INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-22 13:40 UTC

Self-Replicating npm Supply Chain Worm Harvests Developer Credentials and Spreads Across Ecosystems

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0200
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	npm packages: @automagik/genie, pgserve, @fairwords/websocket, @fairwords/loopback-connector-es, @openwebconcept/theme-owc, @openwebconcept/design-tokens (Namastex Labs, 16+ packages total); PyPI ecosystem (secondary); Chrome, Firefox, MetaMask, Exodus, Atomic Wallet, Phantom browser extensions and crypto wallets
Published	2026-04-22T08:57:42
Discovery Source	Rss

Executive Summary

A self-replicating worm discovered April 21-22, 2026, has infected at least 16 npm packages from Namastex Labs by stealing developer publish tokens and autonomously injecting malicious code into every package those tokens control. The worm harvests credentials from browser extensions, cryptocurrency wallets, and CI/CD environment variables, then uses stolen tokens to spread further, creating an exponentially expanding infection surface across npm and secondarily PyPI. Any organization whose developers or build pipelines installed affected packages should treat their development environment and all secrets stored there as compromised. Technical details are compiled from available reporting; security teams should cross-reference against primary research from Socket, StepSecurity, and vendor threat teams before taking final action.

Technical Analysis

Based on preliminary reports from multiple news outlets (April 21-22, 2026), a self-replicating worm has been observed in the npm ecosystem. A supply chain worm, attributed with low confidence to a group with technical similarity to TeamPCP, propagates by stealing npm publish tokens from developer environments and CI/CD pipelines (T1552.004, T1528). Once a token is obtained, the worm autonomously publishes poisoned versions

of every package that token authorizes (T1195.001, T1195.002, T1554), embedding payload code that targets browser extension credential stores (Chrome, Firefox, T1539, T1555.003), cryptocurrency wallets (MetaMask, Exodus, Atomic Wallet, Phantom), and environment variables (T1552.001). Secondary propagation into PyPI has been confirmed, indicating cross-ecosystem reach (T1195.002). The payload communicates via standard HTTP/S (T1071.001) and exfiltrates data via web services (T1567). JavaScript is the primary execution mechanism (T1059.007). Confirmed infected packages include @automagik/genie, pgserve, @fairwords/websocket, @fairwords/loopback-connector-es, @openwebconcept/theme-owc, @openwebconcept/design-tokens, and 10+ additional Namastex Labs packages. No CVE has been assigned. Relevant CWEs: CWE-506 (embedded malicious code), CWE-494 (download without integrity check), CWE-522 (insufficiently protected credentials), CWE-829 (inclusion of functionality from untrusted control sphere), CWE-312 (cleartext storage of sensitive information). No vendor patch is available; removal and token rotation are the required response. Security teams should validate technical details against primary research from Socket, StepSecurity, and vendor threat intelligence publications as they become available.

Action Checklist

1. Step 1: Containment. Immediately audit all package.json, package-lock.json, and requirements.txt files across development, build, and CI/CD environments for the confirmed infected packages: @automagik/genie, pgserve, @fairwords/websocket, @fairwords/loopback-connector-es, @openwebconcept/theme-owc, @openwebconcept/design-tokens, and all other Namastex Labs-scoped packages. Block installation of all flagged packages at your package registry proxy or artifact manager (Artifactory, Nexus, Verdaccio). Do not run or build from any environment where these packages were installed until secrets rotation is complete. If immediate business continuity is required, prioritize secrets rotation in parallel and validate that all suspected environments have undergone token revocation before resuming deployments; treat any build artifact generated during the exposure window (April 21 - present) as potentially compromised.
2. Step 2: Detection. Query CI/CD logs and developer workstation audit logs for npm install, npm publish, or pip install events referencing the confirmed package names. Search environment variable stores and secrets managers for anomalous access events around April 21-22, 2026, and forward. Check browser extension activity logs if available. Look for outbound HTTP/S connections from build agents to unfamiliar destinations during or after package installation (T1071.001, T1567). Scan all npm tokens and PyPI API keys for unauthorized publish activity against packages you own.
3. Step 3: Eradication. Rotate all npm publish tokens immediately, particularly any stored in CI/CD environment variables, .npmrc files, GitHub Actions secrets, or developer dotfiles. Rotate PyPI API tokens if any build environment touched both ecosystems. Revoke and reissue any other secrets (API keys, cloud credentials, wallet seed phrases if stored in environment variables) present in environments where infected packages were installed. Remove affected packages from all lock files and rebuild from a clean dependency tree using only verified, pinned package versions. Re-examine all packages published by your organization between April 21 and present for unauthorized modifications.
4. Step 4: Recovery. After token rotation, verify no unauthorized npm or PyPI packages were published under your organization's scope during the exposure window. Re-run your full test suite against a clean dependency tree before promoting any build artifact to production. Monitor your npm and PyPI publish logs for anomalous activity for at least 30 days. Confirm secrets managers show no active sessions tied to rotated credentials.

5. Step 5: Post-Incident. This worm exploited the absence of publish token scoping, integrity verification on installed packages, and secrets isolation in CI/CD pipelines. Implement npm token granular permissions (publish-only, scoped to specific packages). Enforce package integrity verification using lockfile hash pinning and a tool such as Socket, Sigstore, or Snyk in your pipeline. Store CI/CD secrets in a dedicated secrets manager with least-privilege access and automatic rotation. Establish a baseline of expected outbound network connections from build agents and alert on deviations. Review NIST SP 800-161 (supply chain risk management) for a control framework applicable to this attack class.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO and legal counsel immediately if any evidence confirms unauthorized packages were published under your organization's npm or PyPI scope, if crypto wallet seed phrases or cloud credentials stored as CI environment variables were exposed, or if the worm's network callbacks transmitted secrets to external infrastructure — any of these conditions may trigger breach notification obligations under applicable data protection regulations.
Recovery Notes	After rotating all npm and PyPI tokens and verifying no unauthorized package versions were published under your scope, rebuild all CI/CD pipelines from clean runner images and re-pin every dependency to a verified SHA-512 integrity hash in lockfiles before any artifact is promoted to production. Monitor npm and PyPI publish logs for your organization's package scope daily for a minimum of 30 days, as the worm's stolen tokens may have been shared or cached in attacker infrastructure and could be retried after initial rotation if not fully revoked at the registry level. Any developer workstation where a compromised package was installed and where a browser-based crypto wallet (MetaMask, Phantom, Exodus, Atomic Wallet) was active during that session should be treated as fully compromised and reimaged, not merely remediated.
Forensic Artifacts	npm postinstall script execution records: Sysmon Event ID 1 (Process Create) on Windows build agents showing node.exe or npm spawning child processes (cmd.exe, powershell.exe, sh, curl, wget) during 'npm install' of the flagged Namastex Labs packages — the worm's credential harvesting and token exfiltration are triggered via postinstall hooks in package.json | CI/CD environment variable access traces: GitHub Actions runner diagnostic logs at '/home/runner/runners//diag/' and '.runner' files, combined with any secrets manager CloudTrail/audit events, showing which secrets were visible in the build environment at the time infected packages were installed between April 21, 2026 and present | Outbound network connections from build agents: NetFlow or pcap data showing HTTP POST requests from node.exe to non-npmjs.com destinations during or immediately after 'npm install' of flagged packages, consistent with MITRE T1071.001 (Application Layer Protocol: Web Protocols) and T1567 (Exfiltration Over Web Service) — capture destination IPs, POST body size, and timing relative to package install timestamps | Browser extension LevelDB credential stores: Raw LevelDB files from Chrome/Firefox profiles on developer workstations for MetaMask (extension ID: nkbihfbeogaeaoehlefnkodbefgpgknn), Phantom, Exodus, and Atomic Wallet extensions, located at platform-specific extension storage paths — the worm targets these stores for seed phrase and private key harvesting | .npmrc and dotfile token artifacts: All instances of ~/.npmrc, /.npmrc, ~/.yarnrc, and CI runner home directory dotfiles containing '//registry.npmjs.org/:_authToken=' entries, plus GitHub Actions secrets store exports and any .env files in repository roots — these are the primary credential stores the worm harvests and uses for autonomous npm publish propagation

Per-Action IR Details

Step 1: Containment — Immediately audit all package.json, package-lock.json, and requirements.txt files across development, build, and CI/CD environments for the confirmed infected packages: @automagik/genie, pgserve, @fairwords/websocket, @fairwords/loopback-connector-es, @openwebconcept/theme-owc, @openwebconcept/design-tokens, and all other Namastex Labs-scoped packages. Block installation of all flagged packages at your package registry proxy or artifact manager (Artifactory, Nexus, Verdaccio). Do not run or build from any environment where these packages were installed until secrets rotation is complete.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST CM-7 (Least Functionality), NIST SI-3 (Malicious Code Protection), CIS 2.3 (Address Unauthorized Software), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Run 'grep -r "@automagik/genie\|pgserve\|@fairwords/websocket\|@fairwords/loopback-connector-es\|@openwebconcept/theme-owc\|@openwebconcept/design-tokens" /path/to/repos --include="*.json" --include="*.txt" -l' to enumerate all affected manifests. In Verdaccio, add each flagged package to the blacklist array in config.yaml under 'packages:' with access: deny. For Nexus OSS, create a component blocklist rule via the REST API or UI under Repository > Routing Rules.

Evidence: Before blocking, capture: (1) full contents of every package-lock.json and yarn.lock referencing the flagged packages, preserving the 'resolved' URL and 'integrity' (sha512) fields — these confirm which exact tarball was fetched; (2) the ~/.npm/_cacache directory on developer workstations and build agents, which retains tarballs of previously installed packages for post-mortem hash comparison; (3) node_modules/.package-lock.json inside any project directory for the installed dependency graph at time of compromise; (4) contents of any .npmrc files in project roots, home directories, and CI runner home directories, which may contain the stolen npm publish tokens in plaintext.

Step 2: Detection — Query CI/CD logs and developer workstation audit logs for npm install, npm publish, or pip install events referencing the confirmed package names. Search environment variable stores and secrets managers for anomalous access events around April 21–22, 2026, and forward. Check browser extension activity logs if available. Look for outbound HTTP/S connections from build agents to unfamiliar destinations during or after package installation (T1071.001, T1567). Scan all npm tokens and PyPI API keys for unauthorized publish activity against packages you own.

NIST Phase: Detection Analysis

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

Controls: NIST IR-5 (Incident Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-12 (Audit Record Generation), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: On Linux CI runners, run 'journalctl -u runner --since 2026-04-21 | grep -E "npm (install|publish)|pip install"' and correlate timestamps against outbound DNS with 'tcpdump -r /var/log/capture.pcap -nn port 53' if packet captures exist. On Windows build agents with Sysmon deployed, query Event ID 1 (Process Create) for CommandLine containing 'npm publish' or 'pip upload', and Event ID 3 (Network Connect) from node.exe or python.exe to non-registry destinations. Use the Sigma rule 'proc_creation_win_npm_package_install.yml' from SigmaHQ as a detection baseline. Query npm audit log via 'npm access list packages ' and cross-reference with the publish history at 'https://registry.npmjs.org/-/npm/v1/packages?user=' using your rotated token's predecessor to identify unauthorized publishes.

Evidence: Capture before analysis: (1) GitHub Actions workflow run logs (Settings > Actions > Workflow runs) for all runs between April 21–22, 2026 and present, specifically the raw log files showing environment variable expansion and npm command output — download via 'gh run download --dir ./evidence'; (2) outbound network flow logs from build agents showing connections on port 443 from node.exe/npm processes to non-npmjs.com, non-registry.npmjs.org destinations, consistent with MITRE T1567 (Exfiltration Over Web Service) used to exfiltrate harvested tokens; (3) Chrome/Firefox browser extension storage databases at '%LOCALAPPDATA%/Google/Chrome/User

Data/Default/Local Extension Settings/' (Windows) or '~/.config/google-chrome/Default/Local Extension Settings/' (Linux) for MetaMask, Exodus, and Phantom extension IDs, which the worm targets for credential harvesting; (4) CI/CD secrets manager access logs (e.g., AWS Secrets Manager CloudTrail events 'GetSecretValue' with unexpected IAM principal or source IP) around the infection window.

Step 3: Eradication — Rotate all npm publish tokens immediately, particularly any stored in CI/CD environment variables, .npmrc files, GitHub Actions secrets, or developer dotfiles. Rotate PyPI API tokens if any build environment touched both ecosystems. Revoke and reissue any other secrets (API keys, cloud credentials, wallet seed phrases if stored in environment variables) present in environments where infected packages were installed. Remove affected packages from all lock files and rebuild from a clean dependency tree using only verified, pinned package versions. Re-examine all packages published by your organization between April 21 and present for unauthorized modifications.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST IA-5 (Authenticator Management), NIST SI-2 (Flaw Remediation), NIST CM-7 (Least Functionality), CIS 5.2 (Use Unique Passwords), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Revoke all npm tokens via 'npm token revoke ' for each token listed by 'npm token list --json'; do this from a clean, air-gapped workstation that never touched the infected packages. For GitHub Actions, use the GitHub CLI: 'gh secret delete NPM_TOKEN -R /' followed by 'gh secret set NPM_TOKEN -R / @ && tar -tzf | xargs -I{} tar -xOf {}' and diff against your git-tagged source for the same version. For PyPI, revoke tokens at pypi.org/manage/account/token/ and audit publish history with 'pip index versions ' to detect unexpected version bumps.

Evidence: Preserve before rotation: (1) complete output of 'npm token list --json' showing all active token IDs, creation dates, and CIDR restrictions — this establishes the full exposure surface of tokens the worm may have harvested; (2) contents of all .npmrc files found in CI runner home directories (~/.npmrc), project roots, and developer dotfiles repos, which store tokens in '//registry.npmjs.org/:_authToken=' format and are a primary exfiltration target of this worm; (3) GitHub Actions secrets list (accessible via 'gh secret list -R /') and the timestamps of last modification to detect if the worm already rotated a secret to maintain persistence; (4) diff output of 'git diff HEAD' for each of your organization's published npm/PyPI packages to identify unauthorized postinstall hooks or added network-calling code in package.json scripts.

Step 4: Recovery — After token rotation, verify no unauthorized npm or PyPI packages were published under your organization's scope during the exposure window. Re-run your full test suite against a clean dependency tree before promoting any build artifact to production. Monitor your npm and PyPI publish logs for anomalous activity for at least 30 days. Confirm secrets managers show no active sessions tied to rotated credentials.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-7 (Software, Firmware, and Information Integrity), NIST AU-11 (Audit Record Retention), NIST CP-10 (System Recovery and Reconstitution), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Verify clean publish history by querying 'curl https://registry.npmjs.org/' and inspecting the 'time' object for any version timestamps falling between April 21, 2026 and your token rotation date that do not correspond to authorized release commits. For PyPI, use 'pip index versions ' or query the JSON API at 'https://pypi.org/pypi//json' and audit the 'releases' object. To verify no active sessions exist against rotated credentials, check AWS Secrets Manager CloudTrail for 'GetSecretValue' events after rotation timestamp and confirm all are from expected IAM principals. Rebuild CI pipelines from scratch on a fresh runner image rather than reusing potentially contaminated runner environments.

Evidence: Capture for recovery validation: (1) npm registry publish history JSON for all your organization-scoped packages ('curl https://registry.npmjs.org/-/v1/search?text=maintainer:&size=250') to enumerate every package under your control and confirm no worm-authored versions exist; (2) secrets manager audit trail exports covering the full exposure window (April 21, 2026 to rotation date) showing all GetSecretValue, PutSecretValue, and DeleteSecret API

calls with source IPs and IAM principals for anomaly review; (3) CI/CD pipeline build artifact hashes (SHA-256) for all production deployments during the exposure window, to verify no worm-tainted build artifacts reached production environments.

Step 5: Post-Incident — This worm exploited the absence of publish token scoping, integrity verification on installed packages, and secrets isolation in CI/CD pipelines. Implement npm token granular permissions (publish-only, scoped to specific packages). Enforce package integrity verification using lockfile hash pinning and a tool such as Socket, Sigstore, or Snyk in your pipeline. Store CI/CD secrets in a dedicated secrets manager with least-privilege access and automatic rotation. Establish a baseline of expected outbound network connections from build agents and alert on deviations. Review NIST SP 800-161r1 (supply chain risk management) for a control framework applicable to this attack class.

NIST Phase: Post Incident

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

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SA-12 (Supply Chain Protection), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-7 (Least Functionality), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 6.3 (Require MFA for Externally-Exposed Applications)

Compensating: Implement Sigstore's cosign (free, open-source) to sign all packages your organization publishes: 'cosign sign-blob --output-signature .sig' and verify on install. Add Socket.dev's free GitHub App to all repositories to detect postinstall scripts and network-calling code in dependencies at PR time — this would have flagged the worm's credential-harvesting postinstall hook before merge. Create a YARA rule targeting the worm's known behavior pattern (environment variable enumeration combined with outbound HTTP POST in postinstall scripts) and run it against node_modules on each build via 'yara -r worm_postinstall.yar ./node_modules'. Enforce npm token granularity by creating automation tokens scoped to 'publish' only with CIDR restrictions via 'npm token create --cidr=/32 --type=publish'.

Evidence: For the lessons-learned record: (1) the complete list of all npm and PyPI packages confirmed or suspected to have been installed from infected Namastex Labs packages across all environments, with first-install timestamps from CI logs; (2) a timeline reconstruction showing the worm's self-replication path — specifically which of your organization's npm tokens were potentially exfiltrated and which downstream packages (if any) were subsequently compromised using those tokens, sourced from npm publish audit logs and network flow records; (3) browser extension forensic artifacts from affected developer workstations, specifically the LevelDB files for MetaMask ('chrome-extension://nkbihfbeogaeaoehlefnkodbefgpgknn/'), Phantom, and Exodus extension local storage, documenting whether wallet credentials were accessed by the worm's harvesting routine.

Detection Guidance

Primary detection signals: (1) npm audit logs or registry proxy logs showing installs of @automagik/genie, pgserve, @fairwords/websocket, @fairwords/loopback-connector-es, @openwebconcept/theme-owc, @openwebconcept/design-tokens, or any @namastex-labs-scoped package. (2) Unexpected npm publish events against packages your organization owns; check publish history via 'npm access ls-packages' and compare against authorized publish timestamps. (3) Outbound connections from CI/CD build agents or developer workstations to unfamiliar IPs or domains during or after package install steps; correlate with build logs. (4) Environment variable access anomalies in your secrets manager or CI/CD platform (GitHub Actions audit log, GitLab audit events, Jenkins build logs) around or after April 21, 2026. (5) Browser extension storage access by unexpected processes on developer endpoints (T1539). (6) Presence of .npmrc files containing tokens in home directories or project roots on shared build systems. Behavioral indicators: a build agent initiating network connections not present in baseline, or npm publish commands appearing in logs without corresponding authorized CI/CD job triggers. No public IOC list (malware hashes, C2 IPs, domains) has been

confirmed in sources available at configuration date. For self-replicating supply chain attacks, behavioral and log-based detection (package audit trails, publish events, environment variable access) are the primary detection mechanisms. Contact Socket or StepSecurity for any payload indicators they may have collected.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://www.npmjs.com/package/@automagik/genie`	Confirmed infected npm package — Namastex Labs AI agent tooling	HIGH
URL	`https://www.npmjs.com/package/pgserve`	Confirmed infected npm package — database library	HIGH
URL	`https://www.npmjs.com/package/@fairwords/websocket`	Confirmed infected npm package — Namastex Labs	HIGH
URL	`https://www.npmjs.com/package/@fairwords/loopback-connector-es`	Confirmed infected npm package — Namastex Labs	HIGH
URL	`https://www.npmjs.com/package/@openwebconcept/theme-owc`	Confirmed infected npm package — Namastex Labs	HIGH
URL	`https://www.npmjs.com/package/@openwebconcept/design-tokens`	Confirmed infected npm package — Namastex Labs	HIGH

Framework Mappings

MITRE-ATTACK

- **T1071.001** — Web Protocols
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1195.002** — Compromise Software Supply Chain
- **T1176** — Software Extensions
- **T1555.003** — Credentials from Web Browsers
- **T1539** — Steal Web Session Cookie
- **T1552.004** — Private Keys
- **T1554** — Compromise Host Software Binary
- **T1528** — Steal Application Access Token
- **T1552.001** — Credentials In Files
- **T1059.007** — JavaScript
- **T1567** — Exfiltration Over Web Service

NIST-800-53R5

- **CM-7** — Least Functionality

- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **IA-5** — Authenticator Management
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures
- **A04:2021** — Insecure Design
- **A07:2021** — Identification and Authentication Failures

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **5.2** — Use Unique Passwords
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers

HIPAA-SECURITY

- **164.308(a)(5)(ii)(D)** — Password Management
- **164.312(d)** — Person or Entity Authentication

SOC2-TSC

- **CC6.1** — Logical access security software, infrastructure, and architectures
- **CC9.2** — Manages risks associated with vendors and business partners

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program

ISO-27001-2022

- **A.5.21** — Managing information security in the ICT supply chain

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1071.001	Web Protocols	Command-And-Control
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1195.002	Compromise Software Supply Chain	Initial-Access
T1176	Software Extensions	Persistence

Technique ID	Technique Name	Tactic
T1555.003	Credentials from Web Browsers	Credential-Access
T1539	Steal Web Session Cookie	Credential-Access
T1552.004	Private Keys	Credential-Access
T1554	Compromise Host Software Binary	Persistence
T1528	Steal Application Access Token	Credential-Access
T1552.001	Credentials In Files	Credential-Access
T1059.007	JavaScript	Execution
T1567	Exfiltration Over Web Service	Exfiltration

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/new-npm-supply-chain...	T3
Malicious PyPI and npm Packages Discovered Exploiting ...	https://thehackernews.com/2025/08/malicious-pypi-and-npm-packages.html	T3
A compromised npm package today doesn't just steal a token	https://www.linkedin.com/posts/safedep_a-compromised-npm-package-to...	T3
Multiple Malicious Packages Discovered on PyPI, npm ... - Cloudsmith	https://cloudsmith.com/blog/multiple-malicious-packages-discovered-...	T3
Malicious npm Packages Impersonate Flashbots, Steal Ethereum ...	https://thehackernews.com/2025/09/malicious-npm-packages-impersonat...	T3

Generated 2026-04-22 13:40 UTC by TJS Security Command Center