INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-06 09:04 UTC

Axios npm Library: Prototype Pollution Read-Side Gadgets Enable Credential Injection and Request Hijacking

CVE VULNERABILITY | HIGH | CVSS 8.1

SCC Item ID	SCC-CVE-2026-0131
Type	CVE Vulnerability
CVE ID	CVE-2026-42264
Severity	HIGH
CVSS Base Score	8.1
Affected Products	axios (npm), specific affected versions not confirmed from available sources
Published	2026-05-05T00:18:38Z
Discovery Source	Osv

Executive Summary

A high-severity vulnerability in Axios, a widely-used JavaScript HTTP client library, allows attackers to inject credentials or hijack outbound HTTP requests by exploiting read-side prototype pollution gadgets in its Node.js adapter. Any application or service built with the affected Axios version that processes attacker-influenced input is potentially exposed. The business risk is unauthorized access to downstream APIs, backend services, or authenticated sessions, with cascading exposure across any system that trusts those requests.

Technical Analysis

CVE-2026-42264 (GHSA-q8qp-cvcw-x6jj, CWE-1321) affects the Axios npm library's HTTP adapter for Node.js. The vulnerability involves read-side prototype pollution gadgets, code paths within Axios that read properties from the JavaScript prototype chain rather than writing to it. If an attacker can influence input that reaches these code paths (e.g., via user-supplied headers, config objects, or request parameters), they can cause Axios to read attacker-controlled values from the prototype chain, enabling credential injection into outgoing HTTP requests or full request hijacking. MITRE ATT&CK mappings include T1557 (Adversary-in-the-Middle) and T1565.002 (Stored Data Manipulation). CVSS base score: 8.1 (High). Consult GHSA-q8qp-cvcw-x6jj and CVE-2026-42264 in the NVD for confirmed affected version ranges and patch status.

Action Checklist

1. Step 1: Containment. Audit all Node.js applications and services in your environment for Axios as a direct or transitive dependency. Run 'npm list axios' or review lockfiles (package-lock.json, yarn.lock) across repositories. Isolate or add input validation wrappers around any service that passes user-controlled data into Axios request configuration objects until a patch is confirmed available.
2. Step 2: Detection. Search application logs for anomalous outbound HTTP requests: unexpected Authorization header keys or credential values in request logs (if logs do not contain plaintext credentials due to sanitization, correlate with failed authentication errors in destination services). Query your SIEM for outbound traffic from backend services to atypical destinations. Review Node.js application debug logs for unexpected prototype property resolution if debug logging is enabled.
3. Step 3: Eradication. Confirm the patched version of Axios via the OSV advisory GHSA-q8qp-cvcw-x6jj and NVD entry CVE-2026-42264 once available. Upgrade Axios to the confirmed fixed version across all direct and transitive dependencies. Re-audit lockfiles post-upgrade. Verify the patched version changelog explicitly addresses prototype pollution or GHSA-q8qp-cvcw-x6jj. If patch notes do not reference this issue, contact Axios maintainers for confirmation before deploying.
4. Step 4: Recovery. After upgrading, run regression tests on all services using Axios to verify HTTP request behavior is correct. Monitor outbound request logs for 48-72 hours post-patch for residual anomalies. Rotate any credentials (API keys, tokens, bearer credentials) that were configured into Axios-dependent services during the exposure window, as these credentials may have been misrouted or hijacked in outbound requests by an attacker. Even if no unauthorized access is detected, rotation prevents delayed exploitation of cached or logged credentials.
5. Step 5: Post-Incident. This vulnerability exposes a control gap in dependency inventory and supply-chain visibility. Implement or validate software composition analysis (SCA) tooling (e.g., integrated into CI/CD pipelines) to detect vulnerable transitive dependencies before deployment. Review input sanitization practices for any code that passes user-controlled data into HTTP client configuration objects.

IR / Forensic Enrichment

Triage Priority	URGENT
Escalation Criteria	Escalate to CISO and legal/compliance immediately if post-detection log review confirms that injected credentials were used to authenticate to downstream APIs handling PII, PHI, or financial data, or if any downstream service owner reports unauthorized access during the Axios exposure window, as this may trigger breach notification obligations under GDPR, HIPAA, or PCI-DSS.
Recovery Notes	Post-patch, validate that no vulnerable Axios version remains in any running container or serverless function by querying the runtime environment directly rather than relying solely on updated lockfiles — container images built before the patch deployment may still be in service. Monitor outbound request Authorization headers from all previously affected services for 72 hours using the interceptor logging approach established in Step 2, comparing against a known-good baseline captured immediately after patch deployment. Do not consider recovery complete until all credentials that transited Axios-dependent services during the exposure window have been rotated and confirmed invalidated at the issuing authority.

Forensic Artifacts	Node.js application debug logs containing serialized Axios request config objects: look for unexpected 'auth', 'headers.Authorization', or 'headers.Cookie' fields populated via prototype chain resolution rather than application-assigned values — these indicate successful gadget exploitation under CVE-2026-42264. | Outbound HTTP proxy or WAF access logs from Axios-dependent backend services: filter for requests to destinations outside the application's normal API call baseline that carry Authorization headers, which would indicate credential injection enabling request hijacking to unintended endpoints. | package-lock.json and yarn.lock files with sha256 checksums from all affected repositories at the time of incident, preserving the exact resolved Axios version tree as evidence of which deployments were vulnerable during the exposure window. | Network packet captures (pcap) from backend service egress interfaces during the exposure window: inspect for HTTP requests with injected Authorization or credential headers to endpoints not present in the application's documented upstream dependencies. | Node.js process heap dumps captured before patching (via 'gcore ' or '--heapsnapshot' V8 flag): may contain in-memory evidence of Object.prototype pollution — specifically unexpected enumerable properties added to the global prototype that correspond to Axios request config fields such as 'auth', 'headers', or 'baseURL'.

Per-Action IR Details

Step 1: Containment — Audit all Node.js applications and services in your environment for Axios as a direct or transitive dependency. Run 'npm list axios' or review lockfiles (package-lock.json, yarn.lock) across repositories. Isolate or add input validation wrappers around any service that passes user-controlled data into Axios request configuration objects until a patch is confirmed available.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST CM-8 (System Component Inventory), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: For teams without an SCA platform, run 'find . -name package-lock.json | xargs grep -l "axios"' recursively across all code repositories to identify affected services. Follow with 'npm list axios --depth=10 2>/dev/null | grep axios' in each service root to surface transitive dependencies. As an immediate input validation compensating control, add a middleware shim that strips or rejects any request configuration property whose key resolves via the prototype chain — a two-line check using 'Object.prototype.hasOwnProperty' before passing config objects into Axios can break the pollution gadget path without a patch.

Evidence: Before isolating services, snapshot the current state of all package-lock.json and yarn.lock files across repositories with checksums (sha256sum) to establish a pre-patch baseline. Capture a full dependency tree output ('npm list --all --json > deptree-snapshot-$(date +%F).json') per service so you can prove which Axios version was in production at the time of exposure. If the application runs in a container, export the container image layer manifest ('docker inspect ') to record the exact installed Axios version at runtime, independent of source repo lockfiles.

Step 2: Detection — Search application logs for anomalous outbound HTTP requests: unexpected Authorization headers, unusual credential values in request logs, or requests to unexpected endpoints initiated by Axios-dependent services. Query your SIEM for outbound traffic from backend services to atypical destinations. Review Node.js application debug logs for unexpected prototype property resolution if debug logging is enabled.

NIST Phase: Detection Analysis

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

Controls: NIST SI-4 (System Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-2 (Event Logging), NIST AU-3 (Content of Audit Records), CIS 8.2 (Collect Audit Logs)

Compensating: Without a SIEM, use Node.js application-level request interceptors (Axios interceptors added to the instance) to log the full resolved config object — including headers and auth fields — to a local file before each outbound request; redirect output to a rotating log file using 'winston' or 'pino'. Then grep those logs for anomalous Authorization header values: 'grep -E "Authorization|Bearer|Basic" /var/log/app/axios-requests.log | awk '{print $0}' | sort | uniq -c | sort -rn' to surface repeated or unexpected credential strings. For network-level detection without EDR, run a tcpdump capture on the egress interface of backend hosts ('tcpdump -i eth0 -w /tmp/axios-egress-$(date +%F).pcap port 443 or port 80') and inspect with Wireshark for unexpected destination IPs or injected Authorization headers in plaintext HTTP traffic.

Evidence: Capture Node.js process-level stdout/stderr logs for the period covering the exposure window — specifically look for serialized Axios request config objects in debug output that include unexpected 'auth', 'headers.Authorization', or 'headers.Cookie' fields populated from prototype chain resolution rather than application-set values. Collect network flow logs (NetFlow, VPC flow logs, or firewall session logs) showing outbound connections from Axios-dependent service hosts to destinations not present in a baseline of normal API call targets. If the application uses an HTTP proxy, pull proxy access logs and filter for requests originating from the affected service with Authorization headers present on endpoints that do not normally require authentication.

Step 3: Eradication — Confirm the patched version of Axios via the OSV advisory GHSA-q8qp-cvcw-x6jj and NVD entry CVE-2026-42264 once available. Upgrade Axios to the confirmed fixed version across all direct and transitive dependencies. Re-audit lockfiles post-upgrade. Do not accept an upgrade that does not explicitly reference this CVE as resolved.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-3 (Configuration Change Control), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 7.4 (Perform Automated Application Patch Management), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Without automated patch management tooling, use 'npm audit fix --force' only after manually reviewing the proposed changeset with 'npm audit fix --dry-run' to confirm it resolves CVE-2026-42264 specifically and does not silently downgrade or break peer dependencies. For transitive dependency pinning, use 'overrides' in package.json (npm v8.3+) or 'resolutions' in package.json (Yarn) to force the patched Axios version across all dependency subtrees: add '"overrides": { "axios": ">=" }' where is confirmed from GHSA-q8qp-cvcw-x6jj. After upgrade, re-run 'npm list axios --depth=10' and diff against the pre-patch baseline snapshot to verify no residual vulnerable version remains in any subtree.

Evidence: Before executing the upgrade, preserve a forensic copy of all node_modules directories for affected services (tar and hash with sha256sum) and capture running process memory if the service is actively exploitable — use 'gcore ' on Linux to dump the Node.js process heap, which may contain in-memory evidence of polluted prototype properties (Object.prototype.__proto__ modifications) that would be lost after restart. Document the exact installed Axios version from the running container or host ('node -e "require('axios/package.json').version |> console.log"') as this may differ from lockfile if the image was built without lockfile enforcement.

Step 4: Recovery — After upgrading, run regression tests on all services using Axios to verify HTTP request behavior is correct. Monitor outbound request logs for 48-72 hours post-patch for residual anomalies. Rotate any credentials (API keys, tokens, bearer credentials) that transited through Axios-dependent services during the potential exposure window.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST IR-5 (Incident Monitoring), 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: For credential rotation without an automated secrets management platform, enumerate all API keys and bearer tokens in use by Axios-dependent services by searching application configuration files and environment variables ('grep -rE "(API_KEY|AUTH_TOKEN|BEARER|SECRET)" /etc/app/ /opt/app/.env* 2>/dev/null'). Rotate each

credential at the issuing service (upstream API provider, OAuth server, internal auth service) and redeploy the updated secrets to affected services. For the 48-72 hour monitoring window without a SIEM, schedule a cron job that runs the Axios interceptor log grep command from Step 2 every 30 minutes and emails a diff of new unique Authorization header values to the response team: 'crontab -e' with '*/30 * * * * grep -E "Authorization" /var/log/app/axios-requests.log | sort | uniq > /tmp/auth-headers-current.txt && diff /tmp/auth-headers-baseline.txt /tmp/auth-headers-current.txt | mail -s "Axios auth anomaly check" sec@yourdomain'.

Evidence: Prior to credential rotation, collect a full audit trail of all outbound API calls made by each Axios-dependent service during the exposure window from proxy logs, WAF logs, or application request logs — this establishes which downstream services received requests that may have carried injected credentials and scopes the blast radius for downstream breach notification assessment. Preserve this log set as evidence before rotating credentials, as rotation will make it impossible to retrospectively match injected credential values to specific downstream API sessions.

Step 5: Post-Incident — This vulnerability exposes a control gap in dependency inventory and supply-chain visibility. Implement or validate software composition analysis (SCA) tooling (e.g., integrated into CI/CD pipelines) to detect vulnerable transitive dependencies before deployment. Review input sanitization practices for any code that passes user-controlled data into HTTP client configuration objects.

NIST Phase: Post Incident

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

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SI-2 (Flaw Remediation), NIST SI-10 (Information Input Validation), NIST SA-15 (Development Process, Standards, and Tools), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Without a commercial SCA tool, integrate 'npm audit --audit-level=high' as a mandatory CI/CD pipeline step that fails the build on high or critical severity findings — this is natively available in npm v6+ at zero cost. Supplement with OSV-Scanner (free, Google-maintained, CLI tool) configured to scan package-lock.json files on every pull request merge: 'osv-scanner --lockfile package-lock.json' will detect CVE-2026-42264 and similar prototype pollution CVEs in Axios once indexed. For input validation against prototype pollution specifically, add a shared library function to your Node.js services that rejects any object key equal to '__proto__', 'constructor', or 'prototype' before passing user-supplied data into Axios config: this directly eliminates the read-side gadget attack surface documented in CVE-2026-42264.

Evidence: Produce a post-incident dependency graph report showing all services that carried a vulnerable Axios version, the exposure window (first vulnerable deployment date to patch date), and all downstream APIs those services communicated with — this document supports both internal lessons learned and any regulatory breach notification assessment. Archive the pre-patch lockfile snapshots, the heap dumps (if captured in Step 3), and the 48-72 hour post-patch monitoring logs as the evidentiary record for this incident.

Detection Guidance

Detection is complex because read-side prototype pollution does not always produce obvious errors. Focus on behavioral anomalies: (1) Outbound HTTP request logs - look for unexpected Authorization, Cookie, or Proxy-Authorization header values or unexpected header keys in requests originating from Node.js services. If logs are sanitized and do not contain plaintext credentials, correlate with failed authentication errors in destination services. (2) Application error logs - prototype chain traversal issues may surface as unexpected property types or null-reference errors in Axios internals; search for stack traces referencing Axios adapter code. (3) Network traffic - compare baseline outbound destinations for Axios-dependent services against current traffic; flag requests to unexpected hosts or with unexpected credential material. (4) SIEM query - filter for HTTP requests from application servers where the User-Agent or request signature matches Axios and the destination or credential values deviate from baseline. No public IOCs (IPs, hashes, domains) are available for this vulnerability at this time.

Framework Mappings

MITRE-ATTACK

- **T1557** — Adversary-in-the-Middle
- **T1565.002** — Transmitted Data Manipulation

CIS-V8

- **6.3** — Require MFA for Externally-Exposed Applications
- **8.2** — Collect Audit Logs

HIPAA-SECURITY

- **164.312(d)** — Person or Entity Authentication

SOC2-TSC

- **CC6.1** — Logical access security software, infrastructure, and architectures

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities

NIST-800-53R5

- **SI-4** — System Monitoring

NIST-CSF-2

- **DE.CM-01** — Networks and network services are monitored

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1557	Adversary-in-the-Middle	Credential-Access
T1565.002	Transmitted Data Manipulation	Impact

Sources

Source	URL	Tier
osv	https://osv.dev/vulnerability/GHSA-q8qp-cvcw-x6jj	T3
CVE-2026-42264 - CVE Record	https://www.cve.org/CVERecord?id=CVE-2026-42264	T3
CVE-2026-42641 Detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-42641	T1

Source	URL	Tier
Lenovo Product Security Advisories and Announcements	https://pcsupport.lenovo.com/ky/en/products/laptops-and-netbooks/id...	T3
CVE-2026-41264 | Tenable®	https://www.tenable.com/cve/CVE-2026-41264	T3
NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-42264	T1

Generated 2026-05-06 09:04 UTC by TJS Security Command Center