INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-12 14:16 UTC

Chained SQL Injection and Unsafe Deserialization in LangGraph Enable RCE on Self-Hosted AI Agent Deployments

CVE VULNERABILITY | HIGH | CVSS 7.5

SCC Item ID	SCC-CVE-2026-0297
Type	CVE Vulnerability
CVE ID	CVE-2025-67644, CVE-2026-28277, CVE-2026-27022
Severity	HIGH
CVSS Base Score	7.5
EPSS Score	0.0002 (7th percentile)
Affected Products	langgraph-checkpoint-sqlite < 3.0.1; langgraph < 1.0.10; @langchain/langgraph-checkpoint-redis < 1.0.1 (self-hosted deployments only; LangSmith managed platform not affected)
Published	2026-06-12T05:50:36
Discovery Source	Rss

Executive Summary

Three chained vulnerabilities in LangGraph, an open-source AI agent orchestration framework, allow an unauthenticated attacker to execute arbitrary code on self-hosted deployments using SQLite or Redis checkpointers. Organizations running LangGraph in their own infrastructure, particularly those exposing agent state endpoints externally, face direct risk of full server compromise. Patched versions are available; unpatched self-hosted deployments with exposed get_state_history() endpoints should be patched as a priority.

Technical Analysis

CVE-2025-67644, CVE-2026-28277, and CVE-2026-27022 form an exploit chain targeting LangGraph's state persistence layer. An attacker with network access to an exposed get_state_history() endpoint can supply unsanitized filter parameters to trigger SQL injection (CWE-89, CWE-943) against the SQLite checkpointer, or manipulate Redis-backed state. The injected path delivers a crafted msgpack payload that is passed to an unsafe deserialization routine (CWE-502), resulting in arbitrary code execution with the privileges of the LangGraph process. No authentication is required. MITRE techniques involved include T1190 (Exploit Public-Facing Application), T1203 (Exploitation for Client Execution), T1059 (Command and Scripting

Interpreter), T1565.001 (Stored Data Manipulation), and T1552 (Unsecured Credentials). Affected packages: langgraph-checkpoint-sqlite < 3.0.1, langgraph < 1.0.10, @langchain/langgraph-checkpoint-redis < 1.0.1. The LangSmith managed platform is explicitly unaffected. EPSS score is 0.022% (6.58th percentile) indicating low active exploitation in the wild. However, if the get_state_history() endpoint is exposed to an untrusted network, one crafted request is sufficient to achieve server compromise. Patches are available for all three packages.

Action Checklist

1. Step 1: Containment, Immediately restrict network access to the get_state_history() endpoint on all self-hosted LangGraph deployments. If the endpoint is internet-facing, place it behind a firewall or WAF rule blocking external access until patching is complete. Identify all instances running langgraph < 1.0.10, langgraph-checkpoint-sqlite < 3.0.1, or @langchain/langgraph-checkpoint-redis < 1.0.1.
2. Step 2: Detection, Query application and web server logs for anomalous requests to get_state_history() endpoints, specifically filter parameters containing SQL metacharacters (' " -- ; UNION SELECT), oversized or binary-encoded msgpack payloads, or unexpected process spawning from the LangGraph service process. Check host-level logs for new child processes, outbound connections from the LangGraph process to external IPs, or modified files in the application directory. Monitor for new or modified local accounts created post-exploitation.
3. Step 3: Eradication, Upgrade all affected packages to patched versions: langgraph to >= 1.0.10, langgraph-checkpoint-sqlite to >= 3.0.1, @langchain/langgraph-checkpoint-redis to >= 1.0.1. Apply updates through your standard package manager (pip, npm). After patching, rotate all credentials and API keys accessible to the LangGraph process, including database credentials, Redis authentication tokens, and any downstream API keys stored in environment variables.
4. Step 4: Recovery, After patching, validate installed package versions against patched release numbers. Restore the get_state_history() endpoint to production only after confirming patched versions are running. Monitor application logs and host process trees for 72 hours post-remediation for anomalous behavior indicating a persistent foothold. Confirm firewall or access control rules are in place to restrict get_state_history() access to authorized internal clients only.
5. Step 5: Post-Incident, Conduct a review of all self-hosted AI agent frameworks in the environment for similar exposure patterns: unauthenticated state endpoints, deserialization of external input, and direct database query construction from user-supplied parameters. Document this vulnerability chain as a control gap case study for developer training.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal, and privacy counsel immediately if forensic review of web server access logs, SQLite checkpoint.db, or Redis keyspace dump reveals evidence of successful exploitation (child process spawning from LangGraph, exfiltration of environment variables containing API keys or PII, or attacker-controlled msgpack payloads stored in checkpoint backend), as this constitutes a confirmed breach of a self-hosted AI orchestration system that may have accessed downstream data stores, triggering breach notification obligations under applicable data protection regulations.

Recovery Notes	Before restoring the get_state_history() endpoint to any externally-accessible state, verify installed package versions programmatically (not by inspecting package metadata files, which can be spoofed) and confirm WAF rules actively block requests with SQL metacharacters in thread_id and checkpoint_id parameters. Purge the Redis keyspace and SQLite checkpoint database of all entries created during the suspected exploitation window — attacker-planted checkpoint payloads persist in the backend and can re-trigger deserialization on subsequent legitimate LangGraph reads even after patching. Maintain elevated logging verbosity (full request body capture for get_state_history() POST requests) and process-tree monitoring for a minimum of 72 hours post-restoration, extending to 7 days if any indicators of compromise were confirmed during the detection phase.
Forensic Artifacts	Web server access logs (nginx/access.log, Apache access_log, or uvicorn stdout) containing HTTP requests to paths matching get_state_history — filter for URI-encoded SQL metacharacters (%27, %22, %2D%2D, UNION, SELECT) in query parameters and POST bodies with Content-Length exceeding expected state query sizes, establishing the exploitation attempt timeline | SQLite database WAL file (checkpoint.db-wal) and main database file used by langgraph-checkpoint-sqlite — SQL injection via CVE-2025-67644 targets thread_id and checkpoint_id query construction, leaving attacker-supplied string literals visible in the database query log or embedded as malformed checkpoint records in the schema | Redis RDB snapshot or keyspace dump (redis-cli --rdb dump.rdb or BGSAVE output) from the Redis instance backing @langchain/langgraph-checkpoint-redis — CVE-2026-28277 exploits unsafe msgpack deserialization of checkpoint values, meaning attacker-crafted pickle or msgpack payloads are stored as serialized checkpoint objects and are directly recoverable from the keyspace | Memory image of the LangGraph service process (winpmem/LiME output) captured before process termination — successful RCE via CVE-2026-27022 injects shellcode or spawns a reverse shell from within the Python/Node.js interpreter heap; the injected payload, attacker IP, and any staged commands are recoverable from process memory and are destroyed upon process kill or host reboot | Sysmon Event ID 1 (Process Create) records where ParentImage is the Python or Node.js interpreter running LangGraph and ChildCommandLine contains shell invocations (sh, bash, cmd.exe, powershell.exe) or network utilities (curl, wget, nc, ncat) — this is the primary host-based indicator that deserialization-to-RCE (CVE-2026-27022) was successfully triggered and a command was executed under the LangGraph process context

Per-Action IR Details

Step 1: Containment — Immediately restrict network access to the get_state_history() endpoint on all self-hosted LangGraph deployments. If the endpoint is internet-facing, place it behind a WAF or firewall rule blocking external access until patching is complete. Identify all instances running langgraph < 1.0.10, langgraph-checkpoint-sqlite < 3.0.1, or @langchain/langgraph-checkpoint-redis < 1.0.1. Per NIST AC-4 (Information Flow Enforcement), enforce approved authorizations controlling information flow between external networks and the affected service.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: choose a containment strategy based on criteria including potential damage, need for evidence preservation, and service availability; implement network-level isolation before altering host state.

Controls: NIST AC-4 (Information Flow Enforcement), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 4.5 (Implement and Manage a Firewall on End-User Devices)

Compensating: On Linux hosts: `iptables -I INPUT -p tcp --dport -s 0.0.0.0/0 -j DROP` to block all external traffic, then add explicit ACCEPT rules for trusted internal CIDR ranges. On Windows: `New-NetFirewallRule -DisplayName 'Block LangGraph External' -Direction Inbound -LocalPort -Protocol TCP -RemoteAddress Internet -Action Block`. Enumerate

all running LangGraph instances with `pip show langgraph langgraph-checkpoint-sqlite` and `npm list @langchain/langgraph-checkpoint-redis` on each host. Use a simple bash loop across your inventory: `for host in $(cat hosts.txt); do ssh $host 'pip show langgraph 2>/dev/null | grep Version'; done`.

Evidence: BEFORE applying firewall rules or WAF blocks, capture: (1) active TCP connections from the LangGraph process via `ss -tnp | grep ` or `netstat -ano | findstr ` — these connections disappear once network access is cut; (2) full process tree showing the LangGraph service and any child processes it has spawned (`ps auxf` on Linux or `Get-CimInstance Win32_Process | Select ProcessId,ParentProcessId,CommandLine` on Windows); (3) RAM image of the LangGraph host if any indicators of active exploitation are present, using winpmem (Windows) or LiME kernel module (Linux) — deserialization payloads and injected shellcode exist only in memory and are unrecoverable after process termination or isolation.

Step 2: Detection — Query application and web server logs for anomalous requests to get_state_history() endpoints, specifically filter parameters containing SQL metacharacters (' " -- ; UNION SELECT), oversized or binary-encoded msgpack payloads, or unexpected process spawning from the LangGraph service process. Check host-level logs for new child processes, outbound connections from the LangGraph process, or modified files in the application directory (NIST SI-4, AU-6, CIS 8.2). Use D3-SFA (System File Analysis) to check for modifications to LangGraph application files and D3-LAM (Local Account Monitoring) to detect new or modified local accounts created post-exploitation.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: analyze indicators across log sources, correlate host and network evidence, and estimate scope of compromise before proceeding to containment actions.

Controls: NIST AU-6 (Audit Record Review, Analysis, And Reporting), NIST AU-2 (Event Logging), NIST AU-12 (Audit Record Generation), CIS 8.2 (Collect Audit Logs)

Compensating: Deploy Sysmon (config with SwiftOnSecurity baseline) on LangGraph hosts and filter for Event ID 1 (Process Create) where ParentImage contains the Python or Node.js executable running LangGraph and ChildImage is cmd.exe, sh, bash, or a network utility (curl, wget, nc). Query web server access logs with: `grep -E "get_state_history" access.log | grep -iE "('|\"|--|;|UNION|SELECT|DROP|INSERT)"` for SQLi patterns. For msgpack deserialization abuse, look for HTTP POST bodies to the get_state_history endpoint exceeding 10KB or containing non-UTF8 byte sequences: `grep 'get_state_history' access.log | awk '$10 > 10240'`. Use osquery to detect new local accounts: `SELECT * FROM users WHERE creation_time > (SELECT strftime('%s','now','-24 hours'));` and new SUID binaries: `SELECT * FROM suid_bin WHERE last_change > (SELECT strftime('%s','now','-24 hours'));`.

Evidence: Volatile evidence to capture BEFORE any process termination or isolation: (1) LangGraph web/application server access logs — specifically HTTP requests to paths matching `/*/get_state_history` or equivalent REST route, preserving full request URI, query string, POST body size, source IP, and response code; (2) SQLite database file (`checkpoint.db` or equivalent) used by langgraph-checkpoint-sqlite — capture a read-only copy before any writes occur, as SQL injection payloads may be embedded in thread_id or checkpoint_id parameters visible in the database query log or WAL file; (3) Redis keyspace snapshot (`BGSAVE` or `DEBUG RELOAD`) from the Redis instance backing @langchain/langgraph-checkpoint-redis — deserialized payloads are stored as checkpoint values and may contain attacker-controlled msgpack objects; (4) Sysmon Event ID 3 (Network Connection) records showing outbound connections initiated by the LangGraph Python/Node process — post-exploitation reverse shells or C2 beacons originate from this process; (5) contents of environment variable store accessible to the LangGraph process (`/proc//environ` on Linux) — captures API keys, database credentials, and Redis auth tokens that an attacker would have accessed upon successful RCE.

Step 3: Eradication — Upgrade all affected packages to patched versions: langgraph to >= 1.0.10, langgraph-checkpoint-sqlite to >= 3.0.1, @langchain/langgraph-checkpoint-redis to >= 1.0.1. Apply updates through your standard package manager (pip, npm). After patching, rotate all credentials and API keys accessible to the LangGraph process, including database credentials, Redis authentication tokens, and any downstream API keys stored in environment variables (D3-CRO — Credential Rotation). Per CIS 7.4, perform application patch management and verify installed versions post-update.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: eliminate components of the incident including malware, vulnerabilities exploited, and compromised credentials; verify eradication before beginning recovery.

Controls: CIS 7.4 (Perform Automated Application Patch Management), CIS 7.2 (Establish and Maintain a Remediation Process), NIST AC-2 (Account Management), NIST IA-5 — [no mapped control in knowledge base for credential management beyond AC-2; citing AC-2 for account/credential lifecycle]

Compensating: Run `pip install --upgrade langgraph==1.0.10 langgraph-checkpoint-sqlite==3.0.1` and `npm install @langchain/langgraph-checkpoint-redis@1.0.1` then verify with `pip show langgraph langgraph-checkpoint-sqlite` and `npm list @langchain/langgraph-checkpoint-redis`. For credential rotation without a secrets manager: enumerate environment variables in the LangGraph service unit file (`/etc/systemd/system/.env` or `.env` file in the app directory), rotate each credential at the issuing system (database, Redis `CONFIG SET requirepass `, API provider dashboard), and update the service configuration before restart. If compromise is confirmed or suspected, treat ALL secrets accessible to the process as compromised — not just those in obvious config files.

Evidence: BEFORE running pip/npm upgrades or rotating credentials, capture: (1) a frozen snapshot of the current Python virtual environment or Node.js node_modules directory (`pip freeze > pre-patch-requirements.txt` / `npm list --depth=0 > pre-patch-packages.txt`) to establish the exact vulnerable version baseline for forensic record; (2) contents of all `.env` files, systemd EnvironmentFile entries, and Kubernetes secrets mounted to the LangGraph pod — these document the full credential exposure surface an attacker had access to post-RCE; (3) Redis AUTH configuration and ACL list (`redis-cli ACL LIST`) before rotation, to document pre-incident permission state; (4) file integrity baseline of the LangGraph application directory (`sha256sum -r /app/*.py > pre-patch-hashes.txt`) to detect any attacker-modified source files that patching alone would not remove — a patch overwrites packages but not attacker-planted webshells or backdoors in the application layer.

Step 4: Recovery — After patching, validate installed package versions against patched release numbers. Restore the get_state_history() endpoint to production only after confirming patched versions are running. Monitor application logs and host process trees for 72 hours post-remediation for anomalous behavior indicating a persistent foothold. Confirm firewall or WAF rules are in place to restrict get_state_history() access to authorized internal clients only (NIST AC-3, AC-4). Re-enable external access only if business-required and after WAF rules filtering SQL injection patterns are confirmed active.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery: restore systems to normal operation, confirm that systems are functioning normally, and implement monitoring to detect recurrence; verify integrity before restoring production access.

Controls: NIST AC-3 (Access Enforcement), NIST AC-4 (Information Flow Enforcement), NIST AU-6 (Audit Record Review, Analysis, And Reporting), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Verify patched versions with: `python -c "import langgraph; print(langgraph.__version__)"` and `python -c "import langgraph_checkpoint_sqlite; print(langgraph_checkpoint_sqlite.__version__)"` — confirm output is >= 1.0.10 and >= 3.0.1 respectively. For 72-hour post-recovery monitoring without SIEM: schedule a cron job every 15 minutes running `ps auxf | grep -A5 langgraph > /var/log/langgraph-proctree-$(date +%s).log` and `ss -tnp | grep >> /var/log/langgraph-netconn.log` to capture any re-exploitation spawning child processes or establishing outbound connections. For WAF rule validation without enterprise appliance, use NGINX `ngx_http_lua_module` or ModSecurity with OWASP CRS and test with a benign SQLi probe: `curl -v 'https:///get_state_history?thread_id=test%27+OR+1%3D1--'` and confirm it is blocked with HTTP 403.

Evidence: During the 72-hour monitoring window, continuously collect: (1) LangGraph application stdout/stderr logs for Python tracebacks, unexpected import errors (which may indicate a backdoored module), or deserialization errors that could signal re-exploitation attempts against the patched endpoint; (2) Sysmon Event ID 1 (Process Create) and Event ID 3 (Network Connection) filtered to the LangGraph process PID — a persistent implant or re-exploitation will manifest as anomalous child processes (sh, bash, python -c) or new outbound connections to non-inventory IPs; (3) SQLite WAL file (`checkpoint.db-wal`) for any residual attacker-controlled checkpoint entries that survived the patch but could be re-triggered by LangGraph reading stored state; (4) file modification timestamps in the application directory (`find /app -newer /app/requirements.txt -type f`) to detect late-stage persistence mechanisms (cron backdoors, modified __init__.py files) not removed by the package upgrade.

Step 5: Post-Incident — Conduct a review of all self-hosted AI agent frameworks in the environment for similar exposure patterns: unauthenticated state endpoints, deserialization of external input, and direct database query construction from user-supplied parameters. Document this vulnerability chain as a control gap case study for developer training. Map to CIS 7.1 (Establish and Maintain a Vulnerability Management Process) and evaluate whether AI/ML framework dependencies are included in your software inventory (CIS 2.1) and patch management scope (CIS 7.3, CIS 7.4). Apply D3-CH (Credential Hardening) and D3-UAP (User Account Permissions) reviews to all service accounts used by AI orchestration workloads.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: conduct lessons-learned meetings, produce incident reports, retain evidence per policy, and update detection and response capabilities based on findings from this incident.

Controls: CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported), NIST AC-6 (Least Privilege)

Compensating: Extend your software inventory to explicitly tag AI/ML framework packages: run `pip list --format=json | python -m json.tool | grep -iE 'langgraph|langchain|autogen|crewai|haystack|llamaindex'` across all hosts and add results to your asset inventory. Create a YARA rule targeting msgpack-encoded Python pickle opcodes (byte sequence `0x80 0x05` for protocol 5) in HTTP POST bodies captured by network taps or proxy logs to detect future deserialization abuse attempts against any AI framework endpoint. Write a Sigma rule targeting Sysmon Event ID 1 where ParentImage matches your Python interpreter path and CommandLine contains `sh -c`, `bash -c`, or `os.system` to generalize detection across similar RCE chains in AI orchestration tools. Schedule quarterly `pip-audit` and `npm audit` runs for all Python/Node AI workloads as a compensating patch-cadence control.

Evidence: Preserve for the post-incident record: (1) the pre-patch-requirements.txt and pre-patch-packages.txt snapshots captured during eradication, serving as the definitive record of the vulnerable software configuration at time of discovery; (2) the full timeline of HTTP requests to get_state_history() endpoints reconstructed from web server access logs, annotated with the first appearance of SQLi metacharacters or oversized msgpack payloads — this establishes the earliest possible exploitation window for breach notification and regulatory reporting purposes; (3) the Redis keyspace dump and SQLite checkpoint.db copy captured before remediation — these are the primary forensic record of what attacker-controlled deserialization payloads were stored in the checkpoint backend and must be retained per AU-11 record retention requirements; (4) a signed inventory of all credentials rotated during eradication with rotation timestamps, for audit trail purposes under AC-2 account management obligations.

Detection Guidance

Focus detection on the get_state_history() API endpoint in LangGraph deployments. In web server and application logs, look for filter parameter values containing SQL injection patterns: single quotes, double dashes, UNION SELECT, semicolons, or hex-encoded equivalents. Look for requests with unusually large or binary-encoded request bodies that may carry crafted msgpack payloads. At the host level, monitor for unexpected child processes spawned by the LangGraph or Python/Node process (e.g., /bin/sh, cmd.exe, curl, wget) and outbound network connections from the application process to external IPs. Use file integrity monitoring (e.g., NIST SI-7) to detect unauthorized modification of LangGraph application files, configuration files, or the SQLite checkpoint database. Monitor system logs for any new local accounts or privilege changes on hosts running LangGraph. No public IOCs (IPs, domains, hashes) are associated with this vulnerability chain at this time; detections should be behavior-based.

Framework Mappings

MITRE-ATTACK

- **T1565.001** — Stored Data Manipulation
- **T1078** — Valid Accounts
- **T1059** — Command and Scripting Interpreter
- **T1203** — Exploitation for Client Execution
- **T1190** — Exploit Public-Facing Application
- **T1552** — Unsecured Credentials

NIST-800-53R5

- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-7** — Software, Firmware, and Information Integrity
- **SC-7** — Boundary Protection
- **SI-2** — Flaw Remediation
- **CA-8** — Penetration Testing
- **RA-5** — Vulnerability Monitoring and Scanning
- **SI-10** — Information Input Validation

OWASP-TOP10-2021

- **A03:2021** — Injection
- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **7.3** — Perform Automated Operating System Patch Management
- **7.4** — Perform Automated Application Patch Management

ISO-27001-2022

- **A.8.28** — Secure coding
- **A.8.8** — Management of technical vulnerabilities

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1565.001	Stored Data Manipulation	Impact

Technique ID	Technique Name	Tactic
T1078	Valid Accounts	Defense-Evasion
T1059	Command and Scripting Interpreter	Execution
T1203	Exploitation for Client Execution	Execution
T1190	Exploit Public-Facing Application	Initial-Access
T1552	Unsecured Credentials	Credential-Access

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/06/langgraph-flaw-chain-exposes-self...	T3
CVE-2026-28277 Detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-28277	T1
When Your AI Agent's Memory Becomes a Security Liability	https://blog.checkpoint.com/research/when-your-ai-agents-memory-bec...	T3
CVE-2026-2005: PostgreSQL Buffer Overflow Vulnerability	https://www.sentinelone.com/vulnerability-database/cve-2026-2005/	T3
CVE-2026-27794 - TuxCare Vulnerabilities	https://tuxcare.com/cve-tracker/cve/details/cve-2026-27794/	T3
NVD	https://nvd.nist.gov/vuln/detail/CVE-2025-67644, CVE-2026-28277, CV...	T1

Generated 2026-06-12 14:16 UTC by TJS Security Command Center