logs — this is the evidentiary foundation for any breach notification assessment; (2) full inventory of all packages, repositories, and pipeline runs confirmed or suspected to have been exposed to Miasma-injected code, cross-referenced against the 304 reported affected components and 73 Microsoft GitHub repos; (3) mapping of all credentials, tokens, and API keys confirmed rotated, with rotation timestamps, to demonstrate remediation completeness; (4) gap analysis output documenting which CI/CD service accounts held excessive permissions at time of compromise, to support the AC-6 (Least Privilege) control gap review and inform permission scope reductions. Retain all artifacts per AU-11 (Audit Record Retention) requirements and organizational records retention policy.

## Detection Guidance

Focus detection on four surfaces. First, SCM and registry activity: review GitHub audit logs for unexpected workflow file modifications, unauthorized pushes by service accounts, and new repository collaborators. Second, pipeline secrets access: alert on environment variable reads or secret store accesses that occur outside normal build job patterns, correlated against AU-2 (Event Logging) and AU-6 (Audit Record Review, Analysis, and Reporting) controls. Third, outbound exfiltration: monitor for POST requests from build agents to non-sanctioned external endpoints, consistent with T1567 (Exfiltration Over Web Service) and T1528 (Steal Application Access Token). Fourth, dependency integrity: compare installed package hashes against registry-published checksums; flag any mismatch. D3-SFA (System File Analysis) applied to build configuration files and dependency manifests will surface tampering. D3-LAM (Local Account Monitoring) should cover CI service accounts specifically. No confirmed IOC hashes, domains, or IPs are available from verified sources at this confidence level. Monitor OSV (Open Source Vulnerabilities) and GitHub Security Advisory databases for published indicators or follow-up reports on Hades Campaign components as threat intelligence matures.

## Indicators of Compromise

| Type | Value | Context | Confidence |
|---|---|---|---|
| URL | `https://www.rescana.com/post/threatsday-bulletin-june-2026-miasma-supply-chain-worm-leak-claude-code-github-action-vulnerability-ai-agent-phishing-to` | Rescana ThreatsDay Bulletin June 2026 — primary source describing the Miasma leak and Hades Campaign; Tier 3 source, not independently verified by authoritative databases | LOW |

## Framework Mappings

### MITRE-ATTACK

- **T1552.001** — Credentials In Files
- **T1567** — Exfiltration Over Web Service
- **T1528** — Steal Application Access Token
- **T1552.004** — Private Keys
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1195.002** — Compromise Software Supply Chain
- **T1059** — Command and Scripting Interpreter
- **T1543** — Create or Modify System Process

### NIST-800-53R5

- **CM-7** — Least Functionality
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **IA-5** — Authenticator Management
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan

### OWASP-TOP10-2021

- **A07:2021** — Identification and Authentication Failures
- **A04:2021** — Insecure Design
- **A08:2021** — Software and Data Integrity Failures

### CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **5.2** — Use Unique Passwords
- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **6.3** — Require MFA for Externally-Exposed Applications

- **15.1** — Establish and Maintain an Inventory of Service Providers

**ISO-27001-2022**

- **A.8.28** — Secure coding
- **A.5.21** — Managing information security in the ICT supply chain

**HIPAA-SECURITY**

- **164.308(a)(5)(ii)(D)** — Password Management
- **164.312(d)** — Person or Entity Authentication

**SOC2-TSC**

- **CC6.1** — Logical access security software, infrastructure, and architectures
- **CC9.2** — Manages risks associated with vendors and business partners

**NIST-CSF-2**

- **GV.SC-01** — Cybersecurity supply chain risk management program

## MITRE ATT&CK Mapping

| Technique ID | Technique Name | Tactic |
|---|---|---|
| T1552.001 | Credentials In Files | Credential-Access |
| T1567 | Exfiltration Over Web Service | Exfiltration |
| T1528 | Steal Application Access Token | Credential-Access |
| T1552.004 | Private Keys | Credential-Access |
| T1195.001 | Compromise Software Dependencies and Development Tools | Initial-Access |
| T1195.002 | Compromise Software Supply Chain | Initial-Access |
| T1059 | Command and Scripting Interpreter | Execution |
| T1543 | Create or Modify System Process | Persistence |

## Sources

| Source | URL | Tier |
|---|---|---|
| **JFrog Artifactory: Key Features, Limitations, And Alternatives** | https://octopus.com/devops/jfrog-artifactory/ | T3 |
| **jfrog/jfrog-pypi-trial** | https://github.com/jfrog/jfrog-pypi-trial | T3 |

| Source | URL | Tier |
|---|---|---|
| **Artifactory vs GitHub Package Registry FAQs** | https://jfrog.com/blog/artifactory-vs-github-packages-faq/ | T3 |
| **Poetry fails to publish package to Jfrog's PyPi artifactory via ...** | https://github.com/orgs/python-poetry/discussions/8968 | T3 |
| **ThreatsDay Bulletin June 2026: Miasma Supply Chain ...** | https://www.rescana.com/post/threatsday-bulletin-june-2026-miasma-s... | T3 |



Generated 2026-06-11 19:24 UTC by TJS Security Command Center