CSAI Foundation

Cloud Security Alliance AI Safety Initiative

CanisterSprawl: The Self-Propagating npm Supply Chain Worm

How a Self-Replicating npm Worm Weaponizes Developer Credentials and Decentralized Infrastructure

Unofficial AI-assisted Research

2026-04-23

Key Takeaways

- On April 21–22, 2026, security researchers at Socket and StepSecurity identified a self-propagating supply chain worm, dubbed **CanisterSprawl**, embedded across at least 16 malicious package versions spanning publisher namespaces linked to Namastex Labs and associated accounts. The worm is novel in its fully autonomous propagation: a single developer infection is sufficient to automatically re-infect every npm package the victim's account can publish [1][2].
- The malware executes at install time via a `postinstall` hook, harvesting approximately 40 categories of secrets via regex pattern matching—including cloud provider credentials, AI platform API keys (OpenAI, Anthropic, Cohere), CI/CD tokens, SSH keys, cryptocurrency wallet data, and browser-stored passwords—before exfiltrating them to a dual-channel endpoint [2].
- Exfiltrated data is sent to both a conventional HTTPS webhook and an Internet Computer Protocol (ICP) blockchain canister (`cjn37-uyaaa-aaaac-qgnva-cai`). The ICP canister appears to have been chosen specifically for censorship resistance: it cannot be seized by law enforcement or removed via domain takedown [1][2].
- The worm includes cross-ecosystem propagation logic: if Python package index (PyPI) credentials are discovered on the compromised machine, it generates `.pth` file payloads to infect Python packages via a method that executes automatically at Python interpreter startup [1][3].
- CanisterSprawl exhibits strong technical overlap with the TeamPCP threat actor's prior CanisterWorm campaign (March 2026), including shared ICP infrastructure patterns and an explicit in-code reference to a "TeamPCP/LiteLLM method." However, Socket's researchers stopped short of formal attribution, noting the ICP canister identifiers differ between campaigns [1][4].

Background

The Software Supply Chain as an Attack Surface

The npm registry hosts more than three million packages and serves hundreds of billions of package downloads monthly, making it one of the most widely used distribution networks in software development [5]. Because developers routinely install third-party packages with elevated runtime permissions, npm's `postinstall` script mechanism—designed to automate legitimate setup tasks—has become a well-documented vector for malicious payload execution. Unlike vulnerabilities that require exploitation, postinstall-based attacks require no user interaction beyond a standard `npm install` command.

The broader threat landscape confirms this risk is structural. By most accounts, supply chain attacks targeting package ecosystems have accelerated in frequency and sophistication throughout 2025 and 2026, with attackers moving from simple typosquatting to multi-stage, worm-capable deployments that can propagate without attacker re-engagement. What distinguishes CanisterSprawl is not the use of any single novel technique but the combination: fully autonomous self-replication, censorship-resistant exfiltration infrastructure, and built-in multi-ecosystem propagation logic targeting both npm and PyPI simultaneously.

The CanisterWorm Precedent and TeamPCP

To understand CanisterSprawl in context, it is essential to examine its March 2026 predecessor. In mid-March 2026, a threat actor tracked as TeamPCP compromised Trivy, the widely deployed open-source security scanner, by leveraging stolen maintainer credentials [6][7][9]. Socket's initial disclosure identified 29 affected packages; subsequent threat-hunting by Socket and independent researchers expanded the confirmed scope to 141 malicious artifacts across more than 66 unique npm packages spanning multiple organizational scopes [6][7].

CanisterWorm introduced two techniques that CanisterSprawl inherits: the use of an ICP blockchain canister as a dead-drop command-and-control (C2) resolver (the first publicly documented abuse of ICP infrastructure for C2 purposes), and a self-propagation mechanism that extracts npm tokens from the infected environment, identifies all packages the token can publish, injects the malicious payload, and republishes those packages with incremented version numbers [6]. Early CanisterWorm variants required some degree of manual intervention for initial deployment; the subsequent mutation found in `@teale.io/eslint-config` demonstrated fully autonomous spreading without operator involvement—the closest publicly documented precursor to CanisterSprawl's operational model.

TeamPCP extended its campaign to PyPI in late March 2026, compromising LiteLLM (versions 1.82.7 and 1.82.8) and Telnyx (versions 4.87.1 and 4.87.2). Datadog Security Labs documented how version 1.82.8 deployed a `.pth` file technique—lines in `.pth` files execute at Python interpreter startup—to establish persistent payload execution across Python environments [4]. This PyPI infection method appears directly in CanisterSprawl's cross-ecosystem propagation logic, further underlining the code lineage between campaigns.

Security Analysis

Attack Chain

CanisterSprawl operates through a six-stage sequence that begins at installation and terminates with autonomous worm propagation across the victim's accessible package portfolio.

The entry point is a `postinstall` hook in the package's `package.json`, configured as `"postinstall": "node scripts/check-env.cjs || true"`. The `|| true` suffix ensures that the malicious script's execution—successful or failed—never causes the installation to appear broken to the developer. The obfuscated payload itself spans 1,143 lines embedded in `scripts/check-env.cjs`, accompanied by an RSA public key in `scripts/public.pem` [2].

Once triggered, the payload enters credential harvesting mode. It scans approximately 40 regex patterns across environment variables, targeting secrets for AWS, Azure, GCP, GitHub, GitLab, Docker, Kubernetes, MongoDB, Redis, PostgreSQL, OpenAI, Anthropic, Cohere, Stripe, Twilio, and SendGrid, among others [2]. It then traverses the filesystem for credential stores including `~/.npmrc`, `~/.netrc`, `~/.ssh/`, `~/.aws/credentials`, `~/.azure/accessTokens.json`, GCP service account JSON files, and cryptocurrency wallet files for MetaMask, Phantom, Exodus, and Atomic Wallet. On Linux systems, the payload decrypts Chrome's stored password database using the known GNOME keyring key derivation method [2].

All harvested material is encrypted with a hybrid scheme—an AES-256-CBC session key protected by RSA-4096 with OAEP-SHA256 padding—before exfiltration, making the stolen data inaccessible to defenders who intercept it in transit without possession of the attacker's private key [2]. The SHA-256 fingerprint of the RSA public key embedded across affected packages is `87259b0d1d017ad8b8daa7c177c2d9f0940e457f8dd1ab3abab3681e433ca88e`, providing a current indicator of compromise for artifact scanning—valid for this campaign but subject to change in subsequent variants [1].

Exfiltration proceeds over two independent channels. The primary channel is the ICP canister endpoint `cjn37-uyaaa-aaaac-qgnva-cai.raw.icp0.io/drop`, which returned HTTP 200 responses with JSON bodies confirming receipt (`{"success":true,"id":10,"size":4468}`) during StepSecurity's controlled analysis. The secondary channel is `telemetry.api-monitor.com`, a domain with zero prior threat intelligence history at the time of discovery, registered privately through Bluehost and consistent with purpose-built campaign infrastructure [2].

Upon locating a valid npm publishing token among the harvested credentials, the worm enters propagation mode. It queries the npm registry for every package the compromised account can publish, injects a copy of the malicious `check-env.cjs` payload and `public.pem` key into each package, increments the patch version, and republishes. Each republished package now carries the same worm payload, ready to infect the next developer who installs it. This recursive mechanism means a single initial victim can amplify the infection by the full breadth of that account's publish permissions—without any further attacker action required.

If PyPI credentials are discovered, the payload switches to the `.pth` file technique: it generates a Python path configuration file containing the malicious payload and uploads it via Twine to the PyPI packages the victim can publish, extending the campaign into the Python ecosystem [1][3].

ICP Canister Infrastructure and Its Security Implications

The deliberate choice of an ICP blockchain canister as an exfiltration endpoint represents a meaningful evolution in attacker infrastructure design. Traditional C2 and exfiltration infrastructure relies on domains and IP addresses that defenders can identify and remove through registrar takedowns, DNS sinkholing, or law enforcement action. An ICP canister, by contrast, is a tamper-proof smart contract deployed on the Internet Computer blockchain. Its canister ID cannot be removed through conventional law enforcement channels such as registrar takedowns or DNS sinkholing; removal would require either the canister's own controller or a governance vote by the Internet Computer's Network Nervous System—neither of which defenders can compel [6][7]. The CanisterWorm campaign in March 2026 was the first publicly documented instance of ICP infrastructure being abused for malicious C2 purposes; CanisterSprawl confirms this technique has matured from a novel experiment into a repeatable operational pattern.

The Agentic AI Credential Exposure Risk

The specific victim profile in CanisterSprawl adds a dimension particularly relevant to AI security practitioners. Namastex Labs develops agentic AI tooling—the Automagik suite—designed to build and deploy autonomous AI agents. The package `@automagik/genie` is one of the primary compromised packages. The malware's explicit targeting of Anthropic, OpenAI, and Cohere API keys means that organizations whose CI/CD pipelines or development environments incorporate agentic AI tooling are at elevated risk of losing the credentials that authorize their AI agents to act. A stolen LLM API key can enable unauthorized model inference at the victim's expense, extraction of proprietary prompts and system instructions, or injection of adversarial inputs into agent pipelines if the key is used in production [8].

Affected Packages and Timeline

The initial compromise of `pgserve` was detected through the absence of a corresponding git tag: the last legitimate release (v1.1.10) was published on April 17, 2026 at 21:57 UTC with a matching git tag; versions 1.1.11, 1.1.12, and 1.1.13—all malicious—were published on April 21, 2026 beginning at 22:14 UTC with no corresponding tags in the upstream repository [2]. This discrepancy between the registry and source repository is a detectable anomaly that automated tooling can flag. Six packages with confirmed malicious versions are detailed below; Socket's full campaign analysis identified at least 16 malicious package versions spanning multiple publisher namespaces including @fairwords and @openwebconcept, all bearing the same worm payload [1].

Package	Malicious Versions
`@automagik/genie`	4.260421.33 – 4.260421.39
`pgserve`	1.1.11 – 1.1.13
`@fairwords/websocket`	1.0.38 – 1.0.39
`@fairwords/loopback-connector-es`	1.4.3 – 1.4.4
`@openwebconcept/design-tokens`	1.0.3
`@openwebconcept/theme-owc`	1.0.3

Recommendations

Immediate Actions

Organizations that have installed any of the affected package versions in the past 72 hours should treat the incident as a confirmed credential compromise and rotate all secrets accessible from the affected environment without delay. This includes npm tokens, cloud provider API keys and service account credentials, AI platform API keys (OpenAI, Anthropic, Cohere, and similar), SSH keys, GitHub and GitLab personal access tokens, database connection strings, and any secrets stored in `.env` files or shell history that were accessible from the compromised host. Rotating credentials before auditing the full scope of exposure is preferable to delaying rotation pending a complete inventory.

Audit all internal artifact repositories, caches, and CI/CD pipeline configurations for references to the malicious package versions. Container images, lock files, and dependency snapshots in build pipelines may have captured and re-served the malicious versions. These must be rebuilt from known-clean sources. Search artifact stores for any file matching the RSA key fingerprint `87259b0d1d017ad8b8daa7c177c2d9f0940e457f8dd1ab3abab3681e433ca88e` or the filenames `scripts/check-env.cjs` and `scripts/public.pem` across all cached packages.

Block outbound network connections to both identified exfiltration endpoints: `cjn37-uyaaa-aaaac-qgnva-cai.raw.icp0.io` and `telemetry.api-monitor.com`. While blocking these domains does not remediate an already-completed exfiltration, it prevents ongoing callbacks from potentially persistent implants and disrupts the PyPI propagation logic if it has not yet executed.

Short-Term Mitigations

Implement or enforce npm token scoping as a near-term control. npm automation tokens—the credentials most useful to supply chain worms—can be scoped to read-only access for install operations and to specific package scopes for publish operations. An account whose automation token lacks publish permissions on unrelated packages cannot be weaponized to spread the worm laterally across those packages. Organizations should audit all npm token permissions currently in use across developer workstations and CI/CD systems and remove unnecessary publish rights.

Introduce `postinstall` script controls in dependency management policy. Tools such as npm's `--ignore-scripts` flag, or organizational policies enforced via `.npmrc` with `ignore-scripts=true`, prevent postinstall hooks from executing during installation. For teams that cannot disable scripts universally, runtime security tools capable of alerting on unexpected network connections

during `npm install` provide a meaningful detection layer; StepSecurity's Harden-Runner demonstrated this capability in their controlled analysis of this incident [2]—though readers should note that StepSecurity is also the vendor of that product.

Establish verification requirements for the npm-to-git correspondence of all dependencies. The gap between a published npm version and its corresponding source repository commit is a reliable indicator of tampering. Automated checks that compare registry version tags against repository tags can detect injected versions before they reach production environments.

Strategic Considerations

The use of ICP canister infrastructure in both CanisterWorm and CanisterSprawl suggests this technique may be gaining traction in attacker operational toolkits. If this pattern is adopted beyond TeamPCP, defenders relying exclusively on domain reputation and IP blocklisting will find ICP-backed endpoints difficult to neutralize via traditional takedown processes. Organizations should evaluate whether their network egress controls can selectively restrict or monitor traffic to ICP gateway domains (the `.icp0.io` and `.ic0.app` top-level paths) as a category, without blanket blocking of all ICP traffic, which may affect legitimate decentralized application use.

The cross-ecosystem propagation from npm to PyPI underscores the importance of treating credential exposure as an ecosystem-wide event rather than a package-specific one. A developer whose npm token is compromised may have PyPI credentials on the same workstation; a single postinstall execution can seed infections across both ecosystems simultaneously. GitGuardian's concurrent documentation of related supply chain campaigns during this period illustrates how quickly such infections can propagate across multiple registries in a compressed timeframe [10]. Security operations teams should develop incident response runbooks that treat npm and PyPI compromise as joint events requiring parallel investigation.

AI development environments deserve specific attention. Developers building agentic AI systems routinely handle high-value API keys for production LLM services and may store these credentials in workstation `~/.env` files or shell history files—the exact locations CanisterSprawl targets. Organizations should establish dedicated credential management practices for AI platform keys, including rotation policies, vault-based storage rather than developer workstation files, and environment-specific scoping that prevents development-environment credentials from granting access to production AI workloads.

CSA Resource Alignment

CanisterSprawl maps directly to several layers of CSA's threat modeling and control frameworks for AI and cloud security.

CSA's MAESTRO framework—the Multi-Agent Environment Security Threat, Risk, and Observability model—identifies the integrity of agent tooling and dependency chains as a critical control surface for agentic AI deployments. CanisterSprawl's targeting of `@automagik/genie` and similar agentic AI packages demonstrates that the software supply chain feeding autonomous agent systems is as much an adversarial target as the agents themselves. Organizations applying MAESTRO threat modeling to their agentic AI architectures should extend the scope of that analysis to the full npm and PyPI dependency trees of agent frameworks and toolkits.

The CSA AI Controls Matrix (AICM)—a superset of the Cloud Controls Matrix (CCM)—provides controls relevant to this incident under the Software Development Lifecycle (SLC) and Infrastructure and Virtualization Security (IVS) domains. Specifically, controls addressing dependency integrity verification, privileged credential management in development environments, and build pipeline isolation are directly applicable as countermeasures to postinstall-based supply chain worms.

CSA's Zero Trust guidance applies to the credential exposure risk inherent in developer workstation compromise. The principle of least-privilege access should govern npm and PyPI publishing tokens just as it does cloud IAM roles: tokens should carry only the permissions necessary for their intended operations, scoped to specific packages rather than entire account portfolios. Applying zero trust principles to package publishing credentials directly limits the propagation radius of a worm that relies on stolen publish tokens to spread.

CSA STAR program participants in the software tooling and AI platform sectors should consider adding supply chain worm detection capability—including git-registry correspondence checks and postinstall script auditing—to their STAR Level 2 evidence packages as part of continuous assurance practices.

References

[1] Socket Research Team. "Namastex.ai npm Packages Hit with TeamPCP-Style CanisterWorm Malware." Socket Security, April 2026.

[2] StepSecurity. "CanisterSprawl: pgserve Compromised on npm: Malicious Versions Harvest Credentials and Exfiltrate to a Decentralized ICP Canister." StepSecurity Blog, April 2026.

[3] The Hacker News. "Self-Propagating Supply Chain Worm Hijacks npm Packages to Steal Developer Tokens." The Hacker News, April 22, 2026.

[4] Datadog Security Labs. "LiteLLM and Telnyx Compromised on PyPI: Tracing the TeamPCP Supply Chain Campaign." Datadog Security Labs, April 2026.

[5] BleepingComputer. "New npm Supply-Chain Attack Self-Spreads to Steal Auth Tokens." BleepingComputer, April 2026.

[6] Socket Research Team. "CanisterWorm: npm Publisher Compromise Deploys Backdoor Across 29 Packages." Socket Security, March 2026.

[7] The Hacker News. "Trivy Supply Chain Attack Triggers Self-Spreading CanisterWorm Across 47 npm Packages." The Hacker News, March 2026.

[8] The Register. "Another npm Supply Chain Worm Hits Dev Environments." The Register, April 22, 2026.

[9] Wiz. "Trivy Compromised by 'TeamPCP'." Wiz Blog, March 2026.

[10] GitGuardian. "No Off Season: Three Supply Chain Campaigns Hit npm, PyPI, and Docker Hub in 48 Hours." GitGuardian Blog, April 2026.