INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-15 13:51 UTC

TeamPCP Monetizes Shai-Hulud Fallout: Mistral AI Source Code Listed at $25K as AI Vendor Supply Chain Breach Widens

THREAT CAMPAIGN | **HIGH** | CVSS 7.5

SCC Item ID	SCC-CAM-2026-0317
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	7.5
Affected Products	Mistral AI (internal repositories, SDK/npm packages), OpenAI (internal source code repositories), TanStack (npm packages), UiPath, Guardrails AI, OpenSearch, npm registry, PyPI registry
Published	2026-05-14T18:50:36
Discovery Source	Rss

Executive Summary

TeamPCP, the threat actor behind the Shai-Hulud supply chain attack, is now selling approximately 450 internal Mistral AI source code repositories for $25,000, with a one-week deadline before threatened free public release. The underlying attack compromised CI/CD pipeline credentials to inject malicious code into npm and PyPI packages consumed by Mistral AI, OpenAI, UiPath, Guardrails AI, and OpenSearch, meaning organizations using these vendors' SDKs may have executed malicious code in their own environments. The compounding risk, active extortion, confirmed package contamination, and potential exposure of proprietary AI source code, demands immediate dependency auditing and incident response triage across any environment consuming affected packages.

Technical Analysis

The Shai-Hulud attack (no CVE assigned) exploited hardcoded or stolen CI/CD pipeline credentials (CWE-798) to inject malicious code into npm and PyPI packages, abusing a worm-style propagation pattern. TanStack npm packages served as an initial vector, spreading contamination across 160+ downstream npm packages. Affected ecosystems include Mistral AI SDKs, OpenAI internal tooling, UiPath automation libraries, Guardrails AI, and OpenSearch packages. The attack chain combines supply chain compromise of software libraries (T1195.002), software dependency abuse (T1195.001), credential exposure via CI/CD secrets (T1552.001, T1552.004), and now active financial extortion (T1657) against Mistral AI. Malicious payload delivery follows

T1059 (command execution via compromised packages) with characteristics consistent with T1486 (data impact/exfiltration leverage). CWE-494 (Download of Code Without Integrity Check) and CWE-829 (Inclusion of Functionality from Untrusted Control Sphere) describe the package integrity failure conditions. No CVSS vector or EPSS data is available for this campaign. No patch has been formally issued; mitigation depends on dependency auditing, package version pinning, and CI/CD credential rotation. Mistral AI has published a security advisory at https://docs.mistral.ai/resources/security-advisories (direct vendor confirmation recommended). Source reporting via BleepingComputer and The Hacker News as of the configuration date; details remain active and may evolve.

Action Checklist

1. Step 1: Containment, Immediately audit all npm and PyPI dependencies in your build pipelines for packages originating from or consuming TanStack, Mistral AI SDKs, OpenAI libraries, UiPath packages, Guardrails AI, and OpenSearch. Isolate any build or runtime environment that has executed code from these packages since the attack window. Block outbound connections from affected CI/CD runners pending investigation. Do not install or update affected packages until integrity is confirmed.
2. Step 2: Detection, Query your SIEM and EDR for anomalous process execution spawned by npm or pip install processes, unexpected outbound network connections from CI/CD agents, and any file writes or registry modifications occurring during dependency installation. Review CI/CD pipeline logs for unauthorized credential access or secret reads (CWE-798 exploitation evidence). Cross-reference installed package versions against known-good hashes published in vendor advisories. Check for presence of TanStack packages and any downstream packages introduced via TanStack dependency chains.
3. Step 3: Eradication, Pin all affected dependencies to versions verified as clean per Mistral AI's security advisory and vendor communications from OpenAI, UiPath, Guardrails AI, and OpenSearch. Rotate all CI/CD pipeline credentials, secrets, and API tokens that were accessible in compromised pipeline environments. Remove and reinstall any packages pulled during the contamination window from a clean environment with verified checksums. Revoke and reissue any tokens or keys that may have been exposed during pipeline execution.
4. Step 4: Recovery, Validate rebuilt artifacts using integrity checks against vendor-published hashes before redeployment. Monitor production environments for anomalous behavior consistent with malicious payload execution: unexpected process spawning, unusual outbound connections, and unauthorized data access. Re-run security scans (SCA/SAST) against your full dependency tree post-remediation. Confirm with affected vendors that their published packages are clean before resuming normal dependency update workflows.
5. Step 5: Post-Incident, Conduct a gap assessment against NIST SP 800-161 (C-SCRM) supply chain risk management controls. Implement mandatory integrity verification (checksums, sigstore/sigsum signatures) for all third-party packages before installation. Enforce least-privilege access for CI/CD service accounts to limit credential exposure blast radius. Evaluate software composition analysis (SCA) tooling integration into CI/CD pipelines as a preventive control. Document findings and update your software supply chain incident response playbook.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and executive leadership immediately if forensic analysis confirms malicious payload execution in production (not just CI/CD), if any secrets or credentials exfiltrated from the pipeline are scoped to customer-facing systems or data stores, or if your organization's code or data may be included in the 450 Mistral AI repositories TeamPCP has listed for sale or threatened to release publicly — all three conditions may trigger regulatory breach notification obligations.
Recovery Notes	Before resuming normal dependency update workflows, require explicit vendor confirmation (not just a package version bump) from Mistral AI, OpenAI, UiPath, Guardrails AI, and OpenSearch that their published SDK packages have been re-signed and independently verified clean, and pin all packages to those confirmed-clean versions with exact version locking and integrity hash verification enforced in CI. Monitor production environments running any of the affected SDKs for a minimum of 30 days post-remediation, specifically watching for low-and-slow C2 beacon patterns, periodic outbound connections on non-standard ports, and any lateral movement from the application runtime user account. Treat any CI/CD credential or API token that was in scope during the attack window as permanently compromised regardless of whether active abuse has been confirmed — do not reinstate rotated credentials under any circumstances.
Forensic Artifacts	npm and pip package cache directories containing the original malicious tarballs: Linux (~/.npm/_cacache/, ~/.cache/pip/wheels/), Windows (%APPDATA%\npm-cache\, %LOCALAPPDATA%\pip\Cache\) — these preserve the exact TeamPCP-injected package versions with original sha512 hashes for malware analysis and timeline reconstruction | CI/CD pipeline execution logs for the attack window containing postinstall lifecycle hook execution entries for @tanstack/* packages — these will show the exact command line of the malicious script that ran inside the npm install process and any child processes it spawned (look for curl/wget/fetch calls in the hook output) | CI/CD runner environment variable snapshots and secrets access audit logs (GitHub Actions: Settings > Security log filtered by 'secret'; GitLab CI: audit_events API endpoint) — these establish which credentials (NPM_TOKEN, PYPI_API_TOKEN, cloud provider keys) were in memory and accessible to the malicious postinstall hook at execution time | Network packet captures (pcap) from CI/CD runner hosts during the attack window, filtered for outbound connections from node.exe or python.exe to non-internal, non-CDN destinations — the Shai-Hulud payload's data exfiltration or C2 registration beacon would appear as an anomalous HTTP/S or DNS request immediately following package installation completion | Sysmon Event ID 1 (Process Creation) or Linux auditd EXECVE records showing the full process ancestry tree rooted at the npm or pip install invocation — this reconstructs exactly what code TeamPCP's injected postinstall hooks executed, including any secondary payloads dropped to disk or environment variables harvested and transmitted

Per-Action IR Details

Step 1: Containment — Immediately audit all npm and PyPI dependencies in your build pipelines for packages originating from or consuming TanStack, Mistral AI SDKs, OpenAI libraries, UiPath packages, Guardrails AI, and OpenSearch. Isolate any build or runtime environment that has executed these packages since the attack window. Block outbound connections from affected CI/CD runners pending investigation. Do not install or update affected packages until integrity is confirmed.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-3 (Configuration Change Control), CIS 2.3 (Address Unauthorized Software), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Run 'npm ls --all --json 2>/dev/null | python3 -c "import sys,json; pkgs=json.load(sys.stdin); [print(k) for k in pkgs.get(\"dependencies\",{}).keys()]"' in each pipeline workspace to enumerate the full dependency tree. Use 'pip freeze > installed.txt' and diff against a known-good baseline. For network blocking on Linux CI runners, use 'iptables -I OUTPUT -j DROP' on the runner host and whitelist only your internal artifact registry. On GitHub Actions or GitLab CI, set the runner to offline mode or remove network permissions from the job definition immediately.

Evidence: Before isolating, snapshot the full npm lock file (package-lock.json or yarn.lock) and requirements.txt/poetry.lock from the compromised pipeline workspace — these record the exact resolved versions of TanStack, Mistral AI SDK, OpenAI, UiPath, Guardrails AI, and OpenSearch packages that were installed during the attack window. Capture the CI/CD runner's outbound network connection state using 'ss -tunap' or 'netstat -an' before applying firewall rules. Preserve the runner's process list ('ps auxf' on Linux, 'Get-Process' on Windows) to identify any child processes spawned by npm/pip install that are still running. Archive the runner's environment variable dump (with secrets redacted after noting which variables were set) to establish which CI/CD credentials were in scope.

Step 2: Detection — Query your SIEM and EDR for anomalous process execution spawned by npm or pip install processes, unexpected outbound network connections from CI/CD agents, and any file writes or registry modifications occurring during dependency installation. Review CI/CD pipeline logs for unauthorized credential access or secret reads (CWE-798 exploitation evidence). Cross-reference installed package versions against known-good hashes published in vendor advisories. Check for presence of TanStack packages and any downstream packages introduced via TanStack dependency chains.

NIST Phase: Detection Analysis

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

Controls: NIST SI-4 (System Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-12 (Audit Record Generation), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: On Linux CI runners without EDR, deploy Sysmon for Linux (available from Microsoft's Sysinternals) and configure it to capture ProcessCreate events (Event ID 1) filtering on parent processes matching 'npm', 'node', 'pip', or 'python' spawning unexpected children (curl, wget, bash, sh, nc). Query runner shell history files (~/.bash_history, ~/.zsh_history) and npm debug logs (typically at ~/.npm/_logs/) for evidence of post-install script execution. Use 'grep -r "postinstall\|preinstall" node_modules/.bin/' to identify all hook scripts that ran. For hash verification without a SIEM, run 'npm audit --json' and cross-reference sha512 integrity hashes in package-lock.json against the NIST NVD or vendor advisory published digests. Use 'pip hash --algorithm sha256 .whl' against downloaded packages. For network forensics, run 'tcpdump -i any -w ci_capture.pcap' on the runner (if still live) and filter for connections to non-internal IPs initiated by node or python processes.

Evidence: Pull GitHub Actions audit logs or GitLab CI job logs for the attack window and search for secret variable reads — specifically for any step that accessed CI_JOB_TOKEN, NPM_TOKEN, PYPI_API_TOKEN, or cloud provider credentials (AWS_ACCESS_KEY_ID, AZURE_CLIENT_SECRET). In npm debug logs (default path: ~/.npm/_logs/YYYY-MM-DDTHH-MM-SS-ms-debug-N.log), look for postinstall lifecycle script execution entries for TanStack packages (@tanstack/*) and any Mistral AI or OpenAI SDK versions installed during the contamination window. On Windows runners, query Windows Security Event Log for Event ID 4688 (Process Creation) where ParentProcessName contains 'node.exe' or 'python.exe' and NewProcessName is 'cmd.exe', 'powershell.exe', 'curl.exe', or 'certutil.exe'. For MITRE ATT&CK mapping, this aligns with T1195.001 (Supply Chain Compromise: Compromise Software Dependencies) and T1059 (Command and Scripting Interpreter) for post-install hook payload execution.

Step 3: Eradication — Pin all affected dependencies to versions verified as clean per Mistral AI's security advisory and vendor communications from OpenAI, UiPath, Guardrails AI, and OpenSearch. Rotate all CI/CD pipeline credentials, secrets, and API tokens that were accessible in compromised pipeline environments. Remove and reinstall any packages pulled during the contamination window from a clean environment with

verified checksums. Revoke and reissue any tokens or keys that may have been exposed during pipeline execution.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST IA-5 (Authenticator Management), NIST CM-3 (Configuration Change Control), NIST SI-7 (Software, Firmware, and Information Integrity), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 7.4 (Perform Automated Application Patch Management), CIS 5.2 (Use Unique Passwords)

Compensating: For credential rotation without a secrets management platform: use the GitHub UI or 'gh secret set' CLI to immediately invalidate and replace NPM_TOKEN, PYPI_API_TOKEN, and any cloud provider keys scoped to the affected runners. For npm token revocation, run 'npm token revoke ' via the npm registry CLI. For PyPI, use the PyPI account settings UI to revoke API tokens and generate new project-scoped tokens. To pin and verify package versions, update package-lock.json by running 'npm install @ --save-exact' and verify the integrity field matches the vendor advisory's published sha512. Use 'pip download == --no-deps -d ./verified/' and verify with 'sha256sum' against advisory hashes before installing into a fresh virtual environment. Provision a fresh ephemeral CI runner (new VM image or Docker container from a known-clean base) rather than cleaning the compromised runner in place.

Evidence: Before rotating credentials, document all CI/CD secrets that were in scope by exporting runner environment variable names (not values) from pipeline logs — this establishes the blast radius for downstream credential abuse. Pull the npm registry access logs for your organization's scope (if using a private registry like Verdaccio or JFrog Artifactory) to determine whether the compromised NPM_TOKEN was used to publish any packages under your namespace. Check GitHub's token usage audit log (Settings > Security log, filter by 'token') for any API calls made with the compromised token outside normal pipeline hours. Preserve all original malicious package tarballs downloaded during the attack window in an isolated evidence directory with sha256 hashes documented — these are the primary forensic artifacts for confirming which TeamPCP payload variant was executed.

Step 4: Recovery — Validate rebuilt artifacts using integrity checks against vendor-published hashes before redeployment. Monitor production environments for anomalous behavior consistent with malicious payload execution: unexpected process spawning, unusual outbound connections, and unauthorized data access. Re-run security scans (SCA/SAST) against your full dependency tree post-remediation. Confirm with affected vendors that their published packages are clean before resuming normal dependency update workflows.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST SI-4 (System Monitoring), NIST CP-10 (System Recovery and Reconstitution), NIST CA-7 (Continuous Monitoring), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 8.2 (Collect Audit Logs)

Compensating: For SCA without a commercial tool, run OWASP Dependency-Check (free, CLI-based) against your rebuilt artifact: 'dependency-check.sh --project MyProject --scan ./node_modules --format JSON --out ./dc-report'. For SAST, run Semgrep OSS ('semgrep --config=p/supply-chain ./src') with the supply-chain ruleset which includes rules for detecting injected postinstall hooks and suspicious network calls in JS/Python packages. For production monitoring without EDR, deploy osquery with a pack targeting process lineage — query 'SELECT pid, name, cmdline, parent FROM processes WHERE parent IN (SELECT pid FROM processes WHERE name IN ("node","python3","python"))' on a 60-second interval to catch any dormant payload activation. Use YARA rules targeting known TeamPCP/Shai-Hulud payload signatures (check the ThreatFox and MalwareBazaar databases for community-published rules) against the node_modules directory before redeployment.

Evidence: Before returning any production system to service, capture a baseline process snapshot, full netstat output, and a recursive file hash manifest of the application directory ('find /app -type f -exec sha256sum {} \; > baseline_hashes.txt') to establish a clean-state reference for future comparison. Monitor production application logs specifically for outbound DNS queries to non-CDN, non-vendor domains initiated by the runtime process (node, python) — the Shai-Hulud payload's C2 beacon pattern or data exfiltration channel would appear here. Retain CI/CD pipeline logs for the full attack window under legal hold in case the Mistral AI source code exposure ($25K listing by TeamPCP) triggers regulatory notification obligations for your organization.

Step 5: Post-Incident — Conduct a gap assessment against NIST SP 800-161 (C-SCRM) supply chain risk management controls. Implement mandatory integrity verification (checksums, sigstore/sigsum signatures) for all third-party packages before installation. Enforce least-privilege access for CI/CD service accounts to limit credential exposure blast radius. Evaluate software composition analysis (SCA) tooling integration into CI/CD pipelines as a preventive control. Document findings and update your software supply chain incident response playbook.

NIST Phase: Post Incident

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

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SI-2 (Flaw Remediation), NIST SA-12 (Supply Chain Protection), NIST AC-6 (Least Privilege), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts)

Compensating: For Sigstore/cosign adoption without enterprise tooling, integrate the free 'cosign verify' CLI into your pipeline as a pre-install gate: 'cosign verify --certificate-identity --certificate-oidc-issuer '. For npm, enable 'npm audit signatures' (built into npm v8+) to verify package provenance against the npm registry's Sigstore transparency log. For least-privilege CI/CD scoping, audit GitHub Actions job-level permissions and set 'permissions: {}' at the workflow level with only required permissions granted per-job — this limits the blast radius if a TanStack-style postinstall hook again attempts to read GITHUB_TOKEN. Document the Shai-Hulud attack timeline, TeamPCP TTPs (T1195.001, T1059), and the specific TanStack/Mistral AI/OpenAI package versions confirmed malicious in your internal threat intelligence wiki for future hunt hypothesis development.

Evidence: Compile a final artifact package for lessons-learned and potential regulatory disclosure: the contaminated package list with exact versions and sha256 hashes, the CI/CD credential rotation audit trail with timestamps, all network capture files from affected runners, and the osquery/Sysmon process execution logs from the detection phase. If any of the Mistral AI internal source code repositories (the 450 repositories listed by TeamPCP) contain your organization's proprietary code, API keys, or customer data — because you are a Mistral AI customer or integration partner — assess whether the threatened free public release triggers breach notification obligations under GDPR, CCPA, or applicable state breach notification statutes. Preserve all evidence under a documented chain of custody per NIST 800-61r3 §3.2 evidence handling guidance.

Detection Guidance

Focus detection on three layers. First, CI/CD pipeline integrity: review pipeline execution logs for unexpected secret reads, unusual environment variable access, or anomalous network calls during package installation steps, particularly in jobs that consumed TanStack, Mistral AI, OpenAI, UiPath, Guardrails AI, or OpenSearch packages. Second, package integrity: compare SHA-256 hashes of installed packages against vendor-published clean hashes; any mismatch is a high-confidence indicator. Flag packages with versions not matching your pinned lockfile. Third, runtime behavioral indicators: look for child processes spawned by node, python, or pip processes that initiate outbound TCP connections to non-standard destinations; file writes outside expected application directories following package installation; and credential or token file reads (matching patterns for .npmrc, .pypirc, CI/CD secret paths) from unexpected processes. MITRE T1195.002 and T1552.001 detection logic in your SIEM should be activated for this campaign. No public IOC hash list has been confirmed in the available sources, treat any package from the affected dependency chains installed during the attack window as suspect until hash-verified against vendor advisories.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://docs.mistral.ai/resources/security-advisories`	Mistral AI official security advisory page — review for confirmed affected package versions and hashes	HIGH
DOMAIN	`See Orca Security blog and Mistral advisory for confirmed malicious package names and versions`	No specific malicious domains or hashes confirmed in available sources at time of content generation — consult vendor advisories directly	LOW

Framework Mappings

MITRE-ATTACK

- **T1195.002** — Compromise Software Supply Chain
- **T1657** — Financial Theft
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1586.002** — Email Accounts
- **T1059** — Command and Scripting Interpreter
- **T1552.001** — Credentials In Files
- **T1486** — Data Encrypted for Impact
- **T1552.004** — Private Keys

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
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **CP-9** — System Backup
- **CP-10** — System Recovery and Reconstitution
- **IA-5** — Authenticator Management
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

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

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **2.5** — Allowlist Authorized Software

- **2.6** — Allowlist Authorized Libraries
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers

ISO-27001-2022

- **A.8.28** — Secure coding
- **A.5.21** — Managing information security in the ICT supply chain

HIPAA-SECURITY

- **164.312(d)** — Person or Entity Authentication

SOC2-TSC

- **CC6.1** — Logical access security software, infrastructure, and architectures
- **CC9.2** — Manages risks associated with vendors and business partners

NIST-CSF-2

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1195.002	Compromise Software Supply Chain	Initial-Access
T1657	Financial Theft	Impact
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1586.002	Email Accounts	Resource-Development
T1059	Command and Scripting Interpreter	Execution
T1552.001	Credentials In Files	Credential-Access
T1486	Data Encrypted for Impact	Impact
T1552.004	Private Keys	Credential-Access

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/teampcp-hackers-adve...	T3
Mini Shai-Hulud Worm Compromises TanStack, Mistral AI ...	https://thehackernews.com/2026/05/mini-shai-hulud-worm-compromises....	T3

Source	URL	Tier
Security advisories | Mistral Docs	https://docs.mistral.ai/resources/security-advisories	T3
Mass npm Supply Chain Attack Hits TanStack, Mistral AI, and 170+ ...	https://www.reddit.com/r/cybersecurity/comments/1taposq/mass_npm_su...	T3
TanStack & 160+ npm Packages Compromised - Orca Security	https://orca.security/resources/blog/tanstack-npm-supply-chain-worm/	T3

Generated 2026-05-15 13:51 UTC by TJS Security Command Center