INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-10 18:37 UTC

Next.js Server Components Denial of Service Vulnerability (GHSA-q4gf-8mx6-v5v3)

CVE VULNERABILITY | MEDIUM

SCC Item ID	SCC-CVE-2026-0029
Type	CVE Vulnerability
Severity	MEDIUM
Affected Products	next (npm), specific versions not determinable from available data; see OSV advisory GHSA-q4gf-8mx6-v5v3 for patched/affected version ranges
Published	2026-04-10T15:35:47Z
Discovery Source	Osv

Executive Summary

A Denial of Service vulnerability (GHSA-q4gf-8mx6-v5v3) has been identified in Next.js Server Components, a widely used React framework for web applications. An attacker can send malformed input to trigger resource exhaustion, potentially making affected applications unavailable to users. Organizations running Next.js with Server Components enabled in customer-facing or internal applications should review the OSV advisory and apply available patches promptly.

Technical Analysis

GHSA-q4gf-8mx6-v5v3 describes a Denial of Service condition in Next.js (npm package: next) affecting its Server Components feature. The vulnerability maps to CWE-400 (Uncontrolled Resource Consumption) and MITRE ATT&CK T1499 (Endpoint Denial of Service). An attacker supplying adversarial or malformed input to Server Components can exhaust application resources, causing unavailability. No CVE ID is assigned in source data. CVSS and EPSS scores are not available in the provided data set; qualitative rating assigned from OSV advisory pending NVD publication. Affected version ranges and patch availability must be confirmed against the authoritative OSV advisory at https://osv.dev/vulnerability/GHSA-q4gf-8mx6-v5v3. No known exploitation in the wild or CISA KEV listing is present in this data.

Action Checklist

1. Step 1: Containment, Identify all production services running the next npm package with Server Components enabled. Cross-reference running versions against affected ranges in GHSA-q4gf-8mx6-v5v3 at https://osv.dev/vulnerability/GHSA-q4gf-8mx6-v5v3. If running an affected

version and internet-exposed, consider placing a WAF or rate-limiting proxy in front of the application to restrict adversarial input volume while a patch is staged. Note: this is a temporary mitigation only; priority is to patch the next package to resolve the root cause.

2. Step 2: Detection, Query your software inventory, SBOM, or dependency manifests for the 'next' npm package across all projects. Review application-level logs and web server access logs for anomalous request patterns targeting Server Components endpoints (high-volume or malformed POST/GET requests). Monitor for elevated CPU or memory utilization on Next.js application servers, which may indicate active exploitation attempts.

3. Step 3: Eradication, Upgrade the next package to a patched version as specified in GHSA-q4gf-8mx6-v5v3. Run 'npm audit' or 'yarn audit' against all affected projects to confirm resolution. If a patch is not yet available, evaluate disabling Server Components or restricting access to affected endpoints as a temporary measure.

4. Step 4: Recovery, After upgrading, confirm the patched version is active in all environments (development, staging, production). Re-run 'npm audit' to verify no remaining advisories on the next package. Monitor application performance metrics and error rates for 24-48 hours post-patch to confirm stability.

5. Step 5: Post-Incident, Review SBOM and dependency tracking processes to ensure next and other critical framework packages are inventoried and monitored for advisories. Evaluate whether automated dependency update tooling (Dependabot, Renovate) is configured and active. Assess whether rate limiting and input validation controls on Server Components endpoints are sufficient to reduce future DoS exposure.

IR / Forensic Enrichment

Triage Priority	STANDARD
Escalation Criteria	Escalate to urgent and engage senior IR leadership if active exploitation is confirmed via web server logs showing sustained high-volume malformed requests to Next.js Server Components endpoints resulting in application unavailability, or if the affected application processes PII/PHI/PCI data and downtime triggers breach notification obligations under applicable regulations.
Recovery Notes	After upgrading the next package to the GHSA-q4gf-8mx6-v5v3 patched version, verify the corrected version is running in all environments by inspecting the runtime package version and confirming 'npm audit' returns clean results for this advisory. Monitor Node.js process CPU and memory utilization, HTTP 5xx error rates, and Server Components endpoint response times for a minimum of 48 hours post-patch, comparing against the pre-incident baseline captured during containment. If WAF or rate-limiting mitigations were applied as temporary containment, evaluate whether they should remain as a permanent defense-in-depth layer for Server Components endpoints regardless of patch status.

Forensic Artifacts	Web server access logs (nginx: /var/log/nginx/access.log; Apache: /var/log/apache2/access.log) — exploitation of GHSA-q4gf-8mx6-v5v3 would produce a concentrated spike of POST or GET requests to Next.js Server Components RSC fetch endpoints (URLs containing _rsc= query parameters or /_next/static/chunks/ paths) with abnormally large or malformed payloads, correlated with HTTP 500/503 response codes and elevated $request_time values. | Node.js application stderr logs (PM2: ~/.pm2/logs/-error.log; systemd: journalctl -u nextjs.service) — malformed input triggering the DoS condition in Next.js Server Components would likely produce unhandled promise rejections, React rendering errors, or memory allocation failures logged as stack traces referencing Next.js internal Server Components rendering functions. | System resource utilization snapshots (output of 'top -b -n5 -d2' or 'vmstat 2 10' on Linux; Get-Counter on Windows) — resource exhaustion from the DoS vulnerability would manifest as sustained high CPU or memory consumption in the node process hosting the Next.js application, distinguishable from normal load spikes by duration and correlation with the anomalous request pattern. | package.json, package-lock.json, and yarn.lock files from all affected repositories — these establish the exact vulnerable next version deployed at the time of the incident and are required for the incident record and any regulatory or vendor reporting related to GHSA-q4gf-8mx6-v5v3. | npm audit output (pre- and post-patch JSON exports: 'npm audit --json') — provides a timestamped record of the advisory presence and resolution, linking the specific GHSA-q4gf-8mx6-v5v3 finding to the installed next version and confirming eradication when the advisory no longer appears post-upgrade.

Per-Action IR Details

Step 1: Containment — Identify all production services running the next npm package with Server Components enabled. Cross-reference running versions against affected ranges in GHSA-q4gf-8mx6-v5v3 at https://osv.dev/vulnerability/GHSA-q4gf-8mx6-v5v3. If running an affected version and internet-exposed, consider placing a WAF or rate-limiting proxy in front of the application to restrict adversarial input volume while a patch is staged.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SI-10 (Information Input Validation), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Without a commercial WAF, use nginx rate-limiting directives (limit_req_zone / limit_req) to cap request rates to Next.js Server Components endpoints (typically paths handled by the Next.js server under /_next/ or custom RSC fetch routes). Example nginx snippet: 'limit_req_zone $binary_remote_addr zone=rsc_limit:10m rate=20r/s; limit_req zone=rsc_limit burst=40 nodelay;'. Alternatively, deploy Cloudflare's free tier in front of the origin and enable rate-limiting rules targeting POST requests to Server Components fetch endpoints. For a 2-person team, a 30-minute nginx config change is achievable and reversible.

Evidence: Before placing the WAF or proxy, snapshot the current nginx/Apache/Caddy access logs (e.g., /var/log/nginx/access.log) to preserve the pre-containment baseline of request rates and source IPs. Capture a process list (ps aux or Get-Process) and memory/CPU utilization (top -b -n1 > snapshot.txt or Get-Counter) from each Next.js application server to document whether resource exhaustion is already in progress. These baselines are required to distinguish pre-exploitation noise from active DoS activity post-containment.

Step 2: Detection — Query your software inventory, SBOM, or dependency manifests for the 'next' npm package across all projects. Review application-level logs and web server access logs for anomalous request patterns targeting Server Components endpoints (high-volume or malformed POST/GET requests). Monitor for elevated CPU or memory utilization on Next.js application servers, which may indicate active exploitation attempts.

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), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 8.2 (Collect Audit Logs)

Compensating: Run the following one-liner across all repos to enumerate exposed next versions: 'find /opt /var/www /srv -name package.json -not -path "*/node_modules/*" | xargs grep -l "\"next\"" | xargs grep -h "\"next\"". For log analysis without a SIEM, use awk or grep against nginx access logs to surface anomalous Server Components traffic: 'awk \'$9 == 500 || $9 == 503 {print}\' /var/log/nginx/access.log | awk \'{print $1}\' | sort | uniq -c | sort -rn | head -20'. For CPU/memory monitoring, use a bash loop: 'while true; do ps aux --sort=-%mem | head -5 >> /tmp/nextjs_mem_$(date +%F).log; sleep 60; done'. OWASP Dependency-Check (free, CLI) can also scan node_modules and flag the vulnerable next version against the NVD/OSV feed.

Evidence: Preserve raw web server access logs (nginx: /var/log/nginx/access.log; Apache: /var/log/apache2/access.log) covering at least 72 hours prior to detection, with timestamps intact — Next.js Server Components DoS exploitation would appear as a spike in POST requests (or GET requests with large/malformed RSC payloads) resulting in HTTP 500/503 responses or abnormally long response times logged in the $request_time field. Capture Node.js process logs (typically stdout/stderr redirected via PM2: ~/.pm2/logs/-error.log or journalctl -u nextjs.service) for stack traces or unhandled promise rejection errors triggered by the malformed input. Document the next package version from each package.json and the corresponding package-lock.json or yarn.lock to establish scope.

Step 3: Eradication — Upgrade the next package to a patched version as specified in GHSA-q4gf-8mx6-v5v3. Run 'npm audit' or 'yarn audit' against all affected projects to confirm resolution. If a patch is not yet available, evaluate disabling Server Components or restricting access to affected endpoints as a temporary measure.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), 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: For teams without automated deployment pipelines, perform the upgrade manually per environment: 'npm install next@ --save-exact && npm audit'. Pin the exact patched version in package.json using --save-exact to prevent unintended version drift. Validate eradication with 'npm audit --json | python3 -m json.tool | grep -A5 GHSA-q4gf-8mx6-v5v3' — a clean result confirms the advisory is resolved. If Server Components must be disabled temporarily, set the Next.js configuration option 'experimental.serverComponents: false' (or equivalent for the installed version) in next.config.js and redeploy; confirm the RSC endpoint no longer responds by curling it with a malformed payload in a non-production environment.

Evidence: Before upgrading, preserve a copy of the vulnerable package-lock.json or yarn.lock and the output of 'npm list next --depth=0' to document the pre-patch state for the incident record. Capture the output of 'npm audit --json > npm_audit_pre_patch_$(date +%F).json' as a timestamped artifact. If Server Components are being disabled rather than patched, capture the current next.config.js before modification and retain both versions. These artifacts establish the eradication action timeline required by NIST IR-5 (Incident Monitoring) for post-incident reporting.

Step 4: Recovery — After upgrading, confirm the patched version is active in all environments (development, staging, production). Re-run 'npm audit' to verify no remaining advisories on the next package. Monitor application performance metrics and error rates for 24-48 hours post-patch to confirm stability.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-7 (Software, Firmware, and Information Integrity), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Verify the patched next version is running in production by querying the deployed package at runtime: in a Node.js REPL or startup script, log 'require("next/package.json").version' and confirm it matches the patched release. For 24-48 hour stability monitoring without a commercial APM tool, use PM2's built-in metrics ('pm2 monit')

combined with a simple bash health check that polls the application's health endpoint and logs HTTP response codes and response times to a file: 'while true; do curl -o /dev/null -s -w "%{http_code} %{time_total}\n" https:///api/health >> /tmp/health_$(date +%F).log; sleep 30; done'. Alert thresholds: flag any HTTP 5xx responses or response times exceeding 3x the pre-patch baseline as potential indicators of incomplete eradication or new instability.

Evidence: Capture 'npm audit --json > npm_audit_post_patch_$(date +%F).json' immediately after upgrading in each environment to create a timestamped clean bill of health. Preserve Node.js process restart logs from PM2 or systemd (journalctl -u nextjs.service --since 'post-patch timestamp') to confirm the new version was loaded cleanly. Retain web server access logs for the 24-48 hour monitoring window to provide a post-patch baseline of Server Components request patterns, which can be compared against the pre-exploitation baseline captured in Step 1.

Step 5: Post-Incident — Review SBOM and dependency tracking processes to ensure next and other critical framework packages are inventoried and monitored for advisories. Evaluate whether automated dependency update tooling (Dependabot, Renovate) is configured and active. Assess whether rate limiting and input validation controls on Server Components endpoints are sufficient to reduce future DoS exposure.

NIST Phase: Post Incident

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

Controls: NIST IR-8 (Incident Response Plan), NIST SI-5 (Security Alerts, Advisories, and Directives), NIST SI-2 (Flaw Remediation), 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: For SBOM generation without commercial tooling, use Syft (free, open-source) to generate a CycloneDX or SPDX SBOM from each project's node_modules: 'syft dir:. -o cyclonedx-json > sbom_$(date +%F).json'. Feed the SBOM into Grype (also free, from Anchore) for ongoing advisory matching: 'grype sbom:sbom_$(date +%F).json'. Configure GitHub Dependabot (free for public and private repos) with a next-specific alert rule, or use Renovate (self-hostable, free) to automatically open PRs when new next advisories are published to the OSV/npm advisory feed. Document the GHSA-q4gf-8mx6-v5v3 incident in a lessons-learned report that includes the detection gap (time from advisory publication to internal discovery) as a remediation SLA metric.

Evidence: Compile the full incident artifact package for the post-incident review: pre-patch npm audit JSON, post-patch npm audit JSON, web server access log excerpts showing anomalous Server Components request patterns (if exploitation was observed), the timestamped package-lock.json snapshots, and the 24-48 hour post-patch stability log. This package supports both the lessons-learned review required by NIST IR-4 (Incident Handling) and any regulatory reporting obligations if the DoS resulted in service unavailability affecting SLA commitments or user data accessibility.

Detection Guidance

Query software bills of materials, package.json files, and dependency lock files across all repositories and deployed environments for the 'next' npm package. Compare installed versions to affected ranges in GHSA-q4gf-8mx6-v5v3. In application and infrastructure logs, look for spikes in request volume or error rates on Next.js application servers, particularly HTTP 500 or 503 responses that may indicate resource exhaustion. Monitor system-level metrics (CPU, memory, process restarts) on hosts running Next.js. No specific IOCs (IPs, hashes, domains) are available for this vulnerability from the provided data; behavioral and version-based detection are the primary indicators.

Framework Mappings

MITRE-ATTACK

- **T1499** — Endpoint Denial of Service

NIST-800-53R5

- **SC-5** — Denial-of-Service Protection

CIS-V8

- **13.8** — Deploy a Network Intrusion Prevention Solution
- **7.3** — Perform Automated Operating System Patch Management
- **7.4** — Perform Automated Application Patch Management

ISO-27001-2022

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1499	Endpoint Denial of Service	Impact

Sources

Source	URL	Tier
osv	https://osv.dev/vulnerability/GHSA-q4gf-8mx6-v5v3	T3
Axios NPM Supply Chain Compromise: Malicious Packages Deliver ...	https://www.sans.org/blog/axios-npm-supply-chain-compromise-malicio...	T3
Axios NPM Package Compromised: Supply Chain Attack Hits ...	https://www.trendmicro.com/en_us/research/26/c/axios-npm-package-co...	T3
The npm Supply Chain Problem: Why Installing Packages Executes ...	https://linuxsecurity.com/features/npm-install-security-risk	T3
NPM is the biggest weakness of the internet today and it will still ...	https://www.reddit.com/r/webdev/comments/1s96b4a/unpopular_opinion_...	T3

Generated 2026-04-10 18:37 UTC by TJS Security Command Center