CanisterWorm: Blockchain C2 in CI/CD Supply Chain Attack

ICP Dead-Drop Command Infrastructure Enables Self-Propagating npm Worm

Unofficial AI-assisted Research

Cloud Security Alliance AI Safety Initiative

2026-03-22

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

- CanisterWorm is a self-propagating npm worm discovered March 20, 2026, by Aikido Security researcher Charlie Eriksen. It is the second stage of a two-phase supply chain attack orchestrated by the threat actor TeamPCP (also tracked as DeadCatx3, PCPcat, and ShellForce) [1][2].
- The first phase compromised Aqua Security's Trivy vulnerability scanner GitHub Action on March 19, 2026, infecting CI/CD pipelines in an estimated 10,000+ GitHub workflow files and stealing credentials — including npm authentication tokens — from pipeline environments [3][4].
- CanisterWorm uses stolen npm tokens to self-replicate: upon installation, it harvests any npm tokens present in the environment, enumerates all packages the token's owner maintains, and publishes a malicious patch-incremented version to each — enabling cascading propagation across the npm registry [1][5].
- The campaign introduces a novel command-and-control architecture: a canister deployed on the Internet Computer Protocol (ICP) blockchain functions as a tamper-resistant dead-drop resolver, returning the URL of the current payload binary. Because ICP has no centralized hosting provider, the C2 infrastructure cannot be neutralized by conventional ISP or domain registrar takedown requests; network-level blocking of `*.raw.icp0.io` endpoints provides a blunt defensive option but would affect all ICP-hosted applications [1][6].
- As of the date of this publication, the ICP canister is returning a benign URL (a redirect to a non-malicious video), suggesting the primary payload stage has been withheld or the attacker has intentionally armed a kill switch. The worm propagation mechanism and credential theft components were, however, fully armed in the wild [1][2].
- Organizations should treat any npm token exposed in a CI/CD environment between March 19 and March 21, 2026, as compromised, audit all packages associated with those tokens, and immediately rotate any such tokens [3][5].

Background

On March 19, 2026, TeamPCP compromised the `aquasecurity/trivy-action` and `setup-trivy` GitHub Actions — components of Aqua Security's widely deployed Trivy open-source vulnerability scanner — by pushing a malicious release tagged `v0.69.4` [3][4]. This was not Trivy's first compromise: a prior February 28, 2026 incident involved an automated threat actor exploiting a `pull_request_target` workflow misconfiguration to steal a Personal Access Token, which was used to delete releases and introduce a malicious Visual Studio Code extension. Incomplete credential rotation after that first incident may have left authentication material in place; StepSecurity assessed that TeamPCP leveraged this surviving access to execute the March 19 attack, though the exact access pathway has not been independently confirmed [4][7].

Trivy's GitHub Action is referenced in an estimated 10,000 or more GitHub workflow files, making it a high-leverage target for supply chain compromise [3]. The malicious `v0.69.4` binary — in versions for Linux, macOS, and Windows — functioned as an infostealer. It collected SSH keys, cloud provider credentials, Kubernetes tokens, Docker configurations, database passwords, Terraform state files, cryptocurrency wallet keys, and npm authentication tokens from any CI/CD runner executing the workflow [3][4]. Stolen data was exfiltrated using AES-256-CBC and RSA-4096 hybrid encryption to a typosquatting C2 domain (`scan.aquasecurtiy[.]org`) and, as a fallback, via a publicly accessible repository (`tpcp-docs`) created on victim GitHub accounts [3][7]. The malicious release was live from approximately 17:43 UTC on March 19 until remediation was completed at approximately 05:40 UTC on March 20, 2026 [4]. The incident is documented in GitHub Security Advisory GHSA-69fq-xp46-6x23 (Critical severity); no CVE has been assigned [4].

TeamPCP is a cloud-native threat actor first profiled by Flare in February 2026, with Telegram activity associated with their ShellForce channel dating to July 2025 [8]. Prior to the Trivy campaign, the group was known for targeting exposed Docker APIs, Kubernetes clusters, Ray dashboards, and Redis servers, having compromised an estimated 60,000 or more servers across those campaigns [8]. Their operational model combines cryptomining, credential theft, ransomware deployment, and sale of compromised access — the Trivy compromise represents their first confirmed expansion into developer toolchain supply chain operations.

Security Analysis

The CanisterWorm Attack Chain

CanisterWorm operates as a three-stage payload delivered through malicious npm packages published using npm authentication tokens stolen during the Trivy CI/CD compromise. When a developer or automated pipeline runs `npm install` on an affected package, the attack chain initiates immediately through the npm `postinstall` hook — a standard npm lifecycle mechanism that executes arbitrary scripts automatically upon package installation [1][5].

Stage 1 – Loader (`index.js`): The `postinstall` hook executes a Node.js script that decodes a base64-encoded Python backdoor payload and writes it to `~/.local/share/pgmon/service.py`. The script then creates a systemd user service at `~/.config/systemd/user/pgmon.service` configured with `Restart=always` and a five-second restart delay, establishing persistence without requiring root privileges. The service name `pgmon` closely resembles legitimate PostgreSQL monitoring tooling — a likely deliberate choice to reduce the likelihood of casual detection by system administrators [1][2].

Stage 2 – Python backdoor (`service.py`): After an initial sleep of 300 seconds, the backdoor begins beaconing to the ICP canister C2 at intervals of approximately 3,000 seconds. It contacts the canister with a spoofed `Mozilla/5.0` User-Agent, retrieves the current payload URL, downloads the binary to `/tmp/pglog`, sets executable permissions, and spawns it as a detached process. A state file at `/tmp/.pg_state` tracks previously executed URLs to prevent redundant re-execution. A kill switch is embedded in the beacon logic: if the returned URL contains the string `youtube.com`, the backdoor skips execution — a mechanism that appears to currently be active, with the canister returning a YouTube redirect rather than a live payload URL [1].

Stage 3 – Self-propagating worm (`deploy.js`): This stage represents the defining characteristic of CanisterWorm and the mechanism that elevates it from a conventional credential-stealing npm package to a worm. As Eriksen wrote in his initial disclosure: *"This is the point where the attack goes from 'compromised account publishes malware' to 'malware compromises more accounts and publishes itself.'"* [1] The `deploy.js` script systematically harvests npm authentication tokens from `~/.npmrc`, project-level `.npmrc` files, `/etc/npmrc`, and environment variables matching patterns including `NPM_TOKEN`, `NPM_TOKENS`, and `*NPM*TOKEN*`. For each token recovered, it authenticates to npm's `/-/whoami` endpoint, then queries the npm search API for all packages maintained by that token's associated user account — enumerating the complete set of publishable packages in paginated batches of 250. It then bumps the patch version of each discovered package, fetches the original package README to preserve trust signals, and publishes the new malicious version to the npm registry as the latest tag. In Aikido Security's controlled testing, this process enumerated and prepared 28 packages for malicious publishing in under 60 seconds [1].

Blockchain-Based C2: A Resilient Dead-Drop Architecture

The Internet Computer Protocol, developed by the DFINITY Foundation, is a decentralized blockchain network that hosts "canisters" — tamper-resistant WebAssembly smart contracts (modifiable only by their controller) that expose HTTP endpoints with no centralized hosting infrastructure. TeamPCP deployed a canister with the ID `tdtqy-oyaaa-aaaae-af2dq-cai`, accessible at `https://tdtqy-oyaaa-aaaae-af2dq-cai.raw.icp0.io/`, exposing three methods: `get_latest_link` (returning the current payload URL), `http_request` (the standard canister HTTP interface), and `update_link` (allowing the threat actor to swap the payload target URL at any time) [1][6].

This architecture implements the MITRE ATT&CK technique T1102.001 (Web Service: Dead Drop Resolver) in a form specifically engineered for resilience against conventional takedown. The canister does not serve malware directly; it functions as an indirection layer that returns a plain-text URL pointing to wherever the actual binary is currently hosted. Because the canister runs on ICP's distributed node network rather than any single hosting provider or registrar-controlled domain, conventional law enforcement requests to an ISP, cloud provider, or domain registrar are ineffective — there is no central hosting party to contact. Network-level blocking of `*.raw.icp0.io` endpoints is available as a blunt defensive measure but would affect all ICP-hosted applications. Cooperation from the DFINITY Foundation is theoretically possible but is not a standard abuse-handling pathway. The threat actor retains the ability to update the payload URL and push new binaries to all infected hosts without touching the implant itself, simply by calling the canister's `update_link` method [1][6].

This represents the first publicly confirmed abuse of an ICP canister for malware command-and-control purposes. It is not, however, the first instance of blockchain infrastructure being abused for C2 in supply chain contexts. The EtherHiding technique — storing JavaScript payloads or C2 addresses in BNB Smart Chain smart contracts — was first publicly documented in October 2023 and subsequently adopted by North Korea's Lazarus Group (UNC5342) for infostealer distribution campaigns [9]. In November 2024, Checkmarx documented a typosquatting npm package (`jest-fet-mock`) that used an Ethereum smart contract to dynamically retrieve C2 server addresses — the first documented case of blockchain C2 in an open-source supply chain attack [10]. NeoShadow, a December 2025 campaign involving four npm packages, queried an Ethereum smart contract via Etherscan's API to resolve a dynamic C2 base URL before falling back to a hardcoded domain [11]. The Aeternum C2 botnet, disclosed in February

2026, stored encrypted commands in Polygon blockchain transactions, with operational costs as low as one dollar per 100 to 150 command transactions [12]. CanisterWorm's ICP canister represents an evolution of this pattern: it combines the dead-drop resolver approach with a blockchain network specifically designed for decentralized application hosting, introducing HTTP endpoint exposure directly from the canister and removing the need to interact with public blockchain explorer APIs as an intermediary.

Scope of Compromise

At the time of initial disclosure by Socket.dev, 135 malicious artifacts had been identified across 64 or more unique npm packages [5] (Socket.dev's article was subsequently updated to reflect the expanded enumeration; the original headline cited 29 packages). Independent analyses from Endor Labs and Mend.io identified 47 to 50 packages at first publication, reflecting the ongoing nature of the investigation and variation in enumeration methodologies [2][6]. Affected packages included 28 packages in the `@EmilGroup` scope (56 malicious versions), 16 packages in the `@opengov` scope, `@teale.io/eslint-config` (versions 1.8.9 through 1.8.12), `@airtm/uuid-base32`, `@pypestream/floating-ui-dom`, and standalone packages including `jest-preset-ppf`, `babel-plugin-react-pure-component`, and `eslint-config-service-users` [1][5].

The persistence mechanism — a systemd user service — only activates on Linux systems; macOS and Windows systems running the affected packages experienced credential theft and worm propagation but no persistent backdoor implantation [1]. This distinction is operationally significant for organizations auditing affected developer workstations: Linux-based CI/CD runners and developer machines require active removal of the systemd service and service file in addition to token rotation, while non-Linux systems require credential rotation but may not have a persistent foothold.

CanisterWorm's propagation model has a direct precedent in the Shai-Hulud self-propagating npm worm, first documented in September 2025. That campaign infected approximately 200 packages via the same npm token theft and re-publication mechanism, including CrowdStrike-owned packages, and established the worm propagation pattern that CanisterWorm subsequently employed. A second Shai-Hulud variant in November 2025 (Shai-Hulud 2.0) infected 796 unique npm packages with a combined weekly download count exceeding 20 million, across 25,000 malicious GitHub repositories [13][14]. CanisterWorm extends this lineage by adding the blockchain C2 second stage and the CI/CD compromise vector through Trivy.

Indicators of Compromise

The following artifacts indicate CanisterWorm infection on Linux systems:

Indicator	Type	Notes
`~/.local/share/pgmon/service.py`	File	Python backdoor implant
`~/.config/systemd/user/pgmon.service`	File	Persistence via systemd user service
`/tmp/pglog`	File	Downloaded payload binary
`/tmp/.pg_state`	File	State tracker for executed URLs
`tdtqy-oyaaa-aaaae-af2dq-cai.raw.icp0.io`	Network	ICP canister C2 endpoint
`scan.aquasecurtiy[.]org` / `45.148.10.212`	Network	Trivy-phase C2 (typosquatting domain)
`plug-tab-protective-relay.trycloudflare.com`	Network	GitHub Actions exfiltration relay
`tpcp-docs` public repository	Git artifact	Credential exfiltration fallback
systemd service `pgmon` with `Restart=always`	Process	Masquerading as PostgreSQL monitoring

File hashes for the malicious `index.js` payload vary across distribution waves: `e9b1e069efc778c1e77fb3f5fcc3bd3580bbc810604cbf4347897ddb4b8c163b` (Wave 1, dry run), `61ff00a81b19624adaad425b9129ba2f312f4ab76fb5ddc2c628a5037d31a4ba` (Wave 2, armed), `0c0d206d5e68c0cf64d57ffa8bc5b1dad54f2dda52f24e96e02e237498cb9c3a` (Wave

3, self-propagating test payload), and `c37c0ae9641d2e5329fcdee847a756bf1140fdb7f0b7c78a40fdc39055e7d926` (Wave 4, final) [1].

Recommendations

Immediate Actions

Any organization whose CI/CD pipelines executed `aquasecurity/trivy-action` or `setup-trivy` between 17:43 UTC March 19 and 05:40 UTC March 20, 2026 should treat all secrets accessible from those pipeline runners as potentially exfiltrated. This includes SSH keys, cloud provider credentials, Kubernetes service account tokens, Docker registry credentials, database connection strings, Terraform state containing embedded secrets, and — critically — npm authentication tokens configured in the pipeline environment [3][4]. All such secrets should be rotated immediately, without waiting to confirm whether a specific pipeline was actually compromised. The brevity of the exposure window and the automated, high-volume nature of the credential exfiltration make targeted confirmation impractical as a prerequisite for remediation.

For npm tokens specifically, organizations should query the npm registry to identify all packages associated with any token that may have been present in a compromised pipeline environment. Any packages where the maintainer list or published versions appear unexpected warrant investigation. Organizations should review npm publish audit logs for their packages to detect unauthorized version publications during the March 19 to March 21, 2026 window. The npm registry provides audit event history accessible via the npm CLI and registry API.

On any Linux system — developer workstation or CI/CD runner — that installed an npm package during the exposure window, operators should check for the presence of `~/.local/share/pgmon/service.py`, `~/.config/systemd/user/pgmon.service`, `/tmp/pglog`, and `/tmp/.pg_state`. If found, the systemd service should be stopped and disabled, both files removed, and the machine treated as potentially fully compromised pending forensic review. On macOS and Windows systems, the persistent foothold is not present, but token theft may still have occurred.

Safe versions of the Trivy ecosystem components are: `trivy` version 0.69.2 or 0.69.3 (avoid 0.69.4 entirely), `trivy-action` version 0.35.0 or later, and `setup-trivy` version 0.2.6 (the restored clean version) [4].

Short-Term Mitigations

CI/CD pipeline hardening should be a priority for any organization using GitHub Actions with third-party actions. Pinning actions to specific commit SHAs rather than mutable version tags prevents a compromised tag push from silently altering pipeline behavior — if `aquasecurity/trivy-action@0.34.2` had been referenced by SHA rather than tag, the malicious `v0.69.4` tag would not have been loaded by any pipeline that had previously pinned to a safe version [3][7]. Tools such as StepSecurity's Harden-Runner, which monitors GitHub Actions network egress and runtime behavior, can detect anomalous outbound connections from CI/CD runners at the time they occur rather than after credential exfiltration is complete — though organizations should be aware that StepSecurity is also a primary incident analysis source for this campaign [7].

Pipeline secrets should be scoped with least privilege. npm tokens used in CI/CD should be automation tokens restricted to publish access on specific packages, not user tokens with full account scope. GitHub Actions workflow permissions should be explicitly configured to the minimum required — specifically, the `GITHUB_TOKEN` permissions block should not grant write access to any resource not required by the workflow. Secrets available to pull request workflows deserve particular scrutiny given the `pull_request_target` exploitation that enabled TeamPCP's initial Trivy access in February 2026 [4][7].

For npm packages in particular, organizations should evaluate whether their dependency trees include packages from the affected scopes identified in threat intelligence reporting. Dependabot, Socket.dev's GitHub App, and similar software composition analysis tooling can flag packages associated with known malicious publications at time of install or during scheduled dependency review.

Strategic Considerations

The CanisterWorm campaign illustrates a structural risk in the open-source supply chain ecosystem: the combination of high-trust developer tooling, automated pipeline execution, and npm's postinstall hook mechanism creates a surface where a single compromised upstream component can yield authenticated publishing access to a large and unpredictable set of downstream packages proportional to the breadth of the compromised upstream component's adoption. Traditional dependency auditing approaches — static SBOM generation and scheduled SCA scans — are not designed to detect novel malicious publications in real time. Newer signal-aware tools such as Socket.dev's GitHub App can flag packages associated with known malicious activity at the point of installation, though they depend on prior attribution of a package as malicious. The self-propagating worm model, first established by Shai-Hulud and now extended by CanisterWorm with blockchain C2, represents a qualitative escalation in supply chain attack capability that neither approach fully addresses.

The use of ICP canister infrastructure for C2 represents a meaningful shift in threat actor capability planning. Blockchain-based C2 is not a novel concept, but ICP canisters introduce characteristics — natively exposed HTTP endpoints, WebAssembly execution, and decentralized hosting with no single removable node — that make the infrastructure more resilient than smart contract dead-drop approaches that rely on third-party API intermediaries such as Etherscan. Security teams and threat intelligence programs should consider how their tooling handles indicators associated with ICP canister infrastructure. Conventional per-domain or per-IP blocking cannot target the malicious canister in isolation; the ICP gateway domain (`*.icp0.io`) is shared across all ICP-hosted canisters, meaning any network block broad enough to exclude the malicious canister would also affect legitimate ICP applications.

Organizations with significant open-source publishing footprints should evaluate whether their npm account security postures — including token scoping, multi-factor authentication requirements, and access audit practices — are commensurate with the trust that downstream consumers implicitly extend to their packages. The reputational and operational consequences of an organization's published packages becoming propagation vectors for a supply chain worm extend well beyond the organization's own security posture.

CSA Resource Alignment

CanisterWorm maps to several Cloud Security Alliance frameworks and publications that provide guidance directly applicable to the threat patterns this campaign demonstrates.

The CSA Cloud Controls Matrix (CCM) v4.1 addresses supply chain risk under the Supply Chain Management, Transparency and Accountability (STA) domain, and the Application and Interface Security (AIS) domain covers CI/CD pipeline security controls and software composition analysis requirements. The CCM's Identity and Access Management (IAM) domain controls apply directly to the recommendation that pipeline secrets be scoped with least privilege and that access tokens be treated as high-value credentials subject to lifecycle management controls. Organizations assessing their CCM compliance posture should evaluate STA-08 (Supply Chain Governance) and AIS-04 (Secure Development Policy) in light of the access control and dependency governance failures that enabled CanisterWorm's propagation [15].

The AI Controls Matrix (AICM), as a superset of CCM that extends security controls to AI system development and deployment pipelines, is directly relevant where AI development workflows use npm-based tooling or GitHub Actions. The AICM's supply chain controls apply to the same CI/CD pipeline environments that CanisterWorm targeted: organizations building AI systems on top of JavaScript/Node.js toolchains should assess their AICM compliance against the same control domains identified above.

The CSA's *AI Organizational Responsibilities: Core Security Responsibilities* publication addresses third-party risk management and the organizational accountability structures that govern secure software procurement and pipeline security. The Trivy compromise specifically exploited the implicit trust that organizations extend to security tooling vendors — the vulnerability scanner was itself the attack vector. CSA's guidance on third-party AI and security tool governance is directly applicable [16].

The CSA STAR program provides a framework for assessing cloud provider security postures that is relevant to organizations evaluating ICP as infrastructure. The absence of traditional hosting controls for ICP canisters — including the inability to file conventional abuse or takedown requests — represents a gap in standard third-party risk assessment frameworks that organizations using or evaluating ICP-hosted components should account for in their STAR-aligned vendor risk assessments.

The CSA's Zero Trust guidance, including the *An Executive View on How Zero Trust Protects Organizations* publication, applies to the recommendation that pipeline secrets not be held in execution environments with broader-than-necessary access scope. Zero Trust principles — specifically, never trust implicit authentication and always enforce least privilege — are directly operationally applicable to the CI/CD token management failures that enabled CanisterWorm's propagation [17].

References

1. Eriksen, Charlie / Aikido Security, "TeamPCP deploys CanisterWorm on NPM following Trivy compromise," Aikido Security Blog, March 2026. https://www.aikido.dev/blog/teampcp-deploys-worm-npm-trivy-compromise

2. Endor Labs, "CanisterWorm: Malicious npm Packages Deploy Self-Propagating Supply Chain Worm," Endor Labs Blog, March 2026. https://www.endorlabs.com/learn/canisterworm

3. The Hacker News, "Trivy Supply Chain Attack Triggers Self-Spreading CanisterWorm Across 47 npm Packages," March 2026. https://thehackernews.com/2026/03/trivy-supply-chain-attack-triggers-self.html

4. GitHub Security Advisory, "Trivy ecosystem supply chain briefly compromised," GHSA-69fq-xp46-6x23, published March 21, 2026. https://github.com/aquasecurity/trivy/security/advisories/GHSA-69fq-xp46-6x23

5. Socket.dev, "CanisterWorm: npm Publisher Compromise Deploys Backdoor Across 29 Packages," Socket Security Blog, March 2026. https://socket.dev/blog/canisterworm-npm-publisher-compromise-deploys-backdoor-across-29-packages

6. Mend.io, "CanisterWorm: The Self-Spreading npm Attack That Uses a Decentralized Server to Stay Alive," Mend Security Blog, March 2026. https://www.mend.io/blog/canisterworm-the-self-spreading-npm-attack-that-uses-a-decentralized-server-to-stay-alive/

7. StepSecurity, "Trivy Compromised a Second Time – Malicious v0.69.4 Release," StepSecurity Blog, March 2026. https://www.stepsecurity.io/blog/trivy-compromised-a-second-time---malicious-v0-69-4-release

8. Flare, "Threat Alert: TeamPCP, An Emerging Force in the Cloud Native Landscape," Flare Blog, February 2026. https://flare.io/learn/resources/blog/teampcp-cloud-native-ransomware

9. Google Threat Intelligence Group (GTIG), "DPRK Adopts EtherHiding: Nation-State Malware Hiding on Blockchains," Google Cloud Blog, October 2025. https://cloud.google.com/blog/topics/threat-intelligence/dprk-adopts-etherhiding

10. Infosecurity Magazine, "Supply Chain Attack Uses Smart Contracts for C2 Ops," November 2024. https://www.infosecurity-magazine.com/news/supply-chain-attack-smart/

11. Aikido Security, "NeoShadow npm Supply-Chain Attack: JavaScript, MSBuild & Blockchain," Aikido Security Blog, January 2026. https://www.aikido.dev/blog/neoshadow-npm-supply-chain-attack-javascript-msbuild-blockchain

12. The Hacker News, "Aeternum C2 Botnet Stores Encrypted Commands on Polygon Blockchain," February 2026. https://thehackernews.com/2026/02/aeternum-c2-botnet-stores-encrypted.html

13. Sysdig, "Shai-Hulud: The Novel Self-Replicating Worm Infecting npm," Sysdig Blog, September 2025. https://www.sysdig.com/blog/shai-hulud-the-novel-self-replicating-worm-infecting-hundreds-of-npm-packages

14. Microsoft Security Blog, "Shai-Hulud 2.0 Guidance for Detecting, Investigating, and Defending Against the Supply Chain Attack," December 2025. https://www.microsoft.com/en-us/security/blog/2025/12/09/shai-hulud-2-0-guidance-for-detecting-investigating-and-defending-against-the-supply-chain-attack/

15. Cloud Security Alliance, "CCM v4.1 Implementation Guidelines," CSA, 2025. https://cloudsecurityalliance.org/research/cloud-controls-matrix/

16. Cloud Security Alliance, "AI Organizational Responsibilities: Core Security Responsibilities," CSA AI Safety Initiative, 2025. https://cloudsecurityalliance.org/research/working-groups/ai-organizational-responsibilities/

17. Cloud Security Alliance, "An Executive View on How Zero Trust Protects Organizations by Securely Connecting Users to Resources from Anywhere," CSA, 2024. https://cloudsecurityalliance.org/zt