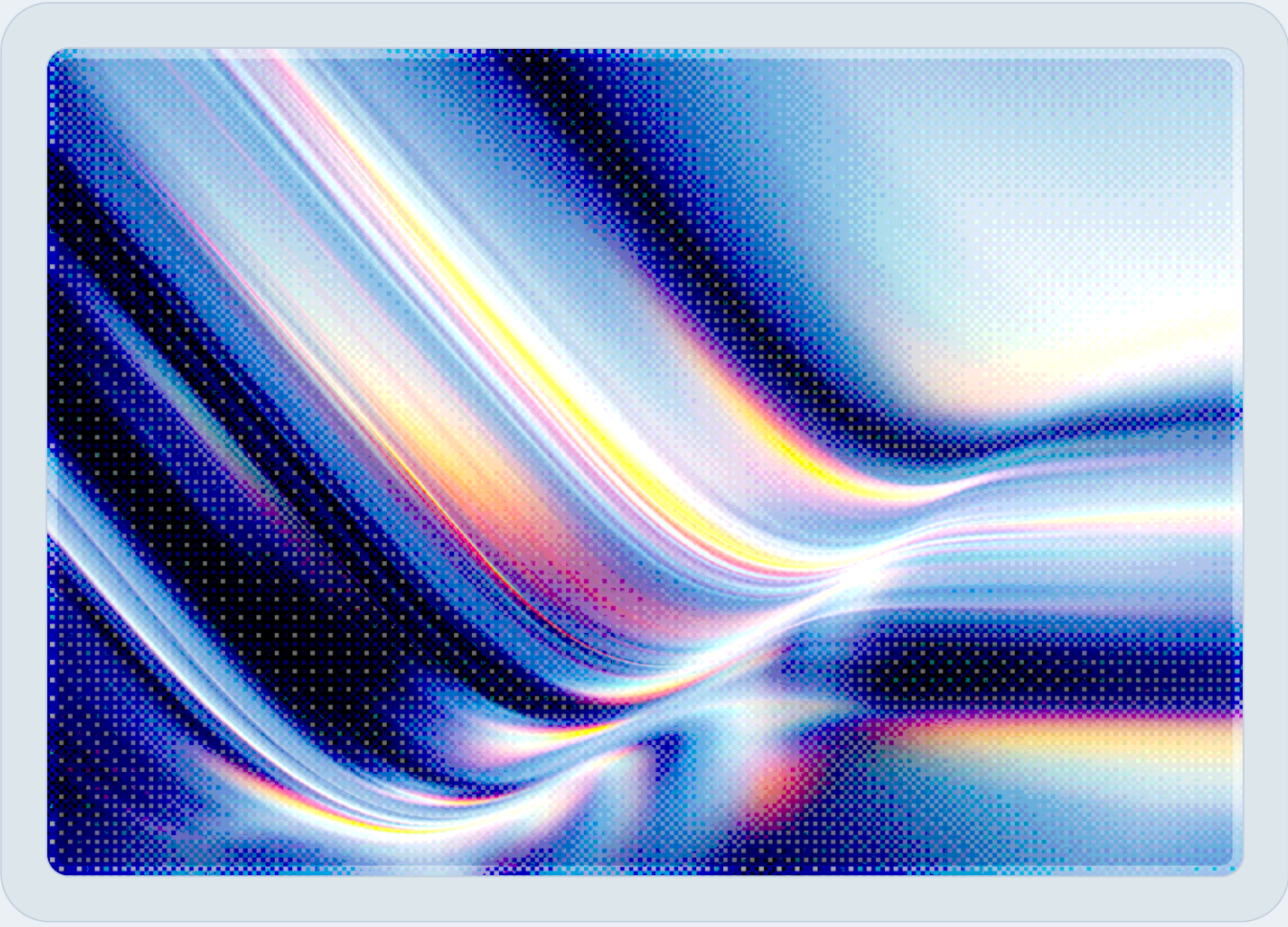
CSAI Foundation | Cloud Security Alliance

Sapphire Sleet Poisons Mastra AI npm Supply Chain

North Korean Threat Actor Backdoors 145 AI Framework Packages via Account Hijack

2026-06-22

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- On June 17, 2026, the North Korean state actor Sapphire Sleet (also tracked as BlueNoroff) compromised 145 npm packages in the Mastra AI framework ecosystem in a rapid 88-minute campaign, affecting packages with a combined weekly download count exceeding 1.1 million [1][2][3].
- The attack originated from a social engineering compromise of a legitimate Mastra contributor account, which was then used to inject a malicious typosquat dependency, "easy-day-js," across the entire @mastra scope as a production dependency [1][3].
- The multi-stage payload targeted cryptocurrency wallet browser extensions, developer credentials, browser history, and host reconnaissance data across Windows, macOS, and Linux — then delivered a persistent PowerShell backdoor granting SYSTEM-level remote access that survived reboots and package cleanup [1].
- AI developer environments represent particularly attractive targets because they tend to aggregate the credential classes that matter most to North Korean financial operations: cloud API keys, LLM API tokens, CI/CD pipeline secrets, and cryptocurrency wallet access — all potentially reachable through a single `npm install` [2][3].
- Organizations using Mastra packages at or above v1.13.1 should treat affected developer systems as potentially compromised, rotate all credentials and tokens accessible from those environments, and implement dependency integrity controls before reinstalling [1][4].

Background: The Mastra AI Ecosystem and Why It Was Targeted

Mastra is an open-source TypeScript framework for building AI agents and agentic workflows. Its modular architecture — distributed as more than a hundred discrete npm packages under the @mastra scope — has made it a popular foundation for teams integrating large language models, tool-use pipelines, and autonomous agent logic into production applications. The @mastra/core package alone receives over 918,000 weekly downloads [3], placing it among the more widely adopted AI development frameworks in the JavaScript ecosystem.

This scale and composition made Mastra a structurally attractive target for Sapphire Sleet. Developer-focused supply chain attacks have historically been selected not for their own intrinsic value but for the credential inventory that developer machines and CI/CD pipelines carry. AI developer environments represent particularly attractive targets because they tend to aggregate the credential classes that matter most to North Korean financial operations: cloud provider API keys for inference workloads, LLM service credentials (OpenAI, Anthropic, Cohere, and others), access tokens for vector databases and data pipelines, cryptocurrency wallet browser extensions used for Web3 integrations, and the elevated CI/CD tokens that control production deployments — all potentially reachable through a single `npm install` [2][3]. Because AI developer environments tend to consolidate LLM service credentials, cloud keys, and cryptocurrency tools alongside standard developer credentials, a single compromised machine in this cohort likely represents a broader credential exposure than a comparably privileged traditional software developer machine.

Sapphire Sleet, which Microsoft and other threat intelligence organizations track as BlueNoroff and link to North Korea's Reconnaissance General Bureau, has maintained a multi-year focus on cryptocurrency theft as a mechanism for generating hard currency for the regime. The Bangladesh Bank heist of 2016 is attributed directly to BlueNoroff in MITRE's ATT&CK catalogue (G0082) [5]; the Ronin Network theft and the Bybit cryptocurrency exchange breach are attributed by security researchers and

law enforcement to the broader Lazarus Group cluster — the financially motivated North Korean umbrella of which Sapphire Sleet is a subunit [5]. The April 2026 compromise of the Axios npm package, attributed to the same group, established a pattern of targeting JavaScript ecosystem packages before the Mastra campaign expanded that pattern into the AI framework space [1] [2].

The Attack: Anatomy of an 88-Minute Campaign

The operation used a two-account structure that separated the credential acquisition phase from the weaponization phase, a pattern consistent with Sapphire Sleet's operational security posture in prior campaigns.

The first account, "sergey2016@tutamail.com," published a benign version of "easy-day-js" (v1.11.21) on June 16 at 07:05 UTC. This package presented itself as a lightweight clone of the popular dayjs date library and initially contained no malicious content, likely to establish publication history and reduce automated detection likelihood [1]. Twenty-four hours later, at 01:01 UTC on June 17, the same account published easy-day-js@1.11.22, which contained an obfuscated postinstall hook.

The second account exploited ehindero, a Mastra contributor account whose npm publishing credentials had been compromised via social engineering. According to Mastra's incident report, the account holder received a connection request from a compromised LinkedIn account, participated in a call, and clicked a suspicious link that delivered credential-harvesting malware [3]. With publishing privileges across the Mastra package scope, the ehindero account was used starting at 01:20 UTC to republish 145 @mastra packages (some sources initially reported 144; The Hacker News confirmed the final count was corrected to 145 [3]) — adding easy-day-js as a production dependency in each. The full package modification campaign completed in approximately 88 minutes [2][3][7].

The sequencing is consistent with a deliberate operational design. By staging a clean version of the typosquat dependency first, the attackers reduced the window during which a security scanner might flag a newly published package with a postinstall hook before any dependents were updated. By using a compromised maintainer account rather than attempting to directly publish malicious packages under a new identity, they inherited the trust and scope access that established contributors accumulate, bypassing the heightened scrutiny that unfamiliar publisher identities typically receive.

Technical Analysis: Payload Architecture and Capabilities

The malicious easy-day-js@1.11.22 delivered a staged payload triggered at installation time. The postinstall hook executed a 4,572-byte dropper file (setup.cjs) that employed rotated string arrays and Base64 encoding to obscure its logic from static analysis tools [1]. Before reaching out to attacker infrastructure, the dropper disabled TLS certificate verification — a step that simplified the use of self-signed certificates on command-and-control servers while also undermining any network monitoring that relied on certificate inspection.

The four-stage payload architecture operated as follows. In Stage 0, the dropper contacted the primary C2 server at 23.254.164.92:8000, retrieved a second-stage payload from the endpoint /update/49890878, and executed it as a detached hidden process before deleting itself from disk [1]. This self-removal step would have the effect of frustrating post-compromise forensics: by the time a developer noticed anomalous behavior, the initial dropper artifact was no longer present.

Stage 1 performed system reconnaissance through PowerShell cmdlets, enumerating installed applications, running processes, and browser extensions to fingerprint the target environment and determine what credential classes were accessible. Stage 2 implemented cross-platform persistence using artifacts designed to blend with legitimate Node Version Manager (NVM)

tooling, then collected cryptocurrency wallet data, browser history, and host information through a custom communication protocol. The malware was specifically configured to identify 166 cryptocurrency wallet browser extensions, including MetaMask, Phantom, Coinbase Wallet, Binance Wallet, and TronLink [2][3]. Persistence mechanisms were OS-specific: Windows Registry run keys, macOS LaunchAgents, and Linux systemd service units.

Stage 3, deployed only to systems that successfully established C2 communication, delivered a dedicated PowerShell backdoor from separate infrastructure (the domains teams.onweblive.org and maskasd.com, distinct from the initial C2 servers) [1]. The backdoor installed a persistent Windows service named "scdev" running at SYSTEM privilege, added Microsoft Defender exclusions to suppress local detection, and used reflective .NET DLL injection — loading malicious code directly into memory without writing a corresponding executable to disk — to evade endpoint detection tools that rely on file-system scanning. The result was full interactive remote access to the developer machine that persisted across reboots and was independent of any user login session.

The following table summarizes the indicators of compromise published by Microsoft as of this writing [1].

Indicator Type	Value	Stage
Malicious package	easy-day-js@1.11.22	Stage 0
Dropper hash (SHA-256)	B122A9873BEDF145AE2A7FD024B5F309007DBB025149F4DC4AC3F7E4F32A36A4	Stage 0
Package archive hash (SHA-256)	AE70DD4F6BC0D1C8C2848E4E6B51934626C4818DCB5AF99D080DDBD7DC337185	Stage 0
C2 server	23.254.164.92:8000	Stage 1–2
C2 server	23.254.164.123:443	Stage 1–2
Post-compromise domain	teams.onweblive.org	Stage 3
Post-compromise domain	maskasd.com	Stage 3
Malicious publisher account	sergey2016@tutamail.com	Initial publish
Compromised maintainer	ehindero (npm)	Package poisoning
Artifact (macOS/Linux)	$TMPDIR/.pkg_history	Stage 2

Indicator Type	Value	Stage
Artifact (macOS/Linux)	$TMPDIR/.pkg_logs	Stage 2
Persistent service	scdev (Windows, SYSTEM)	Stage 3

Attribution: Sapphire Sleet and the Escalating AI Developer Threat

Microsoft assessed with high confidence that this activity is attributable to Sapphire Sleet based on infrastructure overlap with prior campaigns, consistency of tradecraft across the Axios and Mastra operations, and targeting alignment with the group's established mandate to generate hard currency through cryptocurrency theft [1]. The two-stage delivery pattern — benign bait package followed by a weaponized update — and the use of reflective .NET injection for post-compromise persistence have appeared consistently across Sapphire Sleet operations documented over several years.

The group's decision to focus two consecutive supply chain campaigns on the JavaScript AI developer ecosystem suggests a strategic assessment rather than opportunistic targeting. The Axios compromise in April 2026 affected a package with hundreds of millions of weekly downloads across all JavaScript applications; the Mastra compromise in June traded breadth for specificity, targeting developers who work directly with AI agents and are therefore more likely to have LLM API credentials and cryptocurrency wallet extensions in their browser profiles, given that AI framework development typically requires LLM service API keys and, in Web3-adjacent projects, cryptocurrency wallet browser extensions. Both campaigns used the same postinstall hook delivery mechanism and similar C2 infrastructure patterns [2][4].

For organizations' threat models, the shift from broad-reach packages to AI-specific frameworks indicates that Sapphire Sleet has identified AI developer environments as a distinct and particularly lucrative credential category. Organizations that deploy AI framework packages in developer environments or CI/CD pipelines should treat this attack class as a persistent, evolving threat rather than a single incident.

Recommendations

Immediate Actions

Any organization using Mastra packages should first establish whether affected versions were installed. The malicious modifications were introduced at v1.13.1 and equivalent version numbers across the @mastra scope. Teams should audit their dependency lock files for easy-day-js appearing as a transitive dependency and check developer systems for the artifact files listed in the IOC table above ($TMPDIR/.pkg_history and $TMPDIR/.pkg_logs on macOS and Linux; the scdev Windows service) [1][6]. Systems where these artifacts are present or that made outbound connections to the listed C2 IP addresses should be treated as fully compromised, and all credentials, API keys, and tokens accessible from those machines should be rotated immediately — including cloud provider keys, LLM service credentials, and any CI/CD pipeline tokens [1][4].

Npm removed the malicious package versions from the registry, and Mastra revoked all publishing tokens and removed token bypass configurations across its packages following the incident [3]. Pinning to Mastra package versions at or below v1.13.0 in lock files is the safest approach until teams can verify clean versions above that threshold.

Short-Term Mitigations

Among the controls available without architectural changes, disabling postinstall script execution during `npm install` is among the most direct mitigations for this attack class, as it would have blocked Stage 0 payload delivery in this incident. Running `npm install --ignore-scripts` prevents arbitrary code execution triggered by dependency installation. This flag can be set as a permanent project default by adding `ignore-scripts=true` to the project or user `.npmrc` file. Security teams should evaluate whether this configuration can be enforced in CI/CD pipeline definitions and developer environment provisioning scripts, accepting that some packages require postinstall scripts for legitimate compilation steps and may need explicit allowlisting [1][4].

Subresource integrity (SRI) and package lock file pinning are complementary controls. A lock file that records exact content hashes — not merely version strings — provides a mechanism for detecting when a published version's content changes after initial resolution, which is precisely what occurred here when v1.11.22 of easy-day-js replaced v1.11.21. Tools such as Socket.dev, Snyk, and npm audit can flag dependency tree changes and known-malicious packages, but their effectiveness depends on being integrated into the install pipeline rather than run as a periodic review.

Privileged credentials with production access should not be present in developer environments used for framework development and experimentation. Where some credentials are necessary, they should be scoped to the minimum required permissions and time-limited. LLM API keys, cloud provider credentials, and CI/CD tokens that carry production access represent a systemic risk when they coexist with an npm install workflow that can execute arbitrary code. Where possible, development environments should operate with scoped, time-limited credentials, and production tokens should be confined to isolated CI/CD environments where package installation is controlled and logged.

Strategic Considerations

The social engineering compromise of the ehindero maintainer account was the attack's critical entry point, and it was enabled by the broad token scope that contributor accounts in open-source npm ecosystems typically accumulate. Organizations that depend on popular open-source packages should assess the contributor account security posture of their most critical dependencies — specifically whether maintainer accounts are protected by hardware MFA, whether publishing tokens are scoped to individual packages rather than the full organizational scope, and whether the project has implemented npm's recently introduced granular publish attestations. The same assessment applies internally: teams that maintain internal npm registries or scoped packages should audit which developer identities hold publishing rights and ensure those accounts require phishing-resistant authentication.

At a program level, the Mastra incident illustrates why AI framework dependencies warrant the same vendor risk scrutiny applied to production software libraries. The AI developer tooling ecosystem is growing rapidly, and the security maturity of its constituent packages varies considerably. A Mastra-class incident is structurally possible in any AI framework that distributes functionality across dozens of scoped packages managed by contributor accounts with broad publishing permissions.

CSA Resource Alignment

This incident directly implicates several areas of CSA's published guidance on AI and cloud security.

CSA's MAESTRO framework for agentic AI threat modeling addresses the supply chain layer as a distinct threat surface for AI agent systems. The Mastra compromise provides concrete evidence that threat actors are actively targeting the framework layer — the packages developers use to build agents — rather than targeting deployed agents directly. Organizations building on AI

frameworks should apply MAESTRO's Layer 1 and Layer 2 threat analysis to their dependency chains, not only to their agent logic.

The AI Controls Matrix (AICM), as a superset of CSA's Cloud Controls Matrix (CCM), includes supply chain risk controls that map directly to this incident. The AICM's controls for software composition analysis, dependency integrity verification, and third-party component governance apply to AI framework dependencies as much as to traditional software libraries. The Mastra incident provides a concrete, documented case study for calibrating the implementation priority of those controls in organizations with AI development workloads.

CSA's STAR program provides the vendor risk assessment framework most relevant to evaluating open-source AI framework dependencies at scale. While open-source projects do not typically publish STAR assessments, the program's control domains — identity and access management, change control, and incident response — correspond precisely to the gaps that Sapphire Sleet exploited. Enterprises can use STAR criteria as a structured lens for evaluating the security posture of critical open-source dependencies during technology selection.

CSA's guidance on securing non-human identities in AI environments is particularly relevant to the CI/CD dimension of this attack. The tokens and API keys most at risk from a developer machine compromise are often non-human identities — service accounts, pipeline tokens, and API credentials — whose rotation and scoping is frequently less disciplined than human account management. The Mastra incident is an argument for applying the same lifecycle governance to these credentials as to human identities.

Finally, CSA's Zero Trust guidance applies at the network layer: the C2 communication that enabled Stage 2 and Stage 3 payload delivery relied on outbound connections from developer machines to arbitrary IP addresses. Zero Trust network architectures that restrict outbound connectivity from development environments to known egress points would have interrupted the C2 channel even if the initial dropper executed successfully.

References

[1] Microsoft Security. "From package to postinstall payload: Inside the Mastra npm supply chain compromise by Sapphire Sleet." Microsoft Security Blog, June 17, 2026.

[2] Toulas, Bill. "Microsoft links Mastra AI supply chain attack to North Korean hackers." BleepingComputer, June 17, 2026.

[3] Lakshmanan, Ravie. "145 Mastra npm Packages Compromised via Hijacked Contributor Account." The Hacker News, June 2026.

[4] StepSecurity Research. "Mastra npm Supply Chain Attack: 140+ Packages Backdoored via easy-day-js Typosquat." StepSecurity Blog, June 17, 2026.

[5] MITRE. "BlueNoroff (G0082)." MITRE ATT&CK, accessed June 2026.

[6] Kodem Security. "Mastra npm Compromise: easy-day-js Attack & Response — IOCs and Response Runbook." Kodem Security, June 17, 2026.

[7] CybersecurityNews Staff. "Hackers Compromised 140+ Mastra npm Packages to Deploy Password-Stealing Malware." CybersecurityNews, June 17, 2026.