INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-01 18:41 UTC

TeamPCP Supply Chain Campaign: Security Scanners Weaponized in CI/CD Pipeline Attacks with Confirmed Victim and Cloud Enumeration

THREAT CAMPAIGN | **HIGH** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0136
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	9.5
Affected Products	Security scanning tools integrated into CI/CD pipelines; cloud environments accessed post-compromise, specific vendors and versions not identified in available source data
Published	2026-04-01T09:08:26
Discovery Source	Rss

Executive Summary

The TeamPCP group is reported to be compromising organizations by weaponizing security scanning tools integrated into CI/CD pipelines, turning a defensive control into an entry point. Post-compromise activity has been observed to include cloud environment enumeration, indicating attackers move beyond the build pipeline into broader infrastructure. Organizations running security scanners in automated pipelines face elevated risk of supply chain compromise, credential theft, and cloud resource exposure; campaign activity is ongoing.

Technical Analysis

TeamPCP is executing a supply chain attack (MITRE T1195.001, T1195.002) by compromising or staging malicious versions of security scanning tools used within CI/CD pipelines (T1554: Compromise Software Supply Chain, Development Tools). The attack chain exploits four documented weaknesses: CWE-693 (Protection Mechanism Failure, the scanner itself becomes the delivery vector), CWE-494 (Download of Code Without Integrity Checking, pipeline stages pulling tool artifacts without hash or signature verification), CWE-798 (Use of Hard-Coded Credentials, embedded secrets exposed during execution), and CWE-522 (Insufficiently Protected Credentials, runtime secrets accessible in pipeline environment variables or logs). Post-initial access, actors have conducted cloud service enumeration (T1526, T1538) using harvested credentials (T1552, T1552.001), with scripted execution (T1059) and scheduled task persistence (T1053) consistent with dwell-time extension. Staging infrastructure activity (T1608) and valid account abuse (T1078) have also been associated with the

campaign. No specific vendor tools, versions, or patch identifiers are confirmed in available source data. No CVE has been assigned. No CISA KEV listing as of the report date. Attribution is to TeamPCP by campaign naming; no nation-state nexus confirmed. Confidence in campaign staging and enumeration activity: MEDIUM-HIGH (reported by multiple sources; full victim scope unconfirmed at time of reporting).

Action Checklist

1. Step 1: Containment. Immediately audit all security scanning tools integrated into CI/CD pipelines. Identify which tools pull from external package registries, GitHub repositories, or third-party distribution channels. Temporarily isolate or pause pipeline stages that invoke unverified scanner binaries until integrity can be confirmed. Restrict outbound network access from pipeline runners to known-good endpoints only. Cross-reference against any TeamPCP IOCs published by your threat intelligence provider or CISA.
2. Step 2: Detection. Review pipeline execution logs for unexpected outbound connections, cloud API calls, or credential-access events originating from scanner tool processes. Query cloud provider audit logs (AWS CloudTrail, Azure Activity Log, GCP Cloud Audit Logs) for enumeration activity (ListBuckets, DescribeInstances, GetCallerIdentity equivalents) initiated from CI/CD runner identity contexts. Hunt for T1526 and T1538 indicators: unusual ListServices, DescribeRegions, or cloud account discovery calls from service accounts associated with pipeline execution. Check for scheduled task creation (T1053) or new cron entries on runner hosts post-scanner execution. If TeamPCP IOCs are available from your threat intelligence feed, prioritize detection of those artifacts.
3. Step 3: Eradication. Verify the cryptographic integrity (SHA-256 hash or GPG signature) of all scanner tool binaries against vendor-published checksums before re-enabling pipeline stages. Remove any hard-coded credentials (CWE-798) found in pipeline configuration files, Dockerfiles, or scanner config files; rotate all secrets immediately. Replace static credentials with short-lived, scoped tokens using your pipeline platform's secrets management (e.g., GitHub Actions OIDC, GitLab CI/CD variables with masking, HashiCorp Vault dynamic secrets). Enforce dependency pinning with lockfiles and integrity hashes for all tool downloads (addresses CWE-494).
4. Step 4: Recovery. After rotating credentials and re-validating scanner binaries, restore pipeline execution under increased logging verbosity. Verify no persistence mechanisms (scheduled tasks, cron jobs, new service accounts) were established on runner infrastructure during the compromise window. Audit cloud IAM for newly created roles, policies, or access keys that were not provisioned through your standard change process. Monitor cloud enumeration patterns for 30 days post-remediation to confirm actor eviction.
5. Step 5: Post-Incident. Conduct a formal review of the pipeline's trust model: document every external artifact pull point and assign an owner responsible for integrity verification. Implement mandatory artifact signing and verification as a pipeline gate (e.g., Sigstore/Cosign for container images and binaries). Evaluate adoption of a Software Bill of Materials (SBOM) for pipeline tooling to detect future supply chain tampering. Map control gaps to NIST SP 800-161 (C-SCRM) and CISA's Secure Software Development Framework (SSDF) for remediation prioritization.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate to CISO, legal, and cloud platform owner immediately if CloudTrail or equivalent confirms that the compromised runner identity successfully performed IAM enumeration, accessed secrets stores, or created new credentials — any of these conditions indicates the blast radius extends beyond the build pipeline into production cloud infrastructure and may trigger breach notification obligations depending on data classification of cloud resources accessed.
Recovery Notes	Before restoring any pipeline stage, confirm three things specific to this campaign: (1) all scanner binaries have passed cryptographic verification against vendor-published checksums, (2) all cloud IAM roles and access keys created during the compromise window have been identified and either removed or explicitly re-authorized through your change process, and (3) CloudTrail or equivalent shows zero cloud enumeration API calls from runner identities in the 24 hours post-credential rotation. Maintain enhanced cloud API monitoring (alerting on GetCallerIdentity, ListBuckets, DescribeInstances, ListRoles from runner identities) for a minimum of 30 days post-remediation, as TeamPCP has demonstrated persistence motivation evidenced by confirmed victim reporting. If runners are ephemeral (containerized), redeploy all runner infrastructure from a known-good base image rather than attempting in-place remediation.
Forensic Artifacts	CI/CD platform execution logs (GitHub Actions workflow run logs, GitLab CI job trace logs, Jenkins build console output) for all pipeline runs invoking security scanner tools during the 90 days preceding discovery — these will show the exact process tree of the weaponized scanner binary, including any child processes spawned for cloud credential access or outbound C2 communication characteristic of TeamPCP's initial access phase. | AWS CloudTrail management event logs filtered for eventSource in [sts.amazonaws.com, iam.amazonaws.com, s3.amazonaws.com, ec2.amazonaws.com] where userIdentity.arn matches the CI/CD runner IAM role — specifically hunting GetCallerIdentity (T1538 cloud identity discovery), ListBuckets (T1526 cloud service discovery), DescribeInstances, ListRoles, and CreateAccessKey events originating from runner egress IPs during the compromise window. | Runner host filesystem artifacts: modification timestamps on scanner binary files (`stat `), contents of `/tmp/` and runner working directories for dropped payloads, `/etc/cron.d/` and per-user crontabs for T1053 persistence, and `/var/log/auth.log` entries showing any `su`, `sudo`, or new account creation events during scanner execution windows. | Pipeline configuration file Git history for all branches — specifically `git log -p -- '**/*.yml' '**/*.yaml' Jenkinsfile Dockerfile .github/workflows/` showing any unauthorized additions of scanner tool invocations, environment variable exfiltration steps (e.g., `curl` commands posting `$AWS_ACCESS_KEY_ID`), or changes to artifact download URLs redirecting to attacker-controlled infrastructure as the mechanism of supply chain compromise. | Cloud provider IAM credential reports and access key last-used data for all service accounts and roles associated with pipeline execution — this establishes the definitive list of cloud services and resources accessed by the attacker after pivoting from the compromised scanner binary into the cloud environment, which is required for scope determination and any regulatory breach notification analysis.

Per-Action IR Details

Step 1: Containment — Immediately audit all security scanning tools integrated into CI/CD pipelines. Identify which tools pull from external package registries, GitHub repositories, or third-party distribution channels. Temporarily isolate or pause pipeline stages that invoke unverified scanner binaries until integrity can be confirmed. Restrict outbound network access from pipeline runners to known-good endpoints only.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: isolate affected components to prevent further propagation while preserving evidence; choose containment strategy based on potential damage and service availability needs.

Controls: NIST IR-4 (Incident Handling), NIST CM-7 (Least Functionality) — restrict pipeline runner capabilities to approved outbound endpoints only, NIST SC-7 (Boundary Protection) — enforce egress filtering from CI/CD runner network segments, CIS 4.4 (Implement and Manage a Firewall on Servers) — apply host-based firewall rules to runner hosts blocking non-allowlisted outbound destinations, CIS 2.3 (Address Unauthorized Software) — identify and halt execution of any scanner binary not in the approved software inventory

Compensating: For teams without enterprise network controls: immediately add iptables OUTPUT rules on each runner host to DROP all traffic except to explicitly allowlisted IPs (e.g., `iptables -A OUTPUT -d -j ACCEPT && iptables -A OUTPUT -j DROP`). Run `ss -tunap` or `netstat -tunap` on runner hosts to enumerate active outbound connections before applying the block. Use GitHub Actions environment protection rules or GitLab CI/CD protected environments to require manual approval before any pipeline stage that invokes a scanner binary. Export the current pipeline YAML definitions to version control review before pausing.

Evidence: BEFORE isolating runners, capture: (1) Full network connection state on each runner via `ss -tunap > /tmp/runner_netstat_$(hostname)_$(date +%s).txt` — TeamPCP post-compromise activity includes cloud enumeration, so active connections to AWS/Azure/GCP API endpoints (169.254.169.254, 169.254.170.2, metadata.google.internal) are high-value indicators. (2) Running process tree at time of discovery via `ps auxf > /tmp/proctree_$(hostname)_$(date +%s).txt` — capture parent-child relationships showing scanner binary spawning unexpected child processes (shells, curl, python). (3) Full pipeline execution log for the most recent 30 days from your CI platform (GitHub Actions workflow run logs, GitLab CI job logs, Jenkins build console output) before any log rotation occurs. (4) Cloud provider credential files and environment variables accessible to the runner at time of compromise: `env | grep -iE 'aws|azure|gcp|token|key|secret' > /tmp/env_snapshot.txt` (handle with care — store encrypted).

Step 2: Detection — Review pipeline execution logs for unexpected outbound connections, cloud API calls, or credential-access events originating from scanner tool processes. Query cloud provider audit logs (AWS CloudTrail, Azure Activity Log, GCP Cloud Audit Logs) for enumeration activity (ListBuckets, DescribeInstances, GetCallerIdentity equivalents) initiated from CI/CD runner identity contexts. Hunt for T1526 and T1538 indicators: unusual ListServices, DescribeRegions, or cloud account discovery calls from service accounts associated with pipeline execution. Check for scheduled task creation (T1053) or new cron entries on runner hosts post-scanner execution.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: correlate indicators across log sources to establish scope; use threat intelligence to prioritize analysis of cloud enumeration patterns consistent with TeamPCP TTPs.

Controls: NIST IR-4 (Incident Handling), NIST IR-5 (Incident Monitoring) — track and document all cloud enumeration events tied to pipeline runner identity, NIST AU-6 (Audit Record Review, Analysis, and Reporting) — systematically review CloudTrail, Azure Activity Log, and GCP Audit Logs for anomalous API calls, NIST AU-3 (Content of Audit Records) — ensure cloud audit logs capture caller identity, source IP, and API action for all enumeration events, NIST SI-4 (System Monitoring) — monitor pipeline runner processes for anomalous child process spawning and outbound network behavior, CIS 8.2 (Collect Audit Logs) — verify CI/CD platform logs and cloud provider audit logs are enabled and retained, MITRE ATT&CK T1526 (Cloud Service Discovery) — hunt for atypical ListServices, DescribeRegions calls from runner service accounts, MITRE ATT&CK T1538 (Cloud Service Dashboard) — detect console/API access to cloud management planes from runner identities, MITRE ATT&CK T1053 (Scheduled Task/Job) — detect cron and systemd timer creation on runner hosts post-scanner execution

Compensating: Without a SIEM, execute these targeted queries directly: (1) AWS CloudTrail — use AWS CLI: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=ListBuckets --start-time --output json > cloudtrail_listbuckets.json`; repeat for GetCallerIdentity, DescribeInstances, ListRoles. Filter results where `userIdentity.arn` contains your runner's IAM role name. (2) GCP — run `gcloud logging read 'protoPayload.methodName=~"list|describe|get"' --freshness=7d --format=json > gcp_audit.json` scoped to the service account used by your runner. (3) On runner hosts, check for new cron entries: `diff <(crontab -l) <(git show HEAD~30:crontab_baseline.txt)` or manually inspect `/var/spool/cron/`, `/etc/cron.d/`, `/etc/cron.daily/`, and `systemctl list-timers --all`. (4) Use osquery: `SELECT * FROM scheduled_tasks` (Windows runners) or `SELECT * FROM crontab` (Linux) to enumerate persistence mechanisms. (5) Deploy the Sigma rule for cloud enumeration from runner

identities (search SigmaHQ for T1526) against locally exported CloudTrail JSON using `sigma convert`.

Evidence: Before querying and potentially modifying log state: (1) Export immutable copies of AWS CloudTrail logs from S3 to isolated storage — specifically the last 90 days of management events for the account(s) accessible to the compromised runner role. Focus on `eventSource: iam.amazonaws.com`, `sts.amazonaws.com`, `s3.amazonaws.com`, `ec2.amazonaws.com` where `sourceIPAddress` matches the runner's egress IP. (2) Capture `/var/log/syslog` or `/var/log/messages` and `/var/log/auth.log` from runner hosts covering the full compromise window — TeamPCP activity may show scanner binary executing `aws sts get-caller-identity` or similar as a first-stage cloud recon step, which will appear as subprocess execution in auth/syslog. (3) Capture Git history of pipeline configuration files (`.github/workflows/*.yml`, `.gitlab-ci.yml`, `Jenkinsfile`) for unauthorized modifications introducing new scanner invocations or environment variable exfiltration steps. (4) On runner hosts, capture `/proc//cmdline`, `/proc//environ`, and `/proc//net/tcp` before process termination if scanner is still running.

Step 3: Eradication — Verify the cryptographic integrity (SHA-256 hash or GPG signature) of all scanner tool binaries against vendor-published checksums before re-enabling pipeline stages. Remove any hard-coded credentials (CWE-798) found in pipeline configuration files, Dockerfiles, or scanner config files; rotate all secrets immediately. Replace static credentials with short-lived, scoped tokens using your pipeline platform's secrets management (e.g., GitHub Actions OIDC, GitLab CI/CD variables with masking, HashiCorp Vault dynamic secrets). Enforce dependency pinning with lockfiles and integrity hashes for all tool downloads (addresses CWE-494).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: eliminate all components of the incident, including malicious artifacts, compromised credentials, and the vulnerability that enabled initial access; verify eradication before proceeding to recovery.

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation) — remediate the supply chain integrity gap that allowed weaponized scanner binaries to execute without verification, NIST SI-7 (Software, Firmware, and Information Integrity) — employ integrity verification tools to detect unauthorized changes to scanner binaries before re-enabling pipeline stages, NIST IA-5 (Authenticator Management) — rotate all exposed credentials immediately; enforce short-lived token model for pipeline runner authentication, NIST CM-7 (Least Functionality) — scope replacement OIDC tokens and dynamic secrets to minimum required cloud permissions for scanner execution, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — document the supply chain integrity gap as a tracked vulnerability through your remediation process, CIS 7.2 (Establish and Maintain a Remediation Process) — prioritize credential rotation and binary verification as critical-severity remediation items, CIS 5.2 (Use Unique Passwords) — ensure rotated secrets are unique and not reused across pipeline environments

Compensating: Binary integrity verification without enterprise tooling: (1) Download vendor-published SHA-256 checksums from the official release page (not from the same distribution channel that may be compromised) and verify: `sha256sum -c ` against each scanner binary on disk. For GPG-signed releases: `gpg --verify ` after importing the vendor's signing key from their official keyserver. (2) For hard-coded credential hunting in pipeline configs: run `truffleHog filesystem --directory=. --only-verified` or `gitleaks detect --source=. --report-format json --report-path=gitleaks_report.json` across the full repository including history. (3) For dependency pinning enforcement: add a pre-commit hook or CI gate using `pip-audit` (Python), `npm audit` (Node), or `trivy fs .` (multi-ecosystem) that fails the pipeline if any dependency lacks a pinned hash. (4) For GitHub Actions OIDC migration without enterprise tooling: replace `aws-actions/configure-aws-credentials` using static keys with the OIDC variant — this is free and native to GitHub Actions; configure the AWS IAM trust policy to restrict `sub` claim to your specific repository and branch.

Evidence: Before rotating credentials (to establish what was exposed): (1) Extract the full list of environment variables and secrets accessible to the runner during the compromise window from your CI platform's secret audit log (GitHub: Settings > Security > Secret scanning alerts; GitLab: Audit Events filtered by `secrets`). (2) For each exposed AWS IAM key, run `aws iam list-access-keys --user-name ` and `aws iam get-access-key-last-used --access-key-id ` to establish last-use timestamp and service — this determines blast radius for CloudTrail hunting scope. (3) Capture file modification timestamps on all scanner binary files on runner hosts: `find /opt /usr/local/bin /home/runner -name '*.sh' -o -name 'scanner*' -newer /tmp/reference_timestamp -ls` where reference_timestamp is set to the estimated compromise start. (4) Preserve a copy of all pipeline configuration files as they existed during the compromise window

using `git log --all --full-history -- '*.yml' '*.yaml' Jenkinsfile Dockerfile` before any remediation commits overwrite history.

Step 4: Recovery — After rotating credentials and re-validating scanner binaries, restore pipeline execution under increased logging verbosity. Verify no persistence mechanisms (scheduled tasks, cron jobs, new service accounts) were established on runner infrastructure during the compromise window. Audit cloud IAM for newly created roles, policies, or access keys that were not provisioned through your standard change process. Monitor cloud enumeration patterns for 30 days post-remediation to confirm actor eviction.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery: restore systems to normal operation with verified integrity; implement enhanced monitoring to confirm threat actor eviction and detect re-entry attempts during the recovery window.

Controls: NIST IR-4 (Incident Handling), NIST IR-5 (Incident Monitoring) — maintain elevated monitoring of cloud enumeration API calls and pipeline runner behavior for 30 days post-remediation, NIST AU-2 (Event Logging) — increase pipeline runner and cloud provider audit logging verbosity before restoring execution, NIST AU-12 (Audit Record Generation) — ensure all pipeline runner actions generate audit records at increased granularity during recovery window, NIST AC-2 (Account Management) — audit and remove all cloud IAM roles, policies, and access keys created outside standard change process during compromise window, NIST CM-3 (Configuration Change Control) — verify all pipeline configuration files match approved baselines before re-enabling stages, CIS 5.1 (Establish and Maintain an Inventory of Accounts) — reconcile all cloud service accounts and runner identities against pre-compromise baseline, CIS 5.3 (Disable Dormant Accounts) — disable any service accounts created by the attacker that are not actively needed

Compensating: For persistence verification without EDR on runner hosts: (1) Cron/scheduled task audit: `for user in $(cut -f1 -d: /etc/passwd); do crontab -u $user -l 2>/dev/null && echo "user: $user"; done` — compare output against a pre-compromise baseline captured from version control. (2) New service account detection in cloud without CSPM: AWS — `aws iam list-users --query 'Users[?CreateDate>=``]'` and `aws iam list-roles --query 'Roles[?CreateDate>=``]'`; GCP — `gcloud iam service-accounts list --filter='createTime>='`. (3) For 30-day cloud enumeration monitoring without a SIEM: set up a free CloudWatch metric filter on CloudTrail for `eventName = ListBuckets OR DescribeInstances OR GetCallerIdentity` scoped to your runner role ARN, with an SNS email alert on any match — this costs under $1/month. (4) Deploy Falco (free, open-source) on Linux runner hosts to generate real-time alerts for unexpected process spawning from pipeline runner user context during the recovery window.

Evidence: Before restoring pipeline execution: (1) Run a complete cloud IAM credential report — AWS: `aws iam generate-credential-report && aws iam get-credential-report --query 'Content' --output text | base64 -d > iam_credential_report.csv` — review for access keys with last-used dates falling within the compromise window that do not correspond to legitimate pipeline activity. (2) Pull full CloudTrail history for IAM mutation events during the compromise window: `eventName IN [CreateUser, CreateRole, CreateAccessKey, AttachRolePolicy, PutRolePolicy, CreateServiceLinkedRole]` — these are the specific API calls TeamPCP would use to establish persistence in the cloud environment after initial enumeration. (3) On runner hosts, capture `/etc/passwd` and `/etc/shadow` (hash only) diffs against pre-compromise baseline to detect new local accounts created by attacker-controlled scanner code. (4) Export current state of all pipeline YAML files and compare against last known-good commit: `git diff HEAD -- '**/*.yml' '**/*.yaml'` to identify any lingering backdoor steps in pipeline definitions.

Step 5: Post-Incident — Conduct a formal review of the pipeline's trust model: document every external artifact pull point and assign an owner responsible for integrity verification. Implement mandatory artifact signing and verification as a pipeline gate (e.g., Sigstore/Cosign for container images and binaries). Evaluate adoption of a Software Bill of Materials (SBOM) for pipeline tooling to detect future supply chain tampering. Map control gaps to NIST SP 800-161 (C-SCRM) and CISA's Secure Software Development Framework (SSDF) for remediation prioritization.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: conduct lessons-learned review, update IR plan and detection capabilities, share threat intelligence, and implement structural controls to prevent recurrence of supply chain compromise through CI/CD pipeline tooling.

Controls: NIST IR-4 (Incident Handling) — update IR plan to include CI/CD supply chain compromise as a named scenario with specific playbook steps, NIST IR-8 (Incident Response Plan) — revise IR plan to address pipeline trust model gaps exposed by TeamPCP campaign, NIST SI-2 (Flaw Remediation) — formalize integrity verification of pipeline tool dependencies as a recurring remediation process item, NIST SI-7 (Software, Firmware, and Information Integrity) — implement Sigstore/Cosign artifact signing as a permanent integrity control for all pipeline-consumed binaries, NIST SA-12 (Supply Chain Protection) — document all external artifact pull points as supply chain risk items with assigned owners, NIST RA-3 (Risk Assessment) — re-assess risk posture for CI/CD pipeline components in light of confirmed TeamPCP targeting of security scanner tools, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — incorporate supply chain integrity verification into the vulnerability management process, CIS 2.1 (Establish and Maintain a Software Inventory) — extend software inventory to cover all pipeline tooling including security scanner binaries and their upstream sources, CIS 2.2 (Ensure Authorized Software is Currently Supported) — verify all scanner tools integrated into pipelines are actively maintained and sourced from verified distribution channels

Compensating: For teams without enterprise GRC tooling: (1) Trust model documentation — create a markdown file in the repository root (`PIPELINE_TRUST_MODEL.md`) listing every `uses:`, `image:`, `curl | sh`, or package install step with its source URL, expected hash, verification method, and named owner. Commit this as a required review artifact in your change process. (2) Sigstore/Cosign is free and open-source — implement `cosign verify` as a pipeline step before any scanner binary execution: `cosign verify --certificate-identity= --certificate-oidc-issuer= `. For binaries not yet signed by vendors, generate and store your own signatures after verifying the initial download. (3) SBOM generation — use `syft` (free, Anchore) to generate CycloneDX or SPDX SBOMs for all runner container images: `syft -o cyclonedx-json > sbom_$(date +%Y%m%d).json`; store in a dedicated artifact repository and diff on each pipeline run to detect additions. (4) For NIST SP 800-161 C-SCRM gap mapping without a GRC platform, use the NIST C-SCRM Quick Start Guide (free PDF) and map findings from this incident to the relevant practices using a spreadsheet with columns: control ID, gap identified, owner, remediation target date.

Evidence: Artifacts to preserve for lessons-learned and potential regulatory/legal purposes: (1) Complete timeline reconstruction from all log sources (CloudTrail, CI platform logs, runner host logs) covering initial scanner compromise through actor eviction — this is required for any breach notification analysis and serves as the primary input to the lessons-learned review. (2) All versions of pipeline configuration files that existed during the compromise window, preserved as immutable artifacts in isolated storage — these document the attack surface that TeamPCP exploited and support root cause analysis. (3) Final IAM credential report and access key audit output showing all credentials rotated, their creation dates, last-use dates, and disposition — this is required evidence for compliance attestation. (4) Network capture (if available) from runner egress during the compromise window showing cloud API enumeration traffic — even partial pcap preserving DNS queries and TCP flow metadata (no payload required) establishes the scope of cloud enumeration performed by TeamPCP post-compromise.

Detection Guidance

Primary detection surface is CI/CD runner execution logs and cloud provider audit logs. Key behavioral indicators: (1) Scanner tool processes initiating outbound connections to IPs or domains outside the organization's known artifact registries, flag any DNS lookups or TCP connections from scanner process PIDs to external hosts not in an approved allowlist. (2) Cloud API enumeration calls (T1526, T1538) originating from service accounts or IAM roles bound to pipeline runner identities, particularly ListBuckets, DescribeInstances, DescribeRegions, GetAccountAuthorizationDetails, or equivalent cross-service discovery calls. (3) Credential access events (T1552.001) such as reads of environment variable stores, /proc/*/environ on Linux runners, or pipeline secrets APIs outside of expected job steps. (4) New scheduled tasks or cron entries (T1053) created on runner hosts during or after scanner execution windows, compare against baseline. (5) Outbound staging activity (T1608), large data transfers or unusual POST requests from runner hosts to external infrastructure. MITRE techniques to prioritize in SIEM rules: T1195.001, T1195.002, T1526, T1538, T1552.001, T1053. IOC availability: check your threat intelligence platform or CISA alerts for TeamPCP-specific indicators (IPs, domains, hashes). If none are published, detection should rely on behavioral patterns and anomaly baselines.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`[not confirmed in available source data]`	No specific IOCs have been published in available source data for the TeamPCP campaign as of the report date. Monitor threat intelligence feeds for future IOC releases.	LOW

Framework Mappings

MITRE-ATTACK

- **T1554** — Compromise Host Software Binary
- **T1195.002** — Compromise Software Supply Chain
- **T1552.001** — Credentials In Files
- **T1078** — Valid Accounts
- **T1552** — Unsecured Credentials
- **T1526** — Cloud Service Discovery
- **T1059** — Command and Scripting Interpreter
- **T1608** — Stage Capabilities
- **T1538** — Cloud Service Dashboard
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1053** — Scheduled Task/Job

NIST-800-53R5

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

OWASP-TOP10-2021

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

CIS-V8

- **5.2** — Use Unique Passwords
- **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

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.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
- **DE.AE-08** — Incidents are declared when adverse events meet the defined incident criteria

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1554	Compromise Host Software Binary	Persistence
T1195.002	Compromise Software Supply Chain	Initial-Access
T1552.001	Credentials In Files	Credential-Access
T1078	Valid Accounts	Defense-Evasion
T1552	Unsecured Credentials	Credential-Access
T1526	Cloud Service Discovery	Discovery

Technique ID	Technique Name	Tactic
T1059	Command and Scripting Interpreter	Execution
T1608	Stage Capabilities	Resource-Development
T1538	Cloud Service Dashboard	Discovery
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1053	Scheduled Task/Job	Execution

Sources

Source	URL	Tier
Security News	https://www.sans.org/white-papers/when-security-scanner-became-weapon	T3
Assess Vulnerabilities and Misconfigurations in CICD Pipelines: Part 1	https://success.qualys.com/support/s/article/000005841	T3
CI/CD Pipeline Security Best Practices: The Ultimate Guide - Wiz	https://www.wiz.io/academy/application-security/ci-cd-security-best...	T3
Security in CI/CD Pipelines - Medium	https://medium.com/@cyberoptic/security-in-ci-cd-pipelines-8265bfc0...	T3
3 Simple Techniques to Add Security Into the CI/CD Pipeline	https://www.paloaltonetworks.com/blog/2020/10/cloud-add-security-ci...	T3

Generated 2026-04-01 18:41 UTC by TJS Security Command Center