INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-12 14:09 UTC

GHSA-492v-c6pp-mqqv: Next.js has a Middleware / Proxy bypass through dynamic route parameter injectio

CVE VULNERABILITY | HIGH | CVSS 8.1

SCC Item ID	SCC-CVE-2026-0161
Type	CVE Vulnerability
CVE ID	CVE-2026-44574
Severity	HIGH
CVSS Base Score	8.1
Affected Products	next (npm) >= 16.0.0, < 16.2.5
Published	2026-05-11T15:54:08Z
Discovery Source	Osv

Executive Summary

A high-severity vulnerability (CVE-2026-44574, CVSS 8.1) in the Next.js web framework allows attackers to bypass authentication and authorization controls enforced by middleware. Any web application built on Next.js versions 16.0.0 through 16.2.4 may be vulnerable, including customer portals, internal tools, and API gateways. Unauthorized access to protected resources is the primary business risk; upgrade to Next.js 16.2.5 or later immediately.

Technical Analysis

CVE-2026-44574 affects the next npm package (versions >= 16.0.0, = 16.2.5. No CVSS vector string is currently published; the 8.1 base score is sourced from the GitHub Security Advisory (GHSA-492v-c6pp-mqqv). Expected NVD record: https://nvd.nist.gov/vuln/detail/CVE-2026-44574. OSV record: https://osv.dev/vulnerability/GHSA-492v-c6pp-mqqv.

Action Checklist

1. Step 1: Containment, Identify all applications running next npm package versions >= 16.0.0 and < 16.2.5 in your environment. Inventory these immediately using your software composition analysis (SCA) tool or package-lock.json / yarn.lock audits. If internet-facing applications cannot be patched within 24

hours, consider placing a WAF rule to inspect and block malformed dynamic route parameters as a temporary control.

2. Step 2: Detection, Review web application and reverse proxy logs for requests containing unusual or encoded characters in dynamic route segments (e.g., path traversal sequences, null bytes, encoded slashes). Query your SIEM for HTTP 200 responses to routes that should have returned 401 or 403. Check for requests that successfully accessed authenticated endpoints without corresponding valid session tokens in application logs.

3. Step 3: Eradication, Upgrade the next npm package to version 16.2.5 or later. Run 'npm install next@16.2.5' or the equivalent yarn/pnpm command. Verify the installed version in package-lock.json. Rebuild and redeploy all affected applications. Do not rely on WAF-only mitigation as a permanent fix.

4. Step 4: Recovery, After upgrading, validate that middleware-protected routes return the expected 401/403 responses to unauthenticated requests. Replay any suspicious log entries identified in Step 2 against the patched application to confirm bypass is no longer possible. Monitor application access logs for 24-48 hours post-patch for continued anomalous access patterns.

5. Step 5: Post-Incident, Evaluate whether dynamic route parameter validation is enforced at the application layer independently of middleware, as defense in depth. Review your SCA and dependency scanning pipeline to ensure critical npm advisories trigger automated alerts. Assess whether any middleware-only authentication patterns should be hardened with server-side validation as a secondary control.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, Legal, and Privacy/Compliance teams immediately if log analysis from Step 2 identifies HTTP 200 responses to middleware-protected routes (authentication, account data, admin panels, or API endpoints handling PII/PHI/financial data) during the window when vulnerable Next.js versions were running, as confirmed unauthorized access may trigger GDPR Article 33, HIPAA Breach Notification, or PCI DSS Incident Response obligations.
Recovery Notes	After patching to Next.js 16.2.5, validate every middleware-protected route category (authentication gates, role-based access routes, API authorization checks) using both clean unauthenticated requests and the specific encoded dynamic route patterns (null bytes, double-encoded slashes, path traversal sequences) identified during detection. Monitor nginx and application-level access logs for 48 hours post-patch, specifically watching for HTTP 200 responses to routes that should require authentication, as threat actors who discovered the bypass pre-patch may retry after observing changed behavior. If any exploitation is confirmed or cannot be ruled out, treat all sessions issued during the vulnerable window as potentially compromised and force re-authentication across affected applications.

Forensic Artifacts	Web server access logs (nginx /var/log/nginx/access.log or Apache /var/log/apache2/access.log) covering the full deployment window of Next.js 16.x — specifically HTTP 200 responses to routes containing encoded characters (%00, %2F, %5C, semicolons, double dots) in dynamic path segments, which indicate successful middleware bypass exploitation of CVE-2026-44574 | Next.js application stdout/stderr logs captured via PM2 ('pm2 logs --nostream'), systemd journald ('journalctl -u '), or Docker log driver — these logs may show route handler execution reaching authenticated business logic without a preceding middleware auth check, revealing the bypass path taken | CDN or reverse proxy request logs (Cloudflare Firewall Events, AWS CloudFront access logs, or AWS ALB access logs) with full URI fields preserved — these sit upstream of the Next.js application and capture the raw malformed dynamic route parameter payloads before any server-side normalization, which is critical for reconstructing the exact bypass technique used | package-lock.json and yarn.lock files from all Next.js application roots at the time of incident discovery — these establish the exact installed version of the 'next' package and any transitive dependencies, providing forensic confirmation of which vulnerable version was running and for how long based on file modification timestamps | Authentication/session service logs (next-auth debug logs, JWT validation middleware logs, or identity provider access logs such as Auth0, Okta, or Cognito) showing successful resource access events that lack a corresponding token issuance or session creation event — the absence of a paired auth event for a successful route access is the definitive forensic indicator that middleware was bypassed via CVE-2026-44574

Per-Action IR Details

Step 1: Containment — Identify all applications running next npm package versions >= 16.0.0 and < 16.2.5 in your environment. Inventory these immediately using your software composition analysis (SCA) tool or package-lock.json / yarn.lock audits. If internet-facing applications cannot be patched within 24 hours, consider placing a WAF rule to inspect and block malformed dynamic route parameters as a temporary control.

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: Run 'npm ls next 2>/dev/null | grep next' recursively across all Node.js application directories, or use 'find / -name package-lock.json -exec grep -l "\"next\"" {} \;' to locate affected manifests. For WAF-less environments, deploy an nginx or Apache ModSecurity rule matching URI patterns with encoded slashes (%2F), null bytes (%00), semicolons, or repeated path segments targeting dynamic route segments (e.g., '[id]', '[slug]', '[...params]') and return 403. Document all identified instances in a shared spreadsheet before proceeding.

Evidence: Before modifying any environment, snapshot the current package-lock.json and yarn.lock files from all Next.js application roots to preserve the installed version state. Capture current web server access logs (nginx: /var/log/nginx/access.log; Apache: /var/log/apache2/access.log) with timestamps prior to any WAF rule deployment, as rule changes will alter the log baseline. If a WAF is in place, export its current rule set and any existing request logs before adding new rules.

Step 2: Detection — Review web application and reverse proxy logs for requests containing unusual or encoded characters in dynamic route segments (e.g., path traversal sequences, null bytes, encoded slashes). Query your SIEM for HTTP 200 responses to routes that should have returned 401 or 403. Check for requests that successfully accessed authenticated endpoints without corresponding valid session tokens in application logs.

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), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: Without a SIEM, run this against nginx access logs: 'grep -E "(GET|POST|PUT|PATCH|DELETE).*/(\[|%5B|%00|%2F%2F|\.\.|;).*\s200\s" /var/log/nginx/access.log' to surface HTTP 200 responses to routes containing dynamic segment injection patterns. Supplement with: 'awk '$9 == 200 {print $7}' /var/log/nginx/access.log | grep -E "(%00|%2F|%5C|;|\.\./)" | sort | uniq -c | sort -rn' to rank suspicious URIs by frequency. For Node.js application logs, search for requests reaching authenticated route handlers without a valid JWT or session cookie: 'grep -E "(dashboard|admin|api/user|api/account|profile)" /var/log/app/access.log | grep -v "Bearer |session="'. Use GoAccess (free, real-time) to visualize access patterns across the full log timeline.

Evidence: Capture the full nginx or Apache access log covering the period from first deployment of Next.js 16.x through present, preserving original file timestamps with 'cp --preserve=timestamps'. Extract Next.js application-level request logs if the app uses a custom logger (common locations: /app/logs/, /var/log/pm2/, or stdout captured by systemd journald — retrieve with 'journalctl -u your-app.service --since "2026-01-01" > app_requests.log'). If Cloudflare, AWS CloudFront, or another CDN is in front of the application, pull firewall/WAF event logs and origin request logs for the same window. Preserve session token validation logs from the authentication middleware layer specifically, as the bypass mechanism targets Next.js middleware execution order, meaning token validation may be skipped entirely with no error logged at the auth layer.

Step 3: Eradication — Upgrade the next npm package to version 16.2.5 or later. Run 'npm install next@16.2.5' or the equivalent yarn/pnpm command. Verify the installed version in package-lock.json. Rebuild and redeploy all affected applications. Do not rely on WAF-only mitigation as a permanent fix.

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 7.3 (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 a CI/CD pipeline, perform the upgrade manually on a staging replica first: 'npm install next@16.2.5 --save-exact && npm run build'. Verify the lock file with 'cat package-lock.json | python3 -m json.tool | grep -A2 '"next"'' to confirm the resolved version is exactly 16.2.5. After production deployment, use 'node -e "console.log(require('./node_modules/next/package.json').version)"' from the application root to verify the runtime version. If using Docker, rebuild the container image from scratch (not layer cache) with '--no-cache' to ensure the patched package is not shadowed by a cached layer: 'docker build --no-cache -t yourapp:16.2.5-patched .'.

Evidence: Before running the upgrade, archive the current node_modules/next/ directory or capture 'npm list next --depth=0' output as a pre-patch baseline. Preserve a copy of the pre-patch package-lock.json and record the git commit SHA of the deployment being replaced. If the application runs in a container, export the running container image with 'docker export > pre_patch_snapshot.tar' for potential forensic comparison. This preserves evidence of the vulnerable state in case post-incident review requires validating which version was running during a suspected exploitation window.

Step 4: Recovery — After upgrading, validate that middleware-protected routes return the expected 401/403 responses to unauthenticated requests. Replay any suspicious log entries identified in Step 2 against the patched application to confirm bypass is no longer possible. Monitor application access logs for 24-48 hours post-patch for continued anomalous access patterns.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-6 (Security and Privacy Function Verification), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST CA-7 (Continuous Monitoring), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Without an automated testing suite, use curl to replay the bypass patterns identified in Step 2 against the patched application: 'curl -v -o /dev/null -w "%{http_code}" https://yourapp.com/api/protected-route/%00' and 'curl -v -o /dev/null -w "%{http_code}" https://yourapp.com/api/protected-route/..%2F..%2Fadmin' — both should return 401 or 403, not 200. For 24-48 hour post-patch monitoring without a SIEM, set up a cron job running every 15 minutes: '*/15 * * * * grep -E "200" /var/log/nginx/access.log | grep -E "(%00|%2F%2F|\.\.%2F|;)" >> /var/log/security/bypass_watch.log && [-s /var/log/security/bypass_watch.log] && mail -s "ALERT: Possible CVE-2026-44574 bypass attempt" security@yourorg.com < /var/log/security/bypass_watch.log'.

Evidence: Before declaring recovery complete, collect a final snapshot of application access logs covering the 24-48 hour monitoring window. Document the exact curl validation commands run, their outputs, and timestamps as evidence of successful remediation verification. If any suspicious requests identified in Step 2 targeted authenticated endpoints that expose PII, PHI, or financial data, preserve those log entries separately and intact (do not rotate or truncate) as they may constitute breach evidence requiring regulatory notification. Capture the 'npm list next' output from the production environment as a timestamped attestation of the patched version running.

Step 5: Post-Incident — Evaluate whether dynamic route parameter validation is enforced at the application layer independently of middleware, as defense in depth. Review your SCA and dependency scanning pipeline to ensure critical npm advisories trigger automated alerts. Assess whether any middleware-only authentication patterns should be hardened with server-side validation as a secondary control.

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 RA-5 (Vulnerability Monitoring and Scanning), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.3 (Address Unauthorized Software)

Compensating: Integrate 'npm audit' into your CI/CD pipeline or as a pre-commit hook to catch critical npm advisories before deployment: add 'npm audit --audit-level=high' as a build step that fails the pipeline on HIGH or CRITICAL findings. For free SCA scanning, configure Dependabot (GitHub) or Renovate Bot to monitor next package updates and auto-open PRs for security patches. Conduct a code review specifically targeting all Next.js route handlers that rely solely on middleware (defined in middleware.js/middleware.ts at the project root) for authentication — any handler that does not independently verify session validity server-side should be flagged for hardening with a server-side auth check (e.g., using getServerSideProps or API route-level token validation via next-auth or a custom JWT verify call).

Evidence: Compile a lessons-learned report documenting: (1) the time gap between GHSA-492v-c6pp-mqqv publication and detection in your environment, (2) which applications were running vulnerable Next.js versions and for how long, (3) whether any confirmed or suspected exploitation occurred based on the log review in Step 2, and (4) the SCA or dependency monitoring gap that allowed the vulnerable version to reach production. This report supports both internal process improvement and any regulatory disclosure obligations if exploitation of protected data routes is confirmed or cannot be ruled out.

Detection Guidance

Query web/proxy logs for requests to Next.js dynamic route paths ([param] segments) containing percent-encoded characters, double slashes, null bytes, or unexpected special characters. Flag HTTP 200 responses on routes your middleware should protect, particularly where no valid session cookie or authorization header is present in the request. In your SIEM, correlate successful responses to protected API routes against authentication event logs, gaps where a resource was accessed without a corresponding login event are a strong indicator. If you have endpoint detection on your Node.js servers, look for unexpected child processes or file reads triggered from the Next.js runtime. No public IOCs (IPs, hashes, domains) are associated with this vulnerability at time of writing.

Framework Mappings

MITRE-ATTACK

- **T1550** — Use Alternate Authentication Material
- **T1190** — Exploit Public-Facing Application

NIST-800-53R5

- **CA-8** — Penetration Testing
- **RA-5** — Vulnerability Monitoring and Scanning
- **SC-7** — Boundary Protection
- **SI-2** — Flaw Remediation
- **SI-7** — Software, Firmware, and Information Integrity
- **AC-3** — Access Enforcement
- **SI-10** — Information Input Validation

OWASP-TOP10-2021

- **A01:2021** — Broken Access Control
- **A03:2021** — Injection

CIS-V8

- **6.1** — Establish an Access Granting Process
- **6.2** — Establish an Access Revoking Process
- **16.10** — Apply Secure Design Principles in Application Architectures

SOC2-TSC

- **CC6.1** — The entity implements logical access security software, infrastructure, and architectures over protected information assets

HIPAA-SECURITY

- **164.312(a)(1)** — Access Control

ISO-27001-2022

- **A.8.26** — Application security requirements
- **A.8.8** — Management of technical vulnerabilities

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1550	Use Alternate Authentication Material	Defense-Evasion
T1190	Exploit Public-Facing Application	Initial-Access

Sources

Source	URL	Tier
osv	https://osv.dev/vulnerability/GHSA-492v-c6pp-mqqv	T3
CVE-2026-44574 - Docker Scout	https://scout.docker.com/vulnerabilities/id/CVE-2026-44574?t=npm&am...;	T3
CVE-2026-44574 - CVE Record	https://www.cve.org/CVERecord?id=CVE-2026-44574	T3
CVE-2026-4574 Detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-4574	T1
The most severe Linux threat to surface in years catches the world ...	https://arstechnica.com/security/2026/04/as-the-most-severe-linux-t...	T2
NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-44574	T1

Generated 2026-05-12 14:09 UTC by TJS Security Command Center