INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-28 06:45 UTC

JINX-0164: macOS Infostealer Campaign Chains Recruiter Lures to npm Supply Chain and CI/CD Compromise

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0375
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	macOS (Intel and Apple Silicon), npm package @velora-dex/sdk, VeloraDEX DeFi platform, CI/CD infrastructure, iCloud Keychain, Chrome browser, Discord, Slack, Telegram, cryptocurrency wallet browser extensions
Published	2026-05-28T03:54:48
Discovery Source	Rss

Executive Summary

A threat actor tracked as JINX-0164 is running an active campaign against cryptocurrency firms and software developers, combining fake LinkedIn recruiter outreach with a confirmed supply chain compromise of the npm package @velora-dex/sdk. Developers who installed the malicious package received persistent macOS malware capable of stealing credentials, SSH keys, cryptocurrency wallet data, and messaging platform sessions, while also granting the attacker remote access and CI/CD pipeline infiltration. Organizations in the cryptocurrency and DeFi sectors that employ macOS developers or depend on npm-distributed packages face immediate risk of credential theft, source code poisoning, and cascading compromise across their development infrastructure.

Technical Analysis

JINX-0164 delivers two custom macOS malware families through a combination of spearphishing via LinkedIn (T1566.002, T1566.003) and supply chain compromise (T1195.001, T1195.002). AUDIOFIX is an infostealer targeting iCloud Keychain (T1555.001), browser-stored credentials in Chrome (T1555.003), SSH private keys (T1552.004), and session tokens from Discord, Slack, and Telegram. MiniRAT is a Go-based remote access trojan (T1021) with lateral movement capability, dropped via the compromised @velora-dex/sdk npm package. The malicious package version establishes macOS persistence using launchctl (T1543.004), consistent with CWE-494 (Download of Code Without Integrity Check) and CWE-829 (Inclusion of Functionality from Untrusted

Control Sphere). Initial access via social engineering maps to CWE-506 (Embedded Malicious Code). CI/CD infrastructure infiltration (confirmed) enables downstream code poisoning (T1195). Command and control uses HTTP/S (T1071.001). Data is staged and exfiltrated (T1560, T1105). No CVE identifier has been assigned; no patch is available from the affected package maintainer at time of writing. The @velora-dex/sdk package should be treated as fully compromised across all versions published after mid-2025 until the maintainer confirms a clean release with verified provenance.

Action Checklist

1. Step 1: Containment, Audit all macOS developer workstations and CI/CD build nodes for installations of @velora-dex/sdk from npm. Run 'npm ls @velora-dex/sdk' across build environments and developer machines. Isolate any host where the package is present. Block outbound connections from build infrastructure to unrecognized external endpoints pending investigation. Remove or quarantine the package from all package-lock.json and yarn.lock files and lock package managers against reinstalling it (NIST SI-3, CIS 2.3).
2. Step 2: Detection, Search macOS hosts for launchctl persistence entries added after mid-2025: inspect ~/Library/LaunchAgents/ and /Library/LaunchDaemons/ for unfamiliar plist files. Review npm install logs and CI/CD pipeline logs for @velora-dex/sdk installation events. Hunt for outbound connections to unknown IPs from build nodes and developer endpoints. Check for new or modified SSH keys in ~/.ssh/ and review iCloud Keychain access logs where available. Look for process execution of Go-compiled binaries from unusual paths. MITRE techniques to hunt: T1543.004 (launchctl persistence), T1105 (ingress tool transfer), T1071.001 (C2 over HTTP/S), T1552.004 (SSH key access), T1555.001 (Keychain access) (CIS 8.2, NIST AU-6, NIST SI-4).
3. Step 3: Eradication, Remove all instances of @velora-dex/sdk from dependency trees and block the package at the npm registry proxy or internal artifact manager. Delete any launchctl persistence plists tied to the malware. Rotate all credentials stored in iCloud Keychain, Chrome, and messaging platforms on affected hosts. Revoke and regenerate all SSH keys from compromised machines. Revoke CI/CD pipeline secrets, API tokens, and signing certificates that were accessible from affected build nodes. Reimage compromised developer workstations rather than attempting in-place cleanup (NIST IR-4, D3-CRO, D3-CH, CIS 5.2).
4. Step 4: Recovery, Before restoring developer workstations to production use, verify absence of MiniRAT persistence entries and confirm clean launchctl state. Restore CI/CD pipelines only after rotating all secrets and validating pipeline configuration files against known-good versions in version control. Enable integrity verification on npm packages via lockfile pinning and hash validation going forward. Monitor rebuilt environments for 30 days for anomalous outbound connections and unexpected process spawns. Validate that no malicious commits were introduced to codebases accessible from compromised CI/CD nodes (NIST SI-7, NIST AU-6, CIS 8.2).
5. Step 5: Post-Incident, Conduct a dependency audit across all projects to identify other third-party npm packages without verified provenance or integrity pinning, addressing CWE-494 and CWE-829 at the program level. Implement a formal software supply chain policy requiring lockfile integrity enforcement, Sigstore or equivalent package signing verification, and private registry mirroring for critical dependencies (NIST SR-4, CIS 2.1, CIS 2.2). Establish LinkedIn and recruiter-contact awareness training for developers, specifically addressing job-lure social engineering patterns used by JINX-0164 (NIST AT-2). Deploy endpoint detection on macOS developer workstations with coverage for launchctl persistence and credential access behaviors (D3-LAM, D3-SFA, CIS 4.4).

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate immediately to CISO and legal counsel if forensic analysis confirms that CI/CD pipeline secrets, code-signing certificates, or cryptocurrency wallet private keys were exfiltrated, as this constitutes a confirmed supply chain breach with potential downstream customer impact and may trigger breach notification obligations under applicable data protection regulations.
Recovery Notes	CI/CD pipelines must not be restored to production use until every secret, token, and certificate accessible from compromised runners has been rotated and the rotation is confirmed in the secrets management audit log — partial rotation leaves JINX-0164 with persistent access. Rebuilt developer workstations should be monitored for 30 days using daily launchctl baseline diffs and outbound connection logging, as MiniRAT variants may include delayed or conditional persistence mechanisms that survive a naive reimage if the backup image itself is from the compromise window. Additionally, audit all git repositories accessible from compromised CI/CD nodes for unauthorized commits or workflow file modifications introduced during the compromise period, as CI/CD infiltration is a confirmed JINX-0164 capability.
Forensic Artifacts	LaunchAgent/LaunchDaemon plist files in ~/Library/LaunchAgents/ and /Library/LaunchDaemons/ with creation timestamps post-dating @velora-dex/sdk installation — these represent MiniRAT's persistence mechanism (T1543.004) and will contain the path to the dropped Go binary and its scheduled execution parameters | Go-compiled Mach-O binary dropped to disk by the malicious @velora-dex/sdk postinstall script — typically written to /tmp, ~/Library/, or a dot-prefixed directory; recoverable via 'find / -type f -newer | xargs file | grep Mach-O' and should be preserved with SHA-256 hash before any cleanup | macOS Unified Log ('log collect') entries covering securityd/keychain subsystem activity, capturing iCloud Keychain access events triggered by the MiniRAT credential harvester (T1555.001), timestamped relative to the malware execution window | Chrome 'Login Data' SQLite database and 'Cookies' file from '~/Library/Application Support/Google/Chrome/Default/' — JINX-0164 specifically targets these for credential and session token theft; capture with file metadata timestamps to establish whether files were accessed or copied by the malware | CI/CD pipeline run logs (GitHub Actions workflow run history, Jenkins build console output, GitLab CI job logs) for the period spanning @velora-dex/sdk installation, preserving evidence of which secrets were injected as environment variables into jobs that executed on the compromised runner and what external network calls were made during those runs

Per-Action IR Details

Step 1: Containment — Immediately audit all macOS developer workstations and CI/CD build nodes for installations of @velora-dex/sdk from npm. Run 'npm ls @velora-dex/sdk' across build environments and developer machines. Isolate any host where the package is present. Block outbound connections from build infrastructure to unrecognized external endpoints pending investigation. Remove or quarantine the package from all package-lock.json and yarn.lock files and lock package managers against reinstalling it (NIST SI-3, CIS 2.3).

NIST Phase: Containment

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

Controls: NIST SI-3 (Malicious Code Protection), NIST IR-4 (Incident Handling), NIST CM-7 (Least Functionality), CIS 2.3 (Address Unauthorized Software), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: On each macOS host, run: 'find / -name package-lock.json -o -name yarn.lock 2>/dev/null | xargs grep -l "@velora-dex/sdk"' to locate all dependency files referencing the malicious package. Use 'pfctl' or macOS Application Firewall ('socketfilterfw') to block outbound traffic from build nodes to non-allowlisted IPs. For CI/CD runners (GitHub Actions, Jenkins, GitLab CI), disable the runner service immediately: 'sudo systemctl stop github-actions-runner' or equivalent, and revoke runner registration tokens from the control plane. Use 'osquery' with query 'SELECT name, version, path FROM npm_packages WHERE name = "@velora-dex/sdk";' if osquery npm tables are available, or iterate npm workspaces manually.

Evidence: Before isolating any host, capture: full 'npm ls --all' output and package-lock.json/yarn.lock contents to document the exact installed version of @velora-dex/sdk and its dependency tree; macOS Unified Log entries ('log collect --last 72h') preserving launchctl service registration events; a 'netstat -anp tcp' snapshot and 'lsof -i' output capturing active outbound connections from the affected build node at time of discovery; contents of ~/Library/LaunchAgents/ and /Library/LaunchDaemons/ with file creation timestamps ('ls -la@' or 'mdls' for extended metadata); and a memory acquisition (via osxpmem or similar) if the host is actively beaconing, to capture in-memory MiniRAT artifacts before process termination on isolation.

Step 2: Detection — Search macOS hosts for launchctl persistence entries added after mid-2025: inspect ~/Library/LaunchAgents/ and /Library/LaunchDaemons/ for unfamiliar plist files. Review npm install logs and CI/CD pipeline logs for @velora-dex/sdk installation events. Hunt for outbound connections to unknown IPs from build nodes and developer endpoints. Check for new or modified SSH keys in ~/.ssh/ and review iCloud Keychain access logs where available. Look for process execution of Go-compiled binaries from unusual paths. MITRE techniques to hunt: T1543.004 (launchctl persistence), T1105 (ingress tool transfer), T1071.001 (C2 over HTTP/S), T1552.004 (SSH key access), T1555.001 (Keychain access) (CIS 8.2, NIST AU-6, NIST SI-4).

NIST Phase: Detection Analysis

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

Controls: NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST SI-4 (System Monitoring), NIST AU-12 (Audit Record Generation), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Hunt for JINX-0164 persistence using: 'sudo find /Library/LaunchDaemons ~/Library/LaunchAgents -name "*.plist" -newer /tmp/ref_mid2025 -ls' (create ref file with 'touch -t 202506010000 /tmp/ref_mid2025'). Identify Go-compiled binaries dropped by the malware: 'find /tmp /var/tmp ~/Library -type f -newer /tmp/ref_mid2025 | xargs file 2>/dev/null | grep "Mach-O"'. For iCloud Keychain access attempts, query macOS Unified Log: 'log stream --predicate "subsystem == \"com.apple.securityd\" AND category == \"keychain\"" --info'. For SSH key theft (T1552.004), check: 'ls -la ~/.ssh/ && stat -f "%m %N" ~/.ssh/*' for modification timestamps post-installation. Write a YARA rule targeting Go ELF/Mach-O binaries with embedded C2 callback strings common to MiniRAT variants. Use Wireshark or 'tcpdump -i en0 -w capture.pcap "not (dst net 10.0.0.0/8 or dst net 192.168.0.0/16)"' on build nodes to capture C2 beaconing traffic (T1071.001).

Evidence: Capture before analysis: macOS Unified Log archive covering the period from @velora-dex/sdk installation timestamp forward ('log collect --start YYYY-MM-DD' anchored to npm install date from CI/CD logs); npm install history from '~/.npm/_logs/' and CI/CD runner stdout logs showing the specific pipeline run that introduced @velora-dex/sdk; plist file contents and 'plutil -p' output for any LaunchAgent/LaunchDaemon entries created post-installation, which MiniRAT uses for persistence (T1543.004); 'security dump-keychain -d login.keychain' output (requires user auth) to establish a baseline of what credentials existed before compromise; and Chrome 'Login Data' SQLite database ('~/Library/Application Support/Google/Chrome/Default/Login Data') and session cookie files, which this campaign specifically targets for credential theft.

Step 3: Eradication — Remove all instances of @velora-dex/sdk from dependency trees and block the package at the npm registry proxy or internal artifact manager. Delete any launchctl persistence plists tied to the malware. Rotate all credentials stored in iCloud Keychain, Chrome, and messaging platforms on affected hosts. Revoke and regenerate all SSH keys from compromised machines. Revoke CI/CD pipeline secrets, API

tokens, and signing certificates that were accessible from affected build nodes. Reimage compromised developer workstations rather than attempting in-place cleanup (NIST IR-4, D3-CRO, D3-CH, CIS 5.2).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST CM-2 (Baseline Configuration), NIST IA-5 (Authenticator Management), NIST AC-2 (Account Management), CIS 5.2 (Use Unique Passwords), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Block @velora-dex/sdk at the registry level: if using Verdaccio or Nexus as an npm proxy, add the package to the blocklist immediately. For teams using the public registry directly, add '.npmrc' entries: 'ignore-scripts=true' (prevents postinstall execution) and add '@velora-dex' to a blocked scope. Unload and delete MiniRAT LaunchAgent plists: 'launchctl bootout gui/$(id -u) ~/Library/LaunchAgents/.plist && rm ~/Library/LaunchAgents/.plist'. For SSH key rotation, revoke keys from all GitHub/GitLab/Bitbucket accounts associated with the compromised developer: use each platform's API ('gh ssh-key delete ') and push new keys from a clean machine. For CI/CD secret rotation in GitHub Actions: 'gh secret set --body "" ' for every secret accessible to workflows that ran on the compromised runner. Rotate Discord, Slack, and Telegram session tokens by logging out all sessions from account settings on a clean device — these platforms are specifically targeted by JINX-0164 for session hijacking.

Evidence: Before reimaging, preserve as forensic images: full disk image of the compromised macOS workstation using 'dd' or Target Disk Mode to an external drive for later analysis; copy of all LaunchAgent/LaunchDaemon plists with SHA-256 hashes ('shasum -a 256 ~/Library/LaunchAgents/*.plist'); the Go-compiled MiniRAT binary (do not delete before hashing and preserving — submit to VirusTotal and internal malware repo); ~/.ssh/known_hosts and ~/.ssh/id_* files with metadata timestamps to establish when keys were accessed or copied; and CI/CD pipeline run logs (GitHub Actions workflow run IDs, Jenkins build IDs) that show which jobs executed on the compromised runner and what secrets were injected into the environment.

Step 4: Recovery — Before restoring developer workstations to production use, verify absence of MiniRAT persistence entries and confirm clean launchctl state. Restore CI/CD pipelines only after rotating all secrets and validating pipeline configuration files against known-good versions in version control. Enable integrity verification on npm packages via lockfile pinning and hash validation going forward. Monitor rebuilt environments for 30 days for anomalous outbound connections and unexpected process spawns. Validate that no malicious commits were introduced to codebases accessible from compromised CI/CD nodes (NIST SI-7, NIST AU-6, CIS 8.2).

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST CM-3 (Configuration Change Control), NIST SR-4 (Provenance), CIS 8.2 (Collect Audit Logs), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Verify clean launchctl state on rebuilt macOS hosts: 'launchctl list | grep -v com.apple' to surface any non-Apple LaunchAgent entries for review. For npm integrity validation going forward, enforce 'npm ci' instead of 'npm install' in all CI/CD pipelines — this mandates lockfile adherence and fails on dependency tree changes. Add 'npm audit --audit-level=high' as a required pipeline step. For commit validation on repositories accessible from compromised CI/CD nodes, run: 'git log --all --since="" --format="%H %an %ae %s" | grep -v "" ' to surface unexpected commits. Deploy a 30-day monitoring cron on rebuilt macOS workstations: 'crontab -e' with a daily job running 'launchctl list | grep -v com.apple > /tmp/launchctl_$(date +%F).log' and diffing against the post-reimage baseline. Monitor for MiniRAT C2 beaconing patterns (Go HTTP/S callbacks) using pfctl logging or endpoint firewall logs.

Evidence: Before declaring recovery complete, capture: git diff output comparing current HEAD of all repositories against the last known-clean commit prior to the CI/CD compromise, specifically hunting for unexpected changes to package.json, .github/workflows/, Jenkinsfile, or .gitlab-ci.yml (JINX-0164's CI/CD infiltration vector); 'npm audit' output and 'npm list --all' from the rebuilt environment confirming @velora-dex/sdk and its transitive dependencies are fully absent; a signed attestation of the new SSH public keys deployed to GitHub/GitLab and CI/CD systems for

chain-of-custody; and macOS System Integrity Protection status ('csrutil status') and Gatekeeper verification logs from rebuilt hosts to confirm baseline security posture.

Step 5: Post-Incident — Conduct a dependency audit across all projects to identify other third-party npm packages without verified provenance or integrity pinning, addressing CWE-494 and CWE-829 at the program level. Implement a formal software supply chain policy requiring lockfile integrity enforcement, Sigstore or equivalent package signing verification, and private registry mirroring for critical dependencies (NIST SR-4, CIS 2.1, CIS 2.2). Establish LinkedIn and recruiter-contact awareness training for developers, specifically addressing job-lure social engineering patterns used by JINX-0164 (NIST AT-2). Deploy endpoint detection on macOS developer workstations with coverage for launchctl persistence and credential access behaviors (D3-LAM, D3-SFA, CIS 4.4).

NIST Phase: Post Incident

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

Controls: NIST SR-4 (Provenance), NIST AT-2 (Literacy Training and Awareness), NIST SI-7 (Software, Firmware, and Information Integrity), NIST RA-3 (Risk Assessment), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: For the dependency provenance audit (CWE-494, CWE-829), run 'npx better-npm-audit' or 'npm audit --json' across all repositories and flag packages with no integrity hash in package-lock.json. Implement Sigstore cosign verification for critical packages where signing attestations exist: 'cosign verify-attestation --type slsaprovenance '. For teams without a private registry, configure .npmrc with 'package-lock=true' and 'save-exact=true' enforced via a pre-commit hook. For macOS endpoint detection without commercial EDR, deploy an Osquery + Kolide Fleet (free tier) configuration with the following scheduled query for launchctl persistence: 'SELECT name, path, program, run_at_load FROM launchd WHERE run_at_load = 1 AND path NOT LIKE "/System/%";'. For social engineering awareness specific to JINX-0164, brief developers on the confirmed pattern: LinkedIn recruiter contact → coding challenge or SDK integration request → malicious npm package installation. Create a one-page reference card with red flags: unsolicited recruiter outreach requesting package installation, GitHub repos with no issue history, npm packages with low download counts but high dependency claims.

Evidence: For the post-incident lessons learned (NIST 800-61r3 §4), document and retain: the complete timeline from first JINX-0164 LinkedIn contact (if identifiable from developer communications) through detection, mapped against CI/CD pipeline run timestamps to establish the full blast radius; a dependency graph of all npm packages installed on affected build nodes during the compromise window, generated via 'npm list --all --json > dep_snapshot.json', to identify any other packages that may have been trojanized or tampered with by the same threat actor; IOCs extracted from the MiniRAT binary (C2 IP addresses, hardcoded strings, Go build metadata) for submission to your threat intel platform and sharing via ISAC channels relevant to the cryptocurrency/DeFi sector; and a record of all CI/CD pipeline secrets, signing certificates, and API tokens that were in scope during the compromise, with confirmation of rotation dates and responsible parties, to satisfy any regulatory or audit obligations.

Detection Guidance

Primary hunt surface is macOS developer workstations and CI/CD build nodes. Key behavioral indicators: (1) Launchctl persistence, enumerate ~/Library/LaunchAgents/ and /Library/LaunchDaemons/ for plist files created or modified after mid-2025 that reference executables in /tmp/, ~/Library/Application Support/, or other non-standard paths. (2) npm install events, search CI/CD logs and npm debug logs for '@velora-dex/sdk' installation. (3) Go binary execution, look for process trees spawned from npm postinstall scripts that execute compiled Go binaries, particularly from temporary or hidden directories. (4) Credential access, on macOS, audit Security framework calls accessing the Keychain (securityd events), SSH key file reads from ~/.ssh/, and Chrome's Login Data SQLite file access outside normal browser processes. (5) C2 communications, inspect DNS and proxy logs for developer workstations and build nodes making outbound HTTP/S connections to

domains or IPs with no prior organizational history, particularly from node or build agent processes. (6) Lateral movement, review SSH authentication logs across internal hosts for connections originating from developer workstations at unusual times or using recently added keys. MITRE ATT&CK coverage priorities: T1543.004, T1555.001, T1555.003, T1552.004, T1071.001, T1105, T1021. Apply NIST SI-4 (system monitoring) and AU-6 (audit review) to build infrastructure as a priority; these nodes are high-value lateral movement targets.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`@velora-dex/sdk (npm package)`	Compromised npm package delivering MiniRAT and establishing macOS launchctl persistence; treat all versions published post mid-2025 as malicious until maintainer confirms clean release	HIGH
HASH	`Not publicly disclosed at time of writing`	File hashes for AUDIOFIX infostealer and MiniRAT Go RAT binaries have not been confirmed in available T3 sources; monitor threat intel feeds for updates	LOW
URL	`Not publicly disclosed at time of writing`	C2 infrastructure URLs and callback domains for MiniRAT have not been confirmed in available sources; behavioral detection is the primary hunt approach	LOW

Framework Mappings

MITRE-ATTACK

- **T1555** — Credentials from Password Stores
- **T1071.001** — Web Protocols
- **T1059** — Command and Scripting Interpreter
- **T1566.003** — Spearphishing via Service
- **T1566** — Phishing
- **T1078** — Valid Accounts
- **T1071** — Application Layer Protocol
- **T1552.004** — Private Keys
- **T1560** — Archive Collected Data
- **T1566.002** — Spearphishing Link
- **T1059.004** — Unix Shell
- **T1021** — Remote Services
- **T1195.002** — Compromise Software Supply Chain
- **T1056** — Input Capture

- **T1555.003** — Credentials from Web Browsers
- **T1555.001** — Keychain
- **T1059.006** — Python
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1082** — System Information Discovery
- **T1204.002** — Malicious File
- **T1543.004** — Launch Daemon
- **T1195** — Supply Chain Compromise
- **T1105** — Ingress Tool Transfer

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-7** — Software, Firmware, and Information Integrity
- **AT-2** — Literacy Training and Awareness
- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-8** — Spam Protection
- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **AC-17** — Remote Access
- **AC-3** — Access Enforcement
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SR-2** — Supply Chain Risk Management Plan
- **CM-3** — Configuration Change Control

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **6.3** — Require MFA for Externally-Exposed Applications
- **14.2** — Train Workforce Members to Recognize Social Engineering Attacks
- **15.1** — Establish and Maintain an Inventory of Service Providers

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

ISO-27001-2022

- **A.5.21** — Managing information security in the ICT supply chain
- **A.5.23** — Information security for use of cloud services

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1555	Credentials from Password Stores	Credential-Access
T1071.001	Web Protocols	Command-And-Control
T1059	Command and Scripting Interpreter	Execution
T1566.003	Spearphishing via Service	Initial-Access
T1566	Phishing	Initial-Access
T1078	Valid Accounts	Defense-Evasion
T1071	Application Layer Protocol	Command-And-Control
T1552.004	Private Keys	Credential-Access
T1560	Archive Collected Data	Collection
T1566.002	Spearphishing Link	Initial-Access
T1059.004	Unix Shell	Execution
T1021	Remote Services	Lateral-Movement
T1195.002	Compromise Software Supply Chain	Initial-Access
T1056	Input Capture	Collection
T1555.003	Credentials from Web Browsers	Credential-Access
T1555.001	Keychain	Credential-Access
T1059.006	Python	Execution

Technique ID	Technique Name	Tactic
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1082	System Information Discovery	Discovery
T1204.002	Malicious File	Execution
T1543.004	Launch Daemon	Persistence
T1195	Supply Chain Compromise	Initial-Access
T1105	Ingress Tool Transfer	Command-And-Control

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/05/jinx-0164-targets-cryptocurrency-...	T3
@velora-dex/sdk Compromised on npm: Malicious Version Drops ...	https://www.stepsecurity.io/blog/velora-dex-sdk-compromised-on-npm-...	T3
@velora-dex/sdk npm Compromise: macOS Implant via Build Pipeline	https://phoenix.security/velora-dex-sdk-supply-chain-compromise-bui...	T3
Malicious @velora-dex/sdk Delivers Go RAT via npm - SafeDep	https://safedep.io/malicious-velora-dex-sdk-npm-compromised-rat	T3
Charlie Eriksen's Post - Package compromised on Github - LinkedIn	https://www.linkedin.com/posts/charlie-eriksen-a318578_package-comp...	T3

Generated 2026-05-28 06:45 UTC by TJS Security Command Center