INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-05 05:57 UTC

Strapi Plugin Impersonation Campaign Deploys Eight-Stage Attack Chain Targeting Cryptocurrency Platforms via npm Supply Chain

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0150
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	npm registry (36 malicious packages), Strapi CMS, Redis, PostgreSQL, Docker, Kubernetes, GitHub Actions, PyPI, VS Code extensions, Polymarket, bittensor-wallet, express-session, mgc (npm package)
Published	2026-04-05T01:07:00
Discovery Source	Rss

Executive Summary

A coordinated threat actor published 36 malicious npm packages impersonating legitimate Strapi CMS plugins, targeting organizations in the cryptocurrency and digital assets sectors. The campaign deployed an eight-stage attack chain capable of remote code execution, container escape, credential harvesting, and cryptocurrency wallet theft, specifically targeting Polymarket and bittensor-wallet assets. Organizations using Strapi CMS with npm-sourced plugins, particularly those operating crypto trading or DeFi platforms, face direct risk of infrastructure compromise and financial asset loss.

Technical Analysis

SafeDep researchers identified 36 malicious npm packages uploaded across a 13-hour window by four coordinated accounts mimicking legitimate Strapi CMS plugin naming conventions (typosquatting/masquerading, MITRE T1036.001). The attack chain progressed through eight stages, adapting in near-real-time as earlier payloads failed, indicating the attacker actively monitored execution outcomes and adapted payloads in response. Payload capabilities included: JavaScript-based command execution (T1059.007), Redis RCE, PostgreSQL exploitation (T1190), Docker container escape (T1611), persistent implant deployment (T1543, T1505.003), credential harvesting from files and environment variables (T1552.001, T1552.007), exfiltration via HTTP (T1071.001, T1041), and cron-based persistence (T1053.003). The presence of hard-coded target database credentials and hostname ('prod-strapi') within the malicious payload indicates

prior reconnaissance or initial access; the attacker had obtained environment-specific details before deploying this campaign (CWE-798). Cryptocurrency wallet extraction targeted Polymarket and bittensor-wallet integrations. Supply chain insertion was via npm registry (T1195.001). Related weaknesses: CWE-250 (excessive privileges), CWE-506 (embedded malicious code), CWE-829 (inclusion of untrusted functionality), CWE-732 (incorrect permission assignment). No CVE assigned. No patch available; threat is supply chain insertion, not a vendor vulnerability. The complete list of 36 malicious package names is available in the SafeDep threat research report (https://safedep.io/malicious-npm-strapi-plugin-events-c2-agent). Affected ecosystem: npm, Strapi CMS, Redis, PostgreSQL, Docker, Kubernetes, GitHub Actions. Source quality is moderate (0.64); primary technical reporting from SafeDep. No official CISA or law enforcement advisory currently available.

Action Checklist

1. Step 1: Containment. Immediately audit all installed npm packages in Strapi CMS deployments for packages matching Strapi plugin naming patterns not sourced from the official Strapi npm organization (@strapi/*). Cross-reference installed package names and versions against the 36 malicious packages identified by SafeDep (see https://safedep.io/malicious-npm-strapi-plugin-events-c2-agent for the package list). Isolate any host matching the hostname 'prod-strapi' from network egress pending investigation. Block outbound connections to unknown external hosts from Strapi application servers.
2. Step 2: Detection. Search npm audit logs and package-lock.json or yarn.lock files for unrecognized Strapi-adjacent package names installed from accounts other than the official Strapi organization. Query endpoint and container logs for cron job creation events, new user account creation (T1136), and outbound HTTP POST activity from Strapi processes. Check environment variables and file system for exfiltrated credentials. Review Redis, PostgreSQL, and Docker daemon logs for unexpected command execution or privilege escalation. Look for process enumeration (T1057) and file discovery (T1083) activity originating from the Node.js process.
3. Step 3: Eradication. Remove all identified malicious packages using 'npm uninstall ' and purge from node_modules. Rotate all database credentials (Redis, PostgreSQL) and API keys accessible from the compromised environment. Revoke and regenerate any secrets stored in environment variables on affected hosts. Remove any cron jobs, backdoor accounts, or persistent implants identified during detection. Rebuild affected containers from verified base images rather than patching in place.
4. Step 4: Recovery. Validate package integrity post-remediation using npm audit and cross-reference against the npm public registry provenance data. Confirm no unauthorized outbound network sessions remain active. Re-deploy Strapi instances from clean infrastructure using locked, verified dependency manifests. Monitor Redis, PostgreSQL, and Docker daemon activity baselines for 30 days post-recovery for signs of residual persistence. Verify cryptocurrency wallet integrations (Polymarket, bittensor-wallet) for unauthorized transaction history.
5. Step 5: Post-Incident. Implement npm package allowlisting or private registry mirroring to prevent unauthorized package installation. Enforce GitHub Actions trusted publishing workflows to require provenance attestation on published packages. Apply least-privilege principles to Strapi application runtime accounts (CWE-250 remediation). Integrate software composition analysis (SCA) tooling into CI/CD pipelines to flag new or unrecognized transitive dependencies before deployment. Conduct a dependency review for all other projects consuming npm packages in your environment.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to executive leadership, legal counsel, and applicable regulatory bodies if forensic evidence confirms exfiltration of Polymarket or bittensor-wallet private keys, database credential sets, or any PII processed by the Strapi CMS — or if the container escape stages of the eight-stage chain are confirmed to have succeeded, indicating lateral movement beyond the initial Strapi host into broader Kubernetes or cloud infrastructure.
Recovery Notes	Rebuild all affected Strapi instances from verified base container images using a pinned, hash-locked package-lock.json; do not attempt in-place remediation of compromised node_modules, as the eight-stage chain's post-install scripts may have written persistent payloads outside the npm dependency tree. Monitor Redis MONITOR output, PostgreSQL pg_stat_activity, and Docker daemon audit logs at 15-minute intervals for the first 72 hours post-recovery, then daily for 30 days, specifically alerting on any EVAL, CONFIG SET, or COPY TO commands not matching the Strapi application's known query patterns. Continuously query Polymarket and bittensor-wallet transaction histories for 90 days post-recovery, as cryptocurrency theft may be delayed or staged after initial credential harvesting to evade immediate detection.
Forensic Artifacts	node_modules//package.json and associated post-install script files: primary evidence of the initial infection vector, revealing the specific eight-stage chain entry point and any hardcoded C2 URLs or payload download locations embedded in the npm lifecycle scripts. | npm cache artifacts at ~/.npm/_cacache/ and /tmp/ directories: contain downloaded stage payloads (stages 2-8 of the chain) that may persist on disk even after package uninstall, preserving executable evidence of the full attack chain including any container escape or credential harvesting modules. | Redis MONITOR output and /var/log/redis/redis-server.log: specific evidence of C2 persistence or credential staging via Redis — this campaign's eight-stage chain would likely use Redis EVAL (Lua scripting) or keyspace manipulation for inter-stage coordination or data exfiltration staging. | Network capture (pcap) of outbound HTTP POST traffic from the Node.js Strapi process: contains encrypted or plaintext C2 beacon data, exfiltrated environment variables, and potentially harvested Polymarket/bittensor-wallet credentials transmitted to the threat actor's infrastructure during the credential harvesting stage. | Bittensor-wallet JSON keystore files and Polymarket API key references in environment variables or ~/.config/ paths: direct evidence of targeted credential access (the campaign specifically targets these assets), with filesystem atime/mtime metadata revealing the exact moment the malicious package accessed these files.

Per-Action IR Details

Step 1: Containment — Immediately audit all installed npm packages in Strapi CMS deployments for packages matching Strapi plugin naming patterns not sourced from the official Strapi npm organization (@strapi/*). Cross-reference installed package names and versions against the 36 malicious packages identified by SafeDep (see https://safedep.io/malicious-npm-strapi-plugin-events-c2-agent for the package list). Isolate any host matching the hostname 'prod-strapi' from network egress pending investigation. Block outbound connections to unknown external hosts from Strapi application servers.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SC-7 (Boundary Protection), NIST CM-8 (System Component Inventory), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 4.4 (Implement and Manage a Firewall on

Servers)

Compensating: Run 'cat package.json package-lock.json | grep -E "strapi" | sort -u' on each Strapi host to extract all Strapi-adjacent package names, then diff against the known-good @strapi/* scoped package list. Use 'npm ls --all 2>/dev/null | grep -v "@strapi/"' to surface unscoped impostor packages. For network egress blocking on Linux hosts without a managed firewall, use 'iptables -I OUTPUT -p tcp -m owner --uid-owner -j LOG --log-prefix "STRAPI-EGRESS: "' followed by a DROP rule to isolate the Node.js process's outbound traffic immediately while preserving evidence.

Evidence: Before isolating 'prod-strapi' from the network, capture a full memory dump of the Node.js Strapi process using 'gcore $(pgrep -f strapi)' to preserve in-memory C2 callback URLs, decrypted credentials, and any stage-payload code loaded without touching disk. Snapshot active network connections with 'ss -tnpH | grep node' to document live C2 sessions before egress is cut. Preserve the unmodified node_modules directory tree ('tar czf node_modules_evidence.tar.gz ./node_modules') and the package-lock.json as forensic baselines before any uninstall operations. Capture running cron entries with 'crontab -l -u ' and 'cat /etc/cron.d/*' to document persistence before eradication. Export current environment variables for the Strapi process via '/proc/$(pgrep -f strapi)/environ | tr "\0" "\n" > env_evidence.txt' to capture any credentials already loaded into the process environment.

Step 2: Detection — Search npm audit logs and package-lock.json or yarn.lock files for unrecognized Strapi-adjacent package names installed from accounts other than the official Strapi organization. Query endpoint and container logs for cron job creation events, new user account creation (T1136), and outbound HTTP POST activity from Strapi processes. Check environment variables and file system for exfiltrated credentials. Review Redis, PostgreSQL, and Docker daemon logs for unexpected command execution or privilege escalation. Look for process enumeration (T1057) and file discovery (T1083) activity originating from the Node.js process.

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-2 (Event Logging), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs), MITRE ATT&CK T1136 (Create Account), MITRE ATT&CK T1057 (Process Discovery), MITRE ATT&CK T1083 (File and Directory Discovery), MITRE ATT&CK T1059.007 (JavaScript for Automation)

Compensating: Deploy the following targeted detection queries without a SIEM: (1) Scan for suspicious child processes of Node.js with 'ps -eo pid,ppid,comm,args | awk \'$2==$(pgrep -f strapi) && $3!="node"\'' to find shell spawns from the Strapi process consistent with T1059 stage execution. (2) Search for new local accounts created post-install date of suspicious packages with 'lastlog | grep -v "Never"' and 'grep -E "useradd|adduser" /var/log/auth.log'. (3) Detect outbound HTTP POST from Node.js using 'tcpdump -i any -nn -A "src host and tcp port 80 or tcp port 443" -w strapi_egress.pcap' for offline analysis in Wireshark. (4) Query Redis command log via 'redis-cli MONITOR' (run briefly) or parse '/var/log/redis/redis-server.log' for CONFIG SET, SLAVEOF, or EVAL commands not issued by the application. (5) Check PostgreSQL logs at '/var/lib/postgresql/*/main/pg_log/' for COPY TO, pg_read_file(), or CREATE EXTENSION commands originating from the Strapi DB user.

Evidence: Pull the npm install timestamp from package-lock.json ('jq '.packages | to_entries[] | select(.key | test("strapi")) | {(.key): .value.resolved}' package-lock.json') to establish a precise compromise window for log correlation. Extract all outbound HTTP connections from the Strapi process during the compromise window from '/proc//net/tcp' or system-level network logs, correlating destination IPs against known-clean Strapi CDN/registry endpoints. Capture the full Redis slow log ('redis-cli SLOWLOG GET 128') and PostgreSQL pg_stat_activity snapshot to identify anomalous queries issued during the suspected exploitation window. Retrieve Docker daemon logs ('journalctl -u docker --since --until now') for unexpected container exec events, privileged container launches, or volume mounts consistent with container escape techniques. Check '/home/*/.bash_history' and '/root/.bash_history' for commands consistent with bittensor-wallet or Polymarket credential file enumeration (e.g., 'find / -name "*.json" -path "*/bittensor/*"').

Step 3: Eradication — Remove all identified malicious packages using 'npm uninstall ' and purge from node_modules. Rotate all database credentials (Redis, PostgreSQL) and API keys accessible from the

compromised environment. Revoke and regenerate any secrets stored in environment variables on affected hosts. Remove any cron jobs, backdoor accounts, or persistent implants identified during detection. Rebuild affected containers from verified base images rather than patching in place.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST SI-3 (Malicious Code Protection), NIST CM-8 (System Component Inventory), NIST IA-5 (Authenticator Management), CIS 2.3 (Address Unauthorized Software), CIS 5.3 (Disable Dormant Accounts)

Compensating: Before uninstalling, compute SHA-256 hashes of all malicious package files for forensic record: 'find node_modules// -type f -exec sha256sum {} \; > malicious_pkg_hashes.txt'. Use 'npm uninstall --save' for each of the 36 identified packages, then verify removal with 'npm ls 2>&1 | grep '. For credential rotation without a secrets manager, use 'openssl rand -hex 32' to generate new PostgreSQL and Redis passwords, update via 'psql -c "ALTER USER strapi PASSWORD \'\';"' and 'redis-cli CONFIG SET requirepass ', then push updates to application config files with strict 600 permissions. Verify no residual cron jobs remain with 'for user in $(cut -f1 -d: /etc/passwd); do crontab -u $user -l 2>/dev/null && echo $user; done'.

Evidence: Preserve forensic copies of all identified malicious package directories BEFORE uninstalling — specifically the post-install scripts ('package.json' scripts.postinstall, scripts.install fields) that triggered the eight-stage chain, and any downloaded stage payloads cached in '/tmp/', '~/.npm/_cacache/', or custom download paths written by the malicious package. Capture a diff of /etc/passwd and /etc/shadow before account removal to document backdoor account attributes. Export the full list of active Redis keys ('redis-cli KEYS "*"') and any suspicious Lua scripts ('redis-cli SCRIPT LIST') before flushing, as the eight-stage chain may have used Redis for C2 persistence or credential staging. Document exact cron job text verbatim before deletion, including the command, schedule, and owner, as this constitutes primary persistence evidence for post-incident reporting.

Step 4: Recovery — Validate package integrity post-remediation using npm audit and cross-reference against the npm public registry provenance data. Confirm no unauthorized outbound network sessions remain active. Re-deploy Strapi instances from clean infrastructure using locked, verified dependency manifests. Monitor Redis, PostgreSQL, and Docker daemon activity baselines for 30 days post-recovery for signs of residual persistence. Verify cryptocurrency wallet integrations (Polymarket, bittensor-wallet) for unauthorized transaction history.

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 SI-2 (Flaw Remediation), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST CP-10 (System Recovery and Reconstitution), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Validate clean deployment integrity by running 'npm audit --audit-level=high' on the rebuilt environment and cross-referencing each Strapi-adjacent package's '_resolved' URL in package-lock.json against 'https://registry.npmjs.org/' to confirm the tarball hash matches registry provenance. Use 'ss -tnp state established' on the rebuilt host to confirm zero unauthorized egress sessions before returning to production. For 30-day Polymarket and bittensor-wallet transaction monitoring without a dedicated blockchain analytics tool, use the public APIs (Polymarket's CLOB API and bittensor-wallet's on-chain explorer) to programmatically query transaction history for each wallet address associated with the compromised environment, alerting on any transaction not initiated by the application service account. Script this check to run daily via cron.

Evidence: Before re-enabling production traffic, capture a clean-state baseline snapshot of all active network connections ('ss -tnpH > clean_baseline_connections.txt'), running processes ('ps auxf > clean_baseline_processes.txt'), and loaded npm packages ('npm ls --all > clean_baseline_packages.txt') from the rebuilt Strapi host for future drift comparison. Retrieve npm provenance attestation records for each reinstalled package using 'npm view @ dist.integrity' and compare against the package-lock.json 'integrity' field to verify no tampering in transit. Document the exact blockchain transaction IDs and timestamps of the last known-good

Polymarket and bittensor-wallet transactions prior to the compromise window as the forensic baseline for unauthorized activity review.

Step 5: Post-Incident — Implement npm package allowlisting or private registry mirroring to prevent unauthorized package installation. Enforce GitHub Actions trusted publishing workflows to require provenance attestation on published packages. Apply least-privilege principles to Strapi application runtime accounts (CWE-250 remediation). Integrate software composition analysis (SCA) tooling into CI/CD pipelines to flag new or unrecognized transitive dependencies before deployment. Conduct a dependency review for all other projects consuming npm packages in your environment.

NIST Phase: Post Incident

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

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST SA-15 (Development Process, Standards, and Tools), NIST CM-7 (Least Functionality), NIST AC-6 (Least Privilege), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Configure a local Verdaccio npm registry mirror ('npm install -g verdaccio') as a package allowlist proxy — set '.npmrc' to 'registry=http://localhost:4873' and configure Verdaccio to only proxy the official @strapi/* scoped packages plus an explicit allowlist of vetted dependencies, rejecting all others. For SCA without a commercial tool, add 'npx better-npm-audit --level high --fail-on-missing' and 'node --experimental-vm-modules node_modules/.bin/socket' (Socket.dev CLI, free tier) as mandatory steps in the CI/CD pipeline before any npm install. For least-privilege remediation of the Strapi runtime: create a dedicated OS service account with 'useradd -r -s /sbin/nologin strapi-svc', remove write access to node_modules at runtime with 'chmod -R a-w ./node_modules' post-install, and drop Linux capabilities on the Strapi process by adding 'CapabilityBoundingSet=~CAP_SYS_ADMIN CAP_NET_ADMIN CAP_SETUID' to the systemd unit file.

Evidence: Compile the full incident timeline correlating npm package install timestamps from package-lock.json with the first observed C2 beacon, credential access events, and any detected cryptocurrency wallet queries — this timeline is the primary artifact for the post-incident lessons-learned review and any required regulatory disclosure. Preserve the safedep.io IOC list, all captured pcap files, memory dumps, and malicious package archives in an evidence repository with SHA-256 hashes and chain-of-custody documentation per NIST IR-5 (Incident Monitoring). Document the specific eight-stage attack chain stages observed in your environment (which stages executed, which failed, what payloads were delivered) as input for writing environment-specific YARA rules and Sigma detection rules targeting this campaign's indicators in future deployments.

Detection Guidance

Primary behavioral indicators: Node.js or npm postinstall scripts spawning shell commands (T1059.004, T1059.007); outbound HTTP/S connections from Strapi application processes to non-whitelisted external hosts (T1071.001); cron job creation under application service accounts (T1053.003); new local or container user account creation (T1136); Redis or PostgreSQL receiving commands outside normal application query patterns (T1190); credential harvesting enabling potential valid account usage (T1078). IOC patterns: hostname 'prod-strapi' in network traffic or configuration; packages with Strapi-adjacent names from non-@strapi npm publisher accounts; environment variable access attempts for database credentials and wallet keys (T1552.001, T1552.007). Log sources to query: npm install logs, container runtime logs (Docker/Kubernetes event logs), cron/systemd logs, Redis MONITOR output, PostgreSQL pgaudit logs, network flow logs for unexpected egress from app servers. SIEM query approach: alert on any child process spawned by node with arguments containing 'bash', 'sh', 'curl', 'wget', or base64-encoded strings (example Splunk SPL: parent_process=node.exe (child_process=bash OR child_process=sh OR child_process=curl OR child_process=wget)). SCA tools should flag packages published by accounts not matching established Strapi organization publisher history.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`prod-strapi (hostname)`	Hard-coded target hostname found in malicious payloads; indicates a pre-identified victim environment. Presence of this hostname in internal infrastructure warrants immediate investigation.	HIGH
URL	`https://safedep.io/malicious-npm-strapi-plugin-events-c2-agent`	SafeDep primary research report; contains full list of 36 malicious package names and technical payload analysis. Treat as reference for package blocklist — verify at source.	MEDIUM
DOMAIN	`Polymarket (wallet integration target)`	Cryptocurrency platform explicitly targeted for wallet credential extraction. Environments with Polymarket API integrations should treat this as a high-priority exposure indicator.	HIGH
DOMAIN	`bittensor-wallet (npm package integration target)`	bittensor-wallet npm package explicitly targeted for credential and key extraction. Environments using bittensor-wallet should audit for unauthorized access.	HIGH

Framework Mappings

MITRE-ATTACK

- **T1059.007** — JavaScript
- **T1543** — Create or Modify System Process
- **T1041** — Exfiltration Over C2 Channel
- **T1190** — Exploit Public-Facing Application
- **T1036.001** — Invalid Code Signature
- **T1078** — Valid Accounts
- **T1053.003** — Cron
- **T1105** — Ingress Tool Transfer
- **T1611** — Escape to Host
- **T1552.001** — Credentials In Files
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1136** — Create Account
- **T1059.004** — Unix Shell

- **T1057** — Process Discovery
- **T1071.001** — Web Protocols
- **T1505.003** — Web Shell
- **T1083** — File and Directory Discovery
- **T1059.006** — Python
- **T1552.007** — Container API

NIST-800-53R5

- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-4** — System Monitoring
- **CA-8** — Penetration Testing
- **RA-5** — Vulnerability Monitoring and Scanning
- **SI-2** — Flaw Remediation
- **SI-7** — Software, Firmware, and Information Integrity
- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **SI-3** — Malicious Code Protection
- **CM-7** — Least Functionality
- **CM-2** — Baseline Configuration
- **AC-3** — Access Enforcement
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A01:2021** — Broken Access Control
- **A07:2021** — Identification and Authentication Failures

CIS-V8

- **5.4** — Restrict Administrator Privileges to Dedicated Administrator Accounts
- **6.8** — Define and Maintain Role-Based Access Control
- **16.10** — Apply Secure Design Principles in Application Architectures
- **3.3** — Configure Data Access Control Lists
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers

ISO-27001-2022

- **A.8.28** — Secure coding
- **A.5.34** — Privacy and protection of personal information
- **A.5.21** — Managing information security in the ICT supply chain

HIPAA-SECURITY

- **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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1059.007	JavaScript	Execution
T1543	Create or Modify System Process	Persistence
T1041	Exfiltration Over C2 Channel	Exfiltration
T1190	Exploit Public-Facing Application	Initial-Access
T1036.001	Invalid Code Signature	Defense-Evasion
T1078	Valid Accounts	Defense-Evasion
T1053.003	Cron	Execution
T1105	Ingress Tool Transfer	Command-And-Control
T1611	Escape to Host	Privilege-Escalation
T1552.001	Credentials In Files	Credential-Access
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1136	Create Account	Persistence
T1059.004	Unix Shell	Execution
T1057	Process Discovery	Discovery
T1071.001	Web Protocols	Command-And-Control
T1505.003	Web Shell	Persistence
T1083	File and Directory Discovery	Discovery
T1059.006	Python	Execution
T1552.007	Container API	Credential-Access

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/04/36-malicious-npm-packages-exploit...	T3
Someone is actively publishing malicious packages targeting the ...	https://www.reddit.com/r/programming/comments/1sbkx3b/someone_is_ac...	T3
Thirty-Six Malicious npm Strapi Packages Deploy Redis RCE ...	https://safedep.io/malicious-npm-strapi-plugin-events-c2-agent	T3
Malicious npm Packages Disguised as Strapi Plugins Enable Data ...	https://dev.to/kornilovconstru/malicious-npm-packages-disguised-as-...	T3
Security suggestion for trusted publishing in GitHub Actions #191125	https://github.com/orgs/community/discussions/191125	T3

Generated 2026-04-05 05:57 UTC by TJS Security Command Center