INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-29 19:07 UTC

GHSA-35jp-ww65-95wh: axios Vulnerable to Full Man-in-the-Middle via Prototype Pollution Gadget in `co

CVE VULNERABILITY | HIGH | CVSS 8.1

SCC Item ID	SCC-CVE-2026-0240
Type	CVE Vulnerability
CVE ID	CVE-2026-44494
Severity	HIGH
CVSS Base Score	8.1
Affected Products	axios (npm), specific affected versions not confirmed from available sources
Published	2026-05-29T16:04:00Z
Discovery Source	Osv

Executive Summary

A prototype pollution vulnerability in the axios HTTP client library (npm) allows an attacker who can manipulate JavaScript object prototypes to redirect outbound HTTP/HTTPS requests through an attacker-controlled proxy, enabling full interception of traffic. Any Node.js application using axios for external API calls, authentication flows, or data retrieval may be exposing request payloads, session tokens, and credentials to an adversary. Affected version range: [pending OSV advisory confirmation]. Organizations using axios in production Node.js environments should treat this as high severity and apply the patched version identified in GHSA-35jp-ww65-95wh within 72 hours of verification.

Technical Analysis

CVE-2026-44494 (GHSA-35jp-ww65-95wh) describes a prototype pollution gadget within axios's `config.proxy` handling logic. CWE-1321 (Prototype Pollution) combined with CWE-441 (Unintended Proxy or Intermediary) creates a chain: an attacker who achieves prototype pollution in the JavaScript runtime, via a separate injection point, can poison the axios proxy configuration on the `Object.prototype`, causing all subsequent axios requests to route through an attacker-controlled host. This enables MITRE ATT&CK T1557 (Adversary-in-the-Middle), T1040 (Network Sniffing), and T1071.001 (Application Layer Protocol: Web Protocols) techniques. CVSS base score is 8.1 (High); CVSS vector pending NVD publication. Per GHSA-35jp-ww65-95wh, exploitation requires a precondition: the attacker must already have a vector to pollute Object.prototype in the same runtime, making this a chained or second-stage exploit in most realistic scenarios. EPSS data is not yet available (0.0), and this

vulnerability is not currently listed on the CISA KEV catalog. The authoritative source is OSV advisory GHSA-35jp-ww65-95wh.

Action Checklist

1. Step 1: Containment, Identify all Node.js services in your environment that import axios (npm). Cross-reference package-lock.json or yarn.lock files to enumerate affected deployments. Prioritize services that make outbound HTTP/HTTPS calls carrying authentication tokens, API keys, or PII. Temporarily restrict outbound proxy configurations on those services until patching is confirmed. Reference: NIST SI-3, CIS 2.1.
2. Step 2: Detection, Query application dependency manifests and CI/CD artifact stores for axios versions. In runtime environments, inspect Node.js process logs for unexpected `proxy` configuration values in axios request objects. Look for outbound CONNECT requests or HTTP traffic routed to unexpected hosts; check proxy-related log fields in your WAF, NGFW, or outbound TLS inspection logs. MITRE T1557 detection: flag axios requests where the resolved proxy host differs from your approved egress proxy list. Reference: NIST AU-6, CIS 8.2.
3. Step 3: Eradication, Upgrade axios to the patched version identified in GHSA-35jp-ww65-95wh (confirm the fixed version directly from https://osv.dev/vulnerability/GHSA-35jp-ww65-95wh before deploying). Run `npm audit` or `yarn audit` to surface transitive dependencies pulling in the vulnerable axios version. For applications where an immediate upgrade is not feasible, explicitly set and validate `config.proxy` in axios instances rather than relying on inherited prototype values; treat any proxy configuration sourced from user-controlled input as untrusted. Reference: NIST SI-2, CIS 7.3, CIS 7.4.
4. Step 4: Recovery, After patching, re-run `npm audit` and confirm zero findings for GHSA-35jp-ww65-95wh. Enable outbound TLS inspection or logging on the remediated services and monitor for anomalous proxy routing for 7 days post-patch. Rotate any API keys, OAuth tokens, or session credentials that transited axios-based services during the exposure window. Reference: NIST IR-4, D3-CRO (Credential Rotation).
5. Step 5: Post-Incident, Assess whether your software supply chain scanning (SCA tooling) would have flagged this advisory before deployment. If not, integrate OSV and GitHub Advisory Database feeds into your CI/CD pipeline for automated npm dependency scanning. Evaluate whether prototype pollution is a systemic risk across your Node.js codebase by running a prototype pollution audit tool (e.g., `pp-audit`). Document control gaps against NIST SA-15 (Development Process Standards) and CIS 2.2 (Ensure Authorized Software is Currently Supported).

IR / Forensic Enrichment

Triage Priority	URGENT
Escalation Criteria	Escalate to senior IR leadership and legal/compliance immediately if log analysis confirms outbound CONNECT requests or axios traffic routed to non-approved proxy hosts during the exposure window, as this indicates potential credential or PII exfiltration requiring breach notification assessment under applicable data protection regulations (GDPR, CCPA, HIPAA depending on data classification of axios-transited payloads).

Recovery Notes	Post-patch verification must confirm the resolved axios version in node_modules matches the GHSA-35jp-ww65-95wh fixed release across all deployment environments including staging and container images, not just production — prototype pollution gadgets can persist in cached Docker layers or pre-built artifacts. Monitor outbound proxy routing behavior for a minimum of 7 days post-patch using NGFW or tcpdump-based egress logging, specifically watching for HTTP CONNECT method requests to non-approved destinations that would indicate a missed instance of the vulnerable library. Any API keys, OAuth bearer tokens, or session credentials transmitted through axios-based services during the confirmed or estimated exposure window must be treated as compromised and rotated before services are returned to full production status.
Forensic Artifacts	Node.js application stdout/stderr logs (journald or /var/log//*.log) for the exposure window: filter on 'proxy', 'httpsProxy', '__proto__', or 'constructor.prototype' strings which would appear if prototype pollution was actively triggered and logged by the application or a debugging middleware | NGFW/WAF outbound connection logs filtered for HTTP CONNECT method requests and TCP sessions where the SNI hostname in the TLS ClientHello does not match the IP resolved from DNS — the axios prototype pollution gadget routes TLS traffic through an injected proxy, producing this host/IP mismatch artifact | package-lock.json and yarn.lock snapshots from all Node.js services at the time of incident discovery — these establish the exact resolved axios version (including transitive resolution via parent packages) and the full dependency chain that introduced the vulnerable version | Node.js process environment dumps (/proc//environ) and `ss -tnp` connection state captures from affected services at time of containment — these may reveal HTTP_PROXY or HTTPS_PROXY environment variable injection or active connections to attacker-controlled proxy IPs if exploitation was in progress | CI/CD pipeline build artifacts and npm registry fetch logs showing when the vulnerable axios version was first pulled and deployed, establishing the exposure window start date for credential rotation scope and breach notification timeline calculations

Per-Action IR Details

Step 1: Containment — Identify all Node.js services in your environment that import axios (npm). Cross-reference package-lock.json or yarn.lock files to enumerate affected deployments. Prioritize services that make outbound HTTP/HTTPS calls carrying authentication tokens, API keys, or PII. Temporarily restrict outbound proxy configurations on those services until patching is confirmed. Reference: NIST SI-3, CIS 2.1.

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 SC-7 (Boundary Protection), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Run `grep -r 'require.*axios\|from.*axios' /var/www /opt /srv --include='*.js' --include='*.ts' -l` across all Node.js application roots to enumerate axios consumers. Then run `find / -name 'package-lock.json' -o -name 'yarn.lock' 2>/dev/null | xargs grep -l 'axios'` to identify transitive inclusions. To block outbound proxy abuse immediately without a SIEM, apply an iptables OUTPUT rule on each affected host: `iptables -A OUTPUT -p tcp --dport 8080 -j DROP` adjusted to your environment's non-approved proxy ports, logging dropped packets to /var/log/iptables.log via `iptables -A OUTPUT -j LOG --log-prefix 'DROPPED-OUTBOUND:'`.

Evidence: Before restricting outbound proxy routes, capture current axios runtime configuration by dumping Node.js process environment: `cat /proc//environ | tr '\0' '\n' | grep -i proxy` for each identified Node.js PID. Preserve existing outbound connection state with `ss -tnp | grep node` and `netstat -anp | grep ESTABLISHED` snapshots. Capture current package-lock.json and yarn.lock files from all identified services as timestamped forensic copies before any patching activity — these establish the pre-remediation dependency chain for GHSA-35jp-ww65-95wh.

Step 2: Detection — Query application dependency manifests and CI/CD artifact stores for axios versions. In runtime environments, inspect Node.js process logs for unexpected `proxy` configuration values in axios request objects. Look for outbound CONNECT requests or HTTP traffic routed to unexpected hosts — check proxy-related log fields in your WAF, NGFW, or outbound TLS inspection logs. MITRE T1557 detection: flag axios requests where the resolved proxy host differs from your approved egress proxy list. Reference: 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-2 (Event Logging), NIST AU-3 (Content of Audit Records), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: Without a SIEM, use osquery to inspect live Node.js processes for suspicious prototype-inherited proxy config: `SELECT pid, cmdline, env FROM processes WHERE name = 'node';` then correlate PIDs against `lsof -p -i` to surface unexpected outbound TCP connections. For network-level detection of CONNECT tunneling (MITRE T1557 pivot), run Wireshark or tcpdump on egress interfaces with `tcpdump -i eth0 -w /tmp/axios-capture.pcap 'tcp port 443 or tcp port 80' -n` and post-filter for CONNECT method requests to non-approved proxy IPs using `strings /tmp/axios-capture.pcap | grep -i 'CONNECT '`. Write a Sigma rule targeting Node.js application stdout/stderr logs filtering for lines containing `proxy` adjacent to unexpected IP ranges.

Evidence: Extract WAF and NGFW logs for the exposure window filtering on HTTP CONNECT method requests and outbound connections to hosts not on your approved egress list — the prototype pollution gadget in axios redirects requests by injecting a proxy value into `Object.prototype`, so look for axios-generated requests where the Host header destination differs from the TCP destination IP. Capture Node.js application stdout logs (typically journald: `journalctl -u --since --until now > /tmp/-axios-logs.txt`) filtering on 'proxy' or 'httpsProxy' strings. Preserve CI/CD build logs showing which axios version was installed at last deployment to establish exposure window start time.

Step 3: Eradication — Upgrade axios to the patched version identified in GHSA-35jp-ww65-95wh (confirm the fixed version directly from https://osv.dev/vulnerability/GHSA-35jp-ww65-95wh before deploying). Run `npm audit` or `yarn audit` to surface transitive dependencies pulling in the vulnerable axios version. For applications where an immediate upgrade is not feasible, explicitly set and validate `config.proxy` in axios instances rather than relying on inherited prototype values — treat any proxy configuration sourced from user-controlled input as untrusted. Reference: NIST SI-2, CIS 7.3, CIS 7.4.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST CM-8 (System Component Inventory), NIST SA-22 (Unsupported System Components), 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: For teams unable to immediately upgrade, implement a Node.js startup script that freezes Object.prototype before axios loads: add `Object.freeze(Object.prototype);` as the first line of each affected service's entry point, which will throw on prototype pollution attempts and surface exploitation in application error logs. Run `npm audit --json | jq '.vulnerabilities.axios'` to extract the exact affected version range confirmed by npm's advisory database. For transitive dependency discovery without enterprise SCA tooling, run `npm ls axios --all 2>/dev/null` in each project root to display the full dependency tree showing all packages pulling in axios.

Evidence: Before executing the npm upgrade, preserve a complete snapshot of node_modules: `tar czf /tmp/node_modules-pre-patch-$(date +%Y%m%d%H%M%S).tar.gz ./node_modules/axios/` for each affected service — this captures the vulnerable axios source (lib/helpers/buildURL.js, lib/core/buildFullPath.js, and lib/defaults/index.js are the files most relevant to prototype chain resolution) as forensic evidence of the vulnerable code state. Run `npm audit --json > /tmp/npm-audit-pre-patch.json` to preserve the pre-remediation advisory state. Document the exact prototype pollution gadget path if confirmed: the mechanism involves `__proto__` or constructor.prototype manipulation propagating a `proxy` property to all axios config objects.

Step 4: Recovery — After patching, re-run `npm audit` and confirm zero findings for GHSA-35jp-ww65-95wh. Enable outbound TLS inspection or logging on the remediated services and monitor for anomalous proxy routing for 7 days post-patch. Rotate any API keys, OAuth tokens, or session credentials that transited axios-based services during the exposure window. Reference: NIST IR-4, D3-CRO (Credential Rotation).

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST IA-5 (Authenticator Management), NIST AC-2 (Account Management), NIST AU-11 (Audit Record Retention), CIS 5.2 (Use Unique Passwords), CIS 6.1 (Establish an Access Granting Process)

Compensating: Verify patch success without enterprise tooling by running `node -e "const axios = require('axios'); console.log(axios.VERSION);"` on each remediated host and confirming the version matches the GHSA-35jp-ww65-95wh fixed release. For credential rotation tracking, generate a list of all external API endpoints called by axios in each affected service by grepping source for axios call patterns: `grep -rE 'axios\.(get|post|put|patch|delete|request)' /opt//src --include='*.js' -h | grep -oP 'https?://[^'"]+' | sort -u` — this inventory drives the credential rotation scope. Monitor for 7 days post-patch using a cron-driven tcpdump job: `crontab -e` → `*/15 * * * * tcpdump -i eth0 -c 500 -w /tmp/egress-check-$(date +\%s).pcap 'tcp[tcpflags] & tcp-syn != 0 and dst port 443' 2>/dev/null` with daily review for unexpected CONNECT destinations.

Evidence: Post-patch, run `npm audit --json > /tmp/npm-audit-post-patch.json` and diff against the pre-patch capture to confirm GHSA-35jp-ww65-95wh no longer appears. Document the credential rotation log: for each rotated API key or OAuth token, record the service name, the external API endpoint reached by axios, the rotation timestamp, and the issuing authority — this establishes the remediation timeline for any downstream breach notification assessment. Preserve outbound proxy connection logs from the 7-day monitoring window as evidence of clean post-patch behavior.

Step 5: Post-Incident — Assess whether your software supply chain scanning (SCA tooling) would have flagged this advisory before deployment. If not, integrate OSV and GitHub Advisory Database feeds into your CI/CD pipeline for automated npm dependency scanning. Evaluate whether prototype pollution is a systemic risk across your Node.js codebase by running a prototype pollution audit tool (e.g., `pp-audit`). Document control gaps against NIST SA-15 (Development Process Standards) and CIS 2.2 (Ensure Authorized Software is Currently Supported).

NIST Phase: Post Incident

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

Controls: NIST SA-15 (Development Process, Standards, and Tools), NIST RA-5 (Vulnerability Monitoring and Scanning), NIST CA-7 (Continuous Monitoring), NIST SI-2 (Flaw Remediation), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Integrate OSV Scanner (free, Google-maintained) into your CI/CD pipeline with a GitHub Actions step: `uses: google/osv-scanner-action@v1` targeting package-lock.json — this will surface GHSA advisories including GHSA-35jp-ww65-95wh-class issues before deployment. Run `npx pp-audit` in each Node.js project root to enumerate all prototype pollution gadget chains in your full dependency tree, not just axios — this reveals whether other npm packages in your stack are similarly exploitable via the same attack class (MITRE T1557 via supply chain). Document lessons learned against NIST SA-15 by recording: time from GHSA publication to detection in your environment, number of affected services, and whether the CI/CD pipeline would have blocked a vulnerable axios version from shipping.

Evidence: Retrieve CI/CD pipeline build history logs for all Node.js services to determine when the vulnerable axios version was first introduced — this establishes the maximum possible exposure window for any credential compromise assessment. Run `pp-audit` output to a file (`npx pp-audit --json > /tmp/pp-audit-results.json`) and preserve it as the post-incident baseline for prototype pollution risk across your Node.js inventory. Document the delta between GHSA-35jp-ww65-95wh advisory publication date and your organization's detection date as a key metric for the lessons-learned report — this gap directly measures SCA control effectiveness against supply chain vulnerabilities of this class.

Detection Guidance

Primary detection approach: scan all production and staging Node.js environments for axios as a direct or transitive dependency using `npm audit --json` filtered for GHSA-35jp-ww65-95wh. In runtime telemetry, look for HTTP CONNECT or outbound requests where the destination proxy host is not in your approved egress allowlist, this is the behavioral indicator of a successful gadget activation. If you run outbound TLS inspection or a forward proxy, review logs for axios User-Agent strings paired with unexpected proxy hop headers. For prototype pollution detection at runtime, tools such as `--inspect` with Node.js heap snapshots can reveal unexpected properties on `Object.prototype`. SIEM query suggestion (adapt to your log schema): filter on outbound HTTP requests where `proxy_host` is non-null and not in [approved_proxy_list] AND `user_agent` contains 'axios'. No confirmed public IOCs (IPs, domains, hashes) are associated with active exploitation of this advisory at this time. MITRE ATT&CK coverage: T1557 (Adversary-in-the-Middle, via proxy gadget), T1040 (monitor for credential capture patterns on redirected sessions). Reference: NIST AU-2, AU-6, SI-4, D3-LAM.

Framework Mappings

MITRE-ATTACK

- **T1040** — Network Sniffing
- **T1557** — Adversary-in-the-Middle
- **T1071.001** — Web Protocols

ISO-27001-2022

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1040	Network Sniffing	Credential-Access
T1557	Adversary-in-the-Middle	Credential-Access
T1071.001	Web Protocols	Command-And-Control

Sources

Source	URL	Tier
osv	https://osv.dev/vulnerability/GHSA-35jp-ww65-95wh	T3
CVE-2026-6494 Detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-6494	T1

Source	URL	Tier
CVE-2026-27494: n8n Workflow Automation RCE Vulnerability	https://www.sentinelone.com/vulnerability-database/cve-2026-27494/	T3
CVE-2026-26944 - CVE Record	https://www.cve.org/CVERecord?id=CVE-2026-26944	T3
CVE-2026-34944 Common Vulnerabilities and Exposures - SUSE	https://www.suse.com/security/cve/CVE-2026-34944.html	T3
NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-44494	T1

Generated 2026-05-29 19:07 UTC by TJS Security Command Center