INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-19 13:50 UTC

Tag Hijacking in actions-cool Workflows Exposes CI/CD Pipelines to Active Credential Exfiltration

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0337
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	GitHub Actions: actions-cool/issues-helper (all tag-referenced versions), actions-cool/maintain-one-comment (all tag-referenced versions); npm @antv ecosystem packages (scope per Mini Shai-Hulud campaign)
Published	2026-05-19T01:28:06
Discovery Source	Rss

Executive Summary

Threat actors redirected all version tags for two widely-used GitHub Actions, actions-cool/issues-helper and actions-cool/maintain-one-comment, to malicious commits, causing any CI/CD pipeline referencing those actions by tag to automatically execute attacker-controlled code. The payload actively exfiltrates CI/CD secrets, credentials, and tokens to an attacker-controlled domain. The same infrastructure connects to the 'Mini Shai-Hulud' npm supply chain campaign, attributed to TeamPCP with high confidence per Wiz research, confirming a coordinated, multi-ecosystem operation that puts software build pipelines and everything they touch at direct risk of compromise.

Technical Analysis

TeamPCP compromised actions-cool/issues-helper and actions-cool/maintain-one-comment by force-pushing malicious commits and reassigning all existing mutable Git tags to those commits. Any workflow referencing these actions by tag (e.g., @v3, @latest), rather than a pinned commit SHA, executed the malicious payload on its next scheduled or triggered run. The payload performs credential exfiltration (T1552.001) via outbound data transfer (T1041, T1567) to an attacker-controlled domain, targeting CI/CD secrets, API tokens, and environment variables. No CVE has been assigned because this attack exploits Git's mutable tag architecture (a design characteristic), not a patched vulnerability or code flaw. Remediation is architectural (SHA pinning), not a patch deployment. Relevant CWEs: CWE-829 (Inclusion of Functionality from Untrusted Control Sphere), CWE-312 (Cleartext Storage of Sensitive Information), CWE-494 (Download of Code Without Integrity Check). MITRE techniques include T1195.001 (Supply Chain Compromise: Compromise Software Dependencies), T1078.004

(Valid Accounts: Cloud Accounts), T1059/T1059.007 (Command and Scripting Interpreter: JavaScript), and T1553 (Subvert Trust Controls). The exfiltration infrastructure overlaps with the Mini Shai-Hulud campaign targeting the @antv npm scope (per Wiz research), indicating shared tooling or operator overlap within TeamPCP. No patch exists, remediation is architectural: pin all Actions to verified commit SHAs. Affected scope: all tag-referenced versions of both actions; any npm @antv packages in scope of the parallel campaign.

Action Checklist

1. Step 1: Containment, Immediately audit all workflow files (.github/workflows/*.yml) across every repository for references to actions-cool/issues-helper or actions-cool/maintain-one-comment. Disable or block any workflow referencing these actions by mutable tag until SHA pinning is confirmed. If workflows ran against these actions after the tags were compromised (verify exact date against current GitHub or CISA advisory), treat all secrets accessible in those runs as compromised.
2. Step 2: Detection, Review GitHub Actions run logs for any job invoking actions-cool/issues-helper or actions-cool/maintain-one-comment. Look for unexpected outbound network connections in runner logs, environment variable reads, or curl/wget calls to unrecognized domains. Check SIEM and DNS logs for outbound connections to attacker-controlled domains associated with this campaign (consult current threat intelligence advisories for confirmed IOCs). Query secrets managers and cloud IAM logs for anomalous API calls or token usage originating from CI/CD pipeline service accounts around the time of any suspect workflow runs.
3. Step 3: Eradication, Replace all mutable tag references (e.g., @v3) with pinned commit SHAs verified against the known-good state of each action's repository. Rotate all secrets, tokens, API keys, and credentials that were accessible in any workflow environment where these actions executed. Remove and regenerate GitHub Actions secrets, cloud provider access keys, npm publish tokens, and any other credentials scoped to affected runners. Audit and revoke any access tokens that show anomalous usage patterns.
4. Step 4: Recovery, After rotating credentials, re-run affected workflows under controlled conditions and confirm expected behavior. Validate that all workflow files now reference pinned SHAs. Monitor CI/CD pipeline logs and cloud IAM/audit logs for 30 days post-remediation for anomalous access patterns using the rotated credentials. Confirm no unauthorized packages were published to registries under your organization's scope during the exposure window.
5. Step 5: Post-Incident, This attack exploited systemic over-reliance on mutable Git tags in CI/CD pipelines. Implement a policy requiring all third-party GitHub Actions to be pinned by commit SHA (enforced via tooling such as OpenSSF Scorecards, StepSecurity Harden-Runner, or Dependabot for Actions). Add workflow scanning to your CI/CD pipeline to flag mutable tag references before merge. Review the broader software supply chain: audit npm dependencies for @antv scope packages per the Mini Shai-Hulud advisory. Map this incident to NIST SP 800-161 (supply chain risk management) and consider adding third-party action reviews to your software acquisition process.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate to CISO, legal counsel, and potentially affected cloud/registry providers immediately if forensic review of GitHub Actions runner logs or cloud IAM audit trails confirms that GITHUB_TOKEN, AWS access keys, npm publish tokens, or other credentials were read by the malicious actions-cool payload and subsequently used outside of expected CI/CD activity — this constitutes confirmed credential exfiltration and may trigger breach notification obligations if the pipeline had access to customer data, PII, or regulated environments.
Recovery Notes	After credential rotation, validate recovery by confirming that all GitHub Actions secrets, cloud provider access keys, and npm publish tokens show first-use timestamps no earlier than their post-rotation creation date — any earlier usage indicates a credential copy was retained by the attacker. Monitor cloud IAM logs (AWS CloudTrail, GCP Audit Logs, or Azure Activity Log) for the rotated service account identities for a minimum of 30 days, specifically watching for API calls to secrets managers, container registries, and package publish endpoints that deviate from your established CI/CD baseline. Additionally, monitor your organization's npm scope for unauthorized package publishes for the same 30-day window, as TeamPCP/Mini Shai-Hulud attribution suggests the threat actor's end goal includes registry-level supply chain persistence beyond simple credential theft.
Forensic Artifacts	GitHub Actions raw runner logs (.zip archives downloadable per run via GitHub API: GET /repos/{owner}/{repo}/actions/runs/{run_id}/logs) for all workflow runs invoking actions-cool/issues-helper or actions-cool/maintain-one-comment after approximately May 1, 2026 — these logs will contain stdout output of the malicious action steps, including any env variable dumps, curl/wget exfiltration calls, and the specific attacker-controlled domain contacted during the tag-hijack payload execution. | Git object database entries for the attacker-controlled commit SHA that version tags (e.g., v3, v1) were redirected to in actions-cool/issues-helper and actions-cool/maintain-one-comment — capture via 'git cat-file -p MALICIOUS_SHA' and 'git show MALICIOUS_SHA --stat' before GitHub potentially force-pushes remediation, as this preserves the exact payload code that executed in your pipeline. | Cloud provider IAM/audit logs filtered on the CI/CD runner service account identity during the exposure window — specifically AWS CloudTrail events for 'secretsmanager:GetSecretValue', 'iam:CreateAccessKey', 's3:GetObject', and 'sts:AssumeRole'; GCP Audit Logs for 'iam.serviceAccounts.actAs' and 'secretmanager.versions.access'; these reveal whether the exfiltrated GITHUB_TOKEN or cloud credentials were used for secondary access beyond the initial pipeline run. | DNS query logs from runner infrastructure (corporate resolver, Route 53 Resolver Query Logs, or Cloudflare Gateway logs) filtered on source IPs belonging to GitHub-hosted or self-hosted runner subnets, for the exposure window — the TeamPCP/Mini Shai-Hulud payload exfiltrates to attacker-controlled domains, making DNS telemetry the highest-fidelity network indicator of successful exfiltration and the primary artifact for establishing which runs successfully transmitted secrets. | npm registry publish audit trail for your organization's package scope — retrieve full publish history via 'npm view @YOUR-SCOPE/* time --json' for all packages and cross-reference against internal release records; unexpected publish events during the exposure window, particularly for @antv-scoped packages matching the Mini Shai-Hulud IOC list, indicate the attacker leveraged stolen npm publish tokens to inject malicious package versions into your supply chain downstream of the CI/CD compromise.

Per-Action IR Details

Step 1: Containment — Immediately audit all workflow files (.github/workflows/*.yml) across every repository for references to actions-cool/issues-helper or actions-cool/maintain-one-comment. Disable or block any workflow referencing these actions by mutable tag until SHA pinning is confirmed. If workflows ran against these actions after approximately May 2026, treat all secrets accessible in those runs as compromised.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST CM-2 (Baseline Configuration), NIST SI-7 (Software, Firmware, and Information Integrity), CIS 2.3 (Address Unauthorized Software), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Run this one-liner across all repos to enumerate affected workflow files immediately: grep -r 'actions-cool/issues-helper\|actions-cool/maintain-one-comment' .github/workflows/ --include='*.yml' -l. For orgs with many repos, use the GitHub CLI: gh repo list YOUR-ORG --limit 1000 --json name -q '.[].name' | xargs -I{} sh -c 'gh api repos/YOUR-ORG/{}/contents/.github/workflows --jq ".[].name" 2>/dev/null | xargs -I%% gh api repos/YOUR-ORG/{}/contents/.github/workflows/%% --jq ".content" | base64 -d | grep -l actions-cool'. Disable affected workflows immediately by renaming the .yml extension to .yml.disabled via git commit to main, or set branch protection rules to require SHA review before re-enabling.

Evidence: Before disabling any workflow, capture the following: (1) Full text of each affected .github/workflows/*.yml file — screenshot or git-export to preserve the mutable tag reference (e.g., uses: actions-cool/issues-helper@v3) that was in place at time of compromise. (2) GitHub Actions run history for each affected repo via the GitHub UI or API: GET /repos/{owner}/{repo}/actions/runs — capture run IDs, triggered timestamps, triggering actor, and conclusion status for all runs after approximately May 1, 2026. (3) The malicious commit SHA that tags were redirected to — retrieve via: git ls-remote https://github.com/actions-cool/issues-helper.git 'refs/tags/*' and compare against known-good SHAs documented in public disclosures. Preserve this as the attacker-controlled commit reference before GitHub potentially remediates the repo.

Step 2: Detection — Review GitHub Actions run logs for any job invoking actions-cool/issues-helper or actions-cool/maintain-one-comment. Look for unexpected outbound network connections in runner logs, environment variable reads, or curl/wget calls to unrecognized domains. Check SIEM and DNS logs for outbound connections to attacker-controlled domains associated with this campaign (see IOC field). Query secrets managers and cloud IAM logs for anomalous API calls or token usage originating from CI/CD pipeline service accounts around the time of any suspect workflow runs.

NIST Phase: Detection Analysis

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

Controls: NIST IR-5 (Incident Monitoring), 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: For teams without SIEM: (1) Pull raw GitHub Actions logs per run via GitHub CLI: gh run view RUN_ID --log > run_RUN_ID.log, then grep for 'curl\|wget\|http\|GITHUB_TOKEN\|AWS_\|SECRET' across all captured logs. (2) If runners are self-hosted, review /var/log/syslog or journalctl output on runner hosts for outbound TCP connections during the window of suspect runs; use tcpdump -i any -w capture.pcap 'not port 22' if active runner traffic can be replicated. (3) For AWS IAM: run aws cloudtrail lookup-events --lookup-attributes AttributeKey=Username,AttributeValue=CICD-SERVICE-ACCOUNT-NAME --start-time 2026-05-01 to surface API calls made under pipeline credentials. (4) For npm publish token abuse tied to Mini Shai-Hulud/TeamPCP: check npm audit log at ~/.npm/_logs/ and query the npm registry for unexpected publish events under your org scope.

Evidence: Capture these artifacts before any log rotation occurs: (1) Full GitHub Actions runner logs for every run of affected workflows — these are available for 90 days by default and contain stdout/stderr of every step, including any curl/wget exfiltration commands executed by the malicious actions-cool payload. (2) GitHub Actions environment variable exposure events: in runner logs, search for lines matching 'echo ::add-mask::' (legitimate masking) vs. unmasked variable prints or 'env' command output that would reveal GITHUB_TOKEN, AWS credentials, or npm tokens to the attacker's step. (3) DNS query logs from your corporate resolver or cloud DNS (Route 53 Resolver Query Logs, Cloudflare Gateway) for queries to attacker-controlled exfiltration domains from runner IP ranges during the exposure window — this is the primary network indicator of successful exfiltration. (4) Cloud provider IAM access logs (AWS CloudTrail, GCP Audit Logs, Azure Activity Log) filtered on the service account identity used by CI/CD runners, looking for API calls to sensitive services (S3 GetObject, secretsmanager:GetSecretValue, iam:CreateAccessKey) timestamped concurrently with suspect workflow runs. (5) npm registry publish logs for packages under your org scope — unexpected publishes during the exposure window indicate the attacker leveraged an npm publish token obtained

via the TeamPCP/Mini Shai-Hulud payload.

Step 3: Eradication — Replace all mutable tag references (e.g., @v3) with pinned commit SHAs verified against the known-good state of each action's repository. Rotate all secrets, tokens, API keys, and credentials that were accessible in any workflow environment where these actions executed. Remove and regenerate GitHub Actions secrets, cloud provider access keys, npm publish tokens, and any other credentials scoped to affected runners. Audit and revoke any access tokens that show anomalous usage patterns.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST IA-5 (Authenticator Management), NIST CM-2 (Baseline Configuration), CIS 5.2 (Use Unique Passwords), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: SHA pinning procedure for a 2-person team: (1) Resolve the known-good SHA for each action: git ls-remote https://github.com/actions-cool/issues-helper.git HEAD — use only SHAs documented in the vendor's official advisory or pinned to a pre-compromise tag. (2) Replace all instances of 'uses: actions-cool/issues-helper@v3' with 'uses: actions-cool/issues-helper@SHA256HASH' in every .yml file. Use sed -i 's|actions-cool/issues-helper@v[0-9]*|actions-cool/issues-helper@VERIFIED_SHA|g' .github/workflows/*.yml for bulk replacement. (3) Credential rotation sequence — GitHub secrets: Settings > Secrets > delete and recreate each secret; AWS: aws iam create-access-key then aws iam delete-access-key for old key; npm: npm token revoke TOKEN_ID then npm token create --read-only or --automation as appropriate. (4) Use StepSecurity's free Action pinning tool (app.stepsecurity.io/securerepo) to automate SHA pinning PRs across repos without enterprise tooling budget.

Evidence: Preserve before rotating credentials: (1) Screenshot or export of all GitHub Actions secrets names (not values — GitHub never exposes secret values in UI) from Settings > Secrets > Actions for each affected repo — this documents the blast radius of what was accessible to the malicious action. (2) AWS IAM credential report (aws iam generate-credential-report && aws iam get-credential-report) run before key rotation to establish a baseline of key age, last-used timestamps, and access patterns for the compromised service account keys. (3) List of all npm tokens associated with your org's publish scope before revocation: npm token list — capture token IDs and creation dates as evidence of the pre-incident authentication state. (4) Git log of the malicious commit the tags were redirected to, captured via git log --format='%H %an %ae %ai %s' on the actions-cool repos before any remediation by the vendor — this establishes the attacker-controlled code version that executed in your pipeline.

Step 4: Recovery — After rotating credentials, re-run affected workflows under controlled conditions and confirm expected behavior. Validate that all workflow files now reference pinned SHAs. Monitor CI/CD pipeline logs and cloud IAM/audit logs for 30 days post-remediation for anomalous access patterns using the rotated credentials. Confirm no unauthorized packages were published to registries under your organization's scope during the exposure window.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-7 (Software, Firmware, and Information Integrity), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-11 (Audit Record Retention), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: For controlled workflow re-execution: trigger affected workflows manually in a non-production fork first using 'workflow_dispatch' with the pinned SHA in place, and verify the actions-cool steps complete without outbound network calls to non-GitHub domains by reviewing the raw runner log output. For npm registry validation: run npm search --scope=@YOUR-ORG and cross-reference publish timestamps against your release history — any package with a publish date during the exposure window and no corresponding internal release ticket is a suspected unauthorized publish. For 30-day monitoring without SIEM, set up a daily cron job to pull AWS CloudTrail events for the new service account keys: aws cloudtrail lookup-events --start-time $(date -d '1 day ago' +%Y-%m-%dT%H:%M:%S) --lookup-attributes AttributeKey=AccessKeyId,AttributeValue=NEW_KEY_ID >> /var/log/cicd-audit.log.

Evidence: Before declaring recovery complete: (1) Diff of all .github/workflows/*.yml files showing before/after state — mutable @v3 tags replaced with explicit commit SHAs — committed to the repo's git history as a permanent audit record. (2) GitHub Actions run summary for the first post-remediation execution of each previously-affected workflow, confirming the SHA-pinned action resolved correctly and no unexpected steps or network calls appeared. (3) npm registry publish history for your org's @antv-scope or equivalent packages for the full exposure window — retrieve via: npm view @YOUR-SCOPE/PACKAGE-NAME time --json — to confirm no unauthorized versions were published under your credentials. (4) Cloud IAM access review confirming the old compromised keys show 'N/A' or pre-rotation last-used timestamps and the new keys' first-use timestamps align with expected post-rotation pipeline runs.

Step 5: Post-Incident — This attack exploited systemic over-reliance on mutable Git tags in CI/CD pipelines. Implement a policy requiring all third-party GitHub Actions to be pinned by commit SHA (enforced via tooling such as OpenSSF Scorecards, StepSecurity Harden-Runner, or Dependabot for Actions). Add workflow scanning to your CI/CD pipeline to flag mutable tag references before merge. Review the broader software supply chain: audit npm dependencies for @antv scope packages per the Mini Shai-Hulud advisory. Map this incident to NIST SP 800-161 (supply chain risk management) and consider adding third-party action reviews to your software acquisition process.

NIST Phase: Post Incident

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

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SI-2 (Flaw Remediation), NIST SA-12 (Supply Chain Protection), NIST RA-3 (Risk Assessment), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: For teams without enterprise tooling: (1) Add a free GitHub Actions workflow using zizmor (open-source GitHub Actions static analysis) to scan all workflow files on every PR for mutable tag references: run zizmor .github/workflows/ and fail the check if any 'uses:' line references a tag rather than a full 40-character SHA. (2) Use OpenSSF Scorecard's free GitHub Action (github.com/ossf/scorecard-action) to automatically score all third-party actions you depend on and surface those lacking SHA pinning or with low supply chain scores. (3) For @antv npm package audit tied to Mini Shai-Hulud/TeamPCP: run npm audit --json | jq '.vulnerabilities | keys[]' against all repos and cross-reference package names against the public Mini Shai-Hulud IOC list; also run npm ls --all | grep @antv to enumerate the full transitive dependency tree for affected scope packages. (4) Add an osquery scheduled query to runner hosts: SELECT name, version, install_time FROM npm_packages WHERE name LIKE '@antv/%' to continuously inventory @antv-scoped packages on self-hosted runners.

Evidence: For the lessons-learned record and updated threat model: (1) Final inventory of all repositories, workflow files, and run timestamps affected by this campaign — this constitutes the documented scope of the incident per NIST IR-5 (Incident Monitoring) requirements. (2) Timeline reconstruction artifact: a chronological log correlating actions-cool tag redirect timestamps (from public disclosure and git history), your earliest affected workflow run, and credential rotation completion — this establishes dwell time for the attacker's access to your pipeline secrets. (3) The OpenSSF Scorecard output for all third-party GitHub Actions currently referenced across your org, run post-remediation, as a baseline for ongoing supply chain risk tracking. (4) npm dependency tree snapshot (npm ls --all --json > dep-tree-post-incident.json) for all affected repos, preserving the state of @antv-scoped transitive dependencies at time of discovery for comparison against the Mini Shai-Hulud advisory's IOC list.

Detection Guidance

Primary detection vectors: (1) Workflow file audit, grep all .github/workflows/ files for 'actions-cool/issues-helper' or 'actions-cool/maintain-one-comment' not followed by a full commit SHA (40-character hex string). Any @tag reference is suspect. (2) Runner network logs, if self-hosted runners are in use, review outbound connections from runner hosts during workflow execution windows for connections to unrecognized domains. GitHub-hosted runner logs do not expose full network telemetry, so focus on secrets access patterns instead. (3) GitHub audit

log, query for workflow_run events associated with the affected actions, cross-referenced with secrets access events (secret_scanning.alert_created, org.secret_scanning_push_protection_bypass). (4) Cloud provider IAM logs, look for API calls using CI/CD service account credentials at unusual times, from unusual source IPs, or accessing resources outside normal pipeline scope. (5) Secrets manager access logs, flag any reads of CI/CD-scoped secrets by non-human identities outside of expected pipeline execution windows. Behavioral indicator: a workflow step that reads environment variables or secrets and initiates an outbound HTTP request to a domain not in your approved CI/CD destination list. For confirmed attacker-controlled domains and IOCs, consult current CISA advisories or threat intelligence reports.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`See Wiz Mini Shai-Hulud advisory for confirmed exfiltration domains`	Exfiltration infrastructure shared between the actions-cool tag hijacking campaign and the TeamPCP Mini Shai-Hulud npm operation — specific domains confirmed in Wiz research at https://www.wiz.io/blog/mini-shai-hulud-teampcp-hits-antv-supply-chain	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1078.004** — Cloud Accounts
- **T1041** — Exfiltration Over C2 Channel
- **T1552.001** — Credentials In Files
- **T1059** — Command and Scripting Interpreter
- **T1059.007** — JavaScript
- **T1553** — Subvert Trust Controls
- **T1567** — Exfiltration Over Web Service
- **T1195.001** — Compromise Software Dependencies and Development Tools

NIST-800-53R5

- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-4** — System Monitoring
- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **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

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
T1078.004	Cloud Accounts	Defense-Evasion
T1041	Exfiltration Over C2 Channel	Exfiltration
T1552.001	Credentials In Files	Credential-Access
T1059	Command and Scripting Interpreter	Execution
T1059.007	JavaScript	Execution
T1553	Subvert Trust Controls	Defense-Evasion
T1567	Exfiltration Over Web Service	Exfiltration
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/05/github-actions-supply-chain-attac...	T3
The Worm That Keeps on Digging: TeamPCP Hits @antv in Latest ...	https://www.wiz.io/blog/mini-shai-hulud-teampcp-hits-antv-supply-chain	T3
GitHub - actions-cool/issues-helper	https://github.com/actions-cool/issues-helper	T3
Towards a secure by default GitHub Actions #179107	https://github.com/orgs/community/discussions/179107	T3
GitHub Actions Security - StepSecurity	https://www.stepsecurity.io/github-actions-and-stepsecurity	T3

Generated 2026-05-19 13:50 UTC by TJS Security Command Center