INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-17 06:28 UTC

CVE-2025-14869: GitLab has remediated an issue in GitLab CE/EE affecting all versions from 18.5 before 18.9.7, 18.10...

CVE VULNERABILITY | HIGH | CVSS 7.5

SCC Item ID	SCC-CVE-2026-0188
Type	CVE Vulnerability
CVE ID	CVE-2025-14869
Severity	HIGH
CVSS Base Score	7.5
EPSS Score	0.0003 (9th percentile)
Affected Products	GitLab CE/EE versions 18.5 before 18.9.7, 18.10 before 18.10.6, and 18.11 before 18.11.3
Published	2026-05-14T06:16:20.757
Discovery Source	Nvd

Executive Summary

GitLab has patched a denial-of-service vulnerability in GitLab CE/EE that allows unauthenticated attackers to crash or degrade the service by sending crafted API requests. All GitLab instances running versions 18.5 through 18.11.2 are affected, with no login or credentials required to trigger the attack. Organizations using GitLab for software development pipelines, CI/CD, or source code management face potential disruption to development operations until the patch is applied.

Technical Analysis

CVE-2025-14869 is a denial-of-service vulnerability in GitLab CE/EE affecting versions 18.5 through 18.9.6, 18.10 through 18.10.5, and 18.11 through 18.11.2. The flaw is rooted in CWE-1284 (improper validation of specified quantity in input), allowing an unauthenticated remote attacker to send specially crafted payloads to specific API endpoints and cause service disruption. No authentication is required, mapping to MITRE ATT&CK T1499 (Endpoint Denial of Service) and T1498 (Network Denial of Service). CVSS base score is 7.5 (High). EPSS score is 0.00029 (0.03% percentile, 8.6th percentile rank), indicating low current exploitation probability. Patched versions are 18.9.7, 18.10.6, and 18.11.3. No CISA KEV listing as of publication. Source: NVD (https://nvd.nist.gov/vuln/detail/CVE-2025-14869).

Action Checklist

1. Step 1: Identification & Mitigation, Identify all GitLab CE/EE instances in your environment running versions 18.5.x through 18.11.2. If internet-facing instances cannot be patched immediately, restrict API endpoint access via WAF rules or network controls to trusted IP ranges. GitLab's advisory covers versions 18.9.7, 18.10.6, and 18.11.3 as fixed releases.
2. Step 2: Detection, Review web/API access logs on GitLab instances for high-volume, malformed, or anomalous requests to API endpoints from unauthenticated sources. Look for sudden spikes in 4xx/5xx responses or process CPU/memory exhaustion coinciding with inbound API traffic. SIEM searches should target GitLab application logs for repeated unauthenticated API calls with unusual payload sizes or structures.
3. Step 3: Eradication, Upgrade affected GitLab CE/EE instances to version 18.9.7, 18.10.6, or 18.11.3 per GitLab's official release. Follow GitLab's upgrade path documentation to avoid skipping required intermediate versions. Confirm the running version post-upgrade via the GitLab Admin Area or CLI.
4. Step 4: Recovery, After patching, validate GitLab service health, CI/CD pipeline execution, and API responsiveness. Monitor application and system logs for 24-48 hours post-patch for any residual instability. Confirm no active exploitation occurred during the exposure window by reviewing pre-patch API logs for anomalous unauthenticated traffic patterns.
5. Step 5: Post-Incident, Evaluate whether internet-facing GitLab API endpoints require tighter default access controls or rate limiting as a standing control. Review your patch SLA for high-severity CVEs in developer tooling; GitLab vulnerabilities directly affect software supply chain integrity. Consider adding GitLab API traffic to SIEM monitoring baselines if not already present.

IR / Forensic Enrichment

Triage Priority	URGENT
Escalation Criteria	Escalate immediately to CISO and development leadership if pre-patch nginx log analysis confirms successful DoS exploitation (clustered HTTP 500/503 responses from unauthenticated external IPs to /api/v4/ endpoints), if CI/CD pipeline outages during the exposure window are now attributable to CVE-2025-14869 exploitation, or if any GitLab instance hosts code for regulated systems where a supply chain disruption triggers contractual or regulatory notification obligations.
Recovery Notes	After upgrading to GitLab 18.9.7, 18.10.6, or 18.11.3, run 'sudo gitlab-rake gitlab:check SANITIZE=true' and verify all Gitaly, Sidekiq, and Puma workers return healthy status before removing any WAF IP restrictions applied during containment. Monitor /var/log/gitlab/gitlab-rails/production.log and /var/log/gitlab/nginx/gitlab_access.log for continued anomalous unauthenticated API traffic for a minimum of 48 hours post-patch, as adversaries who identified the instance during the vulnerable window may probe again. Validate that all CI/CD pipelines that were queued or failed during any DoS window have been re-triggered and completed successfully before declaring full recovery.

Forensic Artifacts	GitLab nginx access log (/var/log/gitlab/nginx/gitlab_access.log): Contains timestamped records of every inbound API request including source IP, URI path, HTTP method, response code, and bytes transferred — the primary artifact for reconstructing the unauthenticated /api/v4/ request flood pattern specific to CVE-2025-14869 exploitation. | GitLab Rails application log (/var/log/gitlab/gitlab-rails/production.log): Records server-side processing errors, worker timeouts, and exception stack traces that would be generated when crafted API requests trigger the DoS condition — key for confirming exploitation rather than benign traffic spikes. | System resource utilization snapshots (output of 'sar -u', 'vmstat', 'top -b'): Captures CPU and memory exhaustion events correlated with inbound API traffic timestamps, providing forensic evidence that the DoS condition was successfully triggered on a specific GitLab worker process. | GitLab version manifest (/opt/gitlab/version-manifest.txt) and package installation history ('apt list --installed' or 'rpm -qa'): Establishes the exact vulnerable version running during the exposure window and the precise dates it was installed and patched, supporting timeline reconstruction for any breach notification or audit requirement. | Network connection state capture ('ss -tnp' or 'netstat -an' output timestamped at containment): Records any active or TIME_WAIT TCP connections from external IPs to GitLab API ports at the moment of containment, identifying specific source addresses that may have been mid-exploit when WAF restrictions were applied.

Per-Action IR Details

Step 1: Containment — Identify all GitLab CE/EE instances in your environment running versions 18.5.x through 18.11.2. If internet-facing instances cannot be patched immediately, restrict API endpoint access via WAF rules or network controls to trusted IP ranges. GitLab's advisory covers versions 18.9.7, 18.10.6, and 18.11.3 as fixed releases.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SC-7 (Boundary Protection), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory)

Compensating: Run 'gitlab-rake gitlab:env:info' or query the GitLab Admin Area at /admin/version_check to enumerate version strings across all instances. For WAF-less environments, use iptables or nftables to restrict TCP/443 and TCP/80 inbound to the GitLab server to a defined allowlist of corporate egress IPs: 'iptables -I INPUT -p tcp --dport 443 ! -s -j DROP'. If running behind nginx, add 'allow ; deny all;' blocks to the location /api/ directive in /etc/gitlab/nginx.conf and run 'gitlab-ctl reconfigure' to apply without full restart.

Evidence: Before restricting access, capture a full snapshot of current active connections to the GitLab API: run 'ss -tnp | grep :443' and 'netstat -an | grep ESTABLISHED' to record any in-progress connections from external IPs that may represent active exploitation. Export the GitLab application log at /var/log/gitlab/gitlab-rails/production.log and the nginx access log at /var/log/gitlab/nginx/gitlab_access.log covering the prior 72 hours as baseline evidence before WAF rules alter the traffic pattern.

Step 2: Detection — Review web/API access logs on GitLab instances for high-volume, malformed, or anomalous requests to API endpoints from unauthenticated sources. Look for sudden spikes in 4xx/5xx responses or process CPU/memory exhaustion coinciding with inbound API traffic. SIEM searches should target GitLab application logs for repeated unauthenticated API calls with unusual payload sizes or structures.

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-3 (Content of Audit Records), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: Without a SIEM, use the following bash pipeline directly on the GitLab host to identify unauthenticated API flood patterns: 'grep "/api/v4" /var/log/gitlab/nginx/gitlab_access.log | grep " 4[0-9][0-9] \| 5[0-9][0-9] " | awk "{print $1}" | sort | uniq -c | sort -rn | head -20' to surface top offending IPs by error volume. To detect CPU/memory exhaustion correlated with API traffic, run 'sar -u 1 60' or review '/var/log/gitlab/gitlab-rails/production.log' for 'timeout' or 'worker killed' strings. For pipeline-based monitoring, deploy a Sigma rule targeting GitLab nginx logs for any single source IP generating more than 100 unauthenticated /api/v4 requests within a 60-second window (sigma rule category: webserver, product: gitlab).

Evidence: Preserve the full nginx access log (/var/log/gitlab/nginx/gitlab_access.log) and GitLab Rails application log (/var/log/gitlab/gitlab-rails/production.log) before any log rotation occurs. Extract all entries where the HTTP Authorization header is absent and the request path begins with /api/v4/, noting request sizes (bytes_sent field), response codes, and inter-request timing to characterize the crafted payload pattern specific to CVE-2025-14869. Also collect system resource snapshots via 'top -b -n 5 > /tmp/top_snapshot.txt' and 'vmstat 1 30 > /tmp/vmstat_snapshot.txt' to document the denial-of-service impact profile.

Step 3: Eradication — Upgrade affected GitLab CE/EE instances to version 18.9.7, 18.10.6, or 18.11.3 per GitLab's official release. Follow GitLab's upgrade path documentation to avoid skipping required intermediate versions. Confirm the running version post-upgrade via the GitLab Admin Area or CLI.

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 7.2 (Establish and Maintain a Remediation Process)

Compensating: For teams without automated patch management, follow GitLab's upgrade path tool at https://gitlab-com.gitlab.io/support/toolbox/upgrade-path/ to determine if intermediate versions are required before reaching 18.9.7, 18.10.6, or 18.11.3. Execute the upgrade via the OS package manager: 'sudo apt-get install gitlab-ee=18.11.3-ee.0' (Debian/Ubuntu) or 'sudo yum install gitlab-ee-18.11.3' (RHEL/CentOS), preceded by a full backup using 'gitlab-backup create'. Post-upgrade, verify the running version by executing 'sudo gitlab-rake gitlab:env:info | grep "GitLab information" -A 5' and confirm the output shows the target fixed version.

Evidence: Before initiating the upgrade, capture the pre-patch version fingerprint via 'cat /opt/gitlab/version-manifest.txt' and export the GitLab configuration backup ('gitlab-ctl backup-etc') to establish a pre-remediation baseline. Document the package version history ('apt list --installed | grep gitlab' or 'rpm -qa | grep gitlab') to confirm the vulnerable version was running and to support any post-incident timeline reconstruction. Preserve all log files collected in Step 2 in a separate evidence directory before the upgrade process potentially rotates or overwrites them.

Step 4: Recovery — After patching, validate GitLab service health, CI/CD pipeline execution, and API responsiveness. Monitor application and system logs for 24-48 hours post-patch for any residual instability. Confirm no active exploitation occurred during the exposure window by reviewing pre-patch API logs for anomalous unauthenticated traffic 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), CIS 8.2 (Collect Audit Logs)

Compensating: Validate GitLab service health post-upgrade using built-in tooling: run 'sudo gitlab-ctl status' to confirm all service components (puma, sidekiq, postgresql, gitaly, nginx) are running, and 'sudo gitlab-rake gitlab:check SANITIZE=true' to verify Rails environment integrity. To confirm API responsiveness without enterprise tooling, issue a test unauthenticated API call: 'curl -s -o /dev/null -w "%{http_code}" https:///api/v4/version' — a 200 response confirms the API is healthy. For CI/CD pipeline validation, manually trigger a test pipeline on a non-critical project and confirm successful job execution through all stages.

Evidence: Conduct a retrospective analysis of pre-patch nginx access logs to identify any source IPs that sent high-volume unauthenticated requests to /api/v4/ endpoints during the CVE-2025-14869 exposure window (any date

on or after the instance was upgraded into the 18.5–18.11.2 range). Specifically grep for HTTP 503 or 500 responses clustered in time with unauthenticated requests, which would indicate successful DoS triggering: 'grep "/api/v4/" /var/log/gitlab/nginx/gitlab_access.log | grep -E " (500|503) " | grep -v "Authorization"'. Preserve these findings as the exploitation-confirmation artifact for the post-incident record.

Step 5: Post-Incident — Evaluate whether internet-facing GitLab API endpoints require tighter default access controls or rate limiting as a standing control. Review your patch SLA for high-severity CVEs in developer tooling; GitLab vulnerabilities directly affect software supply chain integrity. Consider adding GitLab API traffic to SIEM monitoring baselines if not already present.

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-5 (Security Alerts, Advisories, and Directives), NIST RA-3 (Risk Assessment), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 8.2 (Collect Audit Logs)

Compensating: Without a SIEM, implement GitLab's built-in rate limiting at /admin/application_settings/network under 'Unauthenticated API request rate limit' — set a threshold of no more than 10 requests per minute per IP for unauthenticated API calls as a standing control against future unauthenticated DoS vectors. Subscribe to GitLab security advisories via the GitLab Security Newsletter (https://about.gitlab.com/company/contact/) and the GitLab CE/EE releases RSS feed to reduce time-to-awareness for future CVEs. For patch SLA enforcement, document a formal 72-hour remediation target for CVSS ≥7.0 vulnerabilities in internet-facing developer tooling given supply chain exposure.

Evidence: Produce a lessons-learned document capturing: (1) the date the vulnerable GitLab version was first deployed versus the CVE publication date to calculate actual exposure window duration; (2) whether any CI/CD pipeline disruption occurred during the exposure period that is now attributable to exploitation rather than infrastructure issues; (3) the total count of unauthenticated /api/v4/ requests from external IPs during the exposure window as extracted from preserved nginx logs, to quantify whether opportunistic scanning or targeted exploitation occurred.

Detection Guidance

Query GitLab application logs and web server access logs for unauthenticated requests (no Authorization header or session token) targeting API endpoints, particularly those generating 500-level errors or abnormally long response times. Look for repeated requests from single IP addresses or distributed sources in short timeframes. On the host level, monitor GitLab Puma/Unicorn worker processes for CPU or memory spikes that correlate with inbound API traffic. No public IOCs or specific endpoint paths have been disclosed as of publication; detection relies on behavioral anomalies rather than known signatures. EPSS score of 0.00029 (8.6th percentile) suggests active exploitation is currently unlikely, but unauthenticated DoS vulnerabilities can be trivially scripted once details are public.

Framework Mappings

MITRE-ATTACK

- **T1499** — Endpoint Denial of Service
- **T1498** — Network Denial of Service

NIST-800-53R5

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

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
T1498	Network Denial of Service	Impact

Sources

Source	URL	Tier
nvd	https://nvd.nist.gov/vuln/detail/CVE-2025-14869	T1
(consolidated)	https://nvd.nist.gov/vuln/detail/CVE-2025-14870	T1
CVE-2025-14869 - CVE Record	https://www.cve.org/CVERecord?id=CVE-2025-14869	T3
CVE-2025-14869 Security Vulnerability Analysis & Exploit Details	https://cve.akaoma.com/cve-2025-14869	T3
CVE-2025-14869: GitLab CE/EE DoS Vulnerability - SentinelOne	https://www.sentinelone.com/vulnerability-database/cve-2025-14869/	T3

Generated 2026-05-17 06:28 UTC by TJS Security Command Center