INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-24 18:54 UTC

Cordyceps CI/CD Weakness Class Exposes 300+ GitHub Repositories to Supply Chain Compromise

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.0

SCC Item ID	SCC-CAM-2026-0555
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.0
Affected Products	GitHub repositories with misconfigured CI/CD workflows; reportedly includes repositories maintained by Microsoft, Google, Apache, and Cloudflare
Published	2026-06-24
Discovery Source	Gemini

Executive Summary

Researchers have identified a class of CI/CD configuration weaknesses, dubbed 'Cordyceps,' affecting more than 300 high-impact GitHub repositories, including those maintained by Microsoft, Google, Apache, and Cloudflare. The weakness allows external contributors with little or no privileges to trigger GitHub Actions workflows that execute code with elevated permissions, enabling theft of secrets and credentials stored in CI/CD pipelines. If exploited, attackers can tamper with build artifacts or package releases, turning trusted software into a vehicle for downstream supply chain compromise across any organization consuming those packages.

Technical Analysis

The Cordyceps weakness class targets GitHub Actions workflows that use the pull_request_target trigger (or functionally equivalent triggers) without adequately scoping permissions or sandboxing untrusted code from forked repositories. Because pull_request_target runs in the context of the base repository rather than the fork, it carries the base repository's secrets and elevated GITHUB_TOKEN permissions. An external contributor submits a malicious pull request; the workflow executes attacker-controlled code with write-level access to repository resources, CI/CD secrets, and potentially package publishing credentials. No single CVE has been assigned; this is a systematic configuration weakness class. Relevant CWEs: CWE-250 (Execution with Unnecessary Privileges), CWE-732 (Incorrect Permission Assignment for Critical Resource), CWE-693 (Protection Mechanism Failure), CWE-1357 (Reliance on Insufficiently Trustworthy Component). MITRE

ATT&CK techniques: T1552.001 (Credentials in Files), T1190 (Exploit Public-Facing Application), T1059 (Command and Scripting Interpreter), T1078 (Valid Accounts), T1195.001 (Compromise Software Dependencies and Development Tools). No vendor patch is available because the root cause is misconfiguration, not a software defect; remediation requires workflow permission scoping and safe trigger usage.

Source Confidence Note: Current sources are T3 (news outlets and vendor blogs). Primary research has not been independently verified against original source material. Credibility would be strengthened by T1 sources (CISA advisory or official researcher disclosure) if available.

Action Checklist

1. Step 1: Containment. Audit all GitHub Actions workflow files across your repositories for use of pull_request_target, workflow_run, or similar triggers that execute untrusted fork code in a privileged context. Immediately restrict GITHUB_TOKEN permissions to the absolute minimum required scope (e.g., permissions: read-all for read-only workflows, or omit secrets entirely for external-trigger workflows). Apply NIST AC-6 (Least Privilege) and CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts).
2. Step 2: Detection. Search workflow YAML files for pull_request_target combined with checkout of fork HEAD (actions/checkout with ref: ${{ github.event.pull_request.head.sha }}). Review GitHub Actions audit logs for unexpected workflow runs initiated by external contributors, secret access events, and GITHUB_TOKEN permission escalations. Map findings to AU-2 (Event Logging) and CIS 8.2 (Collect Audit Logs). Apply D3-LAM (Local Account Monitoring) principles to CI/CD service account activity.
3. Step 3: Eradication. Replace pull_request_target with pull_request for workflows that process untrusted code, accepting the loss of secret access as the intended security boundary. Where elevated permissions are genuinely required, gate them behind explicit approval steps using GitHub's environment protection rules with required reviewers. Rotate all secrets and tokens that were accessible to affected workflows, per NIST IR controls and D3-CRO (Credential Rotation). Apply CIS 7.1 (Establish and Maintain a Vulnerability Management Process) to track remediation across all affected repositories.
4. Step 4: Recovery. Re-run affected pipelines from a known-clean state after secrets rotation. Validate that build artifacts and package releases produced during the exposure window match expected checksums and have not been tampered with. Enable branch protection rules and require signed commits. Monitor CI/CD pipeline logs for anomalous behavior for at least 30 days post-remediation. Apply AU-6 (Audit Record Review, Analysis, and Reporting) and D3-SFA (System File Analysis) to verify pipeline integrity.
5. Step 5: Post-Incident. Implement a recurring CI/CD workflow security review process. Adopt GitHub's principle of minimal token permissions as an organizational standard (permissions blocks in every workflow). Integrate workflow YAML linting into code review gates to flag dangerous trigger/permission combinations before merge. Map gaps to NIST AC-3 (Access Enforcement), CIS 7.2 (Establish and Maintain a Remediation Process), and NIST CM controls to formalize configuration hardening of CI/CD infrastructure as a standing control.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate to CISO and legal/compliance if GitHub Actions audit logs confirm any externally-triggered workflow run resolved a `secrets` context or produced a build artifact during the exposure window, as this constitutes potential supply chain compromise with downstream consumer impact and may trigger software bill-of-materials (SBOM) disclosure obligations or breach notification requirements if the affected repositories publish to public package registries (npm, PyPI, Maven, container registries) consumed by regulated-industry customers.
Recovery Notes	After secrets rotation and workflow remediation, re-run all affected CI/CD pipelines from a verified clean commit (one predating any external contributor pull request activity during the exposure window) and republish any packages or container images released during that window with updated provenance attestations via sigstore/cosign. Validate downstream package integrity by comparing SHA-256 checksums of exposure-window releases against clean re-build outputs and notify consuming projects or customers if a tamper discrepancy is detected. Maintain elevated monitoring of GitHub Actions audit logs — specifically filtering for `secret.access` and `workflows.completed` events involving external contributors — for a minimum of 30 days post-remediation, given that supply chain compromise from this weakness class may have pre-positioned persistence in downstream consumer environments that will not be visible in your own pipeline logs.
Forensic Artifacts	GitHub Actions audit log entries (org-level, exportable via API at `GET /orgs/{org}/audit-log`) filtered for `action:workflows.completed`, `action:secret.access`, and `actor_type:external` during the exposure window — the primary source for confirming whether the Cordyceps weakness was passively present or actively exploited by an external contributor. | `.github/workflows/*.yml` files at the HEAD commit of each affected repository at the time of discovery — preserved as git archives to document the exact vulnerable trigger and permissions configuration (pull_request_target + fork HEAD checkout + secrets scope) before remediation modifies the files. | GitHub Actions workflow run logs for all externally-triggered runs during the exposure window, downloaded per-run via `gh run view --log` — searchable for evidence of secret exfiltration patterns such as outbound HTTP requests to attacker-controlled infrastructure, base64-encoded environment variable dumps, or unexpected `curl`/`wget` invocations in step output. | Build artifact SHA-256 checksums and package release metadata (npm `package.json` integrity hashes, PyPI distribution file hashes, container image digests from registry manifests) for all releases published during the exposure window — compared against clean re-build outputs to detect tampered supply chain artifacts delivered to downstream consumers. | GitHub repository secrets inventory (names and scopes, not values) and any associated third-party service audit logs (npm publish log, PyPI upload history, cloud provider CloudTrail/Activity Log) for the exposure window — to trace whether stolen CI/CD secrets were used to authenticate to downstream services after exfiltration from the compromised workflow execution context.

Per-Action IR Details

Step 1: Containment — Audit all GitHub Actions workflow files across your repositories for use of pull_request_target, workflow_run, or similar triggers that execute untrusted fork code in a privileged context. Immediately restrict GITHUB_TOKEN permissions to the minimum required scope (e.g., permissions: read-all) and remove secrets access from workflows triggered by external pull requests. Apply NIST AC-6 (Least Privilege) and CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts).

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: Stop the bleeding by restricting the attack surface before eradication; here, that means cutting off the path by which external fork code gains privileged GITHUB_TOKEN scope and secret access.

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

Compensating: For teams without enterprise tooling, run the following bash one-liner across a cloned copy of all repos to surface dangerous triggers: `grep -rl 'pull_request_target\|workflow_run' .github/workflows/ | xargs grep -l 'secrets\|GITHUB_TOKEN'`. Pair with the free tool 'zizmor' (GitHub Actions static analyzer, open source) to batch-scan YAML files for privilege escalation patterns specific to the Cordyceps weakness class without requiring a SIEM.

Evidence: Before restricting GITHUB_TOKEN permissions or revoking secrets access, capture a point-in-time snapshot of: (1) the GitHub Actions audit log export via GitHub API (`GET /orgs/{org}/audit-log?phrase=action:workflows`) filtered to the past 90 days — this log is volatile in the sense that GitHub retains it for only 180 days and you need it before any workflow reruns overwrite context; (2) the exact YAML content of every `.github/workflows/*.yml` file at HEAD for affected repos (git archive or direct API snapshot), preserving the vulnerable trigger/permission configuration as evidence before remediation changes it; (3) the list of all repository secrets and environment secrets currently scoped to affected workflows via `gh secret list --repo /` for each affected repository.

Step 2: Detection — Search workflow YAML files for pull_request_target combined with checkout of fork HEAD (actions/checkout with ref: ${{ github.event.pull_request.head.sha }}). Review GitHub Actions audit logs for unexpected workflow runs initiated by external contributors, secret access events, and GITHUB_TOKEN permission escalations. Map findings to AU-2 (Event Logging) and CIS 8.2 (Collect Audit Logs). Apply D3-LAM (Local Account Monitoring) principles to CI/CD service account activity.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: Correlate GitHub Actions audit log events with workflow YAML configurations to determine which repositories were exposed, which external contributors triggered privileged runs, and whether any secret exfiltration actually occurred — moving from 'misconfiguration exists' to 'exploitation confirmed or ruled out.'

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

Compensating: Without a SIEM, use the GitHub CLI and API directly: `gh api /orgs/{org}/audit-log --paginate --jq '.[] | select(.action=="workflows.completed" or .action=="secret.access")' > audit_export.json`. Then use `jq` to filter for runs where `actor_type` is an external contributor (not an org member) and `conclusion` is not null. Cross-reference the resulting run IDs against the list of repos with the dangerous `pull_request_target` + fork-checkout pattern identified in Step 1. Flag any run where a `secrets` context was resolved during an externally-triggered workflow.

Evidence: This step is analytical and does not alter live state, but evidence to prioritize before any containment actions modify workflow configurations includes: (1) GitHub Actions workflow run logs for all `pull_request_target`-triggered runs by external contributors — downloadable per-run via `gh run view --log`; (2) audit log entries with action types `workflows.completed`, `secret.access`, and `org.oauth_application_tokens_approve` scoped to the exposure window; (3) any workflow run artifacts (uploaded via `actions/upload-artifact`) produced during external-contributor-triggered runs, as these may contain tampered build outputs or exfiltrated data embedded in artifact payloads.

Step 3: Eradication — Replace pull_request_target with pull_request for workflows that process untrusted code, accepting the loss of secret access as the intended security boundary. Where elevated permissions are genuinely required, gate them behind explicit approval steps using GitHub's environment protection rules with required reviewers. Rotate all secrets and tokens that were accessible to affected workflows, per NIST IR controls and D3-CRO (Credential Rotation). Apply CIS 7.1 (Establish and Maintain a Vulnerability Management Process) to track remediation across all affected repositories.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: Remove the root cause (dangerous trigger/permission combination) from every affected workflow file and invalidate all credentials that transited the compromised CI/CD execution context, ensuring the Cordyceps privilege escalation path no longer exists in any repository.

Controls: NIST AC-3 (Access Enforcement), NIST AC-6 (Least Privilege), 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: Track multi-repo remediation progress using a simple CSV or GitHub Project board with columns: repo name, vulnerable workflow file path, trigger type found, remediation PR merged (Y/N), secrets rotated (Y/N), environment protection rule enabled (Y/N). For secret rotation without a secrets manager, use `gh secret set --repo /` for each affected secret after generating new values — document each rotation with a timestamp and the workflow run ID that had access to the prior value.

Evidence: Because this step includes rotating secrets and tokens — which destroys the prior credential values — capture BEFORE rotation: (1) the full list of secrets scoped to each affected workflow (names only, not values) via `gh secret list` for audit trail purposes; (2) any third-party service tokens (e.g., npm publish tokens, PyPI API keys, cloud provider OIDC or static credentials, Docker Hub tokens) stored as repository or organization secrets that were in scope for affected workflows — document the service, token name, and scope so downstream revocation can be verified; (3) GitHub App installation token logs if any Apps were authorized to affected workflows, as these represent an additional lateral movement vector if the Cordyceps weakness was used to exfiltrate App credentials.

Step 4: Recovery — Re-run affected pipelines from a known-clean state after secrets rotation. Validate that build artifacts and package releases produced during the exposure window match expected checksums and have not been tampered with. Enable branch protection rules and require signed commits. Monitor CI/CD pipeline logs for anomalous behavior for at least 30 days post-remediation. Apply AU-6 (Audit Record Review, Analysis, and Reporting) and D3-SFA (System File Analysis) to verify pipeline integrity.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery: Restore CI/CD pipeline integrity by re-executing builds from verified source state with rotated credentials, then confirm that no tampered artifacts from the exposure window reached downstream consumers (package registries, container images, release binaries) before resuming normal release operations.

Controls: NIST AU-6 (Audit Record Review, Analysis, And Reporting), NIST AU-9 (Protection Of Audit Information), CIS 3.4 (IG1/IG2/IG3) — Enforce Data Retention

Compensating: For artifact integrity validation without a commercial tool, use `sha256sum` or `sigstore/cosign` (free, open source) to compare checksums of published packages on npm, PyPI, Maven Central, or GitHub Releases against checksums generated by a clean re-run of the pipeline post-remediation. For signed commits, enable `git config --global commit.gpgsign true` and enforce via GitHub branch protection's 'Require signed commits' setting at no cost. Set up a free GitHub Actions workflow on a cron schedule (`on: schedule`) to re-run the zizmor YAML scanner weekly and alert on any new dangerous trigger patterns introduced.

Evidence: Before re-running pipelines (which will overwrite current run state), preserve: (1) the checksums and metadata of all build artifacts, container image digests, and package release files published to any registry during the exposure window — download and hash these before a clean re-run makes comparison ambiguous; (2) any GitHub Releases or git tags created during the exposure window, preserved as git bundle archives (`git bundle create repo-exposure-window.bundle --all`) so tag provenance can be audited independently of GitHub's UI; (3) the current state of all branch protection rule configurations per affected repo (exportable via `gh api /repos/{owner}/{repo}/branches/{branch}/protection`) before enabling new protections, to document the pre-remediation security posture for post-incident reporting.

Step 5: Post-Incident — Implement a recurring CI/CD workflow security review process. Adopt GitHub's principle of minimal token permissions as an organizational standard (permissions blocks in every workflow). Integrate workflow YAML linting into code review gates to flag dangerous trigger/permission combinations before merge. Map gaps to NIST AC-3 (Access Enforcement), CIS 7.2 (Establish and Maintain a Remediation Process), and NIST CM controls to formalize configuration hardening of CI/CD infrastructure as a standing control.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: Conduct a lessons-learned review to convert the Cordyceps detection and remediation experience into durable organizational controls — specifically, a CI/CD workflow security standard and automated pre-merge linting gate — so that the same class of pull_request_target privilege escalation

cannot be reintroduced by future contributors.

Controls: NIST AC-3 (Access Enforcement), NIST AC-6 (Least Privilege), NIST AU-2 (Event Logging), CIS 4.6 (IG1/IG2/IG3) — Securely Manage Enterprise Assets and Software, CIS 7.2 (IG1/IG2/IG3) — Establish and Maintain a Remediation Process

Compensating: Implement YAML linting as a required status check using the free open-source tool 'actionlint' (static checker for GitHub Actions workflows) configured as a pre-merge required check via a simple GitHub Actions workflow triggered on `pull_request`. Encode the Cordyceps rule explicitly: actionlint flags `pull_request_target` with unsafe `ref` expressions by default. Additionally, create an org-level reusable workflow template repository with pre-approved, hardened workflow patterns and require teams to extend from it via GitHub's `workflow_call` mechanism, enforcing the `permissions: read-all` default organizationally without relying on individual developer discipline.

Evidence: No volatile live-state evidence capture is required for this phase; the relevant evidence to retain from the incident for post-incident documentation includes: (1) the final audit log export covering the full exposure window, archived to immutable storage (S3 Object Lock, Azure Blob immutability, or equivalent) per NIST AU-11 (Audit Record Retention) requirements; (2) the lessons-learned report documenting which repositories were confirmed vulnerable, which were confirmed exploited (if any), the timeline from configuration introduction to detection, and the remediation gap analysis; (3) the before/after YAML diffs for every remediated workflow file, stored in a post-incident evidence repository as a reference baseline for the recurring CI/CD workflow security review process.

Detection Guidance

Primary detection surface is GitHub Actions workflow YAML files and the GitHub Actions audit log. Search all workflow files (.github/workflows/*.yml) for the string pull_request_target; flag any workflow that also checks out fork code using actions/checkout with a fork HEAD reference. In the GitHub audit log (accessible via organization settings or the REST API), filter for workflow_run events initiated by external contributors (actors outside your organization) and correlate with any secret_scanning or token permission events in the same timeframe. Behavioral indicators include unexpected package releases or artifact uploads following external pull request activity, CI jobs accessing secrets they do not normally require, and workflow runs completing successfully on pull requests from first-time or low-reputation contributors. Apply NIST AU-6 (Audit Record Review, Analysis, and Reporting) review cadence to CI/CD logs. D3-SFA (System File Analysis) applies to workflow YAML monitoring for unauthorized modification. No network-based IOCs are available from current T3 sources; IOC list is empty pending primary research verification.

Framework Mappings

MITRE-ATTACK

- **T1552.001** — Credentials In Files
- **T1190** — Exploit Public-Facing Application
- **T1059** — Command and Scripting Interpreter
- **T1078** — Valid Accounts
- **T1195.001** — Compromise Software Dependencies and Development Tools

NIST-800-53R5

- **CA-8** — Penetration Testing
- **RA-5** — Vulnerability Monitoring and Scanning
- **SC-7** — Boundary Protection

- **SI-2** — Flaw Remediation
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **AC-3** — Access Enforcement
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A01:2021** — Broken Access Control

CIS-V8

- **5.4** — Restrict Administrator Privileges to Dedicated Administrator Accounts
- **6.8** — Define and Maintain Role-Based Access Control
- **3.3** — Configure Data Access Control Lists
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers

HIPAA-SECURITY

- **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

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain

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
T1190	Exploit Public-Facing Application	Initial-Access

Technique ID	Technique Name	Tactic
T1059	Command and Scripting Interpreter	Execution
T1078	Valid Accounts	Defense-Evasion
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access

Sources

Source	URL	Tier
Cordyceps CI/CD Flaws Expose 300+ GitHub Repositories to ...	https://thehackernews.com/2026/06/cordyceps-cicd-flaws-expose-300-g...	T3
'Cordyceps': Malicious Pull Requests Threaten CI/CD Workflows	https://www.darkreading.com/application-security/cordyceps-maliciou...	T3
Achieving DevSecOps with GitHub Advanced Security - YouTube	https://www.youtube.com/watch?v=hln5v7odguE	T3
How we found vulnerabilities in GitHub Actions CI/CD pipelines	https://cycode.com/blog/github-actions-vulnerabilities/	T3
Preventative Measures Against Supply Chain Attacks in GitHub ...	https://github.com/orgs/community/discussions/154124	T3

Generated 2026-06-24 18:54 UTC by TJS Security Command Center