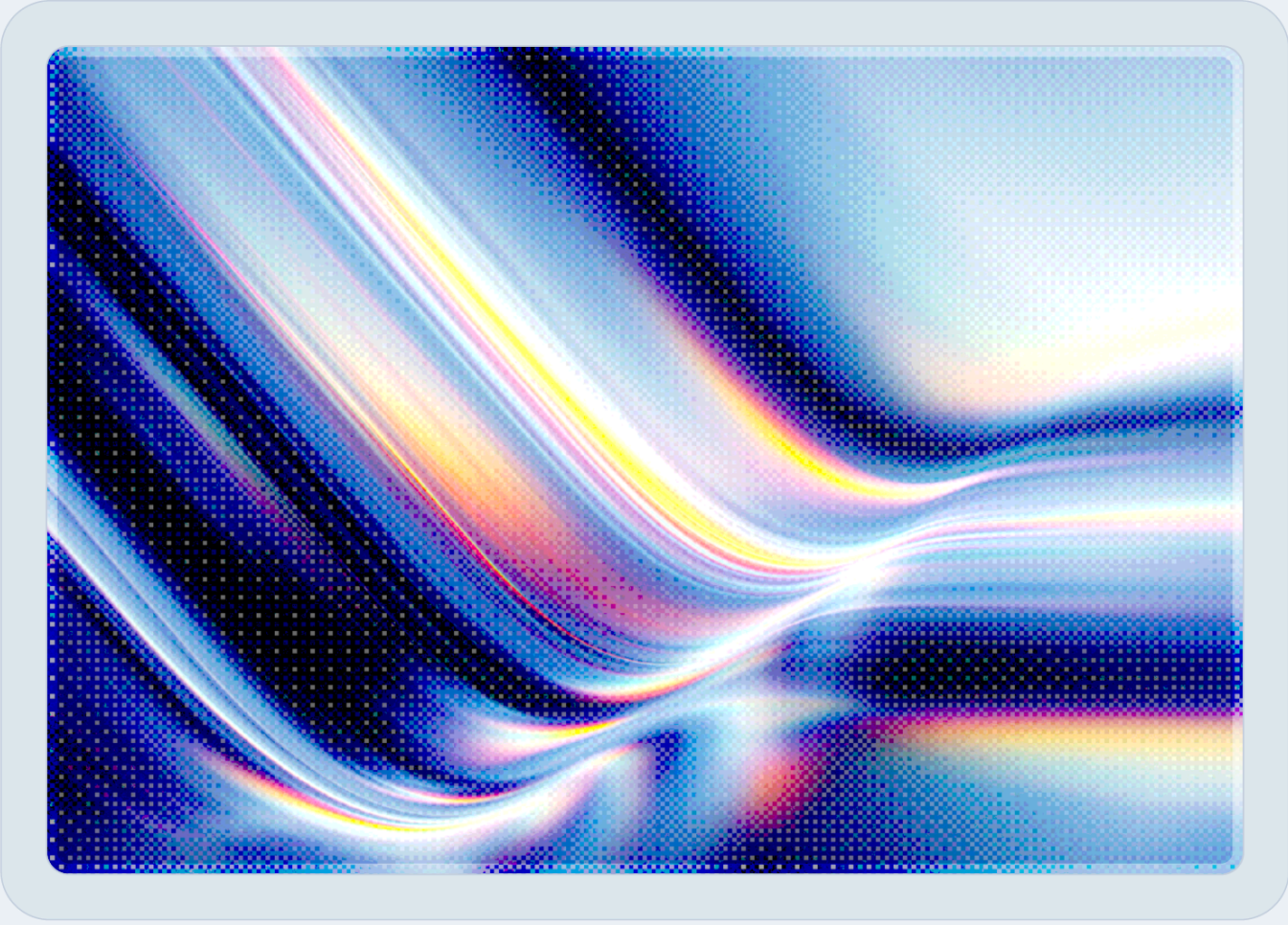
CSAI Foundation | Cloud Security Alliance

Mini Shai-Hulud: Cross-Ecosystem Supply Chain Attack Targets AI Developers

TeamPCP Simultaneously Compromises SAP npm, PyTorch Lightning, and Ruby Gems

2026-05-03

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- Between April 29 and May 1, 2026, a threat actor tracked as TeamPCP executed a coordinated supply chain attack — dubbed "Mini Shai-Hulud" — injecting malicious code simultaneously into official SAP Cloud Application Programming Model npm packages, PyTorch Lightning on PyPI, the `intercom-client` npm package, and Ruby gems, with a combined monthly download reach of more than ten million installs across all affected ecosystems [1][2][6].
- The malware harvests credentials from over 80 file paths — including GitHub tokens, npm authentication tokens, and cloud API keys for AWS, Azure, GCP, and Kubernetes — then exfiltrates stolen material by creating public repositories on the victim's own GitHub account, a technique that evades DLP tools monitoring outbound traffic to unknown destinations [3] [4].
- The attack exploits malicious `preinstall` hooks in `package.json`, using an embedded Bun JavaScript runtime to execute an obfuscated payload at the moment a developer runs `npm install`, requiring no post-install user action and triggering silently in CI/CD pipelines [4][5].
- On Linux CI runners, the payload escalates its credential access by spawning a Python child process that reads `/proc/{pid}/mem` of GitHub Actions Runner workers to extract plaintext secrets directly from process memory — a technique that bypasses environment variable inspection [5].
- The PyTorch Lightning community discovered and quarantined the malicious versions (2.6.2 and 2.6.3) within 42 minutes of publication, crediting automated scanning by Socket's AI-based package monitor [6]. SAP released Security Note 3747787 on April 30 [7]. At time of writing, organizations should treat any environment that installed affected packages between April 29 and May 1 as potentially compromised.

Background

A pattern observed across recent security incident disclosures suggests supply chain attacks have shifted from opportunistic typosquatting toward precision operations that target well-maintained, high-trust packages sustained by legitimate organizations. The campaign now known as Mini Shai-Hulud — named after the Dune science fiction creature, continuing naming conventions established in earlier TeamPCP operations — exemplifies this pattern: a coordinated, cross-registry strike that hit npm, PyPI, and RubyGems within the same 48-hour window.

TeamPCP is assessed by researchers as a financially motivated threat actor group with a documented history of supply chain operations [3]. A predecessor campaign designated "Shai-Hulud 2.0" was documented by Microsoft Security in December 2025, which at that time had affected hundreds of packages [8]. Mini Shai-Hulud builds on that prior infrastructure — attributed, according to Wiz and Snyk researchers, through shared RSA-4096 public keys and encoding routines found in the payloads [3][5] — while tightening the target list to focus on developer ecosystems with direct access to cloud infrastructure and AI/ML pipeline credentials.

Despite its name, the Mini Shai-Hulud campaign's combined reach across affected packages approximates the scale of prior Shai-Hulud operations. By targeting SAP's Cloud Application Programming Model (CAP) npm packages, PyTorch Lightning, and Ruby gems concurrently, TeamPCP demonstrated the capacity to maintain compromised publishing credentials across multiple independent registries simultaneously. The combined monthly download figures for the affected packages exceeded ten million installs at the time of compromise [1].

The selection of PyTorch Lightning is particularly significant for the AI security community. With over 31,100 GitHub stars, the `lightning` package is a primary abstraction layer used by AI and machine learning researchers for model training workflows [6]. Developer environments running PyTorch Lightning frequently contain cloud API keys for GPU clusters and model storage — credential classes that Mini Shai-Hulud was specifically engineered to collect [3][5].

Security Analysis

Attack Mechanism and Delivery

The attack's primary delivery mechanism exploits the `preinstall` lifecycle hook in npm's `package.json` specification. When a developer runs `npm install` on any of the affected SAP CAP packages (`@cap-js/sqlite`, `@cap-js/postgres`, `@cap-js/db-service`, or `mbt`), npm automatically executes the `preinstall` script before the package itself is installed. In the compromised versions, this hook invoked `setup.mjs`, which downloaded a Bun JavaScript runtime binary and used it to execute an obfuscated payload (`execution.js`, approximately 11.6 MB) [4][5].

Using Bun rather than the system Node.js installation is consistent with an intentional evasion strategy: security monitoring agents that instrument Node.js processes will not flag Bun activity [5]. The payload itself uses AES-256-GCM encryption with an embedded RSA-4096 public key, and compresses exfiltrated data with gzip before transmission [5]. This level of operational security suggests a well-resourced threat actor with structured development practices rather than an opportunistic attacker.

For the PyTorch Lightning attack, the malicious versions (2.6.2 and 2.6.3) employed a different entry point: a hidden `_runtime` directory that executed credential-harvesting code at import time, meaning any Python process that ran `import lightning` on the compromised package immediately triggered the malware [6][10]. This technique is notable because it activates in both interactive development sessions and automated training jobs, making it harder to constrain the blast radius once a compromised version enters a shared environment.

Credential Targeting and Exfiltration

The Mini Shai-Hulud payload searches over 80 file paths for authentication material [3]. Its primary targets are GitHub Personal Access Tokens (matching `ghp_` and `gho_` prefixes) and GitHub Actions runner secrets, but it also harvests npm authentication tokens from `.npmrc` files, service account credentials for AWS, Azure, and GCP, Kubernetes configuration files (`~/.kube/config`), SSH private keys, and configuration files belonging to AI coding tools — including Claude Code's `.claude/settings.json` and VS Code task definitions [3][4].

Exfiltration takes place through two channels. The primary channel creates new public repositories on the compromised developer's own GitHub account and pushes the stolen credentials there. This approach is effective at evading DLP controls because outbound traffic to `github.com` is expected in any development environment, bypasses most DLP rules, and gives TeamPCP a credential cache they can retrieve at any time before the victim notices and revokes access. A secondary channel used dedicated infrastructure at `zero.masscan.cloud` for environments where GitHub connectivity was restricted [3].

A behavioral indicator embedded in the payload reveals deliberate geographic restriction: the code checks for Russian-language system locale settings and terminates without executing if detected, a pattern consistent with threat actors operating in jurisdictions where prosecution risk is influenced by domestic victim status [3].

The CI/CD Memory-Scraping Technique

The most technically sophisticated aspect of Mini Shai-Hulud is the Linux-specific memory scraping capability described by Snyk's analysis [5]. On Linux CI runners — the execution environment used by most automated build pipelines — the payload spawns a Python child process that reads `/proc/{pid}/mem` directly from the GitHub Actions Runner.Worker process. This technique extracts secrets that were injected into the runner's process memory by GitHub Actions, bypassing environment variable enumeration and credential file scanning entirely.

This matters because many organizations rely on GitHub Actions' masked secrets feature, believing that secrets injected as environment variables are not readable by untrusted code. The `/proc/mem` technique circumvents this assumption in environments where the runner has not been hardened to restrict process memory access. CI pipelines that install npm dependencies as part of their build process — a pattern common across modern JavaScript and Python workflows — are therefore at risk even if they use GitHub's native secret masking.

The Ruby Gems and Go Modules Wave

Concurrent with the npm and PyPI attacks, security researchers documented a related campaign targeting the RubyGems registry and Go module proxies [9]. Packages published by a GitHub account operating as "BufferZoneCorp" — including `activesupport-logger`, `devise-jwt`, and `config-loader` — were designed as sleeper packages: they appeared functional and benign but contained hooks enabling credential theft and GitHub Actions workflow tampering once installed.

Attribution to TeamPCP's broader infrastructure is indicated by shared encoding patterns and exfiltration endpoints [3][9], though the timing of this wave suggests a parallel operation rather than a strictly simultaneous deployment.

The RubyGems component illustrates an important escalation in supply chain attack sophistication: the use of sleeper packages that remain dormant for days or weeks before activating. Traditional detection approaches that scan packages at install time may pass a package as clean if the malicious behavior is triggered by a runtime condition rather than visible in static analysis. Go module proxy caching further complicates revocation, since cached versions in private proxy servers or CI runner caches may persist even after the upstream module is removed from the public index.

Scale and Impact

SecurityWeek's reporting placed the number of developer repositories containing stolen credentials at over 1,800 at the time of initial disclosure [1]. The combined monthly downloads of the affected packages — approximately 2.5 million across the four SAP CAP packages, plus PyTorch Lightning's 7.9 million monthly PyPI downloads and intercom-client's additional share — mean the exposure window of roughly 48 hours could have touched well over ten million developer environments globally [1][2][6]. The true count of affected installations is likely underreported: developers and CI pipelines that installed the malicious versions and whose credentials were subsequently rotated may not appear in post-incident tallies.

SAP's `mbt` package was identified as the source of the initial maintainer account compromise. Researchers at Snyk traced the breach to the `cloudmtabot` publishing account, which lacked a required approval gate in its publishing workflow — allowing an attacker with temporary access to publish malicious versions without a second-factor confirmation step [5]. This underscores that the attack was not purely a registry-level vulnerability: it exploited a process gap in how package publishing permissions were governed.

Recommendations

Immediate Actions

Organizations should audit their dependency installation logs from April 29 through May 1, 2026 for installs of the following packages at affected versions: `@cap-js/sqlite`, `@cap-js/postgres`, `@cap-js/db-service`, and `mbt` (any version published in this window), `lightning` versions

2.6.2 or 2.6.3, `intercom-client` versions 7.0.4 or 7.0.5, and any Ruby gems matching the BufferZoneCorp publisher account [1][2][7].

Any environment that installed the affected packages should be treated as compromised until credentials are rotated. This means revoking and regenerating all GitHub Personal Access Tokens, npm tokens, GitHub Actions secrets, and cloud service account keys that were present in the developer's environment at the time of installation. Cloud access logs should be audited for anomalous API calls following the installation window — particularly for unexpected repository creation events on GitHub accounts belonging to affected developers.

- Upgrade SAP CAP packages to versions published after April 30, 2026 and confirmed clean against SAP Security Note 3747787 [7]
- Upgrade PyTorch Lightning to version 2.6.4 or later (or revert to 2.6.1, the last confirmed clean version)
- Remove and audit any Go modules or Ruby gems installed from the BufferZoneCorp publisher

Short-Term Mitigations

CI/CD pipelines should be configured to pin exact package versions using lock files (`package-lock.json`, `poetry.lock`, `Gemfile.lock`) and to verify package integrity against published checksums before installation. Continuous Integration environments should use ephemeral, immutable runners that are destroyed after each job rather than persistent runners where a compromised environment can persist across builds.

For npm specifically, organizations should review whether `preinstall` and `postinstall` lifecycle scripts are necessary for their dependencies, and consider using `npm install --ignore-scripts` in CI contexts where lifecycle scripts are not required. This eliminates the primary delivery mechanism Mini Shai-Hulud used for the SAP npm attack vector, though it requires testing to ensure legitimate build tooling is not disrupted.

Linux CI runner hardening should include restricting cross-process memory reads by ensuring runners do not run as root and by applying Linux Security Module policies (AppArmor or SELinux) that block access to `/proc/{pid}/mem` by processes that did not create the target process. GitHub's security hardening documentation recommends using Actions runner isolation at the job level for untrusted dependency installations [11].

Strategic Considerations

Mini Shai-Hulud demonstrates that the AI and ML development ecosystem is now a deliberate target for credential theft, not merely a collateral victim. Organizations building AI systems should treat their development environments — including local developer workstations, CI pipelines, and model training infrastructure — with the same credential hygiene standards applied to production systems. The presence of cloud API keys granting access to GPU clusters and model storage in developer environments creates a high-value target that financially motivated actors will continue to exploit.

Registry-level controls are necessary but insufficient. The 42-minute detection-to-quarantine window for the PyTorch Lightning attack — achieved through Socket's automated scanning infrastructure — demonstrates what automated package monitoring can achieve at scale, yet many organizations will have already installed the malicious versions in that window. A defense-in-depth approach requires combining registry-level scanning (automated tools watching for new versions of trusted packages), pipeline-level isolation (ephemeral runners, network egress restrictions, banned lifecycle scripts for untrusted packages), and credential-level controls (short-lived tokens, OIDC-based cloud authentication that avoids static API keys in CI environments).

Supply chain security programs should extend Software Bill of Materials (SBOM) practices beyond direct dependencies to include transitive dependencies and the publishing account governance practices of upstream maintainers. The `mbt` compromise originated from a missing publishing approval gate — a process control gap that no SBOM or vulnerability scanner would detect. Vendor risk assessments for open-source dependencies should include questions about maintainer account security practices, multi-party publishing approval workflows, and response time commitments for security incidents.

CSA Resource Alignment

The Mini Shai-Hulud campaign maps directly to multiple CSA frameworks and controls that provide organizations with structured guidance for addressing the risks it exposes.

CSA's AI Controls Matrix (AICM), as a superset of the Cloud Controls Matrix, addresses software supply chain risk through controls requiring software composition analysis, SBOM maintenance, and third-party component verification. The AICM's controls on AI system integrity — specifically requirements for verifying the provenance and integrity of training frameworks and inference libraries — apply directly to the PyTorch Lightning compromise, where a widely-used ML framework became a credential-theft vehicle.

CSA's MAESTRO framework for agentic AI threat modeling identifies the AI development pipeline itself as a distinct attack surface, separate from the deployed AI system. Mini Shai-Hulud attacks precisely this layer: not the model, but the development environment in which models are built and trained. MAESTRO's Layer 2 (Data Operations) and Layer 3 (Infrastructure) threat categories cover supply chain injection into training pipelines, and organizations using MAESTRO for threat modeling should include their dependency installation workflows as an explicit attack surface.

The STAR program's third-party risk assessment questionnaire provides a mechanism for organizations to evaluate open-source dependency governance as part of their supplier risk management practice. While STAR was designed primarily for cloud service providers, its principles around access control documentation and incident response SLAs apply to the maintainer account governance gap that enabled the SAP CAP package compromise.

CSA's Zero Trust guidance emphasizes that credentials — including developer tokens and CI/CD pipeline secrets — should be short-lived, scoped to minimum necessary permissions, and issued through federated identity mechanisms wherever possible. Environments using OIDC-based short-lived cloud credentials would have limited the stolen cloud API key exposure, since OIDC tokens cannot be reused after expiry — though GitHub PATs and other long-lived tokens would remain at risk even in a successful compromise.

References

[1] SecurityWeek. "1,800 Hit in Mini Shai-Hulud Attack on SAP, Lightning, Intercom." SecurityWeek, May 1, 2026.

[2] The Hacker News. "SAP-Related npm Packages Compromised in Credential-Stealing Supply Chain Attack." The Hacker News, April 29, 2026.

[3] Wiz. "Mini Shai-Hulud: Supply Chain Campaign Targets SAP npm Packages with Credential-Stealing Malware." Wiz Security Research, April 29, 2026.

[4] Sophos. "Mini Shai-Hulud Supply Chain Attack Targets SAP npm Packages." Sophos X-Ops, April 29, 2026.

[5] Snyk. "'A Mini Shai-Hulud Has Appeared': Bun-Based Stealer Hits SAP @cap-js and mbt npm Packages." Snyk Security Research, April 29, 2026.

[6] Lightning.ai. "How the PyTorch Lightning Community Discovered a Supply Chain Attack and Fixed It in 42 Minutes." Lightning.ai Blog, April 30, 2026.

[7] Onapsis. "Emerging Supply Chain Attack (Mini Shai-Hulud) Targeting SAP Cloud Application Programming Ecosystem." Onapsis Research Labs, April 29, 2026. *SAP Security Note 3747787 is published on SAP's support portal (support.sap.com), which requires authentication; this Onapsis analysis provides a publicly accessible secondary source for the note number and date.*

[8] Microsoft Security. "Shai-Hulud 2.0: Guidance for Detecting, Investigating, and Defending Against the Supply Chain Attack." Microsoft Security Blog, December 9, 2025.

[9] The Hacker News. "Poisoned Ruby Gems and Go Modules Exploit CI Pipelines." The Hacker News, May 1, 2026.

[10] The Hacker News. "PyTorch Lightning and Intercom-client Hit in Supply Chain Attacks to Steal Credentials." The Hacker News, April 30, 2026.

[11] GitHub. "Security Hardening for GitHub Actions." GitHub Documentation, 2026.