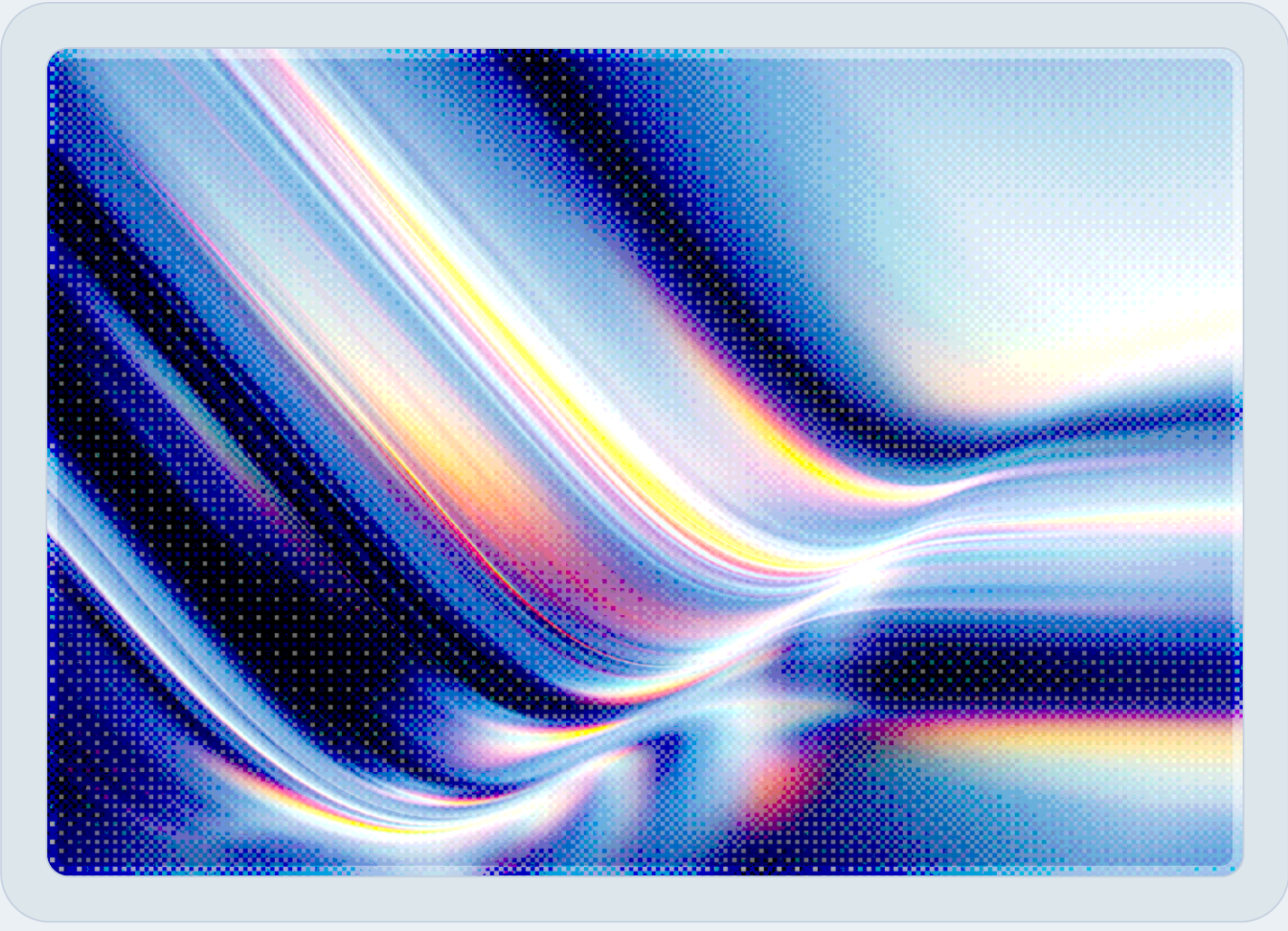
CSAI Foundation | Cloud Security Alliance

Sapphire Sleet Poisons Mastra AI npm Ecosystem

144 Packages Backdoored in 88 Minutes by North Korean APT

2026-06-23

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- On June 17, 2026, the North Korean state actor Sapphire Sleet hijacked an npm maintainer account and published malicious versions of approximately 144 packages in the Mastra AI framework's namespace within an 88-minute automated campaign [1][8].
- The attack weaponized a typosquatted dependency called `easy-day-js` – engineered to be indistinguishable from the legitimate `dayjs` date library – whose postinstall hook deployed a cross-platform credential-stealing remote access trojan (RAT) [2].
- The RAT targeted LLM API keys, cloud provider credentials, CI/CD tokens, and inventoried 166 cryptocurrency wallet browser extensions across Windows, macOS, and Linux [1].
- This is the second npm supply chain attack in 2026 attributed to Sapphire Sleet; a near-identical campaign against the Axios HTTP client was disclosed in April 2026, indicating a deliberate and sustained focus on the JavaScript developer ecosystem [3].
- Organizations using Mastra at the compromised version (v1.13.1) should immediately scan for indicators of compromise, rotate all credentials accessible from affected environments, and pin dependencies to confirmed safe versions.
- This incident demonstrates that AI developer tooling has become a high-value supply chain target: a single developer install can expose an entire constellation of cloud, LLM, and cryptocurrency credentials in one operation.

Background

The Mastra AI Framework

Mastra is an open-source TypeScript framework purpose-built for constructing AI agents. It provides primitives for agent memory, tool orchestration, workflow execution, and LLM provider integration, and has become a foundational dependency in enterprise AI development pipelines. The `@mastra/core` package alone registers approximately 918,000 weekly downloads, and the combined exposure across the Mastra scope exceeded 1.1 million installs per week at the time of the attack [2]. That scale, combined with the nature of environments where Mastra is deployed – developer workstations and

CI/CD pipelines routinely holding LLM API keys, cloud access credentials, and database connection strings — made the framework an exceptionally attractive target for a financially motivated nation-state actor [4].

Sapphire Sleet: A Financially Motivated North Korean APT

Sapphire Sleet is a North Korean state-sponsored threat actor operating under the Reconnaissance General Bureau and active since at least March 2020. The group is tracked under multiple aliases by different intelligence vendors, including BlueNoroff, UNC1069, STARDUST CHOLLIMA, CageyChameleon, Alluring Pisces, and CryptoCore [5]. Some vendors also associate the group with APT38, a related North Korean subgroup whose historical focus is SWIFT banking-system theft; however, many analysts track APT38 as a distinct cluster with a different mission profile, and that mapping is not universally accepted [5]. Its observed mission is the generation of hard currency for the North Korean regime through the theft of cryptocurrency and financial assets, and it has historically targeted cryptocurrency exchanges, decentralized finance protocols, venture capital firms, and blockchain organizations across at least 38 countries [5]. Over time, Sapphire Sleet has expanded beyond direct exchange compromise toward targeting the developer toolchains and credentials that enable access to financial infrastructure — a shift reflecting the reality that AI development environments now concentrate an extraordinary density of valuable credentials in a single location.

Security Analysis

The Attack: Account Compromise and Automated Mass-Publication

The Mastra supply chain campaign began not with a vulnerability in the framework itself but with the compromise of a human maintainer. The npm account `ehindero` — a current Mastra employee with publish rights across the organization's package namespace — was compromised via social engineering. According to Microsoft's analysis, the initial contact was made through a fake LinkedIn account, which directed the maintainer to click a suspicious link during a call [1]. The exact technical bypass that followed has not been fully disclosed, but the maintainer's machine was compromised, and the attacker obtained valid session tokens or credentials sufficient to authenticate as `ehindero` on the npm registry; whether MFA was configured on the account has not been publicly disclosed [2].

From that foothold, the attacker moved with automated speed. Beginning at 01:01 UTC on June 17, 2026, approximately 144 packages across the `mastra` and `@mastra` scopes received new, malicious versions in an automated campaign spanning approximately 88 minutes [1][8]. The publication of each poisoned package was consistent with legitimate release activity in structure and naming, making detection through registry monitoring alone extremely difficult in the window between publication and response.

The Typosquat Mechanism: easy-day-js

The delivery vehicle for the malicious payload was not embedded directly in the Mastra packages themselves; instead, the attacker injected a new dependency declaration pointing to `easy-day-js`, a typosquatted facsimile of the widely used `dayjs` date formatting library [1]. The deception was engineered at the metadata level: `easy-day-js` duplicated `dayjs`'s author name, homepage, repository URL, license, and version numbering. The initial version — `easy-day-js@1.11.21`, published the day before on June 16 — was completely clean, functioning as legitimate date utility code. The malicious version `1.11.22` was published at 01:01 UTC on June 17, one hour before the mass-publication of the poisoned Mastra packages [1].

This sequencing exploited a structural property of npm's semver resolution behavior. When developers or CI/CD pipelines install a package declaring a dependency on `easy-day-js@^1.11.21` or `~1.11.21`, npm's version solver automatically resolves to the highest available compatible version — in this case, the armed `1.11.22` — even though no explicit update was requested and no lockfile entry changed. A developer who had audited their `package.json` and seen no suspicious entries, and who had not recently modified any lockfile, could still receive the malicious build. This is the same class of dependency confusion and semver-exploitation technique used in the April 2026 Axios campaign [3], and its reuse indicates Sapphire Sleet has refined this approach into repeatable tradecraft.

Three-Stage Payload Delivery

The malicious `easy-day-js@1.11.22` carried a `postinstall` script field in its `package.json` that npm executes automatically as part of the standard `npm install` workflow, requiring no additional user interaction. That hook invoked `setup.cjs`, a 4,572-byte obfuscated dropper that performed several functions before yielding to later payload stages [1].

The dropper began by setting `NODE_TLS_REJECT_UNAUTHORIZED=0`, disabling certificate validation for all subsequent Node.js HTTPS connections in the process context. It then wrote filesystem tracking markers — `.pkg_history` and `.pkg_logs` — to the system temporary directory, and

contacted the primary command-and-control server at `23.254.164.92` on port 8000 to retrieve a second-stage payload. The dropper deleted itself after successful delivery [1].

The second stage was an approximately 41 KB Node.js tasking client — the core RAT — designed for cross-platform operation on Windows, macOS, and Linux. Its capabilities included enumeration of installed applications and running processes; extraction of browser history from Chrome, Brave, and Edge via direct SQLite database copying; and a hardcoded inventory sweep of 166 browser extensions associated with cryptocurrency wallets, including MetaMask, Phantom, and Coinbase Wallet [1][2]. The RAT established a polling connection to the C2 infrastructure for follow-on module delivery, creating a persistent remote execution channel.

On Windows systems where the attacker chose to escalate, a third stage delivered a PowerShell backdoor capable of reflective .NET DLL injection into `cmd.exe` — a fileless execution technique that leaves no payload artifact on disk — and manipulated Windows Defender exclusion lists to suppress detection [1]. Persistence was established through three platform-specific mechanisms: a registry Run key on Windows, a LaunchAgent plist at `~/Library/NodePackages/` on macOS, and a systemd user unit at `~/.config/systemd/nvmconf/` on Linux. All persistence artifacts used names closely resembling legitimate Node.js component names — "protocal," "nvmconf" — in a pattern consistent with deliberate obfuscation designed to evade casual inspection [1].

Why AI Developer Tooling Is the New High-Value Target

The Mastra campaign is consistent with a strategic shift in Sapphire Sleet's operational logic — one that threat intelligence analysts have characterized as a deliberate expansion toward developer toolchain targeting. Direct financial theft from cryptocurrency infrastructure historically required targeting exchange accounts, wallet custodians, or DeFi protocol contracts. AI developer environments, by contrast, are ecosystems that passively aggregate the most valuable credential classes in modern software: LLM API keys that control access to large-scale model inference and billing, cloud provider credentials that unlock storage, compute, and database resources, CI/CD pipeline tokens that can modify production deployments, and cryptocurrency wallet extensions that are loaded in the same browser profiles used for development work. A single `npm install` command on a developer's workstation or in a CI/CD job can expose all of these simultaneously — making framework-level supply chain compromise extraordinarily efficient as an initial access strategy [4][6].

The two Sapphire Sleet npm campaigns in 2026 share a pattern that suggests deliberate selection criteria: both targeted widely adopted JavaScript packages with active developer communities, high weekly download counts, and deployment in environments where automation reduces human oversight of individual install operations. The Axios attack in April 2026 affected a package with over 70 million

weekly downloads [3]. The Mastra attack targeted a framework embedded in AI development pipelines where credential density is especially high. The targeting logic is consistent with deliberate selection criteria rather than purely opportunistic exploitation, suggesting the actor may be surveying the JavaScript ecosystem for high-credential-density packages.

Attribution and Confidence

Microsoft Threat Intelligence assessed with high confidence that both the June 2026 Mastra campaign and the April 2026 Axios campaign are attributable to Sapphire Sleet [1][3][7][9]. The attribution rests on infrastructure overlap — both campaigns used the same C2 IP range and operational patterns — as well as the similarity of the multi-stage dropper architecture, the identical technique of injecting a clean decoy package followed by a weaponized patch-version update, and the targeting focus on cryptocurrency wallet credentials consistent with Sapphire Sleet's known mission [1].

Recommendations

Immediate Actions

Organizations that have installed any version of the Mastra framework should immediately audit their environments for the presence of `easy-day-js@1.11.22` in dependency trees and installed `node_modules` directories. The safe version boundary is `mastra@1.13.0` and below; version `1.13.1` is the poisoned release [1]. Developers should inspect their `package-lock.json` and any CI/CD artifact caches for references to `easy-day-js` and remove them. Any system that ran `npm install` on affected packages should be treated as potentially compromised until investigated, and all credentials accessible from that environment — LLM API keys, cloud IAM credentials, CI/CD tokens, database passwords, and any cryptocurrency wallet seed phrases stored in browser profiles — should be rotated immediately.

Security teams should scan for the known C2 indicators in network logs: outbound connections to `23.254.164.92` and `23.254.164.123` on any port, and DNS queries for `teams.onweblive.org` and `maskasd.com` [1]. On affected hosts, investigators should look for the filesystem markers `$TMPDIR/.pkg_history` and `$TMPDIR/.pkg_logs`, the Windows

persistence artifact at `C:\ProgramData\NodePackages\protocal.cjs`, the macOS LaunchAgent at `~/Library/NodePackages/`, and the Linux systemd unit at `~/.config/systemd/nvmconf/` [1].

Short-Term Mitigations

The most immediate structural defense against postinstall-based attacks is to run package installs with the `--ignore-scripts` flag, which prevents npm from executing `postinstall`, `preinstall`, and related lifecycle hooks. This flag can be set as a default in `.npmrc` (`ignore-scripts=true`) for development environments and CI/CD pipelines where lifecycle scripts are not operationally required [1][6]. Organizations should also enforce lockfile integrity by configuring CI/CD systems to run `npm ci` rather than `npm install`, which installs exactly from the lockfile and refuses to resolve newer patch versions automatically. These two controls together would have prevented the Mastra attack from executing on most target systems.

Contributor access to package publishing should be governed on the principle of least privilege. The `ehindero` account held publish rights across the full Mastra namespace; a model where publish access is scoped to specific packages or requires a short-lived token issued at release time would have bounded the blast radius of the account compromise. Organizations publishing npm packages should additionally adopt SLSA provenance attestations, which allow consumers to verify that published packages were built from known repository commits on trusted infrastructure — a control that the malicious Mastra releases would not have been able to satisfy [2].

Strategic Considerations

The two Sapphire Sleet campaigns of 2026 together demonstrate that npm's default trust model — where any published package version is immediately available for resolution, and postinstall hooks execute with full developer privileges — is structurally mismatched to the current threat landscape. The npm security community has been actively discussing changes to tighten these defaults, including restricting automatic postinstall script execution and enforcing stronger provenance attestation requirements. Organizations should monitor the official npm blog and the npm/cli GitHub repository for announced changes to defaults and evaluate whether equivalent restrictions can be adopted ahead of any registry-wide rollout.

More broadly, AI development pipelines represent a new category of supply chain risk that security governance frameworks have not yet fully addressed. The combination of rapid iteration cycles, heavy reliance on open-source dependencies, and environments that concentrate LLM and cloud credentials

creates attack surfaces that differ meaningfully from traditional software supply chains. Security teams should classify AI development pipelines as high-sensitivity environments and apply the same credential isolation, access controls, and audit logging that would apply to production infrastructure — rather than treating developer workstations and CI/CD runners as low-risk endpoints.

CSA Resource Alignment

The Mastra supply chain attack maps directly to several threat categories and control domains addressed by CSA's AI Safety Initiative frameworks.

CSA's **MAESTRO** threat model for agentic AI systems identifies the data and model infrastructure layer and the inter-agent trust boundary layer as primary attack surfaces. The Mastra compromise represents a pre-deployment threat at the toolchain level: an attacker injected malicious behavior into the development environment used to build AI agents, rather than attacking the agents themselves at runtime. MAESTRO's coverage of supply chain integrity threats in AI development environments is directly applicable, and organizations building AI agents with Mastra or similar frameworks should include developer toolchain integrity in their MAESTRO-based threat models.

The **Agentic Trust Framework (ATF)**, stewarded by CSA and MassiveScale.AI, addresses identity verification and access governance for AI agents and the humans who develop them. The Mastra attack succeeded because contributor account access persisted beyond the point where it could be effectively governed, and because the npm trust model provided no mechanism for consumers to verify the provenance of published packages. ATF's guidance on non-human identity lifecycle management and cryptographic supply chain verification offers a control architecture directly responsive to this gap.

CSA's **AI Controls Matrix (AICM)** provides control domains covering software supply chain integrity, third-party dependency management, and developer environment security. The Mastra incident should inform control assessments in the AICM's supply chain and infrastructure security domains, particularly for organizations building on open-source AI frameworks where dependency graphs are large and update cadence is high.

Finally, CSA's **Securing Non-Human Identities in the Age of AI Agents** guidance addresses the credential management challenge that made the Mastra attack consequential: AI development environments hold a high density of non-human identities — API keys, service account tokens, pipeline credentials — that are often under-governed relative to human accounts. The guidance's

recommendations for credential isolation, short-lived token issuance, and secrets management for AI pipelines would have reduced the blast radius of the Mastra compromise even if initial access could not have been prevented.

References

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

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

[3] Microsoft Threat Intelligence. "Mitigating the Axios npm supply chain compromise." Microsoft Security Blog, April 1, 2026.

[4] Upwind Security. "Mastra Supply Chain Compromise: easy-day-js Dropper Pulls a Cross-Platform RAT Into @mastra Installs." Upwind Security, June 2026.

[5] Infosecurity Magazine. "Microsoft Attributes Mastra AI Supply Chain Attack to North Korea." Infosecurity Magazine, June 2026.

[6] SafeDep. "Mastra npm Scope Takeover: 143 Packages Drop a RAT." SafeDep Threat Intelligence, June 2026.

[7] SecurityWeek. "North Korean Hackers Blamed for Mastra NPM Supply Chain Attack." SecurityWeek, June 2026.

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

[9] BleepingComputer. "Microsoft links Mastra AI supply chain attack to North Korean hackers." BleepingComputer, June 2026.

