INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-01 06:14 UTC

GHSA-3g43-6gmg-66jw: axios Vulnerable to Credential Theft and Response Hijacking via Prototype Pollut

CVE VULNERABILITY | HIGH | CVSS 8.1

SCC Item ID	SCC-CVE-2026-0246
Type	CVE Vulnerability
CVE ID	CVE-2026-44495
Severity	HIGH
CVSS Base Score	8.1
Affected Products	axios (npm), specific version range not confirmed from available data
Published	2026-05-29T16:07:31Z
Discovery Source	Osv

Executive Summary

A high-severity vulnerability (CVE-2026-44495, CVSS 8.1) in the axios HTTP client library allows attackers who can influence axios configuration objects to steal authentication credentials and hijack HTTP responses. Axios is one of the most widely used JavaScript/Node.js HTTP libraries, meaning the exposure surface spans virtually any web application or API service built on Node.js or modern front-end frameworks. Organizations running affected axios versions in production environments face credential theft, session compromise, and potential data exfiltration if untrusted input reaches axios configuration logic.

Technical Analysis

CVE-2026-44495 is a prototype pollution vulnerability (CWE-1321) in axios's configuration merge logic. An attacker who controls input to the configuration object passed to axios requests can inject properties into Object.prototype, affecting all objects in the JavaScript runtime. This enables theft of Authorization headers and cookies (credential theft) and manipulation of response handling (response hijacking). MITRE ATT&CK techniques map to T1565.002 (Stored Data Manipulation), T1557 (Adversary-in-the-Middle), and T1040 (Network Sniffing). The vulnerability is tracked under GitHub Security Advisory GHSA-3g43-6gmg-66jw. NVD entry for CVE-2026-44495 is in-progress (CVSS vector not yet published). As of analysis date, the affected version range has not been officially confirmed. Organizations should inventory all axios versions in their environment immediately; a patched release is expected - monitor GHSA-3g43-6gmg-66jw and NVD for the

specific version range and patch release information. Sources: OSV.dev (GHSA-3g43-6gmg-66jw), NVD (CVE-2026-44495), GitLab Advisories.

Action Checklist

1. Step 1: Containment. Audit all Node.js and browser-based applications in your environment for axios dependencies using 'npm ls axios' or 'yarn list axios'. Flag any service that accepts external or user-controlled JSON input and merges it into axios configuration objects (e.g., via request headers, interceptor config, or request body passed to axios config). Standard user input to application logic is lower risk; focus on data that directly influences axios library configuration. Isolate high-risk services from internet exposure until patched, referencing NIST SI-3 (Malicious Code Protection) for interim controls.
2. Step 2: Detection. Search application logs for anomalous Authorization header values or unexpected response content modifications. Query SIEM for unusual outbound HTTP requests from Node.js processes. Review any prototype pollution detection rules in WAF or runtime application self-protection (RASP) tooling. Reference NIST AU-6 (Audit Record Review, Analysis, and Reporting) and CIS 8.2 (Collect Audit Logs) for log coverage verification.
3. Step 3: Eradication. Upgrade axios to the patched version once officially released and confirmed via the GitHub Advisory GHSA-3g43-6gmg-66jw or NVD entry for CVE-2026-44495. Until a patched version is confirmed, validate and sanitize all external input before it reaches axios configuration objects. As a temporary runtime mitigation only if patched version is not immediately available, consider Object.freeze(Object.prototype), but test thoroughly in non-production environments first - this change carries significant risk of breaking application functionality and should only be applied if input validation alone is insufficient. Reference NIST SI-2 (Flaw Remediation) and CIS 7.3/7.4 (Automated Patch Management).
4. Step 4: Recovery. After upgrading axios, re-run 'npm audit' or equivalent to confirm no residual vulnerable version remains in the dependency tree, including transitive dependencies. Rotate any credentials (Authorization tokens, API keys, session cookies) that may have been exposed in affected services, per NIST IA-5 (Authenticator Management) and D3-CRO (Credential Rotation). Monitor application behavior for 30 days post-fix for anomalous HTTP response patterns.
5. Step 5: Post-Incident. Evaluate whether user-controlled input is validated before reaching any third-party library configuration objects across your codebase; this vulnerability class (prototype pollution via config merge) is a recurring pattern in JavaScript ecosystems. Implement NIST AC-6 (Least Privilege) for service accounts whose credentials transit axios requests. Establish or review software composition analysis (SCA) tooling to detect future vulnerable dependency introductions, referencing CIS 2.1 (Software Inventory) and CIS 7.1 (Vulnerability Management Process).

IR / Forensic Enrichment

Triage Priority	URGENT
Escalation Criteria	Escalate to CISO and legal/privacy counsel immediately if application logs from Step 2 reveal Authorization header values or API keys appearing in outbound requests to unintended destinations, or if any service handling PII/PHI credentials transited axios — this meets breach notification threshold analysis under GDPR Article 33 and US state data breach notification statutes.

Recovery Notes	After axios upgrade and credential rotation, verify recovery by running 'npm audit' to confirm zero HIGH/CRITICAL findings for axios across all transitive dependency paths, and perform a heap snapshot comparison to confirm Object.prototype is clean on the patched codebase. Monitor all outbound HTTP Authorization headers from affected Node.js services for 30 days using application-layer request logging, treating any appearance of a previously rotated credential value as an active incident indicator requiring immediate re-escalation. Pay particular attention to transitive dependents — SDKs and internal libraries that bundle axios may require separate patching cycles independent of top-level application upgrades.
Forensic Artifacts	Node.js application stdout/stderr and HTTP access logs (e.g., morgan, winston, or Nginx proxy logs) covering the vulnerability exposure window, filtered for '__proto__', 'constructor.prototype', or unexpected Authorization header values — these capture the prototype pollution injection attempt and any resulting credential leakage in axios-constructed requests. | V8 heap snapshots from affected Node.js processes captured before patching, examined for non-standard properties on Object.prototype such as 'headers', 'baseURL', 'transformRequest', or 'transformResponse' — injected axios config properties that would persist on the prototype after a successful pollution attack. | Preserved node_modules/axios/ directory including package.json (installed version and integrity hash), lib/core/mergeConfig.js, and lib/utils.js from affected services — confirms the vulnerable code path was present and provides the exact merge logic that allowed prototype pollution to propagate into axios request configuration. | package-lock.json or yarn.lock files from all affected projects at the time of discovery — these establish the precise resolved axios version and the full transitive dependency graph, identifying every package that could have introduced the vulnerable axios version as an indirect dependency. | Outbound network flow records or proxy logs for Node.js service egress traffic during the exposure window, specifically capturing destination IPs and hostnames of HTTP requests that carried Authorization headers — anomalous destinations relative to the application's normal API call baseline indicate successful credential exfiltration via response or request hijacking.

Per-Action IR Details

Step 1: Containment — Audit all Node.js and browser-based applications in your environment for axios dependencies using 'npm ls axios' or 'yarn list axios'. Flag any service that accepts external or user-controlled input and passes it to axios configuration objects. Isolate high-risk services from internet exposure until patched, referencing NIST SI-3 (Malicious Code Protection) for interim controls.

NIST Phase: Containment

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

Controls: NIST SI-3 (Malicious Code Protection), NIST AC-4 (Information Flow Enforcement), NIST CM-7 (Least Functionality), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Run 'npm ls axios --all 2>/dev/null | grep axios' recursively in each Node.js project root to enumerate all direct and transitive axios instances, including version strings. For browser-bundled apps without package.json access, grep built JS bundles: 'grep -r "axios/" ./dist --include="*.js" -l'. Block outbound HTTP from high-risk Node.js services at the host firewall using UFW: 'ufw deny out from to any port 80,443' until patched. Use osquery's 'npm_packages' table ('SELECT name, version, directory FROM npm_packages WHERE name = "axios";') for fleet-wide enumeration without SIEM.

Evidence: Before isolating any service, capture a full snapshot of all running Node.js process arguments and environment variables ('ps aux | grep node' or 'Get-Process node | Select-Object -ExpandProperty StartInfo' on Windows) to identify which processes are actively using axios and what config objects they may be receiving from external input. Preserve current application configuration files (e.g., app.config.js, .env files) and any middleware that merges request parameters into axios config objects — these establish the attack surface scope and will be needed to

reconstruct whether exploitation was feasible.

Step 2: Detection — Search application logs for anomalous Authorization header values or unexpected response content modifications. Query SIEM for unusual outbound HTTP requests from Node.js processes. Review any prototype pollution detection rules in WAF or runtime application self-protection (RASP) tooling. Reference NIST AU-6 (Audit Record Review, Analysis, and Reporting) and CIS 8.2 (Collect Audit Logs) for log coverage verification.

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), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: Without a SIEM, enable Node.js HTTP request logging by wrapping axios with a request interceptor that logs outbound URL, headers (excluding credential values), and response status to a structured log file — this creates the audit trail AU-12 requires. For prototype pollution detection, deploy a startup check in each Node.js application: 'if (({}).polluted !== undefined) { process.exit(1); }' — this detects a polluted prototype at launch. Use the free Sigma rule 'proc_creation_win_node_prototype_pollution' adapted for Linux via auditd to catch '__proto__' or 'constructor.prototype' appearing in Node.js process command-line arguments or stdin. Review Node.js application stdout/stderr logs for unexpected header injection patterns: 'grep -E "(Authorization:|__proto__|constructor\.prototype)" /var/log/app/*.log'.

Evidence: Capture web server access logs (e.g., Nginx/Apache access.log, Express morgan logs) for the period covering the vulnerability window, specifically filtering for requests containing '__proto__', 'constructor', or 'prototype' in query parameters, JSON bodies, or URL paths — these are the input vectors that trigger prototype pollution in axios config merging. Preserve WAF logs if available, filtering for the same string patterns. Export any Node.js process heap dumps ('node --inspect' + 'process.memoryUsage()' snapshots or v8 heap snapshots via 'node -e "require("v8").writeHeapSnapshot()"') from affected services, which will reveal whether Object.prototype has been modified with injected properties such as 'headers' or 'baseURL' keys that axios reads during request construction.

Step 3: Eradication — Upgrade axios to the patched version once officially released and confirmed via the GitHub Advisory GHSA-3g43-6gmg-66jw or NVD entry for CVE-2026-44495. Until a patched version is confirmed, validate and sanitize all external input before it reaches axios configuration objects. Apply Object.freeze(Object.prototype) as a runtime mitigation where feasible, with caution regarding application compatibility. Reference NIST SI-2 (Flaw Remediation) and CIS 7.3/7.4 (Automated Patch Management).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST SA-10 (Developer Configuration Management), NIST CM-3 (Configuration Change Control), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Until an official patched axios release is confirmed via GHSA-3g43-6gmg-66jw, apply the following code-level mitigation at the application entry point before any axios calls: use a deep-sanitize function that strips '__proto__', 'constructor', and 'prototype' keys from any user-supplied object before it is passed to axios config — implement with the 'sanitize-object' npm package or a manual recursive key filter. Apply 'Object.freeze(Object.prototype)' at process startup in non-library code with compatibility testing. Pin axios to your last-known-safe version in package.json using an exact version pin ("'axios": "x.x.x"') rather than a semver range, and add a 'preinstall' script that validates the installed axios checksum against a known-good SHA before allowing the application to start.

Evidence: Before upgrading, extract and preserve the exact axios package contents currently installed — copy the full 'node_modules/axios/' directory to an offline forensic store, including package.json (to confirm the installed version hash), lib/core/mergeConfig.js and lib/utils.js (the files most likely modified to fix the prototype pollution merge path), and any lock file entries (package-lock.json or yarn.lock) showing the resolved axios version and its integrity hash. This preserves the vulnerable artifact for post-incident root cause analysis and confirms whether any tampering of the installed package occurred via supply chain compromise.

Step 4: Recovery — After upgrading axios, re-run 'npm audit' or equivalent to confirm no residual vulnerable version remains in the dependency tree, including transitive dependencies. Rotate any credentials (Authorization tokens, API keys, session cookies) that may have been exposed in affected services, per NIST IA-5 (Authenticator Management) and D3-CRO (Credential Rotation). Monitor application behavior for 30 days post-fix for anomalous HTTP response patterns.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IA-5 (Authenticator Management), NIST SI-2 (Flaw Remediation), NIST CP-10 (System Recovery and Reconstitution), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 5.2 (Use Unique Passwords), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Run 'npm audit --audit-level=high' and 'npx better-npm-audit audit' to enumerate any transitive axios dependency paths that still resolve to the vulnerable version range — pay specific attention to packages that bundle axios internally (e.g., SDKs or API client libraries) which will not be caught by a top-level 'npm ls axios'. For credential rotation without enterprise PAM tooling, build a rotation checklist from all Authorization headers found in the application's axios request interceptor logs captured in Step 2, then revoke and reissue each API key or token through the issuing platform's management console. Post-rotation, deploy a lightweight Node.js middleware that logs a SHA-256 hash of each outbound Authorization header value daily — a change in hash without a deliberate rotation event is an indicator of renewed credential theft.

Evidence: Before declaring recovery complete, run a final heap snapshot comparison between a freshly started patched instance and the preserved pre-patch snapshots from Step 3 to confirm Object.prototype no longer contains injected axios config properties ('headers', 'baseURL', 'transformResponse'). Preserve 'npm audit' output and 'package-lock.json' from the post-upgrade state as recovery verification artifacts. Collect and retain outbound HTTP request logs for the 30-day monitoring window, specifically filtering for any Authorization header values that match previously rotated credentials — appearance of a rotated credential in outbound traffic would indicate either incomplete rotation or ongoing exploitation of a secondary vector.

Step 5: Post-Incident — Evaluate whether user-controlled input is validated before reaching any third-party library configuration objects across your codebase — this vulnerability class (prototype pollution via config merge) is a recurring pattern in JavaScript ecosystems. Implement NIST AC-6 (Least Privilege) for service accounts whose credentials transit axios requests. Establish or review software composition analysis (SCA) tooling to detect future vulnerable dependency introductions, referencing CIS 2.1 (Software Inventory) and CIS 7.1 (Vulnerability Management Process).

NIST Phase: Post Incident

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

Controls: NIST AC-6 (Least Privilege), NIST IA-5 (Authenticator Management), NIST SA-15 (Development Process, Standards, and Tools), NIST RA-3 (Risk Assessment), 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: Integrate the free 'better-npm-audit' or OWASP Dependency-Check into your CI/CD pipeline (GitHub Actions, GitLab CI, or a pre-commit hook) to block merges that introduce known-vulnerable npm packages, creating the automated SCA coverage CIS 2.1 requires without enterprise tooling cost. For prototype pollution specifically, add ESLint with the 'eslint-plugin-prototype-pollution' rules to flag unsafe object merge patterns (e.g., direct assignment to user-controlled keys, use of 'Object.assign' or lodash merge with external input) across the entire JavaScript codebase. Scope service account API keys used in axios requests to the minimum required permissions at the issuing platform level — document the permission scope in the service's runbook so the next credential rotation maintains least privilege. Write a YARA rule targeting '__proto__' and 'constructor.prototype' string patterns in JavaScript files and run it as a pre-deployment scan to catch future prototype pollution surface introductions.

Evidence: Produce a lessons-learned artifact documenting: (1) the full list of axios-dependent services identified in Step 1 and their exposure status at the time of discovery, (2) any log evidence from Step 2 indicating whether

exploitation was attempted or successful prior to detection, (3) the time delta between GHSA-3g43-6gmg-66jw publication and your team's detection — this gap measurement directly informs the SCA and monitoring improvements required. Archive the pre-patch node_modules/axios forensic copy from Step 3 alongside this report for future reference if exploitation is later confirmed through external threat intelligence about CVE-2026-44495 active exploitation campaigns.

Detection Guidance

Query application and runtime logs for signs of prototype pollution: unexpected '__proto__', 'constructor', or 'prototype' keys appearing in JSON payloads sent to axios-consuming services. In Node.js APM tooling or RASP agents, alert on Object.prototype property modifications at runtime. In your SIEM, correlate axios-related process activity with anomalous outbound HTTP requests containing modified Authorization headers or unexpected response body content. Review access logs for services that accept user-supplied JSON and pass it to axios config, look for inputs containing prototype chain keys. Reference NIST AU-2 (Event Logging) and AU-6 (Audit Record Review) to ensure logging is enabled on all affected application tiers. D3-SFA (System File Analysis) techniques apply for monitoring configuration file integrity where axios config is stored. D3-LAM (Local Account Monitoring) is relevant where service account credentials may transit affected axios instances.

Framework Mappings

MITRE-ATTACK

- **T1565.002** — Transmitted Data Manipulation
- **T1557** — Adversary-in-the-Middle
- **T1040** — Network Sniffing

CIS-V8

- **6.3** — Require MFA for Externally-Exposed Applications

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

MITRE ATT&CK Mapping

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

Technique ID	Technique Name	Tactic
T1040	Network Sniffing	Credential-Access

Sources

Source	URL	Tier
osv	https://osv.dev/vulnerability/GHSA-3g43-6gmg-66jw	T3
CVE-2026-44495 | Tenable®	https://www.tenable.com/cve/CVE-2026-44495	T3
CVE-2026-44495: Axios PP Auth Theft via Config - Miggo Security	https://www.miggo.io/vulnerability-database/cve/CVE-2026-44495	T3
axios Vulnerable to Credential Theft and Response Hijacking via ...	https://advisories.gitlab.com/npm/axios/CVE-2026-44495/	T3
CVE-2026-44495 - CVE Record	https://www.cve.org/CVERecord?id=CVE-2026-44495	T3
NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-44495	T1

DISCLAIMER

This intelligence report is produced by Tech Jacks Solutions Security Command Center (SCC) for informational purposes only. It does not constitute professional security advice, legal counsel, or an incident response engagement. The information herein is derived from publicly available sources and AI-assisted analysis; while every effort is made to ensure accuracy, Tech Jacks Solutions makes no warranties regarding completeness or timeliness. Organizations should conduct their own validation and consult qualified security professionals before taking action based on this report. Tech Jacks Solutions is not liable for any damages resulting from the use of this information.

Generated 2026-06-01 06:14 UTC by TJS Security Command Center