INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-14 18:50 UTC

Mini Shai-Hulud Supply Chain Campaign Breaches OpenAI Developer Devices via TanStack and npm/PyPI Compromise

THREAT CAMPAIGN | **HIGH** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0314
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	9.5
Affected Products	TanStack (npm), Mistral AI (npm/PyPI), UiPath (npm/PyPI), Guardrails AI (PyPI), OpenSearch (npm/PyPI), OpenAI desktop apps (macOS, Windows, iOS, Android), GitHub Actions workflows, VS Code, Claude Code, 170+ npm and PyPI packages reported affected
Published	2026-05-14T15:07:24
Discovery Source	Rss

Executive Summary

A threat actor (tracked as TeamPCP by security researchers) compromised over 170 open-source npm and PyPI packages used widely in AI and enterprise software development, including TanStack, Mistral AI, and UiPath. The attack reached two OpenAI employee developer machines, exposing code-signing certificates for OpenAI's macOS, Windows, iOS, and Android desktop applications. OpenAI states no customer data or production systems were breached, but organizations consuming any of the 170+ affected packages face potential credential theft and supply chain compromise of their own build pipelines.

Technical Analysis

A threat actor tracked as TeamPCP compromised over 170 npm and PyPI packages via stolen CI/CD pipeline credentials, using dependency confusion and package hijacking techniques. Affected high-profile packages include TanStack (npm), Mistral AI (npm/PyPI), UiPath (npm/PyPI), Guardrails AI (PyPI), and OpenSearch (npm/PyPI). Attack vectors targeted GitHub Actions workflows and VS Code environments, propagating through automated pipeline execution. Downstream compromise reached two OpenAI developer endpoints, exposing code-signing certificates for macOS, Windows, iOS, and Android OpenAI desktop application builds. OpenAI has set a remediation deadline of 2026-06-12 for macOS users; unpatched versions may fail notarization validation after this date. Relevant CWEs: CWE-522 (Insufficiently Protected Credentials), CWE-798 (Hard-coded Credentials), CWE-829 (Inclusion of Functionality from Untrusted Control Sphere), CWE-1104

(Use of Unmaintained Third-Party Components), CWE-312 (Cleartext Storage of Sensitive Information), CWE-506 (Embedded Malicious Code). MITRE ATT&CK coverage includes T1195.001 (Compromise Software Dependencies and Development Tools), T1195.002 (Compromise Software Supply Chain), T1554 (Compromise Client Software Binary), T1552.004 (Private Keys), T1078 (Valid Accounts), and T1059 (Command and Scripting Interpreter), among others. No CVE has been assigned. Full scope and attribution remain under active investigation. Technical facts regarding package compromise, injection vectors, and OpenAI developer endpoint exposure are corroborated across multiple sources (medium-high confidence). Package inventory scope (170+ packages) and threat actor attribution (TeamPCP) remain unconfirmed by official vendor or law enforcement advisories (medium confidence).

Action Checklist

1. Step 1: Containment, Audit your dependency manifests (package.json, requirements.txt, pyproject.toml) for any of the 170+ confirmed affected packages, including TanStack, Mistral AI (npm/PyPI), UiPath (npm/PyPI), Guardrails AI (PyPI), and OpenSearch (npm/PyPI). Freeze package installs from affected namespaces until clean versions are confirmed. If your pipelines consume these packages, treat all artifacts built since initial compromise date as potentially tainted. Since the exact compromise window is still emerging, assume the broadest plausible timeframe: treat all artifacts built from affected dependencies within the past 90 days as potentially tainted and require re-validation.
2. Step 2: Detection, Audit GitHub Actions workflow logs and CI/CD pipeline execution records for unexpected package resolution changes, new or modified install steps, or outbound connections to unfamiliar hosts during build jobs. Review npm and PyPI audit logs for packages installed from namespaces matching affected projects. Check for credential exfiltration indicators: look for environment variable access patterns in pipeline logs (T1552.001), unexpected secrets manager calls, or anomalous git credential usage (T1552.004). On developer endpoints, inspect VS Code extension activity and recently executed scripts (T1059). SafeDep's published IOC list is the most specific public source available at this time; cross-reference it against your installed package inventory.
3. Step 3: Eradication, Rotate all CI/CD secrets, GitHub Actions tokens, npm publish tokens, and PyPI API keys that were accessible to any affected pipeline or developer environment. Upgrade or pin dependencies to clean, verified versions as maintainers of affected packages release remediated builds; verify package integrity via hash comparison where registries provide it. For developers who executed affected packages locally, perform a full endpoint audit (process logs, file system, registry, installed extensions). If audit is not feasible or high-risk code was executed, rebuild the machine from known-clean media. Remove any packages from internal artifact caches that were pulled during the compromise window.
4. Step 4: Recovery, If your organization uses OpenAI desktop applications on macOS, update to the latest version immediately and no later than 2026-06-12 to avoid certificate invalidation causing app failure. Rebuild affected artifacts in an isolated staging environment with pinned, hash-verified dependencies. Run SBOM generation and dependency audit tools (npm audit, pip-audit, Snyk) to confirm no malicious packages are present before promoting to production. Monitor post-remediation pipeline runs for anomalous network calls, unexpected dependency resolution, or build step modifications. Enable dependency integrity checks (npm audit, pip-audit) as a required gate in CI/CD pipelines going forward.
5. Step 5: Post-Incident, This attack exploited CI/CD credential exposure (CWE-522), unmaintained or loosely pinned third-party dependencies (CWE-1104), and supply chain trust assumptions (CWE-829). Conduct a control gap assessment against NIST SP 800-161 (Supply Chain Risk Management) and

CISA's Secure Software Development guidance. Implement mandatory dependency pinning with hash verification, short-lived CI/CD tokens scoped to minimum privilege, and automated SBOM generation for all build artifacts. Map gaps to MITRE ATT&CK T1195.001 and T1195.002 and assess whether existing detection rules cover supply chain compromise vectors.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO and legal counsel immediately if forensic review of CI/CD pipeline logs or developer endpoint artifacts reveals that secrets exposed to affected packages were used to authenticate to production systems, sign released software artifacts distributed to customers, or access environments containing PII/PHI — any of which may trigger breach notification obligations under GDPR Article 33, CCPA, or applicable state data breach statutes.
Recovery Notes	Prioritize rebuilding all software artifacts — container images, binaries, signed installers — that were produced by pipelines touching the 170+ affected packages during the compromise window, using verified-clean dependencies and freshly rotated signing credentials; any artifact not rebuilt must be treated as potentially backdoored until proven otherwise. Monitor all post-remediation pipeline runs for a minimum of 30 days using network-level egress filtering on CI runners and anomaly detection on build step durations (a postinstall exfiltration hook will add measurable latency to install steps). If your organization distributes software to customers, proactively notify downstream consumers of any artifacts built during the compromise window per your responsible disclosure policy and NIST 800-61r3 §4 post-incident communication guidance.
Forensic Artifacts	npm and PyPI lockfile snapshots (package-lock.json, Pipfile.lock, poetry.lock) from all developer workstations and CI runners capturing the resolved integrity hashes for TanStack, Mistral AI, UiPath, Guardrails AI, and OpenSearch packages installed during the compromise window — these establish which exact package versions executed postinstall hooks on each machine | GitHub Actions workflow run logs (full stdout/stderr including Post Run teardown steps) for all jobs that invoked npm install or pip install against affected namespaces — postinstall-hook-based credential harvesters in this campaign execute during the install phase and leave traces in the runner's job log output and network egress records | macOS Unified Log archives (logarchive format) and ~/.npm/_logs/ npm debug logs from the two known-compromised OpenAI developer machines — these would contain process execution traces of any postinstall scripts that accessed ~/.ssh/, ~/.gitconfig, ~/.npmrc, code-signing keychain entries, or Xcode signing certificates exploited to produce the tainted OpenAI desktop app code-signing certificates | GitHub organization audit log (exportable via API at /orgs/{org}/audit-log) filtered for secrets.access, oauth_application.create, and personal_access_token events during the compromise window — captures T1552.001 environment variable access and T1552.004 git credential harvesting activity by any malicious postinstall script running in the Actions runner context | DNS query logs and network proxy egress logs from CI runner subnets and developer workstation VLANs covering npm install and pip install execution times — TeamPCP supply chain malware communicates with C2 infrastructure during package installation, producing DNS lookups and TCP connections to non-registry hosts that are absent from clean install runs and serve as the primary network-layer IOC for confirming active compromise

Per-Action IR Details

Step 1: Containment — Audit your dependency manifests (package.json, requirements.txt, pyproject.toml) for any of the 170+ confirmed affected packages, including TanStack, Mistral AI (npm/PyPI), UiPath (npm/PyPI), Guardrails AI (PyPI), and OpenSearch (npm/PyPI). Freeze package installs from affected namespaces until clean versions are confirmed. If your pipelines consume these packages, treat all artifacts built since initial compromise date as potentially tainted — scope is still emerging, so treat the boundary conservatively.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST CM-3 (Configuration Change Control), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Run `npm ls --all --json > npm_tree_$(date +%Y%m%d).json` and `pip list --format=json > pip_inventory_$(date +%Y%m%d).json` across all developer workstations and CI runners. Diff against SafeDep's published IOC package list using a simple Python script: `python3 -c "import json,sys; inv=json.load(open('pip_inventory.json')); iocs=['guardrails-ai','mistralai','uipath']; print([p for p in inv if p['name'].lower() in iocs])"`. Block outbound installs at the network perimeter by null-routing registry.npmjs.org and pypi.org at the DNS level for affected namespace prefixes until clean hashes are published. Freeze package.json and requirements.txt via `npm ci` (which honors lockfile exactly) and `pip install --no-deps --require-hashes` to prevent transitive resolution of tainted packages.

Evidence: Before freezing, snapshot the current resolved dependency tree including transitive dependencies — this is your ground truth for scoping which artifacts are tainted. Preserve: (1) `npm-shrinkwrap.json` or `package-lock.json` with resolved integrity hashes for all TanStack, OpenSearch, and Mistral AI npm packages installed; (2) `Pipfile.lock` or `poetry.lock` hash entries for guardrails-ai, mistralai, and uipath PyPI packages; (3) CI/CD build logs showing the exact package version and registry URL resolved during the installation step for each affected package namespace. These lockfile snapshots establish the compromise window boundary and will be needed for post-incident SBOM reconstruction.

Step 2: Detection — Audit GitHub Actions workflow logs and CI/CD pipeline execution records for unexpected package resolution changes, new or modified install steps, or outbound connections to unfamiliar hosts during build jobs. Review npm and PyPI audit logs for packages installed from namespaces matching affected projects. Check for credential exfiltration indicators: look for environment variable access patterns in pipeline logs (T1552.001), unexpected secrets manager calls, or anomalous git credential usage (T1552.004). On developer endpoints, inspect VS Code extension activity and recently executed scripts (T1059). SafeDep's published IOC list is the most specific public source available at this time — cross-reference it against your installed package inventory.

NIST Phase: Detection Analysis

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

Controls: NIST AU-2 (Event Logging), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-12 (Audit Record Generation), NIST SI-4 (System Monitoring), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: For GitHub Actions: use the GH CLI to bulk-pull workflow run logs — `gh run list --limit 500 --json databaseId,conclusion,createdAt | jq '.[].databaseId' | xargs -I{} gh run view {} --log > run_{}.txt` — then grep for TeamPCP IOC hostnames and exfiltration patterns: `grep -E '(curl|wget|Invoke-WebRequest).*(pastebin|ngrok|\.onion|\.tk|discord)' run_*.txt`. For developer endpoints without EDR: deploy Sysmon with SwiftOnSecurity's config and query Event ID 1 (Process Create) for node.exe, python.exe, or pip.exe spawning cmd.exe or PowerShell with encoded args (`-enc` flag). Use osquery on macOS developer machines: `SELECT name, path, cmdline FROM processes WHERE name IN ('node', 'python3', 'pip3') AND cmdline LIKE '%curl%';`. For VS Code extension activity: inspect `~/.vscode/extensions/` modification timestamps against the compromise window and review `~/.vscode/logs/` for extension host crash logs or unexpected network calls.

Evidence: Preserve before any remediation: (1) GitHub Actions workflow run logs for all jobs that installed TanStack, Mistral AI, UiPath, Guardrails AI, or OpenSearch packages — specifically the `Post Run` and teardown steps where

exfiltration payloads commonly execute in compromised npm postinstall hooks (MITRE T1195.001); (2) On macOS developer endpoints known to be compromised, collect `~/Library/Logs/DiagnosticReports/` for crash logs from OpenAI desktop app or VS Code processes, and `~/.npm/_logs/` for npm lifecycle script execution traces showing postinstall hook commands; (3) Network proxy or DNS logs showing outbound connections from CI runner IPs or developer workstation IPs to non-registry hosts during `npm install` or `pip install` execution windows — TeamPCP-style supply chain malware typically beacons to C2 during package installation via postinstall scripts; (4) GitHub Actions secrets access audit log from the organization's audit log API (`GET /orgs/{org}/audit-log?phrase=action:secrets.access`) covering the compromise window, which would capture T1552.001 credential access attempts.

Step 3: Eradication — Rotate all CI/CD secrets, GitHub Actions tokens, npm publish tokens, and PyPI API keys that were accessible to any affected pipeline or developer environment. Upgrade or pin dependencies to clean, verified versions as maintainers of affected packages release remediated builds — verify package integrity via hash comparison where registries provide it. For developers who ran affected packages locally, consider reimaging or fully auditing endpoint state. Remove any packages from internal artifact caches that were pulled during the compromise window.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST IA-5 (Authenticator Management), NIST CM-3 (Configuration Change Control), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Secret rotation without a secrets manager: use `gh secret set` via GitHub CLI to immediately overwrite all Actions secrets — do not delete and recreate, overwrite in place to avoid pipeline downtime. For npm publish tokens: `npm token revoke ` for each token listed under `npm token list`. For PyPI: revoke API tokens at pypi.org/manage/account/token/ for every project in the affected namespaces. Verify package hash integrity without a commercial tool using: `pip download guardrails-ai== --no-deps -d ./verify/ && pip hash ./verify/*.whl` and compare SHA256 output against the hash published in PyPI's JSON API (`curl https://pypi.org/pypi/guardrails-ai//json | python3 -m json.tool | grep sha256`). For internal Artifactory/Nexus/Verdaccio caches: identify tainted artifacts by querying cache metadata for packages installed within the compromise window and purge them — `find ~/.npm -name '*.tgz' -newer -exec rm -v {} \;` as a manual fallback.

Evidence: Before rotating secrets, document the full scope of exposure: extract the list of every GitHub Actions workflow that referenced each secret using `grep -rn 'secrets\.' .github/workflows/ > secrets_usage_map.txt` — this establishes which pipelines, and therefore which build artifacts and downstream systems, must be treated as compromised. Preserve GitHub's organization audit log export showing the last access timestamp and actor for each secret, npm token, and PyPI key in scope. On the two known compromised OpenAI developer endpoints (and any developer machines in your org that ran affected packages), before reimaging collect: macOS Unified Log (`log collect --last 72h --output endpoint_log_$(hostname).logarchive`) and the contents of `~/.ssh/`, `~/.gitconfig`, `~/.npmrc`, and `~/.pypirc` — these credential files are the primary targets of supply chain credential harvesters (T1552.004) and their contents establish whether code-signing certificate material or registry publish credentials were accessible.

Step 4: Recovery — If your organization uses OpenAI desktop applications on macOS, update to the latest version immediately and no later than 2026-06-12 to avoid certificate invalidation causing app failure. Validate clean builds end-to-end in a staging environment before promoting artifacts to production. Monitor post-remediation pipeline runs for anomalous network calls, unexpected dependency resolution, or build step modifications. Enable dependency integrity checks (npm audit, pip-audit) as a required gate in CI/CD pipelines going forward.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CP-10 (System Recovery and Reconstitution), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated

Application Patch Management), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Validate clean builds without enterprise tooling by implementing a two-step gate in CI: (1) add `npm audit --audit-level=high || exit 1` and `pip-audit --requirement requirements.txt --fail-on-vuln` as pre-build steps that fail the pipeline on any known-bad package; (2) add a post-install hash verification step using `cat package-lock.json | python3 -c "import json,sys; data=json.load(sys.stdin); [print(k,v.get('resolved'),v.get('integrity')) for k,v in data.get('packages',{}).items()]" > build_hashes_$(date +%Y%m%d_%H%M%S).txt` and diff against a known-good baseline. For OpenAI desktop app update verification on macOS: `codesign -dv --verbose=4 /Applications/ChatGPT.app 2>&1 | grep -E '(TeamIdentifier|Signature|Authority)'` and confirm the signing certificate chain is current and not the compromised certificate. Monitor post-remediation pipeline runs using Falco (free, open-source) on Linux CI runners to alert on unexpected outbound connections from build processes: `falco -r supply_chain_rules.yaml` with a rule triggering on network activity from node or python processes connecting to non-registry IPs.

Evidence: During recovery validation, generate and preserve a signed SBOM for each clean build artifact using `cyclonedx-npm --output-file sbom_$(git rev-parse --short HEAD).json` or `cyclonedx-py -o sbom_$(date +%Y%m%d).json` — this becomes your integrity baseline for future incident comparison. Capture network traffic from the first post-remediation clean build run using tcpdump on the CI runner (`tcpdump -w clean_build_$(date +%Y%m%d).pcap -i any host registry.npmjs.org or pypi.org`) to establish a known-good DNS resolution and connection baseline. Document the exact versions, SHA256 hashes, and registry URLs for all replaced TanStack, Mistral AI, UiPath, Guardrails AI, and OpenSearch packages as the verified-clean dependency state.

Step 5: Post-Incident — This attack exploited CI/CD credential exposure (CWE-522), unmaintained or loosely pinned third-party dependencies (CWE-1104), and supply chain trust assumptions (CWE-829). Conduct a control gap assessment against NIST SP 800-161 (Supply Chain Risk Management) and CISA's Secure Software Development guidance. Implement mandatory dependency pinning with hash verification, short-lived CI/CD tokens scoped to minimum privilege, and automated SBOM generation for all build artifacts. Map gaps to MITRE ATT&CK T1195.001 and T1195.002 and assess whether existing detection rules cover supply chain compromise vectors.

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 SI-7 (Software, Firmware, and Information Integrity), NIST SA-12 (Supply Chain Protection), NIST RA-3 (Risk Assessment), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: For a 2-person team with no SIEM budget, implement three durable controls: (1) Dependency pinning enforcement — add a pre-commit hook using `pre-commit` framework with the `pip-audit` and `npm-audit` hooks; enforce in CI with `--frozen-lockfile` (yarn) or `--ci` (npm) flags so unpinned installs fail the build; (2) Short-lived token enforcement — replace long-lived GitHub Actions secrets with OIDC-based ephemeral tokens using `permissions: id-token: write` in workflow YAML, eliminating the static credential exposure that TeamPCP exploited; (3) Detection rule coverage — deploy the public Sigma rule `supply_chain_compromise_postinstall.yml` (available in the SigmaHQ repository) to detect npm/pip postinstall hook process spawning shells or making network connections, runnable against Windows Event Logs via `sigma convert -t windows-sysmon supply_chain_rule.yml | Invoke-Expression`. Write a YARA rule targeting malicious postinstall script patterns observed in this campaign (base64-encoded curl/wget payloads in package.json `scripts.postinstall` fields) and scan your npm cache: `yara postinstall_harvest.yar ~/.npm/_cacache/`.

Evidence: For the post-incident lessons-learned record, compile: (1) A complete timeline of package compromise dates versus your first install of each affected package — this gap analysis quantifies your exposure window and supports any regulatory notification decisions; (2) The full list of build artifacts (container images, binaries, published packages) produced during the compromise window that incorporated any of the 170+ tainted packages, with their registry publication URLs and SHA256 hashes — these artifacts may need to be recalled or re-signed; (3) Documentation of which GitHub Actions workflows had access to secrets during the compromise window and whether any of those secrets were subsequently used in production system authentication, code signing, or external service

access — this scopes the lateral movement risk from T1195.001/T1195.002 into production infrastructure. Retain all evidence per NIST AU-11 (Audit Record Retention) for a minimum period consistent with your incident retention policy, typically 1-3 years.

Detection Guidance

Primary detection surface is CI/CD pipeline logs and package manager audit records. Query GitHub Actions logs for runs that installed packages from affected namespaces (TanStack, mistralai, uipath, guardrails-ai, opensearch-project) during the approximate window of the past 90 days. Consult SafeDep's technical advisory for the most current timeline. Look for anomalous outbound HTTP/S calls during build steps, particularly to non-registry endpoints, which may indicate credential exfiltration. On developer endpoints, review shell history, VS Code extension logs, and any recently executed Python or Node scripts for unexpected network activity or file writes to sensitive directories (T1059, T1547). For code-signing exposure: if your organization signs software using certificates obtained from or present on developer machines that ran affected packages, treat those certificates as potentially compromised and initiate revocation review. SafeDep (safedep.io) has published technical analysis with package-level indicators; treat that as your primary IOC reference until official advisories from affected maintainers are published. Behavioral indicators to prioritize: (1) package.lock or requirements.lock changes not associated with a developer commit, (2) CI jobs with new or modified install commands, (3) build artifacts with unexpected file additions or size changes, (4) pipeline jobs accessing secrets outside their defined scope.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`See SafeDep published IOC list at safedep.io/mass-npm-supply-chain-attack-tanstack-mistral`	SafeDep has published package-level indicators. Specific malicious domains and hashes should be pulled directly from their analysis — reproducing unverified values here would risk fabrication.	MEDIUM
URL	`https://safedep.io/mass-npm-supply-chain-attack-tanstack-mistral`	Primary public IOC and technical analysis source for this campaign as of report date. Human validation of current content recommended.	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1552.004** — Private Keys
- **T1588.003** — Code Signing Certificates
- **T1078** — Valid Accounts
- **T1543** — Create or Modify System Process
- **T1195.001** — Compromise Software Dependencies and Development Tools

- **T1098** — Account Manipulation
- **T1554** — Compromise Host Software Binary
- **T1547** — Boot or Logon Autostart Execution
- **T1650** — Acquire Access
- **T1485** — Data Destruction
- **T1552.001** — Credentials In Files
- **T1195.002** — Compromise Software Supply Chain
- **T1078.001** — Default Accounts
- **T1059** — Command and Scripting Interpreter

NIST-800-53R5

- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **SA-4** — Acquisition Process
- **SR-2** — Supply Chain Risk Management Plan
- **SC-13** — Cryptographic Protection

OWASP-TOP10-2021

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

CIS-V8

- **5.2** — Use Unique Passwords
- **16.10** — Apply Secure Design Principles in Application Architectures
- **16.4** — Establish and Manage an Inventory of Third-Party Software Components
- **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

SOC2-TSC

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

NIST-CSF-2

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1552.004	Private Keys	Credential-Access
T1588.003	Code Signing Certificates	Resource-Development
T1078	Valid Accounts	Defense-Evasion
T1543	Create or Modify System Process	Persistence
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1098	Account Manipulation	Persistence
T1554	Compromise Host Software Binary	Persistence
T1547	Boot or Logon Autostart Execution	Persistence
T1650	Acquire Access	Resource-Development
T1485	Data Destruction	Impact
T1552.001	Credentials In Files	Credential-Access
T1195.002	Compromise Software Supply Chain	Initial-Access
T1078.001	Default Accounts	Defense-Evasion
T1059	Command and Scripting Interpreter	Execution

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/openai-confirms-secu...	T3

Source	URL	Tier
Compromised Mistral AI and TanStack packages may have exposed ...	https://www.tomshardware.com/tech-industry/cyber-security/compromis...	T3
TanStack, Mistral AI, UiPath Hit in Fresh Supply Chain Attack	https://www.securityweek.com/tanstack-mistral-ai-uipath-hit-in-fres...	T3
Mass npm Supply Chain Attack Hits TanStack, Mistral AI, and 170+ ...	https://www.reddit.com/r/cybersecurity/comments/1taposq/mass_npm_su...	T3
Mass Supply Chain Attack Hits TanStack, Mistral AI npm and PyPI ...	https://safedep.io/mass-npm-supply-chain-attack-tanstack-mistral	T3

Generated 2026-05-14 18:50 UTC by TJS Security Command Center