INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-01 07:10 UTC

TeamPCP Mini Shai-Hulud Campaign: Cross-Ecosystem Supply Chain Attack Targets PyTorch Lightning, Intercom npm, and Intercom PHP

THREAT CAMPAIGN | CRITICAL | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0247
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	PyTorch Lightning (PyPI, versions 2.6.2, 2.6.3), intercom-client (npm, version 7.0.4), intercom/intercom-php (Packagist/Composer, version 5.0.2), pyannote-audio (transitive vector); GitHub, npm, and Packagist ecosystems
Published	2026-04-30T12:31:00
Discovery Source	Rss

Executive Summary

A threat actor cluster has simultaneously poisoned packages across three major software ecosystems - PyPI, npm, and Packagist - embedding credential-stealing, self-propagating malware into PyTorch Lightning (versions 2.6.2 and 2.6.3), the Intercom npm client (version 7.0.4), and the Intercom PHP library (version 5.0.2). Any organization whose developers or CI/CD pipelines installed these versions is at risk of having cloud credentials, API keys, and pipeline secrets exfiltrated. Given PyTorch Lightning's presence in AI/ML workflows and Intercom's enterprise customer communication footprint, the potential blast radius spans both production AI infrastructure and customer-facing platforms.

Technical Analysis

This campaign represents a coordinated, multi-ecosystem supply chain attack with worm-like propagation characteristics. Confirmed compromised versions: PyTorch Lightning 2.6.2 and 2.6.3 (PyPI), intercom-client 7.0.4 (npm), intercom/intercom-php 5.0.2 (Packagist/Composer). The pyannote-audio package has been identified as a transitive infection vector within affected PyTorch Lightning environments. Malware behavior maps to: T1195.001 (Compromise Software Supply Chain), T1554 (Compromise Client Software Binary), T1552.001 and T1552.004 (credential harvesting from files and private keys), T1059/T1059.007 (script

execution), T1020 (automated exfiltration), and T1567.001 (exfiltration to code repositories). The self-propagating mechanism - whereby a compromised developer workstation becomes an active poisoning node for additional packages within that environment - represents lateral movement via T1570 (Lateral Tool Transfer) following initial T1195.001 compromise. Primary payload targets: developer secrets, CI/CD credentials (e.g., GitHub Actions tokens, environment variables), and cloud provider tokens (AWS, GCP, Azure). Relevant CWEs: CWE-829 (Inclusion of Functionality from Untrusted Control Sphere), CWE-494 (Download of Code Without Integrity Check), CWE-506 (Embedded Malicious Code), CWE-522 (Insufficiently Protected Credentials). No CVE has been assigned to this supply chain poisoning campaign. CVE-2024-5980 (a prior path traversal flaw in PyTorch Lightning) is unrelated to these malicious package versions. Source quality is T3 (secondary trade press); no authoritative vendor advisory or CISA alert confirmed at time of writing. CVSS 9.5 (analyst-assigned, not vendor-confirmed). Treat all attribution and scope details as preliminary pending confirmation from official vendor advisories or threat intelligence authorities.

Action Checklist

1. Step 1: Containment, Immediately block installation or execution of PyTorch Lightning 2.6.2/2.6.3, intercom-client 7.0.4, and intercom/intercom-php 5.0.2 across all developer workstations, build servers, and CI/CD pipelines. Freeze any pipelines that pulled these versions in the last 30 days. Isolate affected developer environments from internal networks pending investigation.
2. Step 2: Detection, Audit pip, npm, and Composer lock files across all repositories for the affected versions. Search CI/CD pipeline logs for install commands referencing these version strings. Look for anomalous outbound connections from build agents and unexpected cloud API calls. (See Detection Guidance for specific behavioral indicators and technique mappings.)
3. Step 3: Eradication, Downgrade or upgrade to confirmed-clean versions: for PyTorch Lightning, revert to 2.6.1 or await an official patched release from Lightning-AI. For intercom-client and intercom/intercom-php, remove affected versions and hold installation until Intercom publishes a verified clean release. Remove pyannote-audio from environments that installed it alongside compromised PyTorch Lightning versions. Purge local pip, npm, and Composer caches on all affected systems.
4. Step 4: Recovery, Rotate all secrets, API keys, and cloud provider credentials accessible from any environment that installed the affected packages; treat them as fully compromised. Revoke and reissue GitHub tokens, AWS IAM keys, GCP service account keys, and any CI/CD secrets stored as environment variables. Validate package integrity on replacement installs using hash verification against official registry checksums. Re-run dependency audits (pip-audit, npm audit, composer audit) post-remediation before restoring pipeline operations.
5. Step 5: Post-Incident, Implement dependency pinning with hash verification (PEP 508 hashes for pip, integrity fields in package-lock.json, composer.lock hash validation) to prevent silent version substitution in future installs. Enforce a private mirror or artifact proxy (e.g., Artifactory, Nexus) with allow-listing for approved packages. Add supply chain security scanning (e.g., SLSA provenance checks, Sigstore/cosign verification where available) to CI/CD pipelines. Review secrets management posture: CI/CD secrets should use short-lived, scoped tokens, not long-lived static keys accessible to build environments.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate to CISO, legal counsel, and breach notification team immediately if AWS CloudTrail, GCP Audit Logs, or GitHub audit logs confirm that exfiltrated credentials were used to access production systems, customer data stores, or any environment containing PII/PHI/PCI-scoped data, as this converts a supply chain compromise into a reportable data breach under applicable regulations (GDPR Article 33, HIPAA 45 CFR §164.410, or state breach notification statutes).
Recovery Notes	Before restoring any CI/CD pipeline to production operation, verify three gates: (1) all four affected package versions are absent from every lock file and cache, confirmed by 'pip-audit', 'npm audit', and 'composer audit' returning no findings for the poisoned versions; (2) all credentials in scope during the exposure window have been revoked and replaced with newly issued, least-privilege tokens, with AWS CloudTrail and GCP Audit Logs showing zero activity from the old key IDs post-rotation; and (3) hash-pinned dependency files are committed and enforced in pipeline configuration. Maintain enhanced monitoring of outbound network connections from all build agents for a minimum of 30 days post-recovery, specifically watching for connections to domains or IPs associated with TeamPCP infrastructure identified during the incident. Re-run the full lock file audit on a weekly basis for the first 30 days to catch any reintroduction of affected versions through dependency tree updates or developer workstation installs.
Forensic Artifacts	Malicious wheel/tarball files preserved from pip, npm, and Composer cache directories (~/.cache/pip/wheels/, ~/.npm/_cacache/, ~/.composer/cache/) containing the TeamPCP-embedded credential-stealing payload — primary artifact for hash IOC generation and YARA rule development specific to this campaign | CI/CD pipeline execution logs (GitHub Actions run logs, GitLab CI job traces, Jenkins build logs) for the 30-day window containing 'pip install pytorch-lightning==2.6.2' or '==2.6.3', 'npm install intercom-client@7.0.4', or 'composer require intercom/intercom-php:5.0.2' commands — establishes definitive install timestamps for credential rotation scope determination | AWS CloudTrail and GCP Cloud Audit Logs filtered to CI/CD service account identities for the exposure window, specifically GetSecretValue, AssumeRole, storage.objects.get, and secretmanager.versions.access events from source IPs outside known pipeline infrastructure — evidence of whether exfiltrated credentials were actively weaponized by TeamPCP post-exfiltration | DNS resolution logs and network flow records from build agent hosts showing outbound connections initiated by python3, node, or php processes during or immediately after package install events — captures T1567.001 exfiltration traffic to attacker-controlled endpoints that may masquerade as legitimate registry or CDN infrastructure | Git reflog and commit history from internal package repositories and CI/CD configuration files for the 30-day window, filtered to commits by service account or bot identities, to detect T1554 (Compromise Software Supply Chain) lateral movement where TeamPCP may have used exfiltrated GitHub tokens to inject malicious code into internal dependencies

Per-Action IR Details

Step 1: Containment — Immediately block installation or execution of PyTorch Lightning 2.6.2/2.6.3, intercom-client 7.0.4, and intercom/intercom-php 5.0.2 across all developer workstations, build servers, and CI/CD pipelines. Freeze any pipelines that pulled these versions in the last 30 days. Isolate affected developer environments from internal networks pending investigation.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy (choose a containment strategy based on the need to keep the attacker unaware, preserve evidence, and prevent further damage)

Controls: NIST IR-4 (Incident Handling), NIST CM-7 (Least Functionality) — restrict package registries to deny affected version strings, NIST SI-3 (Malicious Code Protection) — block execution of known-malicious package

versions at endpoint and pipeline level, CIS 2.3 (Address Unauthorized Software) — treat poisoned package versions as unauthorized software requiring immediate removal or blocking, CIS 4.4 (Implement and Manage a Firewall on Servers) — block outbound connections from build agents to attacker-controlled exfiltration endpoints

Compensating: On developer workstations without EDR: run 'pip index versions pytorch-lightning' to confirm registry still serves 2.6.2/2.6.3, then immediately add version exclusions to ~/.pip/pip.conf or /etc/pip.conf using 'constraint' files pointing to a local constraints.txt containing 'pytorch-lightning!=2.6.2,!=2.6.3'. For npm, run 'npm config set ignore-scripts true' globally on all build agents and add '.npmrc' deny rules. For Composer, add 'conflict' entries in composer.json. Block outbound TCP 443/80 from CI runner hosts to pypi.org, registry.npmjs.org, and packagist.org at the host firewall using 'iptables -A OUTPUT -p tcp --dport 443 -d pypi.org -j DROP' (resolve to IP first) pending proxy configuration. Freeze GitHub Actions, GitLab CI, and Jenkins pipelines by disabling the runner service: 'systemctl stop gitlab-runner' or equivalent.

Evidence: Before isolating any affected developer workstation or CI runner, capture: (1) Full memory dump of the build agent process using 'procdump -ma ' (Windows) or 'gcore ' (Linux) for any Python, Node, or PHP interpreter process that executed during the poisoned install window — the malware payload may reside in heap memory. (2) Snapshot of all active outbound network connections from the build host using 'ss -tunap > /tmp/connections_$(date +%s).txt' or 'netstat -anob' on Windows, capturing any established sessions to non-corporate IPs at the time of isolation. (3) Copy of the pip, npm, and Composer cache directories in their current state before purging: ~/.cache/pip/, ~/.npm/_cacache/, ~/.composer/cache/ — these preserve the malicious wheel/tarball artifacts for hash comparison against registry records.

Step 2: Detection — Audit pip, npm, and Composer lock files (requirements.txt, package-lock.json, composer.lock) across all repositories for the affected versions. Search CI/CD pipeline logs for install commands referencing these version strings. Look for anomalous outbound connections from build agents to code repositories or unexpected cloud API calls (T1567.001, T1071). Check for new or modified scripts in developer home directories and CI runner environments. Review git history on internal packages for unauthorized commits (T1554).

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis (use all available data sources to determine the scope, origin, and impact of the incident; prioritize analysis based on functional impact)

Controls: NIST IR-4 (Incident Handling) — scope analysis across all three affected ecosystems simultaneously, NIST AU-6 (Audit Record Review, Analysis, and Reporting) — review CI/CD pipeline logs and package manager logs for install events referencing 2.6.2, 2.6.3, 7.0.4, and 5.0.2, NIST AU-12 (Audit Record Generation) — verify that package installation events were logged by CI/CD systems during the 30-day exposure window, NIST SI-4 (System Monitoring) — correlate outbound network events from build agents with package install timestamps for T1567.001 (Exfiltration to Code Repository) and T1071 (Application Layer Protocol) indicators, CIS 8.2 (Collect Audit Logs) — ensure CI/CD runner logs, package manager verbose output, and network flow data are collected and centralized before log rotation, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — use this detection sweep to build a definitive list of affected systems for scoped remediation

Compensating: For git repository scanning without SIEM: run 'grep -r "pytorch-lightning==2.6.2\|pytorch-lightning==2.6.3\|intercom-client.*7.0.4\|intercom/intercom-php.*5.0.2" --include="*.txt" --include="*.json" --include="*.lock" /path/to/repos/' recursively across all local clones. For pipeline log search: 'grep -E "pytorch.lightning.*(2\.6\.2|2\.6\.3)|intercom.client.*7\.0\.4" ~/.jenkins/logs/ -r' or equivalent GitLab/Actions log directories. For network artifact detection without SIEM: parse system DNS logs using 'journalctl -u systemd-resolved | grep -E "(github\.com|api\.github\.com|pypi\.org|registry\.npmjs\.org)"' filtered to the install timestamp window, then cross-reference with 'ausearch -sc connect' (Linux auditd) for socket calls from python3, node, or php processes. For T1554 (Compromise Software Supply Chain via Modify Repository) detection in internal repos: 'git log --all --author-date-order --diff-filter=A -- "*.py" "*.js" "*.php"' to find files added by unexpected committer identities in the past 30 days.

Evidence: Before running detection sweeps that may alter timestamps: (1) Preserve CI/CD pipeline run logs in their original format — for GitHub Actions export via API ('gh run list --json databaseId,conclusion,createdAt | gh run download'), for Jenkins copy $JENKINS_HOME/jobs//builds//log files with 'rsync -a --times'. (2) Export pip install history from all developer hosts: 'pip list --format=json > pip_installed_$(hostname).json' and 'cat ~/.local/share/pip/pip_history'

(if maintained) — on macOS check '~/Library/Logs/pip/'. (3) For T1567.001 detection, extract DNS query logs from the build host's resolver for the 30-day window: 'resolvectl query' history (systemd) or /var/log/named/ (bind) to identify resolutions of attacker-controlled exfiltration domains that may masquerade as legitimate package registry infrastructure. (4) Capture git reflog from any internal package repositories: 'git reflog --all > reflog_$(date +%s).txt' before any force-push or history rewrite operations occur during cleanup.

Step 3: Eradication — Downgrade or upgrade to confirmed-clean versions: for PyTorch Lightning, revert to 2.6.1 or monitor the official Lightning-AI GitHub releases page for a patched release announcement before upgrading forward. For intercom-client and intercom/intercom-php, remove the affected versions and hold installation until Intercom publishes a verified clean release. Remove pyannote-audio from environments that installed it alongside compromised PyTorch Lightning versions. Purge local pip, npm, and Composer caches on all affected systems.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication (eliminate components of the incident such as malware and inappropriate access, and mitigate the vulnerabilities that were exploited)

Controls: NIST IR-4 (Incident Handling) — eradication must cover all three ecosystems and the transitive pyannote-audio vector simultaneously to prevent reinfection, NIST SI-2 (Flaw Remediation) — replace poisoned package versions with verified-clean versions; do not upgrade forward until vendor confirms clean release, NIST SI-7 (Software, Firmware, and Information Integrity) — verify replacement package hashes against official PyPI, npm, and Packagist registry checksums before reinstalling, NIST CM-7 (Least Functionality) — remove pyannote-audio from any environment where it was not explicitly required; its presence is a transitive infection vector specific to this campaign, CIS 2.1 (Establish and Maintain a Software Inventory) — update software inventory to reflect removal of all four affected packages and record clean replacement versions, CIS 7.4 (Perform Automated Application Patch Management) — revert to PyTorch Lightning 2.6.1 via controlled downgrade, not automated update, until Lightning-AI confirms patch integrity

Compensating: For cache purge verification without enterprise tooling: Python — 'pip cache purge && pip cache list' to confirm empty; manually verify '~/.cache/pip/wheels/' and '~/.cache/pip/http/' are empty. npm — 'npm cache clean --force && npm cache verify'. Composer — 'composer clear-cache && ls ~/.composer/cache/'. For hash verification of replacement installs without Artifactory: after installing pytorch-lightning==2.6.1, run 'pip download pytorch-lightning==2.6.1 --no-deps -d /tmp/verify/ && pip hash /tmp/verify/*.whl' and compare SHA256 output against the PyPI JSON API response at 'https://pypi.org/pypi/pytorch-lightning/2.6.1/json' (field: releases[].digests.sha256) — do this comparison on an air-gapped or clean workstation. For pyannote-audio removal: 'pip show pyannote-audio | grep Location' to find install path, then 'pip uninstall pyannote-audio -y' and verify removal with 'pip show pyannote-audio' returning 'not found'.

Evidence: Before purging caches or uninstalling packages, preserve: (1) Malicious wheel and tarball artifacts from the pip cache — 'cp -r ~/.cache/pip/wheels/ /evidence/pip_cache_$(hostname)/' — these files contain the embedded credential-stealing payload and are needed for YARA rule development and hash-based IOC generation. (2) The installed package file tree for each poisoned package before removal: 'pip show pytorch-lightning -f > /evidence/ptl_filelist_$(hostname).txt' then copy the listed files to an evidence directory — focus on any .py files in the package that differ from the 2.6.1 release (diff against clean 2.6.1 source from the official Lightning-AI GitHub tag). (3) For npm: 'npm ls intercom-client --json > /evidence/npm_tree_$(hostname).json' and copy node_modules/intercom-client/ in full before removal. (4) For Composer: copy vendor/intercom/intercom-php/ before 'composer remove'.

Step 4: Recovery — Rotate all secrets, API keys, and cloud provider credentials accessible from any environment that installed the affected packages — treat them as fully compromised. Revoke and reissue GitHub tokens, AWS IAM keys, GCP service account keys, and any CI/CD secrets stored as environment variables. Validate package integrity on replacement installs using hash verification against official registry checksums. Re-run dependency audits (pip-audit, npm audit, composer audit) post-remediation before restoring pipeline operations.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery (restore systems to normal operation, confirm that the systems are functioning normally, and implement additional monitoring to watch for signs of re-compromise)

Controls: NIST IR-4 (Incident Handling) — recovery must include credential rotation before pipeline restoration; restoring pipelines with unrotated secrets re-exposes exfiltrated material, NIST IA-5 (Authenticator Management) — revoke and reissue all GitHub PATs, AWS IAM access keys, GCP service account JSON keys, and CI/CD environment variable secrets that were in scope during the infection window, NIST AC-2 (Account Management) — audit which CI/CD service accounts had access to production systems during the exposure window and apply least-privilege constraints on reissued credentials, NIST SI-7 (Software, Firmware, and Information Integrity) — validate SHA256 hashes of all replacement packages against PyPI, npm registry, and Packagist JSON APIs before pipeline restoration, CIS 5.2 (Use Unique Passwords) — reissued secrets must be unique and not reused from any pre-incident credential store that may have been exfiltrated, CIS 7.2 (Establish and Maintain a Remediation Process) — run pip-audit, npm audit, and composer audit as gate checks before re-enabling pipelines; document pass results as evidence of clean state

Compensating: For credential rotation without a secrets management platform: AWS — 'aws iam list-access-keys --user-name ' to enumerate, 'aws iam delete-access-key --access-key-id ' then 'aws iam create-access-key --user-name ' for replacement; immediately update GitHub Actions secrets via 'gh secret set AWS_ACCESS_KEY_ID' and 'gh secret set AWS_SECRET_ACCESS_KEY'. GCP — 'gcloud iam service-accounts keys list --iam-account=@.iam.gserviceaccount.com' to enumerate, 'gcloud iam service-accounts keys delete ' then create new key and update CI secrets. GitHub PATs — enumerate via 'gh auth token' and rotate via GitHub Settings > Developer Settings > Personal Access Tokens. For pip-audit on systems without SIEM: 'pip-audit -r requirements.txt --output json > audit_$(date +%s).json' and review for any remaining vulnerable packages; repeat for each requirements file in the repository.

Evidence: Before restoring pipelines, document the credential rotation audit trail: (1) Export AWS CloudTrail logs for the 30-day window filtered to the compromised IAM user/role ARNs — 'aws cloudtrail lookup-events --lookup-attributes AttributeKey=Username,AttributeValue= --start-time ' — to determine whether exfiltrated keys were used to make unauthorized API calls (look for CreateUser, PutUserPolicy, AssumeRole, GetSecretValue events from unexpected source IPs). (2) For GCP, export Cloud Audit Logs for the service account: 'gcloud logging read "protoPayload.authenticationInfo.principalEmail=@.iam.gserviceaccount.com" --format json' filtered to the exposure window, looking for storage.objects.get, secretmanager.versions.access, or container.clusters.get events from non-pipeline source IPs. (3) GitHub audit log export for the organization filtered to the CI bot account for the 30-day window, specifically looking for unexpected repository clones, secret reads, or workflow modifications attributable to the compromised PAT.

Step 5: Post-Incident — Implement dependency pinning with hash verification (PEP 508 hashes for pip, integrity fields in package-lock.json, composer.lock hash validation) to prevent silent version substitution in future installs. Enforce a private mirror or artifact proxy (e.g., Artifactory, Nexus) with allow-listing for approved packages. Add supply chain security scanning (e.g., SLSA provenance checks, Sigstore/cosign verification where available) to CI/CD pipelines. Review secrets management posture: CI/CD secrets should use short-lived, scoped tokens — not long-lived static keys accessible to build environments.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity (use the knowledge gained during the incident to improve incident handling procedures, defenses, and detection capabilities; conduct a lessons-learned meeting and update IR plans)

Controls: NIST IR-4 (Incident Handling) — update IR playbook to include a supply chain poisoning scenario with specific detection steps for pip/npm/Composer lock file audits as a standing triage procedure, NIST IR-8 (Incident Response Plan) — revise the IR plan to define package ecosystem compromise as a named incident category with defined severity thresholds and credential rotation timelines, NIST SI-7 (Software, Firmware, and Information Integrity) — implement hash-pinned dependency files (pip --require-hashes, package-lock.json integrity SHA512, composer.lock hash verification) as a preventive integrity control, NIST SA-12 (Supply Chain Protection) — enforce private artifact proxy with package allow-listing to eliminate direct developer/pipeline access to public PyPI, npm, and Packagist registries, NIST IA-5 (Authenticator Management) — replace long-lived CI/CD static keys with short-lived OIDC-federated tokens (GitHub Actions OIDC for AWS/GCP) so that future credential exfiltration yields tokens with

bounded TTL, CIS 2.2 (Ensure Authorized Software is Currently Supported) — add PyPI, npm, and Packagist packages to software inventory with version tracking; flag any package version not on the approved list, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — integrate pip-audit, npm audit, and composer audit into CI/CD pipeline as blocking gates; fail builds on HIGH or CRITICAL findings

Compensating: For teams without Artifactory/Nexus budget: configure pip to use a local PyPI mirror via 'devpi' (free, open source) — 'pip install devpi-server && devpi-server --start' — and set pip.conf to point to it with an allow-list of approved packages. For npm, use 'verdaccio' (free) as a local registry proxy with package white-listing. For hash pinning without enterprise tooling: generate pip hash-pinned requirements with 'pip-compile --generate-hashes requirements.in > requirements.txt' using pip-tools (free). For SLSA provenance checks in GitHub Actions without paid tools: add the free 'sigstore/cosign-installer' GitHub Action and verify PyPI package provenance where sigstore attestations exist using 'cosign verify-attestation --type slsaprovenance '. For short-lived AWS credentials in GitHub Actions: implement the free 'aws-actions/configure-aws-credentials' OIDC action to replace static IAM keys with role assumption tokens that expire in 15 minutes.

Evidence: For the lessons-learned meeting and control validation, collect: (1) The complete timeline of affected version pulls across all repositories, reconstructed from git history, CI/CD logs, and package manager logs — this becomes the definitive blast radius document. (2) A comparison diff of the malicious PyTorch Lightning 2.6.2/2.6.3 package contents versus the clean 2.6.1 release (use 'pkgdiff' or manual 'diff -rq' against extracted wheel contents) — document exactly which files were modified and what credential-stealing code was injected, to inform YARA rule development for future detections. (3) Network flow records showing any successful exfiltration connections from build agents during the exposure window — source IP, destination IP/domain, bytes transferred, timestamps — to determine whether credentials were successfully exfiltrated or only at risk. (4) Updated threat model documentation noting TeamPCP as a confirmed supply chain threat actor targeting Python ML, JavaScript API client, and PHP library ecosystems simultaneously, for inclusion in the organization's threat intelligence register.

Detection Guidance

Primary detection targets: CI/CD pipeline logs and package manager audit logs. Query for installs of pytorch-lightning==2.6.2, pytorch-lightning==2.6.3, intercom-client@7.0.4, or intercom/intercom-php:5.0.2 in pip install logs, npm install outputs, and composer install records. On developer workstations, check ~/.cache/pip, node_modules/.package-lock.json, and vendor/composer/installed.json for version strings. Behavioral indicators: unexpected outbound HTTPS or SSH-based git traffic from build agents to external repositories (especially GitHub, GitLab, or Bitbucket endpoints not in your standard dependency graph); new cron jobs or systemd services created on developer machines (T1543); anomalous reads of .env files, ~/.aws/credentials, ~/.config/gcloud, or CI environment variable stores (T1552.001, T1552.004); commits or pushes to package repositories not initiated by known maintainer accounts (T1608.003). SIEM queries should flag: process executions of python, node, or php spawning child processes that initiate outbound network connections during package install phases. Worm behavior indicator: monitor internal package repositories and artifact proxies for unexpected new package versions or modified checksums on previously trusted packages.

Indicators of Compromise

Type	Value	Context	Confidence
HASH	`[not confirmed in available sources]`	No package hashes for malicious versions confirmed in T3 sources at time of analysis — verify against official PyPI, npm, and Packagist registry checksums for affected versions	LOW

Type	Value	Context	Confidence
URL	`[not confirmed in available sources]`	No C2 domains or exfiltration endpoints published in available T3 sources — monitor for anomalous outbound git-protocol or HTTPS connections from build environments	LOW

Framework Mappings

MITRE-ATTACK

- **T1040** — Network Sniffing
- **T1543** — Create or Modify System Process
- **T1176** — Software Extensions
- **T1554** — Compromise Host Software Binary
- **T1552.001** — Credentials In Files
- **T1071** — Application Layer Protocol
- **T1608.003** — Install Digital Certificate
- **T1552.004** — Private Keys
- **T1059** — Command and Scripting Interpreter
- **T1059.007** — JavaScript
- **T1020** — Automated Exfiltration
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1567.001** — Exfiltration to Code Repository
- **T1036.003** — Rename Legitimate Utilities

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
- **IA-5** — Authenticator Management
- **AT-2** — Literacy Training and Awareness
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

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

- **A07:2021** — Identification and Authentication Failures

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **5.2** — Use Unique Passwords
- **6.3** — Require MFA for Externally-Exposed Applications
- **14.2** — Train Workforce Members to Recognize Social Engineering Attacks
- **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
- **164.312(e)(1)** — Transmission Security

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

NIST-CSF-2

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1040	Network Sniffing	Credential-Access
T1543	Create or Modify System Process	Persistence
T1176	Software Extensions	Persistence
T1554	Compromise Host Software Binary	Persistence
T1552.001	Credentials In Files	Credential-Access
T1071	Application Layer Protocol	Command-And-Control
T1608.003	Install Digital Certificate	Resource-Development
T1552.004	Private Keys	Credential-Access

Technique ID	Technique Name	Tactic
T1059	Command and Scripting Interpreter	Execution
T1059.007	JavaScript	Execution
T1020	Automated Exfiltration	Exfiltration
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1567.001	Exfiltration to Code Repository	Exfiltration
T1036.003	Rename Legitimate Utilities	Defense-Evasion

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/04/pytorch-lightning-compromised-in-...	T3
PyTorch Lightning package compromised in supply-chain attack	https://letsdatascience.com/news/pytorch-lightning-package-compromi...	T3
Lightning vulnerability CVE-2024-5980 · Issue #20032 - GitHub	https://github.com/Lightning-AI/pytorch-lightning/issues/20032	T3
Releases · Lightning-AI/pytorch-lightning - GitHub	https://github.com/Lightning-AI/pytorch-lightning/releases	T3
CVE-2024-5980: PyTorch Lightning Path Traversal Flaw - SentinelOne	https://www.sentinelone.com/vulnerability-database/cve-2024-5980/	T3

Generated 2026-05-01 07:10 UTC by TJS Security Command Center