INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-20 18:46 UTC

Sapphire Sleet Escalates npm Campaign: 140+ Mastra AI Packages Weaponized to Harvest Credentials and Crypto Wallets

THREAT CAMPAIGN | **HIGH** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0525
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	9.5
Affected Products	Mastra AI npm packages (140+ packages in @mastra scope); npm ecosystem; Windows, Linux, macOS developer environments; cryptocurrency wallets: MetaMask, Phantom, Coinbase Wallet, Binance Wallet, TronLink
Published	2026-06-20T10:09:19
Discovery Source	Rss

Executive Summary

North Korean state-sponsored group Sapphire Sleet (BlueNoroff) compromised a dormant contributor account in the Mastra AI npm package scope and pushed malicious updates across 140+ packages, weaponizing them to steal developer credentials, API keys, and cryptocurrency wallet data on Windows, Linux, and macOS. Any organization whose developers installed or updated @mastra-scoped npm packages on or after the compromise date is at risk of credential theft and potential lateral movement into CI/CD pipelines, cloud environments, and financial accounts. Microsoft attributed this attack with high confidence and linked it to the same actor behind the April 2026 Axios npm supply chain compromise, signaling an active, escalating campaign targeting the developer toolchain.

Technical Analysis

Sapphire Sleet gained access to the @mastra npm scope by compromising a forgotten contributor account that lacked modern MFA controls, bypassing protections on the primary maintainer account. Malicious postinstall hooks were injected into 140+ packages across the @mastra namespace. The payload is a cross-platform infostealer targeting: developer credentials and API keys from filesystem and environment variables; cryptocurrency wallet extensions (MetaMask, Phantom, Coinbase Wallet, Binance Wallet, TronLink); and SSH keys and browser-stored secrets. C2 communication uses standard HTTP/S (T1071.001). The payload achieves persistence via LaunchAgent/LaunchDaemon on macOS (T1543.001/T1543.002), Run keys on

Windows (T1547.001), and systemd units on Linux (T1543.003). Obfuscation (T1027) is used to evade static detection. PowerShell (T1059.001) and shell scripting (T1059.004) are leveraged cross-platform. The attack chain aligns with MITRE T1195.001 (Supply Chain Compromise: Compromise Software Dependencies) and T1078 (Valid Accounts, dormant maintainer credential). Stolen tokens are exfiltrated via T1528. Credential theft from files uses T1552.001 and T1555. No CVE is assigned; this is a supply chain campaign. Relevant CWEs: CWE-494 (Download of Code Without Integrity Check), CWE-346 (Origin Validation Error), CWE-522 (Insufficiently Protected Credentials), CWE-312 (Cleartext Storage of Sensitive Information). Microsoft attribution is high-confidence based on shared PowerShell backdoor code and C2 infrastructure overlap with the April 2026 Axios compromise. No patch exists; remediation requires package removal and environment revalidation.

Action Checklist

1. Step 1: Containment. Immediately audit all CI/CD pipelines, developer workstations, and build servers for installed @mastra-scoped npm packages. Quarantine any systems that installed or updated @mastra packages after the compromise window (reference Microsoft Security Blog 2026-06-17 and vendor IOC releases for specific date ranges). Block outbound connections to known Sapphire Sleet C2 infrastructure at the perimeter firewall and endpoint controls. Suspend any API keys, tokens, or credentials that existed on affected developer systems.
2. Step 2: Detection. Search npm package-lock.json, yarn.lock, and node_modules directories across all developer and build environments for @mastra-scoped packages. Review postinstall script execution logs in npm debug logs (~/.npm/_logs) and CI/CD runner logs for unexpected subprocess spawning during npm install. Hunt endpoint telemetry for: PowerShell or shell child processes spawned from node.exe or npm; outbound HTTP/S to non-standard IPs from build agents; file access to browser extension storage directories (MetaMask, Phantom, Coinbase Wallet, Binance Wallet, TronLink); reads of ~/.ssh, environment variable files, and .env files by npm processes. Cross-reference C2 indicators from Microsoft, Kodem, Snyk, and Orca Security IOC releases against DNS, proxy, and firewall logs (NIST AU-6, CIS 8.2).
3. Step 3: Eradication. Remove all @mastra-scoped packages from affected environments and do not reinstall until Mastra project leadership confirms a clean, re-keyed release. Rotate all credentials, API keys, SSH keys, and tokens present on any system where the malicious packages executed. Revoke and reissue cloud provider credentials (AWS, GCP, Azure) sourced from affected machines. Invalidate all active sessions for developer accounts on affected systems. Rebuild affected CI/CD runners from a known-clean image rather than attempting in-place cleaning (NIST IR family; CIS 4.6).
4. Step 4: Recovery. Validate clean builds by verifying package integrity hashes against a known-good lockfile predating the compromise window. Re-enable CI/CD pipelines only after confirming no @mastra packages are present and all credentials have been rotated. Monitor newly issued credentials for anomalous use for a minimum of 30 days post-rotation (NIST AU-6, AU-12). Confirm cryptocurrency wallet recovery phrases stored on affected systems are treated as fully compromised; transfer assets to new wallets generated on clean, air-gapped devices. Apply NIST AC-2 account review to all service accounts that had access from affected build systems.
5. Step 5: Post-Incident. Conduct a review of all third-party npm dependencies for dormant or unmonitored contributor accounts; require package maintainers to demonstrate MFA on all accounts with publish rights (CIS 6.3, CIS 6.5). Implement npm package integrity verification (subresource integrity or private registry mirroring) to enforce CWE-494 controls. Establish a Software Composition Analysis (SCA) gate in CI/CD

pipelines to flag newly added or updated packages for review before installation (CIS 2.1, CIS 2.3, NIST CM family). Review and tighten least-privilege on CI/CD service accounts so build processes cannot access developer credential stores or cryptocurrency wallet directories (NIST AC-6). Document this event as a case study for developer security training on supply chain risk.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to executive leadership, legal counsel, and (where applicable) regulatory notification authorities immediately if forensic evidence confirms exfiltration of cloud provider credentials (AWS/GCP/Azure), SSH private keys, or cryptocurrency wallet seed phrases from affected systems, as these constitute potential financial loss events and may trigger breach notification obligations under applicable data protection regulations if developer PII was co-located in accessed credential stores.
Recovery Notes	Re-enable CI/CD pipelines only after all @mastra packages are purged, runners are rebuilt from verified-clean images, and all credentials sourced from affected systems are rotated and confirmed inactive in cloud provider access logs. Monitor all newly issued credentials, tokens, and SSH keys for anomalous usage patterns — particularly sts:AssumeRole chains, cross-region API calls, and cryptocurrency exchange API activity — for a minimum of 30 days post-rotation, as Sapphire Sleet (BlueNoroff) is known to stage harvested credentials and delay their operational use. Treat any cryptocurrency wallet whose seed phrase or keystore file was accessible on a compromised system as permanently compromised regardless of whether on-chain theft has been observed yet; transfer assets to wallets generated on air-gapped devices before resuming normal operations.
Forensic Artifacts	npm debug logs at ~/.npm/_logs/*.log on developer workstations and CI/CD runners: these record postinstall script stdout/stderr output and will contain execution evidence of Sapphire Sleet's malicious hook running during `npm install @mastra/*`, including any subprocess spawning or network call attempts made by the payload. | Chromium-based browser extension LevelDB stores for MetaMask (extension ID nkbihfbeogaeaoehlefnkodbefgpgknn), Phantom (bfnaelmomeahhmniihlkjnapifbmcelj), Coinbase Wallet (hnfanknocfeofbddgcijnmhnfnkdnaad), Binance Wallet (fhbohimaelbohpjbbldcngcnapndodjp), and TronLink (ibnejdfjmmkpcnlpebklmnkoeoihofec) located under the user's Chromium Default/Extensions directory: file-system access timestamps on these stores will reveal whether the malicious npm postinstall script read wallet data. | Cloud provider credential files at ~/.aws/credentials, ~/.config/gcloud/credentials.db, and ~/.azure/accessTokens.json with last-accessed (atime) filesystem timestamps: these confirm whether the Sapphire Sleet payload accessed cloud credentials during the npm install execution window, and their contents define the exact scope of credential exposure. | Windows Security Event Log Event ID 4688 (Process Creation) or Linux auditd EXECVE records filtered for node.exe or npm as parent process: these capture the exact command lines of any child processes (PowerShell, cmd.exe, bash, curl, python) spawned by the malicious @mastra postinstall hooks, which is the primary execution evidence for this supply chain attack vector. | Network flow records (firewall logs, proxy logs, or Zeek/Wireshark pcap) for outbound HTTP/S connections from developer workstations and CI/CD build agent IPs during the compromise window: Sapphire Sleet C2 exfiltration of harvested credentials and wallet data will appear as POST requests to non-standard IPs or domains not present in pre-compromise baseline traffic, correlatable against IOCs published by Microsoft, Kodem, Snyk, and Orca Security.

Per-Action IR Details

Step 1: Containment — Immediately audit all CI/CD pipelines, developer workstations, and build servers for installed @mastra-scoped npm packages. Quarantine any systems that installed or updated @mastra packages after the compromise window (reference Microsoft Security Blog 2026-06-17 for specific date ranges). Block outbound connections to known Sapphire Sleet C2 infrastructure at the perimeter firewall and endpoint controls. Suspend any API keys, tokens, or credentials that existed on affected developer systems.

NIST Phase: Containment

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

Controls: NIST AC-4 (Information Flow Enforcement), NIST AC-12 (Session Termination), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 4.5 (Implement and Manage a Firewall on End-User Devices)

Compensating: Run `find / -path '*/node_modules/@mastra*' -maxdepth 8 2>/dev/null` on Linux/macOS and `Get-ChildItem -Path C:\ -Recurse -Filter '@mastra' -ErrorAction SilentlyContinue` on Windows to enumerate installs without EDR. Block Sapphire Sleet C2 IPs/domains using host-level iptables/Windows Firewall rules immediately while perimeter rules are being pushed. Use `npm ls --depth=0 --json` in each project root to confirm @mastra presence.

Evidence: Before isolating any host or suspending credentials, capture: (1) full RAM dump using winpmem (Windows) or avml (Linux) to preserve in-memory credential material and any injected payloads loaded by the malicious postinstall script; (2) active network connections via `netstat -ano` (Windows) or `ss -tulnp` (Linux) to document live C2 channels to Sapphire Sleet infrastructure before firewall blocks terminate them; (3) running process list with parent-child relationships via `Get-WmiObject Win32_Process | Select ProcessId,ParentProcessId,CommandLine` or `ps auxf` to capture node.exe/npm child processes before quarantine kills them; (4) contents of ~/.npm/_logs/ and CI runner job logs showing postinstall hook execution timing relative to the compromise window.

Step 2: Detection — Search npm package-lock.json, yarn.lock, and node_modules directories across all developer and build environments for @mastra-scoped packages. Review postinstall script execution logs in npm debug logs (~/.npm/_logs) and CI/CD runner logs for unexpected subprocess spawning during npm install. Hunt endpoint telemetry for: PowerShell or shell child processes spawned from node.exe or npm; outbound HTTP/S to non-standard IPs from build agents; file access to browser extension storage directories (MetaMask, Phantom, Coinbase Wallet, Binance Wallet, TronLink); reads of ~/.ssh, environment variable files, and .env files by npm processes. Cross-reference C2 indicators from Microsoft, Kodem, Snyk, and Orca Security IOC releases against DNS, proxy, and firewall logs (NIST AU-6, CIS 8.2).

NIST Phase: Detection Analysis

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

Controls: NIST AU-6 (Audit Record Review, Analysis, And Reporting), NIST AU-12 (Audit Record Generation), CIS 8.2 (Collect Audit Logs)

Compensating: Deploy Sysmon with a config targeting ProcessCreate (Event ID 1) filtering on ParentImage containing `node.exe` or `npm` with CommandLine spawning `cmd.exe`, `powershell.exe`, `bash`, or `sh`. Use osquery with `SELECT pid, name, path, cmdline, parent FROM processes WHERE parent IN (SELECT pid FROM processes WHERE name IN ('node', 'npm'))` to enumerate suspicious npm child processes. Write a YARA rule targeting the malicious @mastra postinstall script pattern (base64-encoded payload blobs or hardcoded Sapphire Sleet C2 strings) and scan all node_modules directories. For wallet access detection, monitor file read events on `%APPDATA%\MetaMask`, `~/.config/google-chrome/Default/Extensions` (Phantom/Coinbase), and equivalent Chromium extension storage paths using Sysmon FileAccess (Event ID 23).

Evidence: This step is read-only detection; it does not alter live state. Key artifacts to collect during this phase: (1) all npm debug log files at ~/.npm/_logs/*.log preserving postinstall hook stdout/stderr; (2) CI/CD runner job logs (GitHub Actions workflow run logs, GitLab job traces, Jenkins console output) for any `npm install` runs touching @mastra packages during the compromise window; (3) DNS query logs from resolvers or endpoint hosts for domains associated with Sapphire Sleet C2 infrastructure published by Microsoft, Kodem, Snyk, and Orca; (4) file system access audit logs (Windows Security Event ID 4663 or Linux auditd `openat` syscall records) for npm process reads of ~/.ssh/id_rsa, .env files, and Chromium-based wallet extension LevelDB stores; (5) browser extension local storage paths for MetaMask (`nkbihfbeogaeaoehlefnkodbefgpgknn`), Phantom (`bfnaelmomeahhmniihlkjnapifbmcelj`), Coinbase Wallet (`hnfanknocfeofbddgcijnmhnfnkdnaad`), Binance (`fhbohimaelbohpjbbldcngcnapndodjp`), and TronLink

(`ibnejdfjmmkpcnlpebklmnkoeoihofec`) under the Chromium Default/Extensions directory.

Step 3: Eradication — Remove all @mastra-scoped packages from affected environments and do not reinstall until Mastra project leadership confirms a clean, re-keyed release. Rotate all credentials, API keys, SSH keys, and tokens present on any system where the malicious packages executed. Revoke and reissue cloud provider credentials (AWS, GCP, Azure) sourced from affected machines. Invalidate all active sessions for developer accounts on affected systems. Rebuild affected CI/CD runners from a known-clean image rather than attempting in-place cleaning (NIST IR family; CIS 4.6).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST AC-2 (Account Management), NIST AC-12 (Session Termination), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 6.2 (Establish an Access Revoking Process)

Compensating: For CI/CD runner rebuilds without enterprise tooling, use a pinned, hash-verified base container image (e.g., `docker pull node:20-alpine@sha256:`) and redeploy from infrastructure-as-code (Dockerfile or Packer template stored in version control) rather than patching in place. Rotate AWS credentials via `aws iam delete-access-key` and `aws iam create-access-key`; GCP via `gcloud iam service-accounts keys delete`; Azure via `az ad app credential reset`. Audit and revoke all GitHub/GitLab personal access tokens and deploy keys associated with affected developer accounts using each platform's security settings UI or API.

Evidence: Before reimaging CI/CD runners or rotating credentials (both alter live state), capture: (1) a full disk image of affected runners using `dd if=/dev/sda | gzip > runner_disk.img.gz` or a VM snapshot if virtualized, to preserve any persistence mechanisms (cron jobs, systemd units, or npm global package hooks) written by the Sapphire Sleet postinstall payload; (2) the current state of ~/.ssh/known_hosts and ~/.ssh/authorized_keys on affected systems, which may reveal lateral movement staging by the threat actor; (3) a dump of all active shell environment variables (`env` / `Get-ChildItem Env:`) to document which secrets were in-memory at time of compromise; (4) cloud provider credential files at ~/.aws/credentials, ~/.config/gcloud/credentials.db, and ~/.azure/accessTokens.json before rotation, as these confirm the scope of credential exposure to Sapphire Sleet exfiltration.

Step 4: Recovery — Validate clean builds by verifying package integrity hashes against a known-good lockfile predating the compromise window. Re-enable CI/CD pipelines only after confirming no @mastra packages are present and all credentials have been rotated. Monitor newly issued credentials for anomalous use for a minimum of 30 days post-rotation (NIST AU-6, AU-12). Confirm cryptocurrency wallet recovery phrases stored on affected systems are treated as fully compromised; transfer assets to new wallets generated on clean, air-gapped devices. Apply NIST AC-2 account review to all service accounts that had access from affected build systems.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST AU-6 (Audit Record Review, Analysis, And Reporting), NIST AU-12 (Audit Record Generation), NIST AC-2 (Account Management), CIS 5.1 (Establish and Maintain an Inventory of Accounts)

Compensating: Verify package integrity by running `npm audit --audit-level=high` against a lockfile snapshot from before the Mastra compromise date and diffing resolved package hashes using `cat package-lock.json | jq '.packages | to_entries[] | select(.key | startswith("node_modules/@mastra")) | {key, value: .value.integrity}'`. Monitor newly rotated AWS IAM keys for anomalous API calls using CloudTrail with a free EventBridge rule alerting on calls from unfamiliar source IPs or unusual service patterns (e.g., `sts:AssumeRole`, `s3:GetObject` on sensitive buckets). For cryptocurrency wallets, generate new seed phrases exclusively on an air-gapped device that has never connected to any network or installed npm packages.

Evidence: Recovery steps alter live state (pipeline re-enablement, credential issuance); before re-enabling pipelines, confirm: (1) a clean baseline scan of all node_modules directories using `npm ls --json 2>/dev/null | grep @mastra` returns no results across all project roots on rebuilt runners; (2) cloud provider access logs (AWS CloudTrail, GCP Audit Logs, Azure Monitor) show no API activity from the old compromised credentials after their rotation timestamp, confirming Sapphire Sleet has not cached and reused them; (3) cryptocurrency wallet transaction histories for

MetaMask, Phantom, Coinbase Wallet, Binance Wallet, and TronLink are reviewed for any unauthorized transfers initiated during or after the compromise window before those wallets are abandoned.

Step 5: Post-Incident — Conduct a review of all third-party npm dependencies for dormant or unmonitored contributor accounts; require package maintainers to demonstrate MFA on all accounts with publish rights (CIS 6.3, CIS 6.5). Implement npm package integrity verification (subresource integrity or private registry mirroring) to enforce CWE-494 controls. Establish a Software Composition Analysis (SCA) gate in CI/CD pipelines to flag newly added or updated packages for review before installation (CIS 2.1, CIS 2.3, NIST CM family). Review and tighten least-privilege on CI/CD service accounts so build processes cannot access developer credential stores or cryptocurrency wallet directories (NIST AC-6). Document this event as a case study for developer security training on supply chain risk.

NIST Phase: Post Incident

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

Controls: NIST AC-6 (Least Privilege), CIS 6.3 (Require MFA for Externally-Exposed Applications), CIS 6.5 (Require MFA for Administrative Access), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.3 (Address Unauthorized Software), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Implement a private npm registry mirror using Verdaccio (free, self-hosted) configured to allow-list only approved package scopes and pin versions; any @mastra package update would require explicit approval before proxying to developers. Integrate `npm audit` and the free tier of Socket.dev CLI (`npx socket check`) as a pre-install CI step to detect postinstall script anomalies and newly introduced dependency risks in @mastra or any other scoped package. Enforce MFA on npm publish rights by auditing package team membership via `npm team ls :` and cross-referencing against npm's two-factor auth enforcement settings in the organization's npm account security settings. Use osquery scheduled queries to enforce least-privilege on CI/CD service account processes: `SELECT * FROM processes WHERE uid = (SELECT uid FROM users WHERE username = 'ci_runner') AND on_disk = 0` to detect memory-only payloads running as the build service account.

Evidence: This phase does not alter active incident state; no volatile capture is required. Document for the lessons-learned record: (1) a timeline of @mastra package version hashes from the npm registry audit log showing exactly when malicious versions were published by the compromised contributor account, to establish the definitive compromise window; (2) the full list of internal projects referencing @mastra packages extracted from all package-lock.json and yarn.lock files, as the authoritative blast radius inventory; (3) a mapping of which CI/CD service accounts had file-system access to developer credential stores (~/. ssh, .env files, cloud credential files) during the compromise window, to document the maximum possible lateral movement surface Sapphire Sleet could have exploited.

Detection Guidance

Primary detection surface is npm postinstall hook execution. Query endpoint detection logs for node.exe or npm spawning child processes (PowerShell, bash, sh, cmd) during package installation. On Linux/macOS, look for new LaunchAgent plists, LaunchDaemon plists, or systemd unit files created within minutes of an npm install invocation. On Windows, hunt for new Run key entries (HKCU\Software\Microsoft\Windows\CurrentVersion\Run) written by node or npm processes. File access indicators: reads of ~/.ssh/*, ~/.config/metamask, browser extension local storage directories for MetaMask/Phantom/Coinbase Wallet/Binance Wallet/TronLink, and .env files in project directories by npm child processes. Network indicators: outbound HTTP/S connections from build agents or developer workstations to non-CDN, non-registry IPs immediately following npm install, cross-reference against Sapphire Sleet C2 infrastructure published by Microsoft (2026-06-17), Kodem, Snyk, and Orca Security. SIEM query approach: correlate process creation events (parent: npm/node) with file writes to persistence directories and outbound network connections within a 60-second window. Review npm debug logs at ~/.npm/_logs/ for postinstall script

content. D3FEND countermeasures: D3-SFA (System File Analysis) for persistence directory monitoring; D3-LAM (Local Account Monitoring) for anomalous credential access; D3-UAP (User Account Permissions) to restrict npm postinstall execution context. NIST AU-2 event logging must cover process creation and file access; NIST AU-6 review should include CI/CD runner logs. CIS 8.2 audit log collection must be confirmed active on all build systems.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`See Microsoft Security Blog 2026-06-17 and Kodem IOC runbook for confirmed C2 domains`	Sapphire Sleet C2 infrastructure identified via Microsoft attribution and corroborated by Kodem, Snyk, Orca Security — specific values not reproduced here to avoid transcription error; retrieve directly from primary source	HIGH
HASH	`See Microsoft Security Blog 2026-06-17 and Kodem IOC runbook for payload hashes`	Malicious postinstall payload hashes published by Microsoft and Kodem — retrieve directly from primary source to ensure accuracy	HIGH
URL	`https://www.microsoft.com/en-us/security/blog/2026/06/17/postinstall-payload-inside-mastra-npm-supply-chain-compromise/`	Microsoft T1 source — primary IOC and attribution reference for this campaign; treat as canonical for C2 infrastructure and payload hashes	HIGH
URL	`https://www.kodemsecurity.com/resources/mastra-npm-packages-compromised-easy-day-js-supply-chain-attack-iocs-and-response-runbook`	Kodem IOC runbook and response guidance — includes structured IOC list and incident response steps	MEDIUM
URL	`https://snyk.io/blog/a-forgotten-contributor-account-compromised-the-entire-mastra-npm-package-scope/`	Snyk analysis of the dormant contributor account compromise vector and scope of affected packages	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1543.001** — Launch Agent
- **T1059.004** — Unix Shell
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1071.001** — Web Protocols
- **T1059.001** — PowerShell
- **T1528** — Steal Application Access Token

- **T1543.002** — Systemd Service
- **T1027** — Obfuscated Files or Information
- **T1562.001** — Disable or Modify Tools
- **T1608.003** — Install Digital Certificate
- **T1078** — Valid Accounts
- **T1543.003** — Windows Service
- **T1555** — Credentials from Password Stores
- **T1547.001** — Registry Run Keys / Startup Folder
- **T1552.001** — Credentials In Files

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **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
- **CM-3** — Configuration Change Control
- **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

ISO-27001-2022

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

NIST-CSF-2

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1543.001	Launch Agent	Persistence
T1059.004	Unix Shell	Execution
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1071.001	Web Protocols	Command-And-Control
T1059.001	PowerShell	Execution
T1528	Steal Application Access Token	Credential-Access
T1543.002	Systemd Service	Persistence
T1027	Obfuscated Files or Information	Defense-Evasion
T1562.001	Disable or Modify Tools	Defense-Evasion
T1608.003	Install Digital Certificate	Resource-Development
T1078	Valid Accounts	Defense-Evasion
T1543.003	Windows Service	Persistence
T1555	Credentials from Password Stores	Credential-Access
T1547.001	Registry Run Keys / Startup Folder	Persistence
T1552.001	Credentials In Files	Credential-Access

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/microsoft-links-mast...	T3

Source	URL	Tier
Mastra npm Compromise: easy-day-js Attack & Response | Kodem	https://www.kodemsecurity.com/resources/mastra-npm-packages-compro m...	T3
144 Mastra npm Packages Compromised via Supply Chain Attack	https://orca.security/resources/blog/mastra-npm-supply-chain-attack/	T3
Mastra npm Scope Takeover | Snyk	https://snyk.io/blog/a-forgotten-contributor-account-compromised-th...	T3
From package to postinstall payload: Inside the Mastra npm supply ...	https://www.microsoft.com/en-us/security/blog/2026/06/17/postinstal...	T1

Generated 2026-06-20 18:46 UTC by TJS Security Command Center