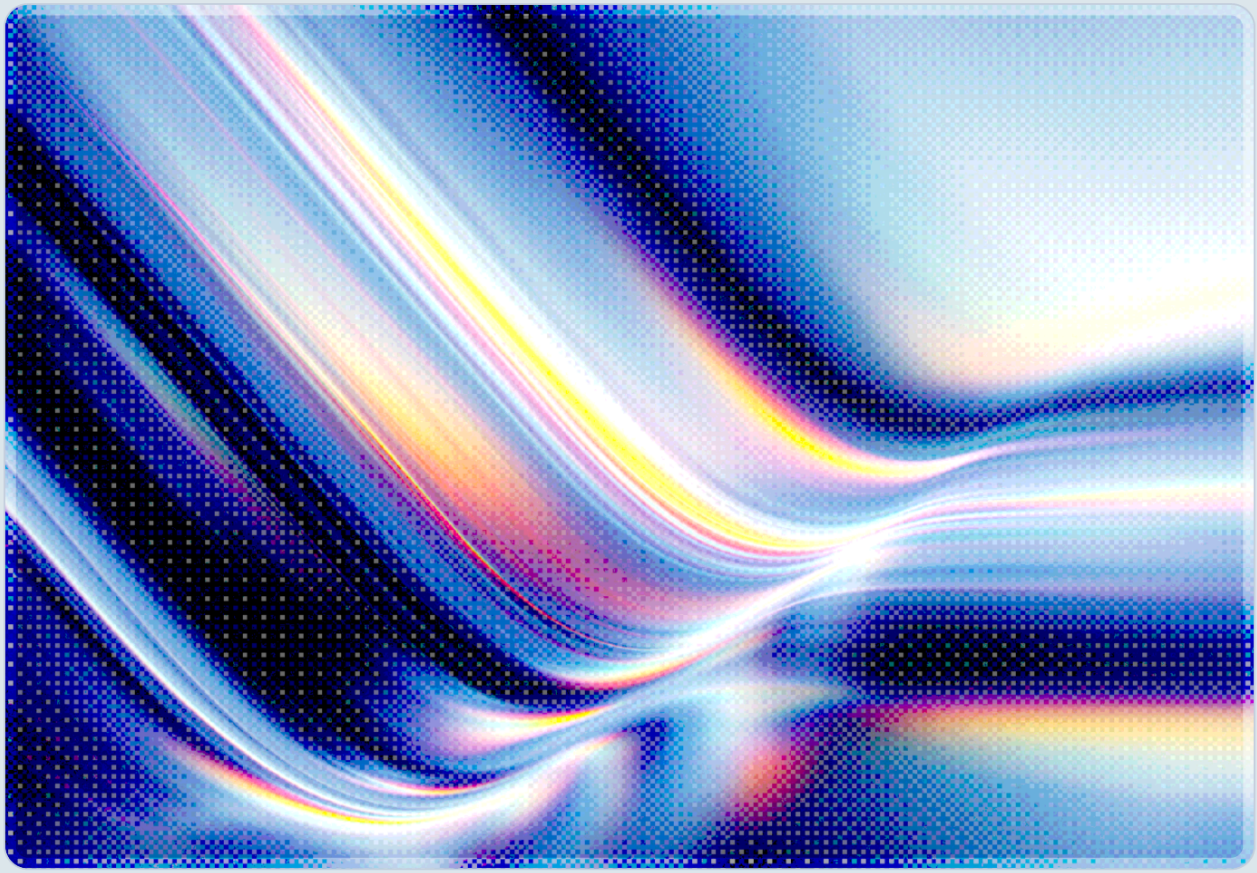
CSAI Foundation | Cloud Security Alliance

Mastra npm Takeover: 144 AI Packages Backdoored

Supply Chain Compromise via Hijacked Contributor Account Targets AI Development Environments

2026-06-18

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

© 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

- On June 17, 2026, an attacker hijacked a dormant former contributor account (ehindero) and used it to republish 144 packages across the entire @mastra npm scope within an 88-minute window, injecting a malicious dependency named easy-day-js into each package.
- The malicious dependency impersonated the legitimate dayjs date library, used a postinstall hook to drop an obfuscated loader, downloaded a second-stage remote access trojan (RAT) from a raw IP address, and then self-deleted to remove forensic traces.
- The RAT targets credentials specific to AI development pipelines — LLM API keys (OPENAI_API_KEY, ANTHROPIC_API_KEY, GOOGLE_API_KEY), cloud provider tokens, CI/CD secrets, and database connection strings — in addition to cryptocurrency wallet browser extensions.
- Any developer workstation, CI runner, or build environment that executed `npm install` against an affected @mastra package between 01:01 UTC on June 17 and 02:39 UTC on June 17, 2026 must be treated as potentially compromised and all credentials rotated immediately.
- The attack exploited two structural weaknesses in the npm registry: scope publish permissions are not revoked on inactivity, and caret-range dependency resolution silently upgrades consumers to new versions without any change to upstream source repositories.
- Tradecraft similarities — particularly the staging of a clean decoy package followed by malicious version injection — overlap with techniques attributed to Sapphire Sleet (BlueNoroff), a North Korean APT known for developer-targeting cryptocurrency theft operations, though attribution to this specific campaign remains unconfirmed.

Background

Mastra is an open-source JavaScript and TypeScript framework for building AI-powered applications, including autonomous agents, retrieval-augmented generation (RAG) pipelines, and multi-step workflows. The framework is developed and maintained under the @mastra npm scope and has become a dependency for a significant segment of the AI developer ecosystem, with @mastra/core alone exceeding 918,000 weekly downloads [1].

The attack vector entered through easy-day-js, a package published to npm on June 16, 2026 at 07:05 UTC by the account "sergey2016." When first published, easy-day-js was a functionally complete and clean copy of the popular dayjs date-formatting library, which receives more than 50 million weekly downloads [2]. This clean initial publication is consistent with a deliberate staging technique: it allows the package to accumulate an innocuous version history and may help it pass static analysis tools that inspect registry metadata. At 01:01 UTC on June 17, the account modified easy-day-js, publishing version 1.11.22 with an obfuscated postinstall dropper concealed inside what otherwise appeared to be a routine dayjs clone [3].

The compromised npm account — ehindero — was listed as a legitimate maintainer of the entire @mastra scope. Critically, the account belonged to a former contributor whose publish permissions were never revoked after they were no longer actively involved in the project. The npm registry provides no automated mechanism to expire scope-level publish access for inactive accounts, meaning the account remained a valid credential for mass publishing without any ongoing activity to signal its risk [4][5]. Between 01:12 and 02:39 UTC on June 17, the ehindero account executed an automated publishing campaign, releasing new versions of 144 @mastra packages within 88 minutes [1] (vendor counts range from 141 to 144, reflecting differences in enumeration timing), each seeded with easy-day-js as a new dependency [6].

Security Analysis

Attack Mechanics and Staging

The Mastra compromise followed a two-account, two-phase staging strategy that is characteristic of deliberate, multi-stage supply chain operations. The "sergey2016" account prepared the malicious payload by publishing and maturing easy-day-js independently, creating a clean version history that would not immediately flag anomaly-detection systems. The "ehindero" account, operating with pre-existing legitimate authority over the @mastra scope, then served as the publishing vehicle — separating credential access from weaponization in a way that complicates attribution and detection [5][7].

The npm caret-range dependency declaration (e.g., `^1.11.0`) is central to why this technique succeeds at scale. When package authors declare dependencies using caret notation, npm resolves to the latest compatible minor or patch version at install time. This means consumers who had previously installed clean versions of @mastra packages would automatically receive easy-day-js@1.11.22 on their

next `npm install` without any change to the @mastra source repository, without a new commit to review, and without any visible indication in the dependency graph that a third-party dependency had changed behavior [3][6].

Payload Delivery Chain

Once easy-day-js@1.11.22 was installed, a postinstall script executed automatically — a standard npm feature that package authors use to compile native bindings or run setup routines, and which attackers exploit because it fires without user interaction. The postinstall hook was a 4,572-byte obfuscated dropper that performed several operations in sequence: it disabled TLS certificate verification, established a connection to attacker-controlled command-and-control (C2) infrastructure at the raw IP address 23.254.164[.]92, retrieved a second-stage payload over HTTPS using a self-signed certificate, launched the payload as a detached background process with no console output, and then deleted itself from disk [4][7].

The second stage is a 41-kilobyte obfuscated Node.js script described by multiple security vendors as a cross-platform remote access trojan. On first execution, the RAT beacons to the C2 server with a reconnaissance profile that includes the username, hostname, operating system and architecture, Node.js version, installed application inventory, running process list, browser history hostnames, and an inventory of cryptocurrency wallet browser extension IDs matched against a list of 166 known wallet identifiers [3][7]. This reconnaissance profile is immediately actionable for attackers: it identifies developer machines with access to high-value credentials and cryptocurrency assets, enabling prioritized follow-on operations against the most valuable targets.

Credential Targets in AI Development Environments

The payload's targeting profile closely matches the credential ecosystem present in AI application development environments. Build environments and developer workstations running Mastra-based applications commonly hold credentials that represent significant financial and operational risk: LLM API keys for services including OpenAI, Anthropic, and Google; cloud provider credentials for AWS, GCP, and Azure; CI/CD tokens with repository-write or deployment authority; database connection strings; and cryptocurrency wallet credentials [1][3]. The combination of these credential classes in a single developer environment reflects the reality of modern AI development stacks, where building an intelligent agent requires simultaneously managing model access, infrastructure, data, and often on-chain value transfer.

The RAT further establishes OS-level login persistence on Windows, macOS, and Linux, ensuring re-execution after system reboot even if the malicious npm packages are subsequently removed [4]. It also opens a remote module execution channel, enabling the attacker to dynamically load and run additional payloads without reinfection — a capability that allows the campaign to expand its footprint or adjust objectives after initial access [7].

npm Registry Structural Weaknesses Exploited

This incident simultaneously exploited two structural weaknesses in the npm ecosystem that are not unique to Mastra. First, the npm registry does not expire or audit scope-level publish permissions when an account becomes inactive. Contributor churn is a normal part of open source project maintenance, but the access rights accumulated during a contributor's active period persist indefinitely unless explicitly revoked by a project owner. This creates a long tail of valid but unmonitored credential holders who represent latent takeover risk for every project they once contributed to [4][5].

Second, npm's dependency resolution model — which the entire JavaScript ecosystem depends on — prioritizes developer convenience through automatic minor-version upgrades while creating an implicit trust chain between a package and every future version of its transitive dependencies. A project's source code can remain entirely unchanged while the effective software its users install diverges significantly from what developers reviewed and tested. The Mastra attack required no compromise of Mastra's GitHub repository, no merge of a malicious pull request, and no change that any repository-scanning tool would have detected [3][6].

Attribution Context

Multiple security vendors noted tradecraft overlaps between the easy-day-js campaign and techniques previously attributed to Sapphire Sleet (BlueNoroff), a North Korean threat actor tracked by Microsoft that conducts developer-targeting operations to fund state programs through cryptocurrency theft [3]. The similarities include the use of a staging account to mature a clean decoy package before weaponizing it, the cross-platform Node.js payload architecture, and the emphasis on cryptocurrency wallet credential harvesting. Microsoft Threat Intelligence attributed an earlier 2026 Axios npm compromise to Sapphire Sleet using closely related techniques [8]. Attribution for the Mastra-specific campaign has not been publicly confirmed as of this writing, and organizations should treat the assessment as an intelligence indicator rather than a settled fact.

Recommendations

Immediate Actions

Organizations that use or used any @mastra npm package — including within CI/CD pipelines, cloud build systems, or developer workstations — should treat any environment that executed `npm install` between 01:01 and 02:39 UTC on June 17, 2026 as potentially compromised, regardless of whether they observed anomalous behavior. The RAT's self-deletion and detached background execution mean that affected systems are unlikely to show obvious symptoms. Compromise assessment should proceed by auditing npm install logs and lock files from the period in question and checking for the presence of easy-day-js in the dependency tree; both Endor Labs and Orca Security have published indicators of compromise (IOCs) including the C2 IP address 23.254.164[.]92 [4][1].

All credentials that may have been accessible during affected builds or sessions should be rotated immediately and treated as stolen rather than merely at risk. This includes npm publish tokens, GitHub and CI/CD tokens, cloud provider access keys (AWS, GCP, Azure), LLM service API keys (OpenAI, Anthropic, Google AI), database connection strings, and SSH keys present on affected systems [6]. Cryptocurrency funds held in browser extension wallets that were present on affected machines should be transferred to new wallets generated on verified-clean devices; the RAT's wallet inventory capability makes affected wallets high-risk for draining [3].

Persistence artifacts installed by the second-stage RAT should be removed. On affected systems, this requires reviewing OS-level startup mechanisms — scheduled tasks and registry run keys on Windows, LaunchAgents and LaunchDaemons on macOS, systemd units and cron jobs on Linux — for any entries referencing Node.js processes that were not deliberately installed [4].

Short-Term Mitigations

Package lock files and software bill of materials (SBOM) practices are the most immediately actionable structural control for preventing this class of attack. Committing and enforcing `package-lock.json` or `yarn.lock` files in CI/CD pipelines pins transitive dependency versions, preventing caret-range resolution from silently upgrading to newly published malicious versions. Automated SBOM generation on each build creates an auditable record of the exact dependency versions actually installed, making it possible to quickly assess exposure when new supply chain incidents are disclosed [9].

Dependency monitoring tools from vendors including Snyk, Endor Labs, OX Security, and Socket [13] should be enabled with real-time alerts for newly published versions of any package in the dependency graph. These tools can detect anomalies such as the sudden introduction of a postinstall hook in a package version that previously lacked one — the specific signal that easy-day-js@1.11.22 would have produced in a monitored environment [5][10].

CI/CD pipelines should be configured to use network egress controls that alert or block outbound connections to raw IP addresses from build processes. The easy-day-js dropper contacted C2 infrastructure at a bare IP address rather than a domain name; a network policy that logs or blocks outbound HTTPS connections from build jobs to IP addresses without associated hostnames would have detected the anomalous connection at execution time [7].

Strategic Considerations

The npm registry's permission model requires organizational attention beyond per-incident response. Project owners should audit their package scopes and revoke publish access for any account that is no longer actively contributing to the project. This is not a one-time exercise: contributor churn makes this a recurring governance obligation. The npm registry supports granular publish-access controls, and well-governed projects should restrict publish rights to the minimum set of currently active maintainers and consider adopting provenance attestation features that link published packages to specific repository commits [5].

This incident is consistent with an emerging pattern of attackers targeting AI development toolchains — a trend illustrated by the earlier Microsoft-attributed Sapphire Sleet Axios campaign [8] and warranting supply chain security attention commensurate with the value of the credentials these environments hold. The credential classes present in a typical AI development environment — model API keys, cloud credentials, vector database access, and CI/CD deployment authority — collectively represent a higher-value target than the session tokens and user credentials typical of web application builds, and often carry direct production access. Organizations building on open-source AI frameworks should apply the same supply chain rigor to their AI dependencies that they apply to security-critical infrastructure code.

CSA Resource Alignment

This incident maps directly to several areas of the CSA AI Safety Initiative's existing framework guidance. The MAESTRO threat modeling framework for agentic AI systems identifies the development and deployment supply chain as an attack surface requiring explicit threat modeling; the Mastra compromise

demonstrates that threat at scale, showing how a single compromised npm account can inject adversary code into an agentic AI framework before it ever reaches a production environment [11].

CSA's AI Controls Matrix (AICM), which extends and supersedes the Cloud Controls Matrix (CCM) for AI-specific contexts, addresses dependency integrity and software supply chain risk as distinct control domains. The Mastra incident provides a concrete case for the AICM's software supply chain controls, including requirements for dependency version pinning, transitive dependency auditing, and contributor access lifecycle management. Organizations using the AICM as a compliance reference should review controls related to third-party software integrity, build environment isolation, and credential scope minimization in light of this incident.

CSA's Zero Trust guidance is directly applicable to the credential exposure dimension of this attack. A Zero Trust posture treats every credential as potentially compromised and enforces just-in-time, minimally scoped access — a model that limits the blast radius when a build environment is compromised. Specifically, LLM API keys used in CI/CD pipelines should be scoped to the minimum permissions required and should be issued as short-lived secrets rather than long-lived static credentials, reducing the value of any single credential to an attacker who exfiltrates it [12].

Reports from the security community suggest many organizations found it difficult to quickly assess whether affected @mastra versions had been installed in their build environments. This difficulty illustrates a gap that SBOM practices — which CSA has advocated as baseline supply chain hygiene [9] — are specifically designed to address, by generating a machine-readable inventory of exact dependency versions at build time. CSA's Software Transparency guidance, aligned with SBOM requirements emerging from US executive orders and international standards, establishes the audit trail capability that enables rapid exposure assessment when incidents like this occur.

References

[1] Orca Security. "144 Mastra npm Packages Compromised via Supply Chain Attack." Orca Security Blog, June 2026.

[2] SafeDep. "Mastra npm Scope Takeover: 141 Packages Drop a RAT." SafeDep, June 2026.

[3] Phoenix Security. "easy-day-js / EASY_DAY_JS_MASTRA_2026: Typosquatted Dependency Delivers Cross-Platform RAT to 144 npm Packages." Phoenix Security, June 2026.

[4] Endor Labs. "Mastra npm Org Compromised: Multiple Packages Trojanized to Drop a Remote Payload via easy-day-js." Endor Labs Blog, June 2026.

[5] Snyk. "A Forgotten Contributor Account Compromised the Entire Mastra npm Package Scope." Snyk Blog, June 2026.

[6] The Hacker News. "144 Mastra npm Packages Compromised via Hijacked Contributor Account." The Hacker News, June 17, 2026.

[7] StepSecurity. "Mastra npm Supply Chain Attack: 140+ Packages Backdoored via easy-day-js Typosquat." StepSecurity Blog, June 2026.

[8] Microsoft Threat Intelligence. "Mitigating the Axios npm Supply Chain Compromise." Microsoft Security Blog, April 1, 2026.

[9] CSA. "Software Transparency: Securing the Digital Supply Chain." Cloud Security Alliance, 2025.

[10] OX Security. "easy-day-js Supply Chain Attack Hits Mastra AI in npm." OX Security Blog, June 2026.

[11] CSA AI Safety Initiative. "Agentic AI Threat Modeling Framework: MAESTRO." Cloud Security Alliance, February 2025.

[12] CSA. "Zero Trust Guidance for Achieving Operational Resilience." Cloud Security Alliance, 2025.

[13] Socket. "140+ Mastra npm Packages Compromised in Coordinated Supply Chain Attack." Socket Security Blog, June 2026.