INTELLIGENCE BRIEFING
Security Command Center

TLP:CLEAR
2026-03-31 18:23 UTC

TeamPCP Supply Chain Campaign Compromises Trivy GitHub Action, Exposing Cisco Source Code and AWS Credentials

THREAT CAMPAIGN | CRITICAL | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0131
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	Cisco Unified Intelligence Center, Cisco AI Assistants, Cisco AI Defense, Trivy vulnerability scanner (GitHub Action), LiteLLM (PyPI package), Checkmarx KICS, GitHub Actions CI/CD pipelines, AWS cloud accounts
Published	2026-03-31T13:53:04
Discovery Source	Rss

Executive Summary

The TeamPCP threat group compromised the Trivy GitHub Action, a widely used open-source vulnerability scanner, and used it as a vector to steal credentials from CI/CD pipelines at Cisco and other organizations. Cisco confirmed the theft of source code from over 300 internal repositories, including proprietary AI products and third-party customer code from banks, government agencies, and BPOs; exfiltrated AWS keys were subsequently used against Cisco cloud infrastructure. The incident is not fully contained, and any organization running Trivy, LiteLLM, or Checkmarx KICS in CI/CD pipelines should treat credentials exposed through those pipelines as compromised.

Technical Analysis

TeamPCP executed a supply chain attack by injecting malicious code into the aquasecurity/trivy-action GitHub Action, a broadly trusted open-source component invoked by CI/CD pipelines to perform container and filesystem vulnerability scanning. The compromised Action harvested CI/CD secrets, environment variables, and cloud credentials at pipeline execution time, exploiting the implicit trust workflows grant to security tooling. The same campaign vector confirmed compromise of LiteLLM (PyPI package for LLM API abstraction) and Checkmarx KICS. Using stolen credentials, actors gained access to Cisco's internal development environment, cloned 300+ repositories containing source code for Cisco Unified Intelligence Center, AI Assistants, and AI Defense, along with unreleased software and third-party customer code. Exfiltrated AWS access keys were weaponized against a subset of Cisco cloud infrastructure. No CVE has been assigned; the attack chain maps to CWE-1104 (use of unmaintained third-party components), CWE-522 (insufficiently protected credentials), CWE-506 (embedded malicious code), and CWE-798 (hardcoded credentials). MITRE ATT&CK techniques include T1195.001 (supply chain compromise: compromise software dependencies), T1552.001 (credentials in files), T1552.004 (private keys), T1552.005 (cloud instance metadata API), T1528 (steal application access token), T1530 (data from cloud storage), T1078 (valid accounts), T1059 (command and scripting interpreter), T1213 (data from information repositories), T1588.001 (malware acquisition), T1586.002 (email accounts), and T1650 (acquire access). The incident is ongoing as of reporting; no vendor-confirmed patch version or remediation timestamp has been published for the compromised Action.

Action Checklist

1. Containment, Pin all GitHub Actions workflow references to a verified, immutable commit SHA rather than a mutable tag (e.g., replace uses: aquasecurity/trivy-action@v0.x.x with the specific commit SHA confirmed clean by your security team). Temporarily disable or isolate any pipeline that invoked trivy-action, litellm, or checkmarx/kics until credential rotation is complete. Prioritize this step before resuming any new pipeline builds that may consume compromised credentials.
2. Detection, Audit GitHub Actions workflow logs for any pipeline that referenced aquasecurity/trivy-action, litellm (PyPI), or checkmarx/kics during the exposure window. Review CI/CD secret access logs for unexpected reads or exports of AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, and any GITHUB_TOKEN usage. In AWS CloudTrail, query for API calls (GetCallerIdentity, ListBuckets, AssumeRole, DescribeInstances) originating from unexpected principal ARNs or source IPs. In GitHub audit logs, filter for unexpected workflow_run events or secret access patterns. Look for outbound network connections from pipeline runners to unfamiliar IPs or domains during Action execution.
3. Eradication, Rotate all secrets, tokens, and cloud credentials (AWS IAM keys, GitHub tokens, service account keys) that were accessible to any pipeline invoking the affected components. Revoke and reissue, do not simply regenerate. Remove the compromised Action versions from all workflow files; replace with SHA-pinned references only after confirming the upstream source. Remove any LiteLLM or KICS versions installed during the exposure window from build environments. Audit IAM roles associated with affected pipelines for unauthorized policy attachments or access grants and revoke anomalous permissions.
4. Recovery, Re-run security scans of affected repositories using a verified, isolated scanning environment to check for persistence (backdoors, modified build scripts, injected dependencies). Validate that rotated credentials are not cached or logged in any artifact store, container image layer, or log aggregator. Monitor AWS CloudTrail and GitHub audit logs for continued use of compromised credentials after rotation. Verify that no cloned repositories or exfiltrated code has been used to establish persistent access to downstream systems. Confirm third-party customers whose code was stored in Cisco repositories have been notified per contractual and regulatory obligations.
5. Post-Incident, Implement mandatory SHA pinning for all third-party GitHub Actions; reject mutable tag references via policy enforcement (e.g., GitHub Actions required workflow or OPA-based CI gate). Establish a software supply chain inventory tracking all GitHub Actions, PyPI packages, and third-party CI tools with version and hash records. Add secret scanning at the pipeline level to detect and block credential exposure before build artifacts are stored. Review SSDF (NIST SP 800-218) practices for supply chain risk, specifically PO.1 (define security requirements for software supply chain) and PS.1 (protect software supply chain processes). Conduct tabletop exercise covering supply chain compromise scenarios for CI/CD tooling, specifically adversary use of trusted security tools as the attack vector.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate immediately to executive leadership, legal counsel, and external IR retainer if AWS CloudTrail confirms any API activity from the compromised keys post-issuance (indicating active adversary access to cloud infrastructure), if any of the 300+ exfiltrated repositories are confirmed to contain PII, PHI, PCI data, or government-classified material triggering mandatory breach notification under GDPR, CCPA, HIPAA, or federal contractor requirements, or if the IR team lacks the capacity to rotate credentials across all affected pipelines and audit all IAM roles within a 4-hour window.
Recovery Notes	After credential rotation, maintain 30-day elevated monitoring on AWS CloudTrail for all API calls using the new IAM keys issued to pipeline service accounts, specifically alerting on any \`AssumeRole\` chains that reach sensitive S3 buckets or internal services — TeamPCP is known to establish secondary persistence using initially stolen credentials before primary keys are rotated. Validate repository integrity for all 300+ affected repos by comparing current HEAD commit hashes against pre-exposure baseline hashes stored in your CI inventory; any delta in build configuration files (\`Makefile\`, \`Dockerfile\`, \`requirements.txt\`, \`package.json\`, \`.github/workflows/\`) requires manual review before the repo is returned to production build pipelines. Maintain GitHub audit log and CloudTrail log retention at minimum 12 months for this incident to support any downstream regulatory investigations or customer breach notifications triggered by the exfiltration of third-party source code.
Forensic Artifacts	GitHub Actions runner logs for all workflow runs referencing aquasecurity/trivy-action, litellm, or checkmarx/kics during the exposure window — specifically any step output containing outbound \`curl\`/\`wget\` calls to non-GitHub IPs, unmasked secret values, or base64-encoded data blobs consistent with credential exfiltration payloads sent to TeamPCP C2 infrastructure | AWS CloudTrail event history filtered for \`GetCallerIdentity\`, \`AssumeRole\`, \`ListBuckets\`, and \`DescribeInstances\` API calls originating from source IPs outside GitHub Actions published IP ranges (api.github.com/meta), with principal ARNs mapped to the specific IAM roles used by affected CI/CD pipelines | Git commit history diffs for \`.github/workflows/\` YAML files and dependency manifests (\`requirements.txt\`, \`package.json\`, \`go.sum\`) across all 300+ affected repositories, comparing the state at exposure window start to current HEAD to identify any TeamPCP-injected dependency substitutions or build script modifications | Container image layer inspection artifacts from ECR or GitHub Container Registry for all images built during the exposure window — \`docker history --no-trunc\` output and environment variable dumps (\`docker inspect\`) to detect AWS credentials or GitHub tokens cached in image layers by the compromised pipeline steps | GitHub organization audit log exports (JSON) filtered for \`secret.access\`, \`workflow_run\`, and \`repo.download\` events during the exposure window, specifically identifying any service account or bot token that accessed secrets stores or triggered unexpected repository clone/download operations consistent with the bulk exfiltration of 300+ repository contents

Per-Action IR Details

Containment — Pin all GitHub Actions workflow references to a verified, immutable commit SHA rather than a mutable tag (e.g., replace uses: aquasecurity/trivy-action@v0.x.x with the specific commit SHA confirmed clean by your security team). Temporarily disable or isolate any pipeline that invoked trivy-action, litellm, or checkmarx/kics until credential rotation is complete. Do this before running any new builds.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: isolate affected components to prevent further credential harvesting from CI/CD pipeline runners executing the compromised aquasecurity/trivy-action, litellm, or checkmarx/kics steps

Controls: NIST IR-4 (Incident Handling), NIST CM-3 (Configuration Change Control), NIST CM-7 (Least Functionality), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 2.3 (Address Unauthorized Software)

Compensating: Run `grep -r 'aquasecurity/trivy-action\|litellm\|checkmarx/kics' .github/workflows/` across all repos to enumerate affected pipeline files. Use the GitHub CLI (`gh workflow list --repo /` and `gh run list`) to identify recent executions. Disable pipelines immediately via `gh workflow disable ` or by renaming the `.github/workflows/` YAML file to `.disabled` and committing the change. For SHA verification, use `git log --oneline` against the upstream aquasecurity/trivy-action repo to confirm the last known-good commit hash before the compromise window.

Evidence: Before disabling any pipeline, archive the full GitHub Actions workflow YAML files from `.github/workflows/` to preserve the exact `uses:` tag reference that was active during the exposure window — this establishes which version of trivy-action, litellm, or KICS was invoked. Capture GitHub Actions runner logs (downloadable via `gh run view --log`) for all workflow runs referencing the affected Actions; look specifically for steps that exported environment variables containing `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `GITHUB_TOKEN`, or any secrets injected via `env:` blocks. Preserve these logs before disabling workflows, as GitHub purges runner logs after 90 days by default.

Detection — Audit GitHub Actions workflow logs for any pipeline that referenced aquasecurity/trivy-action, litellm (PyPI), or checkmarx/kics during the exposure window. Review CI/CD secret access logs for unexpected reads or exports of AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, and any GITHUB_TOKEN usage. In AWS CloudTrail, query for API calls (GetCallerIdentity, ListBuckets, AssumeRole, DescribeInstances) originating from unexpected principal ARNs or source IPs. In GitHub audit logs, filter for unexpected workflow_run events or secret access patterns. Look for outbound network connections from pipeline runners to unfamiliar IPs or domains during Action execution.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: correlate GitHub Actions runner logs, AWS CloudTrail, and GitHub audit logs to establish the exposure window for TeamPCP credential harvesting via the compromised trivy-action supply chain vector

Controls: NIST IR-5 (Incident Monitoring), NIST AU-2 (Event Logging), 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), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: For AWS CloudTrail without a SIEM, use the AWS CLI: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=GetCallerIdentity --start-time --end-time ` and repeat for `AssumeRole`, `ListBuckets`, `DescribeInstances`. Pipe output through `jq '.Events[] | {time: .EventTime, user: .Username, sourceIP: .CloudTrailEvent | fromjson | .sourceIPAddress}'` to surface anomalous source IPs. For GitHub audit logs, use `gh api /orgs//audit-log --paginate --field phrase='action:workflows.run'` filtered to the exposure window. For runner network activity (GitHub-hosted runners), query the GitHub-provided runner IP ranges and flag any outbound connections in workflow logs to IPs outside documented GitHub Actions IP blocks (published at `api.github.com/meta`). Use `jq` or Python to diff actual source IPs against that allowlist.

Evidence: Capture AWS CloudTrail event history for the full exposure window, specifically filtering on `userAgent` strings associated with GitHub Actions runners (e.g., `aws-sdk-go`, `Boto3`) combined with unexpected `sourceIPAddress` values outside GitHub's published IP ranges. In GitHub audit log exports (JSON via REST API), look for `workflow_run` events where the triggering repository or actor does not match expected CI service accounts. Within GitHub Actions runner logs, search for lines containing `::add-mask::` failures or unmasked secret values, and any `curl`, `wget`, or `python` invocations within the trivy-action, litellm install, or KICS execution steps that contacted non-GitHub, non-AWS endpoints — these would indicate the exfiltration callback from the compromised Action. Preserve all raw runner log archives before the 90-day GitHub retention window expires.

Eradication — Rotate all secrets, tokens, and cloud credentials (AWS IAM keys, GitHub tokens, service account keys) that were accessible to any pipeline invoking the affected components. Revoke and reissue, do

not simply regenerate. Remove the compromised Action versions from all workflow files; replace with SHA-pinned references only after confirming the upstream source. Remove any LiteLLM or KICS versions installed during the exposure window from build environments. Audit IAM roles associated with affected pipelines for unauthorized policy attachments or access grants and revoke anomalous permissions.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: remove TeamPCP-implanted credential access by revoking all AWS IAM keys and GitHub tokens exposed to compromised trivy-action, litellm, and KICS pipeline steps, and eliminate the supply chain foothold by replacing mutable Action tag references with verified SHA pins

Controls: NIST IR-4 (Incident Handling), NIST AC-2 (Account Management), NIST IA-5 (Authenticator Management), NIST CM-3 (Configuration Change Control), NIST SI-2 (Flaw Remediation), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 6.2 (Establish an Access Revoking Process), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: For AWS IAM key rotation at scale without enterprise tooling: `aws iam list-access-keys --user-name ` to enumerate all keys per IAM user, then `aws iam update-access-key --access-key-id --status Inactive` immediately, followed by `aws iam delete-access-key` after confirming new key is functional. For IAM role policy auditing: `aws iam get-role --role-name ` and `aws iam list-attached-role-policies --role-name ` — compare output against your last known-good IAM policy baseline (stored in version control if following CIS 4.6). For GitHub token revocation, use `gh auth token` to identify active tokens and revoke via GitHub Settings > Developer Settings > Personal Access Tokens. For LiteLLM and KICS removal from self-hosted runners, run `pip show litellm` and `pip uninstall litellm -y` on each runner host; for container-based runners, rebuild images from a pre-exposure base image hash.

Evidence: Before rotating AWS IAM keys, run `aws iam get-access-key-last-used --access-key-id ` for every key accessible to affected pipelines to establish last-use timestamps — this is critical forensic evidence for determining whether exfiltrated keys were actively weaponized against Cisco cloud infrastructure post-theft. Capture `aws iam list-attached-role-policies` and `aws iam list-role-policies` output for all roles used by affected pipelines before any modifications, as TeamPCP may have attached additional policies or created new IAM entities using the stolen keys. In GitHub, export the full secret access audit trail before rotation: `gh api /repos///actions/secrets` to inventory what secrets were defined and therefore accessible to the compromised Action steps.

Recovery — Re-run security scans of affected repositories using a verified, isolated scanning environment to check for persistence (backdoors, modified build scripts, injected dependencies). Validate that rotated credentials are not cached or logged in any artifact store, container image layer, or log aggregator. Monitor AWS CloudTrail and GitHub audit logs for continued use of compromised credentials after rotation. Verify that no cloned repositories or exfiltrated code has been used to establish persistent access to downstream systems. Confirm third-party customers whose code was stored in Cisco repositories have been notified per contractual and regulatory obligations.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery: restore verified pipeline integrity by scanning all 300+ affected repositories for TeamPCP-injected persistence, validating credential rotation completeness across artifact stores and container layers, and maintaining elevated CloudTrail and GitHub audit log monitoring for residual use of compromised AWS keys or tokens

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), NIST CP-10 (System Recovery and Reconstitution), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: For repository backdoor scanning without commercial tooling, use `git log --all --full-history -- '*.yml' '*.yaml' '*.sh' '*.py' 'requirements*.txt' 'package*.json'` to surface all changes to build configuration and dependency files during the exposure window, then diff against pre-exposure commits using `git diff HEAD`. Use `truffleHog3` (free, open-source) with `trufflehog git file:// --since-commit ` to detect any credentials that may have been committed to source. For container image layer inspection, use `docker history --no-trunc ` and `dive` (open-source) to inspect each layer for cached environment variables containing key patterns. For CloudTrail post-rotation monitoring, set a

CloudWatch metric filter on `AssumeRole` and `GetCallerIdentity` events from the now-revoked key ARNs and alert on any match — a hit after rotation confirms active adversary use.

Evidence: Before re-scanning repositories, preserve git commit history snapshots (full `git bundle create .bundle --all`) for all 300+ affected repositories to maintain forensic integrity of the state at time of discovery. Inspect container image registries (ECR, Docker Hub, GitHub Container Registry) used by affected pipelines for image layers built during the exposure window — use `docker inspect` to check image creation timestamps and `docker run --entrypoint env ` against build-time images to detect cached AWS credentials baked into layers. For artifact stores (S3 buckets, GitHub Packages), run `aws s3api list-object-versions` filtered to the exposure window and retrieve any build artifact that could contain embedded credentials from the compromised pipeline steps.

Post-Incident — Implement mandatory SHA pinning for all third-party GitHub Actions; reject mutable tag references via policy enforcement (e.g., GitHub Actions required workflow or OPA-based CI gate). Establish a software supply chain inventory tracking all GitHub Actions, PyPI packages, and third-party CI tools with version and hash records. Add secret scanning at the pipeline level to detect and block credential exposure before build artifacts are stored. Review SSDF (NIST SP 800-218) practices for supply chain risk, specifically PO.1 (define security requirements for software supply chain) and PS.1 (protect software supply chain processes). Conduct tabletop exercise covering supply chain compromise scenarios for CI/CD tooling, specifically adversary use of trusted security tools as the attack vector.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: document lessons learned from the TeamPCP trivy-action supply chain compromise, implement structural controls to prevent re-compromise via mutable CI/CD Action references, and update the incident response plan to include supply chain tooling as a high-priority attack vector scenario

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST IR-2 (Incident Response Training), NIST IR-3 (Incident Response Testing), NIST SI-7 (Software, Firmware, and Information Integrity), NIST SA-12 (Supply Chain Protection), NIST CM-3 (Configuration Change Control), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: For SHA pinning enforcement without enterprise policy tooling, deploy the open-source `step-security/harden-runner` GitHub Action and `step-security/secure-repo` scanner (both free) to audit all org repos for mutable tag references and auto-generate PR fixes with SHA pins. For a supply chain inventory on zero budget, use a Python script with `PyGithub` to enumerate all `uses:` lines across `.github/workflows/` in every org repo and output a CSV of action name, current tag, and pinned SHA — run weekly via cron. For pipeline secret scanning, enable GitHub's native secret scanning (free for public repos, included with GitHub Advanced Security for private) and add `truffleHog3` as a required pre-commit hook and CI step. For the tabletop exercise, use the CISA Tabletop Exercise Package (CTEP) for supply chain scenarios, adapting the TeamPCP scenario: attacker compromises a trusted security scanner (trivy-action) to harvest CI secrets and exfiltrate source code from 300+ repositories.

Evidence: For the lessons-learned record, compile the full timeline of: (1) first affected pipeline execution referencing the compromised trivy-action tag, (2) earliest evidence of credential harvesting in CloudTrail, (3) first confirmed exfiltration event, and (4) detection time — this gap analysis is required input for the NIST 800-61r3 §4 post-incident review and drives measurable improvement targets. Archive all GitHub Actions workflow YAML files, CloudTrail exports, GitHub audit log exports, and IAM policy snapshots in an immutable evidence store (S3 with Object Lock, or equivalent) with SHA-256 hashes recorded for chain-of-custody before any remediated versions are committed. Document which of the 300+ affected repositories contained third-party customer code from regulated sectors (banks, government agencies) as this drives the regulatory notification timeline under applicable breach notification requirements.

Detection Guidance

Primary detection pivot is pipeline execution logs for the affected components. Query GitHub Actions workflow run logs for any job step invoking aquasecurity/trivy-action (any tag), litellm via PyPI install, or checkmarx/kics. For AWS credential abuse, query CloudTrail with filter: eventSource=sts.amazonaws.com AND eventName=GetCallerIdentity paired with unexpected sourceIPAddress values, or any AssumeRole event referencing a pipeline-associated role from an IP outside your runner IP range. For credential harvesting behavior, look for environment variable enumeration or file read operations targeting .env, ~/.aws/credentials, or /proc/*/environ during Action execution in runner debug logs. Behavioral indicator: a security-tooling Action step that initiates outbound HTTP POST or DNS queries to non-vendor infrastructure. In SIEM, correlate: (1) pipeline execution timestamp, (2) AWS API calls within the same time window, (3) GitHub secret access events. IOC confidence is low for specific IPs and domains given no vendor-published IOC list at time of reporting; treat all credential material accessible during affected pipeline runs as compromised regardless of IOC match.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://github.com/aquasecurity/trivy-action`	Compromised GitHub Action; any workflow reference using a mutable tag during the exposure window should be treated as tainted. Pin to a verified commit SHA.	HIGH
URL	`https://pypi.org/project/litellm/`	PyPI package confirmed compromised in the same campaign. Versions installed during the exposure window should be treated as untrusted.	HIGH
URL	`https://github.com/Checkmarx/kics`	Checkmarx KICS confirmed affected by the same campaign vector. Review pipeline installations for the exposure window.	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1552.001** — Credentials In Files
- **T1650** — Acquire Access
- **T1588.001** — Malware
- **T1078** — Valid Accounts
- **T1530** — Data from Cloud Storage
- **T1552.004** — Private Keys
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1213** — Data from Information Repositories
- **T1528** — Steal Application Access Token
- **T1552.005** — Cloud Instance Metadata API

- **T1586.002** — Email Accounts
- **T1059** — Command and Scripting Interpreter

NIST-800-53R5

- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-7** — Software, Firmware, and Information Integrity
- **SA-4** — Acquisition Process
- **SA-9** — External System Services
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A06:2021** — Vulnerable and Outdated Components
- **A04:2021** — Insecure Design
- **A07:2021** — Identification and Authentication Failures

CIS-V8

- **16.4** — Establish and Manage an Inventory of Third-Party Software Components
- **5.2** — Use Unique Passwords
- **16.10** — Apply Secure Design Principles in Application Architectures
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers

HIPAA-SECURITY

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

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
- **A.5.23** — Information security for use of cloud services

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
T1650	Acquire Access	Resource-Development
T1588.001	Malware	Resource-Development
T1078	Valid Accounts	Defense-Evasion
T1530	Data from Cloud Storage	Collection
T1552.004	Private Keys	Credential-Access
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1213	Data from Information Repositories	Collection
T1528	Steal Application Access Token	Credential-Access
T1552.005	Cloud Instance Metadata API	Credential-Access
T1586.002	Email Accounts	Resource-Development
T1059	Command and Scripting Interpreter	Execution

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/cisco-source-code-st...	T3
Security Update: Suspected Supply Chain Incident	https://docs.litellm.ai/blog/security-update-march-2026	T3
Your AI Stack Just Handed Over Your Root Keys	https://www.trendmicro.com/en_us/research/26/c/your-ai-stack-just-h...	T3
From Scanner to Stealer: Inside the trivy-action Supply ...	https://www.crowdstrike.com/en-us/blog/from-scanner-to-stealer-insi...	T3
How a Poisoned Security Scanner Became the Key to ...	https://snyk.io/articles/poisoned-security-scanner-backdooring-lite...	T3