INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-09 14:20 UTC

Hades Campaign Escalates PyPI Supply Chain Attack with AI Evasion, Cross-Platform Memory Scraping, and Wiper Capability

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0433
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	PyPI (19 packages, 37 malicious wheel artifacts), GitHub Actions (Linux/macOS/Windows runners), npm, RubyGems, JFrog, CircleCI, Anthropic Claude MCP, OpenAI Codex, Google Gemini, Microsoft Copilot, Cline, Aider, Tabby, Amazon Q, Cody, Bolt, Continue, Pythagora-io/gpt-pilot, AWS, GCP, Azure, Kubernetes, Docker, HashiCorp Vault, SSH
Published	2026-06-09T05:13:32
Discovery Source	Rss

Executive Summary

A threat actor operating as the 'Hades campaign', linked with medium confidence to the Miasma/Shai-Hulud supply chain attack lineage, poisoned 19 PyPI packages across 37 malicious wheel artifacts, deploying credential-stealing and destructive capabilities that target the full modern software development and CI/CD stack. Organizations using Python-based developer tooling, AI coding assistants, and cloud platforms (AWS, GCP, Azure) face credential theft, lateral movement via SSH key harvesting, and potential data destruction via an embedded wiper module. The business risk is severe: a single developer installing one affected package can expose cloud credentials, CI/CD secrets, and source code repositories, with the wiper capability adding irreversible data loss to an already high-impact compromise. **Note: Specific package names require validation from primary source before operational use.**

Technical Analysis

The Hades campaign targets the Python Package Index (PyPI) via 19 compromised packages distributed across 37 malicious wheel artifacts. The primary payload leverages Bun-runtime JavaScript execution, reportedly engineered to trigger during package installation before import-time inspection occurs (source: The Hacker News report; direct verification recommended). Key capability upgrades over prior Miasma/Shai-Hulud iterations include: (1) LLM prompt injection payloads embedded in package metadata to manipulate AI-based

security scanners (CWE-88, CWE-693); (2) cross-platform memory scraping targeting GitHub Actions runners on Linux, macOS, and Windows (CWE-312, T1056.004); (3) backdooring of runtime environments used by popular AI coding assistants, including Anthropic Claude MCP, OpenAI Codex, Google Gemini, Microsoft Copilot, and others (T1176, T1547); (4) SSH key harvesting and lateral movement (T1552.004, T1021.004); (5) a destructive wiper module triggering 'rm -rf' upon detection of a revoked GitHub token, functioning as anti-forensic and punitive capability (T1485); (6) obfuscated packaging to evade static analysis (T1027, T1036); and (7) exfiltration via Codex/AI assistant channels (T1567.001). Credential targets span GitHub, AWS, GCP, Azure, Kubernetes, PyPI, npm, RubyGems, JFrog, CircleCI, Anthropic, Docker, HashiCorp Vault, and SSH (T1552.001, T1555, T1078, T1078.004). Supply chain insertion is via compromised package distribution (T1195.001, CWE-494, CWE-829, CWE-506). No CVE ID is assigned. CVSS base score of 9.5 reflects attack scope (supply chain, multi-platform), integrity/confidentiality impact (credential theft, data destruction), and authentication bypass. Attribution to Hades campaign with Miasma/Shai-Hulud lineage is assessed at MEDIUM confidence (50-70% likelihood based on TTP overlap and infrastructure correlation). The 19 malicious package names are not enumerated in this report; obtain the authoritative list from the primary source before operational deployment. Source: The Hacker News (T3 secondary; human validation of URL and content confirmation recommended before operational use). Recommend obtaining corroboration from CISA, vendor threat intelligence, or primary security research before full operational deployment.

Action Checklist

1. PREREQUISITE: Obtain the authoritative list of 19 malicious package names and 37 wheel artifact hashes from the primary source (The Hacker News report or vendor security advisory). All detection and containment steps reference this list; operational execution cannot proceed without it.
2. Step 1: Containment. Immediately audit all Python packages installed in developer environments, CI/CD pipelines, and container build processes. Cross-reference installed package names and versions against the 19 confirmed malicious PyPI packages from this campaign (exact package list requires human-validated source confirmation via The Hacker News report or a primary vendor advisory). Block installation of unsigned or unverified PyPI packages across the organization. Revoke and rotate all cloud credentials (AWS, GCP, Azure) and CI/CD secrets (GitHub Actions, CircleCI) accessible from any environment where affected packages may have been installed. Disable affected GitHub Actions workflows pending investigation. [NIST IR control family; NIST AC-6, Least Privilege]
3. Step 2: Detection. Audit Python environment package manifests (pip list, requirements.txt, pyproject.toml, poetry.lock) for the 19 malicious package names (confirm list from validated source). Search CI/CD runner logs (GitHub Actions, CircleCI) for unexpected network egress, Bun runtime process execution, or JavaScript execution outside expected workflows. Check SSH authorized_keys files across all hosts for unrecognized entries added after any developer installed Python packages. Review AWS CloudTrail, GCP Audit Logs, and Azure Activity Logs for credential use from unexpected IPs or regions. Inspect AI coding assistant extension directories (Claude MCP, Copilot, Codex, Gemini, Cline, Aider, Tabby, Amazon Q, Cody, Bolt, Continue, Pythagora-io/gpt-pilot) for unauthorized modifications or new files. Behavioral indicators: Bun runtime spawned by Python processes, 'rm -rf' execution in CI environments, unexpected outbound connections from runner hosts. [NIST AU-6, Audit Record Review, Analysis, and Reporting; NIST SI-4, Information System Monitoring; CIS 8.2, Collect Audit Logs]
4. Step 3: Eradication. Remove all identified malicious packages from every affected environment using 'pip uninstall' and verify removal. Rebuild affected CI/CD runner images from known-good base images rather than cleaning in place. Purge and regenerate SSH key pairs on any host where harvesting may

have occurred. Rotate all exposed secrets: GitHub tokens, AWS IAM keys, GCP service account keys, Azure service principals, PyPI API tokens, npm tokens, RubyGems tokens, JFrog credentials, CircleCI environment variables, Anthropic API keys, Docker registry credentials, and HashiCorp Vault tokens. Remove and reinstall AI coding assistant extensions from verified sources only. Enforce package allowlisting or use a private package mirror with vetted packages. [NIST CM control family; D3-CRO, Credential Rotation; D3-CH, Credential Hardening; CIS 2.3, Address Unauthorized Software]

5. Step 4: Recovery. Validate clean environments by running package integrity checks against known-good hashes before restoring CI/CD pipelines. Confirm all rotated credentials are active and former credentials are fully revoked in each platform's IAM or secrets management console. Re-enable GitHub Actions workflows only after runner images are rebuilt and package sources are verified. Monitor cloud environments and CI/CD logs continuously for 72 hours post-remediation for re-infection indicators or credential misuse. Verify SSH authorized_keys files are clean across all affected hosts. Confirm AI coding assistant extensions are at verified versions with no unexpected files. [NIST CP control family; CIS 7.2, Establish and Maintain a Remediation Process; D3-LAM, Local Account Monitoring]

6. Step 5: Post-Incident. This campaign exposed gaps in software supply chain verification, AI-assisted development pipeline trust boundaries, and runtime execution controls for CI/CD environments. Implement package signature verification or a curated private package repository (CIS 2.1, Establish and Maintain a Software Inventory; CIS 2.2, Ensure Authorized Software is Currently Supported). Establish a formal process to vet AI coding assistant extensions before enterprise deployment. Review and tighten GitHub Actions permissions using least-privilege token scopes (NIST AC-6, Least Privilege). Conduct threat hunt for Miasma/Shai-Hulud lineage IOCs across historical logs. Evaluate whether LLM-assisted security scanning is used in your pipeline, if so, assess its susceptibility to prompt injection via package metadata. Brief development and DevOps teams on supply chain attack indicators. [D3-SFA, System File Analysis; D3-UAP, User Account Permissions; CIS 7.1, Establish and Maintain a Vulnerability Management Process]

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and cloud platform security teams immediately if CloudTrail, GCP Audit Logs, or Azure Activity Logs confirm any use of credentials accessible from affected environments — particularly IAM key creation, secrets retrieval from Vault or cloud secrets managers, or container registry pushes — as this constitutes confirmed data exfiltration triggering breach notification assessment under applicable regulations (GDPR 72-hour window, US state breach laws); also escalate if the wiper component is confirmed to have executed (evidenced by mass file deletion events or `rm -rf` in CI runner logs), as this constitutes a destructive incident requiring executive notification and potential law enforcement contact.

Recovery Notes	Before re-enabling any CI/CD pipeline, validate every package in the dependency graph — not just direct dependencies — against SHA-256 hashes from your private mirror, because the Hades campaign targeted transitive supply chain trust and a clean top-level package may still pull a poisoned transitive dependency if version pinning is absent. Monitor rebuilt environments for a minimum of 72 hours with alerts on: new IAM key usage, unexpected Bun or Node process spawns in CI runners, and any modification to AI coding assistant extension directories. Given the Miasma/Shai-Hulud campaign lineage, treat this as a persistent threat actor with demonstrated re-attack capability — conduct a second threat hunt at 30 days post-recovery using updated IOCs from the original advisory source to confirm no re-infection through a different package vector.
Forensic Artifacts	Python package install logs and site-packages directory trees with inode timestamps: `find $(python3 -c 'import site; print(site.getsitepackages()[0])') -maxdepth 2 -type f -newer /tmp/reference_date -ls` — timestamps reveal exactly when the Hades wheel artifact was unpacked and whether post-install hooks ran | Bun runtime binary artifacts in non-standard paths: search `/tmp/`, `/var/tmp/`, `~/.local/bin/`, and `%APPDATA%\Local\Temp\` for ELF/PE binaries matching Bun's signature — Hades drops Bun to execute its cross-platform JavaScript memory scraper stage without requiring a pre-installed Node environment | SSH authorized_keys modification events: `find /home -name authorized_keys -newer -exec stat {} \; -exec cat {} \;` correlated with sshd authentication logs — the Hades campaign's SSH harvester injects attacker-controlled public keys, so new entries added during the compromise window are high-confidence IOCs | Cloud provider credential access logs scoped to CI runner identities: AWS CloudTrail `GetSecretValue` and `AssumeRole` events, GCP `data_access` audit logs for `iam.serviceAccounts.actAs`, and Azure Activity Log `Microsoft.KeyVault/vaults/secrets/read` operations — these reveal whether the Hades memory scraper successfully exfiltrated and then used harvested cloud credentials | AI coding assistant extension directories with file hash inventories: `find ~/.vscode/extensions ~/.cursor/extensions ~/.config/claude ~/.config/cline -type f -exec sha256sum {} \;` compared against vendor-published extension manifests — Hades targets these directories specifically to implant persistence or intercept LLM API keys (Anthropic, OpenAI, Google) stored in extension config files

Per-Action IR Details

Step 1: Containment — Immediately audit all Python packages installed in developer environments, CI/CD pipelines, and container build processes. Cross-reference installed package names and versions against the 19 confirmed malicious PyPI packages from this campaign (exact package list requires human-validated source confirmation via The Hacker News report or a primary vendor advisory). Block installation of unsigned or unverified PyPI packages across the organization. Revoke and rotate all cloud credentials (AWS, GCP, Azure) and CI/CD secrets (GitHub Actions, CircleCI) accessible from any environment where affected packages may have been installed. Disable affected GitHub Actions workflows pending investigation. [NIST IR control family; NIST AC-6 — Least Privilege]

NIST Phase: Containment

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

Controls: NIST AC-6 (Least Privilege), NIST AC-2 (Account Management), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Run `pip list --format=json > pip_inventory_$(hostname).json` on every developer workstation and CI runner to snapshot installed packages before any changes. Use `pip show ` to extract install location and metadata for each suspect package. For CI/CD, download GitHub Actions workflow run logs via the GitHub CLI (`gh run list --limit 50 --json databaseId,conclusion,workflowName` then `gh run view --log`) and archive them before disabling workflows. Block PyPI at the network perimeter using a hosts-file or DNS sinkhole entry for `pypi.org` and `files.pythonhosted.org`

on affected segments immediately — no enterprise proxy required.

Evidence: BEFORE revoking credentials, capture: (1) AWS CloudTrail `LookupEvents` filtered to the last 30 days for the IAM user/role associated with any affected CI runner — look for `AssumeRole`, `GetSecretValue`, `ListBuckets` events from runner IP ranges; (2) GitHub Actions workflow run logs showing the full job execution timeline, especially any `run:` steps that invoke pip install; (3) `pip cache dir` contents and `~/.local/lib/pythonX.X/site-packages//` directory trees including timestamps; (4) `/proc//maps` or Windows handle snapshots for any Python processes active at time of discovery to confirm in-memory loaded modules; (5) `~/.ssh/authorized_keys` with `stat` timestamps before any modification.

Step 2: Detection — Audit Python environment package manifests (pip list, requirements.txt, pyproject.toml, poetry.lock) for the 19 malicious package names (confirm list from validated source). Search CI/CD runner logs (GitHub Actions, CircleCI) for unexpected network egress, Bun runtime process execution, or JavaScript execution outside expected workflows. Check SSH authorized_keys files across all hosts for unrecognized entries added after any developer installed Python packages. Review AWS CloudTrail, GCP Audit Logs, and Azure Activity Logs for credential use from unexpected IPs or regions. Inspect AI coding assistant extension directories (Claude MCP, Copilot, Codex, Gemini, Cline, Aider, Tabby, Amazon Q, Cody, Bolt, Continue, Pythagora-io/gpt-pilot) for unauthorized modifications or new files. Behavioral indicators: Bun runtime spawned by Python processes, 'rm -rf' execution in CI environments, unexpected outbound connections from runner hosts. [NIST AU-6 — Audit Record Review, Analysis, and Reporting; NIST SI-4 referenced by role but not confirmed in knowledge base — no mapped control; CIS 8.2 — Collect Audit Logs]

NIST Phase: Detection Analysis

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

Controls: NIST AU-6 (Audit Record Review, Analysis, And Reporting), NIST AU-2 (Event Logging), NIST AU-12 (Audit Record Generation), CIS 8.2 (Collect Audit Logs)

Compensating: Deploy Sysmon on Windows developer workstations with a config that logs Event ID 1 (Process Create) — hunt for `bun.exe` or `node.exe` spawned as a child of `python.exe` or `pip.exe`, which is the Hades campaign's Bun-runtime-based JS execution stage. On Linux/macOS CI runners, use `auditd` or `execsnoop` (from BCC tools) to trace process ancestry: `execsnoop -u ` filtered for `bun`, `rm`, `curl`, or `wget` children of Python. Write a Sigma rule targeting parent-child relationship: ParentImage endswith 'python3' AND Image endswith 'bun'. For SSH key auditing across many hosts, run: `for h in $(cat hosts.txt); do ssh $h 'stat -c "%n %Y" ~/.ssh/authorized_keys && cat ~/.ssh/authorized_keys'; done > ssh_audit.txt` and compare modification timestamps against the earliest suspected package install date.

Evidence: Capture before analysis is complete: (1) Full `~/.vscode/extensions/`, `~/.cursor/extensions/`, `~/.config/claude/`, and equivalent AI assistant config directories with `find -newer -type f` to surface files modified after malicious package install; (2) Network flow logs or `ss -tunap` / `netstat -ano` output from runner hosts showing outbound connections — Hades exfiltrates to C2 over HTTPS so look for anomalous destination IPs/domains on port 443 from non-browser processes; (3) `poetry.lock` and `pyproject.toml` git history (`git log -p -- poetry.lock`) to identify when the malicious dependency was introduced and by whom; (4) CircleCI build artifact logs and environment variable access logs from the CircleCI audit log API; (5) GCP Audit Logs `data_access` entries for service account key usage (`method: 'google.iam.admin.v1.GetServiceAccountKey'`) from runner-associated service accounts.

Step 3: Eradication — Remove all identified malicious packages from every affected environment using 'pip uninstall' and verify removal. Rebuild affected CI/CD runner images from known-good base images rather than cleaning in place. Purge and regenerate SSH key pairs on any host where harvesting may have occurred. Rotate all exposed secrets: GitHub tokens, AWS IAM keys, GCP service account keys, Azure service principals, PyPI API tokens, npm tokens, RubyGems tokens, JFrog credentials, CircleCI environment variables, Anthropic API keys, Docker registry credentials, and HashiCorp Vault tokens. Remove and reinstall AI coding assistant extensions from verified sources only. Enforce package allowlisting or use a private package mirror with vetted packages. [NIST CM control family; D3-CRO — Credential Rotation; D3-CH — Credential Hardening; CIS 2.3 — Address Unauthorized Software]

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST AC-2 (Account Management), NIST AC-6 (Least Privilege), CIS 2.3 (Address Unauthorized Software), CIS 5.3 (Disable Dormant Accounts), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Do NOT clean in place — Hades campaign packages execute code at install time via `setup.py` or wheel `__init__.py` hooks, so the malicious code may have persisted beyond the package files (dropped payloads, modified `.bashrc`/`.profile`, cron entries, or SSH keys). Instead: snapshot the dirty container/runner image for forensics (`docker commit hades-forensic-image:$(date +%s)`), then destroy and rebuild from a pinned base image hash. For HashiCorp Vault token rotation without enterprise tooling, use `vault token revoke -self` for each affected token and audit with `vault audit list`. Verify SSH key purge with `ssh-keygen -lf ~/.ssh/authorized_keys` post-cleanup and confirm zero unrecognized fingerprints against your known-good key inventory. Use `pip hash` to generate SHA-256 hashes of all packages in your private mirror before deploying: `pip hash --algorithm sha256 `.

Evidence: Before rebuilding runner images, capture disk images or at minimum: (1) All cron entries (`crontab -l` for all users, `/etc/cron*` directories) — Hades or its Miasma lineage predecessors have used cron for persistence; (2) `.bashrc`, `.bash_profile`, `.profile`, `.zshrc` for any eval/curl/exec injections added by the malicious package post-install hook; (3) Full process tree snapshot (`ps auxf` on Linux, `Get-Process` with parent IDs on Windows) at time of discovery to catch any Bun or scraper process still running; (4) Memory dump of any running Python or Bun process using `gcore ` or WinPmem — the cross-platform memory scraper component of Hades may have credentials resident in heap; (5) Docker layer history (`docker history --no-trunc`) to identify which layer introduced the malicious package.

Step 4: Recovery — Validate clean environments by running package integrity checks against known-good hashes before restoring CI/CD pipelines. Confirm all rotated credentials are active and former credentials are fully revoked in each platform's IAM or secrets management console. Re-enable GitHub Actions workflows only after runner images are rebuilt and package sources are verified. Monitor cloud environments and CI/CD logs continuously for 72 hours post-remediation for re-infection indicators or credential misuse. Verify SSH authorized_keys files are clean across all affected hosts. Confirm AI coding assistant extensions are at verified versions with no unexpected files. [NIST CP control family; CIS 7.2 — Establish and Maintain a Remediation Process; D3-LAM — Local Account Monitoring]

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST AU-6 (Audit Record Review, Analysis, And Reporting), NIST AU-11 (Audit Record Retention), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory)

Compensating: For package integrity verification without a paid artifact repository, stand up a local devpi or Nexus OSS instance and populate it only with packages whose SHA-256 wheel hashes you have manually verified against PyPI's JSON API (`https://pypi.org/pypi///json` — verify this URL resolves before use). Validate hash pinning in `requirements.txt` using `pip install --require-hashes -r requirements.txt` — if any hash mismatches, halt pipeline. For 72-hour post-recovery monitoring without a SIEM, configure AWS CloudWatch Log Insights with a saved query for new IAM key usage (`filter eventSource = 'iam.amazonaws.com' | filter eventName in ['CreateAccessKey','AssumeRole']`) and set a CloudWatch Alarm threshold of >0 events to page on-call. For GCP, enable Log-Based Alerts on `data_access` logs for the rotated service accounts.

Evidence: During the 72-hour monitoring window, continuously collect: (1) AWS CloudTrail events for the newly issued IAM keys — any usage of the OLD (rotated) keys after revocation confirms an adversary had harvested and cached them; (2) GitHub Actions workflow run summaries for the first 10 pipeline executions post-recovery, checking for any new unexpected `run:` steps or modified workflow YAML files (Hades could attempt to re-poison via a compromised developer committing a workflow change); (3) `pip audit` output run daily against rebuilt environments to catch any newly published malicious versions of the same package names; (4) SSH authentication logs (`/var/log/auth.log` or `journalctl -u sshd`) for logins using keys that were present during the compromise window; (5) Docker registry push logs to confirm no unauthorized images were pushed to your registry using harvested Docker credentials during or after the incident.

Step 5: Post-Incident — This campaign exposed gaps in software supply chain verification, AI-assisted development pipeline trust boundaries, and runtime execution controls for CI/CD environments. Implement package signature verification or a curated private package repository (CIS 2.1 — Establish and Maintain a Software Inventory; CIS 2.2 — Ensure Authorized Software is Currently Supported). Establish a formal process to vet AI coding assistant extensions before enterprise deployment. Review and tighten GitHub Actions permissions using least-privilege token scopes (NIST AC-6 — Least Privilege). Conduct threat hunt for Miasma/Shai-Hulud lineage IOCs across historical logs. Evaluate whether LLM-assisted security scanning is used in your pipeline — if so, assess its susceptibility to prompt injection via package metadata. Brief development and DevOps teams on supply chain attack indicators. [D3-SFA — System File Analysis; D3-UAP — User Account Permissions; CIS 7.1 — Establish and Maintain a Vulnerability Management Process]

NIST Phase: Post Incident

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

Controls: NIST AC-6 (Least Privilege), NIST AU-6 (Audit Record Review, Analysis, And Reporting), 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), CIS 6.5 (Require MFA for Administrative Access)

Compensating: For the Miasma/Shai-Hulud lineage threat hunt without a commercial threat intel platform: write YARA rules targeting the Bun runtime binary dropped to non-standard paths (`/tmp/`, `/var/tmp/`, `~/.local/bin/`) and scan all developer endpoints using `yara -r /home /tmp /var`. Publish a Sigma rule to your community-shared detection repo targeting `CommandLine contains 'bun' AND ParentCommandLine contains 'python'`. For prompt injection assessment in LLM-integrated CI pipelines (e.g., if using Copilot or Claude MCP for automated code review), manually submit a test package `setup.py` containing a benign prompt injection string (`# IGNORE PREVIOUS INSTRUCTIONS`) to your LLM scanner and verify whether it alters the security verdict — document findings for your post-incident report. Use `pip-audit` (free, from PyPA) as a standing weekly cron job against all `requirements.txt` files in your repos: `find . -name 'requirements*.txt' | xargs pip-audit -r`.

Evidence: For the post-incident review and lessons-learned documentation, preserve: (1) The full timeline reconstructed from Git commit history, CI/CD pipeline logs, and CloudTrail — specifically the delta between when the malicious package was first installed in your environment and when it was detected, to calculate dwell time; (2) A record of every credential that was in scope for rotation with its first-issued and rotation timestamp — this is your evidence of scope for any regulatory breach notification assessment; (3) Any network PCAP or flow data capturing C2 communications from the Hades campaign's exfiltration stage, preserved in a write-once location for potential law enforcement referral; (4) The forensic container image snapshots taken during eradication, retained for a minimum of 90 days per your IR evidence retention policy; (5) AI coding assistant extension directory snapshots from affected developer workstations showing what files were present, their hashes, and modification timestamps — this establishes whether the MCP/Copilot/Gemini attack surface was actively leveraged in your environment.

Detection Guidance

Primary detection surfaces for the Hades campaign: (1) Package inventory: diff installed Python packages against allowlists in all developer workstations, container images, and CI/CD runner environments, flag any of the 19 malicious package names (confirm exact list from validated source). (2) Process telemetry: hunt for Bun runtime ('bun', 'bun.exe') executed as a child process of Python or pip, this is anomalous in most environments and a high-fidelity indicator. (3) CI/CD logs: review GitHub Actions and CircleCI logs for unexpected JavaScript execution, runtime downloads, or network connections to non-standard egress destinations during package install phases. (4) Memory scraping artifacts: on GitHub Actions runners, look for processes reading '/proc/[pid]/mem' (Linux), unusual memory APIs (macOS), or cross-process memory access (Windows) outside expected debugging tools. (5) SSH lateral movement: audit '/root/.ssh/authorized_keys' and all user '~/.ssh/authorized_keys' files for entries added outside your provisioning process; correlate timestamps with package installation events. (6) Cloud credential abuse: query AWS CloudTrail for 'ConsoleLogin' or API calls

from unrecognized source IPs or user agents within 24-48 hours of any Python package install activity; repeat for GCP Audit Logs and Azure Activity Logs. (7) Wiper trigger: look for 'rm -rf' or equivalent destructive shell commands spawned from Python, Node, or Bun processes, treat any such event as an active incident. (8) IDE/extension tampering: check file modification timestamps in AI coding assistant extension directories for changes that do not correspond to authorized updates (D3-SFA, System File Analysis). (9) Prompt injection artifacts: if you use AI-based package scanning, audit scanner outputs for anomalous or contradictory verdicts on recently installed packages, these may indicate successful LLM manipulation. **Source quality note: IOCs (specific package names, hashes, C2 infrastructure) require human validation from the primary source report before operational deployment. All hunt queries require prior acquisition of the authoritative 19-package list from a primary source.**

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://thehackernews.com/2026/06/hades-pypi-attack-19-packages-poisoned.html`	Primary news report on the Hades campaign — specific package names, hashes, and C2 infrastructure should be extracted from this source after human URL validation. URL is T3 tier; confirm it resolves before operational use.	LOW

Framework Mappings

MITRE-ATTACK

- **T1552.001** — Credentials In Files
- **T1021.004** — SSH
- **T1078** — Valid Accounts
- **T1078.004** — Cloud Accounts
- **T1555** — Credentials from Password Stores
- **T1056.004** — Credential API Hooking
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1055** — Process Injection
- **T1552.004** — Private Keys
- **T1059.007** — JavaScript
- **T1547** — Boot or Logon Autostart Execution
- **T1036** — Masquerading
- **T1485** — Data Destruction
- **T1027** — Obfuscated Files or Information
- **T1567.001** — Exfiltration to Code Repository
- **T1176** — Software Extensions

NIST-800-53R5

- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **SC-7** — Boundary Protection
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **CM-7** — Least Functionality
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **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
- **8.2** — Collect Audit Logs

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

ISO-27001-2022

- **A.5.34** — Privacy and protection of personal information
- **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
- **DE.CM-01** — Networks and network services are monitored

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1552.001	Credentials In Files	Credential-Access

Technique ID	Technique Name	Tactic
T1021.004	SSH	Lateral-Movement
T1078	Valid Accounts	Defense-Evasion
T1078.004	Cloud Accounts	Defense-Evasion
T1555	Credentials from Password Stores	Credential-Access
T1056.004	Credential API Hooking	Collection
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1055	Process Injection	Defense-Evasion
T1552.004	Private Keys	Credential-Access
T1059.007	JavaScript	Execution
T1547	Boot or Logon Autostart Execution	Persistence
T1036	Masquerading	Defense-Evasion
T1485	Data Destruction	Impact
T1027	Obfuscated Files or Information	Defense-Evasion
T1567.001	Exfiltration to Code Repository	Exfiltration
T1176	Software Extensions	Persistence

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/06/hades-pypi-attack-19-packages-poi...	T3
Hosting of models for GitHub Copilot	https://docs.github.com/en/copilot/reference/ai-models/model-hosting	T3
37 Vulnerabilities in Google Gemini, OpenAI Codex & More - LinkedIn	https://www.linkedin.com/posts/ugoenyioha_37-vulnerabilities-expose...	T3
3rd Party CLI Agents - Claude Code, OpenAI Codex, and Amazon Q	https://www.youtube.com/watch?v=jQXHDCGlHwM	T3
Supported AI models in GitHub Copilot - GitHub Enterprise Cloud ...	https://docs.github.com/enterprise-cloud@latest/copilot/reference/a...	T3

DISCLAIMER

This intelligence report is produced by Tech Jacks Solutions Security Command Center (SCC) for informational purposes only. It does not constitute professional security advice, legal counsel, or an incident response engagement. The information herein is derived from publicly available sources and AI-assisted analysis; while every effort is made to ensure accuracy, Tech Jacks Solutions makes no warranties regarding completeness or timeliness. Organizations should conduct their own validation and consult qualified security professionals before taking action based on this report. Tech Jacks Solutions is not liable for any damages resulting from the use of this information.

Generated 2026-06-09 14:20 UTC by TJS Security Command Center