INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-27 06:37 UTC

Megalodon Campaign Poisons 5,500+ GitHub Repos in Six Hours, Developer Credentials and Secrets at Scale

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0370
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	GitHub public repositories (5,500+ confirmed affected); developers with credentials or secrets stored in repository environments
Published	2026-05-26T15:47:14
Discovery Source	Rss

Executive Summary

A threat campaign designated 'Megalodon' compromised more than 5,500 public GitHub repositories within a six-hour window, extracting GitHub Actions secrets and developer credentials at scale using purpose-built automation. Any organization whose CI/CD pipelines consume or depend on affected repositories faces downstream exposure; stolen credentials could enable unauthorized access to production environments, cloud accounts, and internal systems. This is a high-priority software supply chain event requiring immediate audit of repository secrets and CI/CD pipeline dependencies.

Technical Analysis

The Megalodon campaign executed a mass supply chain attack against GitHub public repositories, injecting malicious commits across 5,500+ repos in approximately six hours, a tempo indicating purpose-built automation. The primary objective was exfiltration of GitHub Actions secrets and developer credentials to attacker-controlled infrastructure. Attack methodology exploited three root-cause weakness classes: hardcoded credentials (CWE-798), plaintext storage of sensitive information (CWE-312), and insufficient verification of software provenance during dependency resolution (CWE-494). A fourth contributor, reliance on insufficiently trusted components (CWE-1357), amplifies downstream impact. MITRE ATT&CK techniques observed include T1195.001 (Compromise Software Dependencies and Development Tools), T1552.001 (Credentials in Files), T1552.004 (Private Keys), T1078.004 (Cloud Accounts), T1213 (Data from Information Repositories), and T1567 (Exfiltration Over Web Service). T1059 (Command and Scripting Interpreter) applies to injected malicious

workflow steps executed via GitHub Actions runners; T1053 (Scheduled Task/Job) applies to the automated campaign execution cadence. No CVE is assigned; this is a process and configuration failure, not a discrete software defect. The campaign was identified and reported by StepSecurity. Source quality is moderate (0.64); the StepSecurity blog post is the primary technical reference. Organizations should treat any secret stored in affected repos as fully compromised and rotate immediately.

Action Checklist

1. Step 1: Containment. Identify all GitHub repositories your organization owns or consumes in CI/CD pipelines. Cross-reference against the StepSecurity Megalodon affected repository list (https://www.stepsecurity.io/blog/megalodon-mass-github-actions-secret-exfiltration-across-5-500-public-repositories). Suspend pipeline execution against any confirmed or suspected affected upstream dependency until verified clean. Revoke and rotate all GitHub Actions secrets, API tokens, and cloud credentials stored in repository environments; treat them as compromised regardless of confirmation status. Apply NIST IR-4 (Incident Handling) procedures immediately.
2. Step 2: Detection. Query GitHub audit logs for unexpected commit activity, workflow modifications, or secret access events in the six-hour window of the campaign (exact window reported by StepSecurity). Search CI/CD pipeline logs for outbound connections to unknown or unrecognized infrastructure originating from Actions runner processes (NIST AU-6, AU-12). Review workflow YAML files in all consumed repositories for unauthorized modifications to run steps, added curl/wget calls, or newly introduced third-party Actions references. Check cloud provider access logs (AWS CloudTrail, Azure Monitor, GCP Audit Logs) for API calls using credentials stored in GitHub Actions secrets; look for calls from unexpected IPs or at unusual times (NIST AU-3). Indicator pattern: unexpected secrets access events in GitHub audit log + outbound data transfer from Actions runners to non-organizational endpoints.
3. Step 3: Eradication. Rotate all exposed secrets immediately: GitHub Personal Access Tokens, cloud IAM keys (AWS, Azure, GCP), deployment keys, and any API credentials stored in repository secrets or environment variables (CIS 5.2, D3-CRO). Remove hardcoded credentials from all repository code and history using tools such as git-filter-repo or BFG Repo Cleaner. Enforce secret scanning on all repositories; enable GitHub's native secret scanning and push protection to block future credential commits (NIST CM-7, CIS 4.6). For any repository confirmed in the affected list, audit the full commit history for injected malicious content and revert unauthorized commits.
4. Step 4: Recovery. Validate that all rotated credentials have been updated in every consuming system, pipeline, and service before resuming CI/CD operations. Re-run pipeline builds from a known-clean state using verified dependency versions; pin Actions to commit SHAs rather than mutable tags (addresses CWE-494). Monitor cloud and system access logs for 30 days post-rotation for any continued unauthorized access attempts using previously valid credentials. Confirm secret scanning alerts are active and routing to your security team (NIST SI-4, AU-6). Verify no malicious workflow steps remain in any pipeline configuration.
5. Step 5: Post-Incident. Conduct a secrets hygiene audit across all repositories and CI/CD systems; implement a secrets management solution (e.g., HashiCorp Vault, AWS Secrets Manager, Azure Key Vault) to eliminate plaintext credential storage (addresses CWE-312, CWE-798; aligns with NIST IA-5, CIS 3.6). Establish a software provenance verification process for all third-party Actions and dependencies; require SHA-pinning and maintain an approved Actions allowlist (addresses CWE-494, CWE-1357; aligns with NIST SA-12, CIS 2.1, CIS 2.3). Implement StepSecurity Harden-Runner or equivalent to monitor and restrict outbound network calls from GitHub Actions runners. Document lessons learned and update CI/CD

security policy (NIST IR-8).

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal, and cloud infrastructure leadership immediately if CloudTrail, Azure Monitor, or GCP Audit Logs confirm that credentials stolen via Megalodon were used to access production environments, cloud storage containing PII/PHI, or customer-facing infrastructure — triggering breach notification obligations under GDPR Article 33, HIPAA §164.412, or applicable state breach laws.
Recovery Notes	Before resuming any CI/CD pipeline, verify every secret reference in every workflow file resolves to a newly rotated credential and that all third-party Actions are pinned to audited commit SHAs — do not accept mutable tag references as clean. Monitor AWS CloudTrail, Azure Monitor, and GCP Audit Logs daily for 30 days post-rotation specifically for API calls using the previously compromised key IDs, as threat actors may have cached credentials for delayed use. Confirm StepSecurity Harden-Runner or equivalent egress monitoring is active on all runners before re-enabling automated deployments to production environments.
Forensic Artifacts	GitHub audit log JSON export filtered for `action:secret_accessed`, `action:workflow_run.completed`, and `action:git.push` events during the Megalodon six-hour campaign window — primary artifact establishing which repos and secrets were accessed by the automation | Raw `.github/workflows/*.yml` files at the malicious commit SHA from all affected upstream repos — contain the injected `run:` block with the base64-encoded or plaintext `curl`/`wget` exfiltration command that characterizes the Megalodon payload delivery mechanism | GitHub Actions runner logs (downloadable via `gh run download `) for workflow executions during the campaign window — may contain stdout output showing data exfiltration attempts, outbound POST request targets, or error messages from the attacker's collection infrastructure | AWS CloudTrail / Azure Monitor / GCP Audit Log entries for the campaign window and 72 hours post-campaign, filtered on API calls originating from source IPs outside GitHub Actions published IP ranges (`https://api.github.com/meta`, field `actions`) using credentials matching those stored in affected repository secrets | IAM credential last-used reports (`aws iam get-access-key-last-used`) for all rotated keys — establishes the precise timestamp of last adversary use, critical for breach notification timelines and determining scope of unauthorized cloud resource access

Per-Action IR Details

Step 1: Containment — Identify all GitHub repositories your organization owns or consumes in CI/CD pipelines. Cross-reference against the StepSecurity Megalodon affected repository list (https://www.stepsecurity.io/blog/megalodon-mass-github-actions-secret-exfiltration-across-5-500-public-repositories). Suspend pipeline execution against any confirmed or suspected affected upstream dependency until verified clean. Revoke and rotate all GitHub Actions secrets, API tokens, and cloud credentials stored in repository environments — treat them as compromised regardless of confirmation status. Apply NIST IR-4 (Incident Handling) procedures immediately.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST AC-2 (Account Management), NIST AC-17 (Remote Access), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Use the GitHub CLI (`gh`) to enumerate all repos your org owns: `gh repo list YOUR_ORG --limit 1000 --json name,url`. For consumed dependencies, parse all `.github/workflows/*.yml` files in your repos with: `grep -r 'uses:' .github/workflows/ | grep -v '#'` to extract third-party Actions references. Cross-reference the resulting list against the StepSecurity affected repo list using a simple bash diff. To revoke credentials without an enterprise vault, iterate GitHub Actions secrets via `gh secret list --repo REPO` and rotate manually via the GitHub Settings UI or API (`gh secret set`). For AWS IAM keys exposed in Actions secrets, use `aws iam delete-access-key` immediately, then create replacement keys and update all pipeline secret stores.

Evidence: Before suspending pipelines or revoking credentials, snapshot the current state: export GitHub Actions secret metadata (names, last-used timestamps) via the GitHub REST API (`GET /repos/{owner}/{repo}/actions/secrets`) — note you cannot retrieve secret values, but names and update timestamps establish a baseline. Export the GitHub audit log for the campaign window using `gh api /orgs/{org}/audit-log?phrase=action:secret&limit=100`. Capture `.github/workflows/*.yml` file hashes (`sha256sum .github/workflows/*.yml`) before any rollback, as these files are the primary tamper artifact in Megalodon-style workflow injection attacks.

Step 2: Detection — Query GitHub audit logs for unexpected commit activity, workflow modifications, or secret access events in the six-hour window of the campaign (exact window reported by StepSecurity). Search CI/CD pipeline logs for outbound connections to unknown or unrecognized infrastructure originating from Actions runner processes (NIST AU-6, AU-12). Review workflow YAML files in all consumed repositories for unauthorized modifications to run steps, added curl/wget calls, or newly introduced third-party Actions references. Check cloud provider access logs (AWS CloudTrail, Azure Monitor, GCP Audit Logs) for API calls using credentials stored in GitHub Actions secrets — look for calls from unexpected IPs or at unusual times (NIST AU-3). Indicator pattern: unexpected secrets access events in GitHub audit log + outbound data transfer from Actions runners to non-organizational endpoints.

NIST Phase: Detection Analysis

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

Controls: NIST AU-3 (Content of Audit Records), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-12 (Audit Record Generation), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: Query GitHub audit log via CLI for secret access events during the Megalodon campaign window: `gh api '/orgs/{org}/audit-log?phrase=action:secret_accessed&per_page=100'`. Use `jq` to filter by timestamp: `jq '.[] | select(.@timestamp >= 1700000000000)'` (replace with actual epoch for the campaign window per StepSecurity advisory). For workflow YAML diff analysis, use `git log --all --full-history -- '.github/workflows/*.yml'` on each cloned dependency repo to surface unauthorized commits. For AWS CloudTrail, run: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=AssumeRole --start-time --end-time ` filtering for source IPs not belonging to GitHub Actions IP ranges (published at `https://api.github.com/meta`, field `actions`). Use `gron` or `jq` to flatten JSON output for manual review without a SIEM.

Evidence: Capture before analysis: (1) GitHub audit log export covering the full six-hour Megalodon window in JSON format — specifically filter for `action:secret_accessed`, `action:workflow_run.completed`, and `action:git.push` event types. (2) Raw `.github/workflows/*.yml` content from all affected upstream repos at HEAD and at the commit SHA immediately preceding the campaign window — the diff isolates injected `run:` steps containing `curl`/`wget` exfiltration commands characteristic of Megalodon. (3) GitHub Actions runner logs (downloadable via `gh run download`) for any workflow executions that ran during the campaign window — look for stdout containing base64-encoded outbound POST requests. (4) AWS CloudTrail / Azure Monitor / GCP Audit Log entries showing API calls made with credentials matching those stored in GitHub Actions secrets, particularly `sts:AssumeRole`, `s3:GetObject`, or equivalent cross-cloud privilege escalation calls from non-organizational source IPs.

Step 3: Eradication — Rotate all exposed secrets immediately: GitHub Personal Access Tokens, cloud IAM keys (AWS, Azure, GCP), deployment keys, and any API credentials stored in repository secrets or environment variables (CIS 5.2, D3-CRO). Remove hardcoded credentials from all repository code and history using tools such as git-filter-repo or BFG Repo Cleaner. Enforce secret scanning on all repositories — enable GitHub's native secret scanning and push protection to block future credential commits (NIST CM-7, CIS 4.6).

For any repository confirmed in the affected list, audit the full commit history for injected malicious content and revert unauthorized commits.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST CM-7 (Least Functionality), NIST IA-5 (Authenticator Management), NIST SI-2 (Flaw Remediation), CIS 5.2 (Use Unique Passwords), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: For hardcoded credential removal, run BFG Repo Cleaner (free, Java-based): `java -jar bfg.jar --replace-text credentials.txt YOUR_REPO.git` where `credentials.txt` contains the exposed token values. Follow with `git reflog expire --expire=now --all && git gc --prune=now --aggressive` and force-push. For secret rotation without a vault, script AWS key rotation: `aws iam create-access-key --user-name SERVICE_USER` then `aws iam delete-access-key --access-key-id OLD_KEY_ID`. Enable GitHub secret scanning via the API: `gh api --method PUT /repos/{owner}/{repo}/vulnerability-alerts` and `gh api --method PUT /repos/{owner}/{repo}/automated-security-fixes`. Use `truffleHog` (free, open source) to scan full git history for any remaining credential patterns: `trufflehog git file://./repo --only-verified`.

Evidence: Before rewriting git history or rotating keys, preserve: (1) Full `git log --all --oneline --graph` output from each affected repo to document the unauthorized commit SHAs introduced during Megalodon — these are forensic evidence of the injection chain. (2) Output of `git show ` capturing the exact injected workflow step payload (typically a `run:` block with base64-encoded exfiltration script). (3) GitHub API response from `GET /repos/{owner}/{repo}/actions/secrets` capturing secret names and `updated_at` timestamps at time of discovery — establishes which secrets were present and accessible during the campaign window. (4) Cloud provider IAM access advisor output (`aws iam generate-service-last-accessed-details`) for all compromised IAM keys, documenting which services were accessed before rotation.

Step 4: Recovery — Validate that all rotated credentials have been updated in every consuming system, pipeline, and service before resuming CI/CD operations. Re-run pipeline builds from a known-clean state using verified dependency versions — pin Actions to commit SHAs rather than mutable tags (addresses CWE-494). Monitor cloud and system access logs for 30 days post-rotation for any continued unauthorized access attempts using previously valid credentials. Confirm secret scanning alerts are active and routing to your security team (NIST SI-4, AU-6). Verify no malicious workflow steps remain in any pipeline configuration.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-4 (System Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST CM-3 (Configuration Change Control), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Validate SHA-pinning compliance across all workflow files using: `grep -r 'uses:' .github/workflows/ | grep -v '@[a-f0-9]\{40\}'` — any result not matching a full 40-character SHA is unpinned and must be remediated. To verify no malicious `run:` steps remain, use Semgrep (free) with a custom rule targeting `curl`, `wget`, or `base64` calls inside `jobs.*.steps[].run` blocks: `semgrep --config auto .github/workflows/`. For 30-day post-rotation cloud monitoring without a SIEM, schedule a daily cron job running AWS CloudTrail Insights queries for `ErrorCode: UnauthorizedAccess` filtering on previously compromised key IDs. Use `gitleaks` (free) in pre-commit hooks to block re-introduction of credentials: `gitleaks protect --staged`.

Evidence: Before resuming CI/CD, collect: (1) Diff output comparing current `.github/workflows/*.yml` files against the last known-clean commit SHA (pre-campaign) for every first-party and consumed dependency repo — confirms no residual injected steps. (2) GitHub Actions runner network logs or StepSecurity Harden-Runner audit output from a test pipeline run showing zero outbound connections to non-organizational endpoints (establishes clean baseline). (3) Cloud IAM credential report (`aws iam generate-credential-report`) confirming all old access keys are in `Inactive` state with `last_used` timestamps predating rotation — validates revocation was complete. (4) Alert configuration export from GitHub Advanced Security showing secret scanning is enabled and alert routing is confirmed to the security team's notification channel.

Step 5: Post-Incident — Conduct a secrets hygiene audit across all repositories and CI/CD systems; implement a secrets management solution (e.g., HashiCorp Vault, AWS Secrets Manager, Azure Key Vault) to eliminate plaintext credential storage (addresses CWE-312, CWE-798; aligns with NIST IA-5, CIS 3.6). Establish a software provenance verification process for all third-party Actions and dependencies — require SHA-pinning and maintain an approved Actions allowlist (addresses CWE-494, CWE-1357; aligns with NIST SA-12, CIS 2.1, CIS 2.3). Implement StepSecurity Harden-Runner or equivalent to monitor and restrict outbound network calls from GitHub Actions runners. Document lessons learned and update CI/CD security policy (NIST IR-8).

NIST Phase: Post Incident

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

Controls: NIST IA-5 (Authenticator Management), NIST IR-8 (Incident Response Plan), NIST SA-12 (Supply Chain Protection), NIST CM-7 (Least Functionality), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.3 (Address Unauthorized Software), CIS 3.6 (Encrypt Data on End-User Devices)

Compensating: For secrets hygiene audit without enterprise tooling, run `truffleHog` and `gitleaks` across all org repos in parallel using a bash loop: `for repo in $(gh repo list YOUR_ORG --json name -q '.[].name'); do gitleaks detect --source=./repos/$repo --report-format json --report-path=./reports/$repo.json; done`. For an approved Actions allowlist without enterprise policy enforcement, create an OPA (Open Policy Agent) Rego policy checked in CI that rejects any workflow referencing an Action not in a maintained `allowlist.txt` — enforced via a required status check. For StepSecurity Harden-Runner alternative on a zero budget, add a `block-egress` step using `iptables` in a self-hosted runner setup to deny all outbound except known GitHub and registry endpoints. Document lessons learned in a structured format referencing NIST 800-61r3 §4 and submit IOCs (malicious Action SHAs, exfiltration endpoints) to CISA's CSRB or GitHub Security.

Evidence: For the lessons-learned record, preserve: (1) Final inventory of all secrets confirmed exposed during Megalodon — mapped to the systems and services they granted access to, establishing the true blast radius for breach notification assessment. (2) Timeline reconstruction from GitHub audit logs documenting when each affected upstream repo was compromised, when your pipelines consumed the malicious version, and when containment actions were completed — this is the core artifact for any regulatory disclosure timeline. (3) Completed secrets inventory report from `gitleaks`/`truffleHog` post-eradication scans (clean run output) confirming zero residual plaintext credentials across all org repos. (4) Network egress baseline from StepSecurity Harden-Runner or equivalent for all GitHub Actions runners — documents approved outbound endpoints as the foundation for future anomaly detection.

Detection Guidance

Primary detection surface is GitHub audit logs and CI/CD pipeline execution logs. Query GitHub audit logs for: (1) unexpected commits from unfamiliar accounts or automated actors during the campaign window; (2) workflow file modifications (`.github/workflows/*.yml`) not authorized by your team; (3) secrets access events outside normal pipeline execution patterns. In cloud provider logs, search for API calls using credentials stored as GitHub Actions secrets originating from IP ranges outside GitHub's documented Actions runner IP blocks; GitHub publishes these via the API (`https://api.github.com/meta`). Behavioral indicators: outbound HTTP/S requests from Actions runners to non-organizational domains, especially during build steps that should not generate external traffic; newly introduced `curl`, `wget`, or PowerShell download commands in workflow YAML. For local repository audit, diff workflow YAML files against your last known-good commit to identify injected steps. If using a SIEM, correlate GitHub audit log events (Actions secret access) with cloud provider API call logs for the same time window; a secret accessed in GitHub Actions that immediately appears in a cloud API call from an unexpected IP is a high-confidence indicator of compromise. Reference NIST AU-6 for audit review requirements and CIS 8.2 for audit log collection baselines.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://www.stepsecurity.io/blog/megalodon-mass-github-actions-secret-exfiltration-across-5-500-public-repositories`	StepSecurity primary research report — contains affected repository list and campaign technical details	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1059** — Command and Scripting Interpreter
- **T1053** — Scheduled Task/Job
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1552.001** — Credentials In Files
- **T1552.004** — Private Keys
- **T1078.004** — Cloud Accounts
- **T1213** — Data from Information Repositories
- **T1567** — Exfiltration Over Web Service

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-7** — Software, Firmware, and Information Integrity
- **AC-3** — Access Enforcement
- **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
- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **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.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain

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

NIST-CSF-2

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1059	Command and Scripting Interpreter	Execution
T1053	Scheduled Task/Job	Execution
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1552.001	Credentials In Files	Credential-Access
T1552.004	Private Keys	Credential-Access
T1078.004	Cloud Accounts	Defense-Evasion
T1213	Data from Information Repositories	Collection
T1567	Exfiltration Over Web Service	Exfiltration

Sources

Source	URL	Tier
Security News	https://www.darkreading.com/application-security/megalodon-malware-...	T3
Over 5,500 GitHub Repositories Infected in 'Megalodon' Supply ...	https://www.securityweek.com/over-5500-github-repositories-infected...	T3

Source	URL	Tier
Security researchers say 5,500 GitHub repositories have ... - Facebook	https://www.facebook.com/mashablefutureshift/posts/security-researc...	T3
Megalodon: Mass GitHub Actions Secret Exfiltration ... - StepSecurity	https://www.stepsecurity.io/blog/megalodon-mass-github-actions-secr...	T3
Megalodon cyberattack infects 5,500 GitHub repositories, report says	https://mashable.com/tech/megalodon-cyberattack-github-repositories...	T3

Generated 2026-05-27 06:37 UTC by TJS Security Command Center