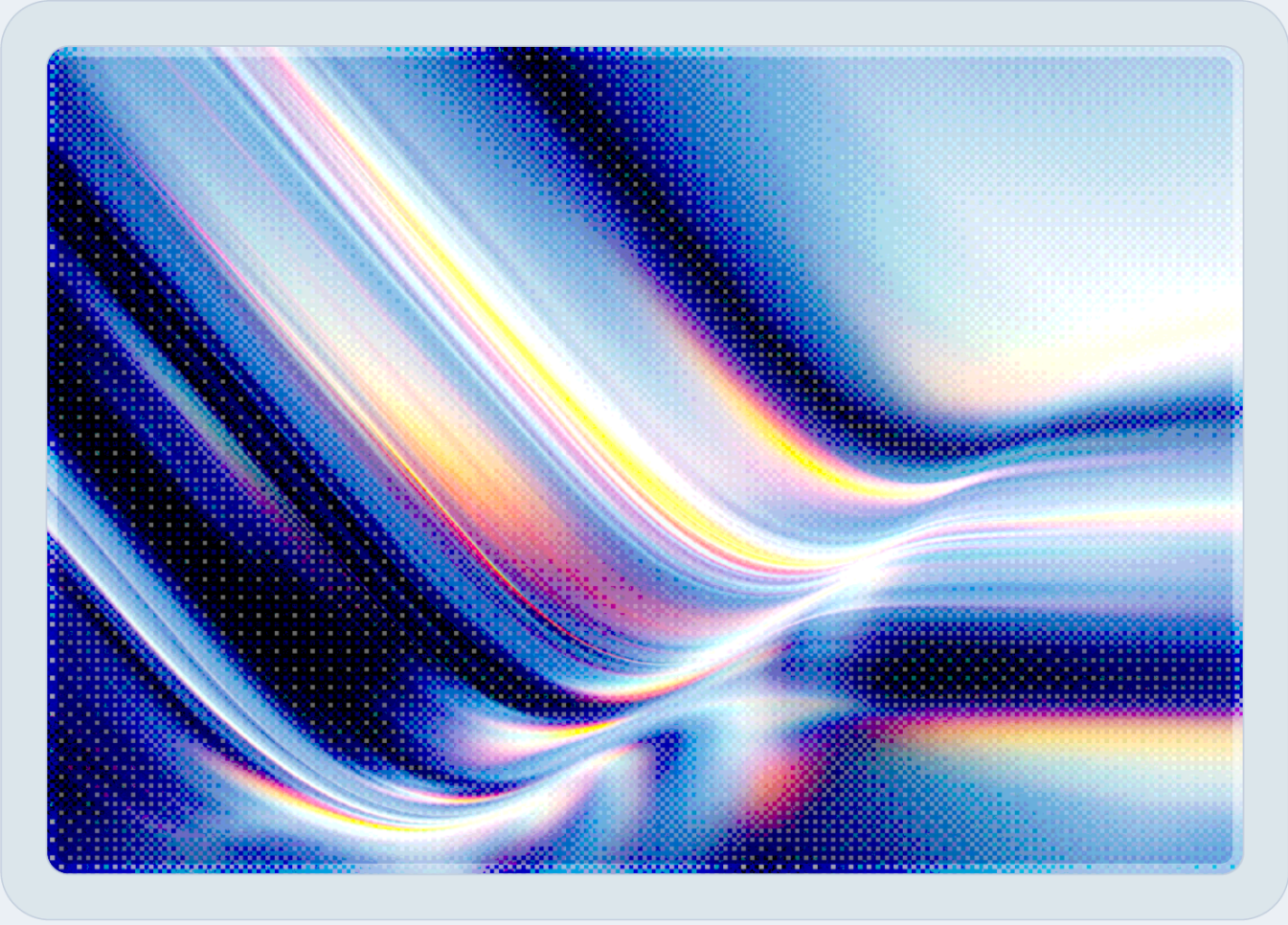
CSAI Foundation | Cloud Security Alliance

ShaiWorm: ML Framework Backdoor in PyTorch Lightning

Compromised PyPI Releases 2.6.2 and 2.6.3 Deploy Multi-Stage Credential-Harvesting Worm

2026-05-05

AI-assisted Rapid Research



CSAI | CSA cloud security alliance®

Key Takeaways

- On April 30, 2026, malicious versions 2.6.2 and 2.6.3 of the `lightning` package — the official PyPI distribution for PyTorch Lightning — were published to PyPI with a multi-stage credential-harvesting payload. Socket's research team detected the compromise approximately 18 minutes after publication; the packages remained available for approximately 42 minutes before quarantine [1][2][6][7]. Version 2.6.1, released January 30, 2026, is the last known clean release.

- The malware, dubbed ShaiWorm by BleepingComputer, activates on `import lightning` rather than at install time, executing a hidden `_runtime` module that downloads the Bun JavaScript runtime and runs an approximately 11.4 MB obfuscated payload. The payload targets cloud credentials (AWS, Azure, GCP), GitHub and npm tokens, browser credentials across Chrome, Firefox, and Brave, SSH keys, Kubernetes configurations, Docker credentials, CI/CD environment variables across numerous platforms, and cryptocurrency wallet files [1][2][3].

- ShaiWorm contains a self-propagating npm worm component: if it locates npm publish credentials, it injects a malicious `postinstall` hook into locally discoverable npm packages, bumps their patch version, and republishes them — extending the infection chain beyond Python into the JavaScript ecosystem [1][3]. Stolen GitHub tokens are additionally used to commit backdoor files to repositories, with commits authored to impersonate `claude@users.noreply.github.com` [1][2].

- Snyk published advisory SNYK-PYTHON-LIGHTNING-16323121 (CVSS 9.3, CWE-506) for the compromised versions [3]. Any environment where `lightning` 2.6.2 or 2.6.3 was installed should be treated as potentially compromised and subjected to immediate credential rotation, regardless of whether `import lightning` was confirmed to have executed.

- Attribution remains contested across vendors. Kodem Security and Aikido link the campaign to a threat actor cluster they call "Mini Shai-Hulud," previously associated with attacks on SAP-related npm packages [4][5]. A group calling itself "Team PCP" claimed responsibility via a Tor onion link posted to the public GitHub issue thread for the attack [1][9]. The contested attribution should not delay response, as the technical indicators and remediation steps are consistent across all analyses.

Background

PyTorch Lightning – distributed as the `lightning` package on PyPI – is one of the most widely adopted training frameworks in the machine learning community. It provides a structured abstraction over raw PyTorch that accelerates model development across research and production workflows. The package receives approximately 11 million downloads per month, making it a high-value target for supply chain attacks: a brief window of malicious availability translates directly into a large pool of potentially compromised developer environments [1][10].

The attack followed a pattern that the security community has documented with increasing frequency across both traditional software ecosystems and AI-specific tooling. Rather than creating a lookalike typosquatting package – the technique that characterized earlier waves of supply chain poisoning on PyPI – the attacker obtained publishing credentials for the legitimate package and pushed malicious builds directly as new releases of the official `lightning` package. Evidence examined by Socket's team points to compromise of the `pl-ghost` GitHub account, a service account used by the Lightning-AI project for automated PyPI publishing [1][9]. The Lightning-AI maintainers confirmed they were investigating how publishing credentials were obtained and audited surrounding releases for similar payloads [6].

The infection executes entirely after a successful `pip install lightning`, without triggering any warning at install time. Conventional security guidance that instructs developers to inspect packages for suspicious `setup.py` or post-install scripts does not protect against this technique: the malicious payload is loaded as a hidden submodule triggered only when the developer's code calls `import lightning`. This characteristic means the attack surface includes any developer who installed a vulnerable version and subsequently ran their training or experimentation code – a step that may occur hours, days, or weeks after installation, depending on how frequently an environment is used.

The campaign's timing and target profile are consistent with deliberate targeting of the ML development community. ML practitioners in production engineering environments frequently operate in environments that concentrate the credential categories ShaiWorm targets: cloud compute credentials for GPU-backed training, GitHub tokens for dataset and checkpoint management, environment variable files containing API keys for model providers, and CI/CD platform tokens for automated training pipelines. Compromising any of these would provide the attacker with access to cloud infrastructure, intellectual property in the form of model weights and training data, and potentially a lateral movement path into the broader software projects where ML researchers contribute.

Security Analysis

Stage-by-Stage Execution Chain

ShaiWorm's multi-stage architecture separates the initial delivery vehicle from the primary payload in a way that complicates both static detection and post-incident reconstruction. The malicious wheels contained a hidden `_runtime` directory absent from the legitimate 2.6.1 release. When the infected package is imported, a Python module within that directory immediately fetches Bun — a legitimate JavaScript runtime maintained by Oven — directly from GitHub releases as version 1.3.13 [2][3]. This use of a trusted, signed binary download as a staging mechanism appears calculated to evade network monitoring: tools that allowlist traffic to GitHub may not flag the initial retrieval, and endpoint security tools will encounter a legitimately signed runtime binary rather than recognizable malware.

Once Bun is in place, the `_runtime` module executes the primary JavaScript payload: `router_runtime.js`, a file of approximately 11.4 MB that Socket and Snyk describe as carrying double-layer obfuscation [1][3][8]. The outer layer employs a "javascript-obfuscator"-style encoding; the inner layer uses a custom routine named `__decodeScrambled()` that applies PBKDF2 with SHA-256 across 200,000 iterations against a hardcoded salt (`ctf-scramble-v2`) — a naming artifact that may indicate the obfuscation tooling was adapted from a CTF-context prototype, though this is speculative without additional context — followed by gzip decompression of base64-encoded string blobs. This level of obfuscation complexity is not consistent with automated tooling applied opportunistically; it reflects deliberate investment in detection evasion.

The credential harvesting phase targets a broad and prioritized set of credential categories. The payload invokes `gh auth token` to retrieve any GitHub CLI tokens, scans file system paths for `.env` files and npm configuration, queries AWS Instance Metadata Service (IMDS) endpoints to extract EC2 instance role credentials, reads Azure and GCP credential files from standard locations, and enumerates environment variables across numerous CI/CD platform formats including GitHub Actions, GitLab CI, CircleCI, and similar systems [1][2][3]. Browser credential extraction targets stored passwords and session cookies in Chrome, Firefox, and Brave profiles. Notably, the payload validates stolen credentials before exfiltration — it checks whether captured tokens are live rather than transmitting expired credentials — indicating that the payload was designed to prioritize high-fidelity, live credentials over raw data volume.

Cross-Ecosystem Worm Propagation

The npm self-propagation component elevates ShaiWorm from a credential stealer to an active worm in a meaningful operational sense. Upon locating npm publish credentials in `~/.npmrc` or equivalent locations, the payload searches for local `.tgz` npm package tarballs, injects a `postinstall` hook referencing a `setup.mjs` dropper and `router_runtime.js`, increments the patch version number, and republishes the modified package to the npm registry under the original package identity [1] [3]. The effect is that a single developer environment can become an unwitting distribution node, propagating the infection to any downstream user who installs the now-contaminated npm package version.

This cross-ecosystem propagation design is notable for several reasons. It is consistent with awareness that ML developers frequently work across both Python and JavaScript tooling — common in environments that combine PyTorch-based training with Node.js-based serving or tooling layers. It also creates a secondary infection surface that is harder to trace back to the original PyPI compromise, because the npm packages carrying ShaiWorm will not show PyTorch Lightning in their dependency tree. Organizations performing post-incident triage must therefore account for npm packages they publish or consume, not just their Python dependency graphs.

Repository Poisoning via Stolen GitHub Tokens

ShaiWorm's use of stolen GitHub tokens for repository poisoning adds a persistence and lateral movement dimension beyond immediate credential exfiltration. The payload, using a token retrieved from the local environment, pushes backdoor files to the victim's GitHub repositories — targeting multiple branches per repository — and creates commits attributed to `claude@users.noreply.github.com` [1][2]. The email `claude@users.noreply.github.com` matches the format used by Claude Code's automated GitHub operations. In environments where AI-assisted development is common, developers reviewing commit logs may attribute these commits to their own tooling rather than recognizing them as unauthorized — an ambiguity the attacker's choice of commit identity appears designed to create.

The backdoor files are placed in `.claude/`, `.vscode/`, and `.github/workflows/` directories — locations that persist across subsequent development sessions and affect any collaborator who clones or pulls from the poisoned repository [1]. A compromised `.github/workflows/` directory is particularly significant because GitHub Actions workflows execute in CI/CD contexts that may have access to elevated repository secrets and deployment credentials, extending the attacker's reach beyond the initially compromised developer workstation.

Detection Challenges and the Import-Time Activation Pattern

The activation timing — triggering on `import lightning` rather than on `pip install` — suggests awareness of how developer security workflows operate in practice. Post-install script execution is logged and inspectable; import-time module execution occurs silently within the Python interpreter during routine code execution. A developer who installs the malicious version on a Friday, runs no ML code until Monday, and then executes a training script will trigger the malware at a point in time behaviorally disconnected from the package installation — reducing the likelihood that the developer associates any anomalous system behavior with the installation event.

This pattern also means that static analysis of the installed wheel's top-level scripts is insufficient for detection. The malicious `_runtime` directory must be identified through a comparison of the installed package contents against the known-good manifest for version 2.6.1, or through behavioral monitoring that observes unexpected network connections and file system access during a training run. The 42-minute window before PyPI quarantine, combined with package manager caches and offline environments, means that some number of installations will have occurred after the packages' technical removal from the index.

Recommendations

Immediate Actions

Any environment where `lightning` version 2.6.2 or 2.6.3 was installed — whether or not `import lightning` was explicitly executed — should be treated as potentially compromised. The threshold for initiating credential rotation should be installation, not confirmed import execution, because logs of import-time events may be incomplete and the downside risk of delayed rotation is disproportionately high. Organizations should immediately rotate GitHub tokens, npm tokens, AWS access keys and IAM role credentials, Azure service principal credentials, GCP service account keys, API keys stored in environment files or CI/CD platform secrets, and SSH keys accessible from the affected environment [1][2][3].

GitHub repositories accessible from any compromised developer environment should be audited for unauthorized commits, with particular attention to commits attributed to `claude@users.noreply.github.com` or other unexpected automated identities added after April 30, 2026 [1][2]. Any `.github/workflows/` files modified in this window should be reviewed

for injected payload retrieval steps before any CI/CD pipeline execution resumes. Similarly, any npm packages published from a potentially compromised environment since April 30 should be reviewed for injected `postinstall` hooks in `package.json` and unexpected files in the package tarball.

- Downgrade `lightning` to version 2.6.1 or upgrade to any release newer than 2.6.3 that Lightning-AI has confirmed as clean following their internal audit [6].
- Search installed packages for the hidden `_runtime` directory: `find $(pip show lightning | grep Location | cut -d' ' -f2)/lightning -name '_runtime' -type d`.
- Inspect outbound network connections from ML training environments around the time the malicious versions were in use for connections to unexpected hosts or to GitHub releases for Bun downloads.

Short-Term Mitigations

Organizations that publish packages to PyPI or npm should immediately audit their publishing workflows for credential exposure. The Lightning-AI compromise appears to have originated from a service account credential – `pl-ghost` – that was accessible outside of a tightly controlled secret management environment [1]. Publishing credentials should be scoped using PyPI's Trusted Publisher mechanism, which uses OIDC identity federation to grant temporary upload tokens to specific GitHub Actions workflows without storing long-lived API keys in repository secrets or CI/CD configuration. This substantially reduces the long-lived credential artifact that ShaiWorm targets, replacing static API keys with short-lived OIDC tokens scoped to specific workflow identities. It does not eliminate credential risk entirely, particularly if an attacker has already gained the ability to modify CI/CD workflow files.

For ML development environments, the import-time activation of ShaiWorm points to the value of process-level behavioral monitoring that operates independently of installation-time scanning. Tools that observe file system access, subprocess spawning, and network connections during the execution of Python scripts can identify the anomalous pattern – downloading an external binary and executing obfuscated JavaScript from within a `lightning` import – regardless of the obfuscation applied to the payload. This class of monitoring is especially important in shared or multi-tenant ML training clusters, where a single compromised dependency imported in a training job can harvest credentials from the cluster's instance metadata service.

Package pinning and hash verification should be standard practice in any ML project that depends on packages with high download volume. Specifying `lightning==2.6.1` in `requirements.txt` is a start; adding the SHA256 hash of the known-good wheel in a lock file or pip-tools–managed

requirements provides a verification step that a compromised version cannot pass regardless of its version number. Projects using `pip-compile`, Poetry, or Pipenv should verify that their lock files were generated from verified sources and have not been tampered with.

Strategic Considerations

The PyTorch Lightning compromise follows from the same structural dynamic that has produced supply chain attacks across npm, PyPI, RubyGems, and now AI-native tooling: the implicit trust that developers extend to packages from well-known, high-reputation projects, and the concentration of that trust in a small number of publishing credentials that become high-value targets. The attack did not exploit a vulnerability in PyPI's infrastructure; it exploited the trust model — the assumption that because `lightning` 2.6.1 was safe, `lightning` 2.6.2 would be safe too, and that the package index reliably enforces this continuity.

The self-propagating npm component signals a maturation in attacker tradecraft: campaigns are no longer designed to exploit a single ecosystem but to leverage the polyglot nature of modern development environments. ML practitioners, data engineers, and applied AI developers routinely work across Python, JavaScript, and container-based tooling. A payload that propagates across ecosystem boundaries is harder to contain and harder to attribute because the npm infection can be discovered independently of the original Python compromise. Security teams conducting incident response on npm supply chain events should consider whether the upstream compromise was initiated through a Python dependency rather than a JavaScript one.

Platform operators at PyPI and npm face increasing pressure to reduce the window between malicious publication and quarantine. An 18-minute detection-to-alert time by Socket's automated analysis is notable; the 42-minute total window before removal still represents a meaningful exposure period given the download volumes involved [1][2][6][7]. Mandatory two-factor authentication for PyPI publishers — which PyPI began requiring for critical packages in 2023 and has progressively expanded [11] — reduces the risk of credential compromise but does not eliminate it if the compromise occurs upstream of the publishing workflow. Trusted Publisher OIDC federation, cryptographic signing of package releases using the Sigstore ecosystem, and faster automated quarantine pipelines for flagged releases are platform-level controls directly relevant to preventing similar events.

CSA Resource Alignment

The PyTorch Lightning supply chain compromise maps to multiple dimensions of CSA's AI and cloud security frameworks, each of which addresses a distinct aspect of the attack surface the campaign exploited.

CSA's AI Controls Matrix (AICM) [12] treats software supply chain security as a cross-cutting concern spanning the model provider, application provider, and infrastructure domains. The ShaiWorm campaign specifically violates AICM controls requiring provenance verification for third-party components, integrity validation of runtime dependencies, and credential management discipline in automated deployment workflows. Organizations applying the AICM to their ML engineering practices should interpret this incident as a test case for the controls governing AI training infrastructure: how are dependencies for training pipelines reviewed, locked, and verified? How are the credentials used by those pipelines scoped and rotated? The answers to these questions determine whether a similar compromise would be detected or would remain latent until credentials are abused.

CSA's MAESTRO framework for agentic AI threat modeling [13] identifies the development toolchain and its dependencies as a threat surface at Layer 1 (Model and Inference Infrastructure). A backdoored training framework is a MAESTRO Layer 1 threat because it compromises the environment in which models are built — with potential consequences that extend into the produced model artifacts if, for example, a dataset or checkpoint written by a compromised environment is later loaded in a clean environment. The cross-repository backdoor injection ShaiWorm performs through stolen GitHub tokens represents a MAESTRO Layer 3 (Infrastructure and Orchestration) threat: by modifying CI/CD workflow files, the attacker positions for persistent code execution in automated environments that may have access to production model serving infrastructure.

The STAR program's supply chain security questionnaire provides a structured basis for evaluating ML framework vendors and PyPI publishers against the expectations this incident establishes. Security teams performing vendor risk assessments on ML tooling providers should ask specifically about publishing credential management practices, whether Trusted Publisher OIDC federation is in use, what monitoring exists for unauthorized or anomalous package releases, and what the incident response process is for supply chain compromises — including the expected time to quarantine and the communication commitments to affected users.

CSA's Zero Trust guidance applies to the trust assumption that ShaiWorm exploited at the developer-to-package-registry boundary. Treating a versioned package from a well-known project as inherently trusted because previous versions were safe is precisely the implicit trust model that Zero Trust principles reject. Operationally, Zero Trust for dependency consumption means: verify the integrity of every dependency at every update, scope the credentials accessible from dependency execution

environments to the minimum necessary for the task, and monitor for behavioral deviations from the expected execution profile regardless of the dependency's stated provenance. The inability of import-time malware to exfiltrate credentials it cannot reach is the most reliable mitigation available when detection cannot be guaranteed before execution.

References

[1] Socket Research Team. "Lightning PyPI Package Compromised." Socket.dev, April 2026.

[2] BleepingComputer. "Backdoored PyTorch Lightning package drops credential stealer." BleepingComputer, April 2026.

[3] Snyk Security Research. "Lightning PyPI Compromise: Bun-Based Credential Stealer." Snyk, April 2026.

[4] Kodem Security. "Mini Shai-Hulud Strikes: PyTorch Lightning and intercom-client Inside the Cross-Ecosystem Supply Chain Attack." Kodem Security, April 2026.

[5] Aikido Security. "Popular PyTorch Lightning Package Compromised by Mini Shai-Hulud." Aikido, April 2026.

[6] Lightning-AI. "PyTorch Lightning Supply Chain Attack — Official Response." Lightning.ai, April 2026.

[7] The Hacker News. "PyTorch Lightning and Intercom-client Hit in Supply Chain Attacks." The Hacker News, April 2026.

[8] Semgrep. "Malicious Dependency in PyTorch Lightning Used for AI Training." Semgrep, April 2026.

[9] GitHub. "Possible supply chain attack on version 2.6.3 (Issue #21689)." Lightning-AI/pytorch-lightning, April 2026.

[10] Sonatype. "Malicious PyTorch Lightning Packages Found on PyPI." Sonatype Blog, April 2026.

[11] PyPI Security Team. "2FA Enforcement for Critical Projects." PyPI Blog, December 2023.

[12] Cloud Security Alliance. "AI Controls Matrix (AICM)." Cloud Security Alliance, 2025.

[13] Cloud Security Alliance. "MAESTRO: Multi-Agent Environment and System Threat and Risk Operational Framework." Cloud Security Alliance, 2024.

