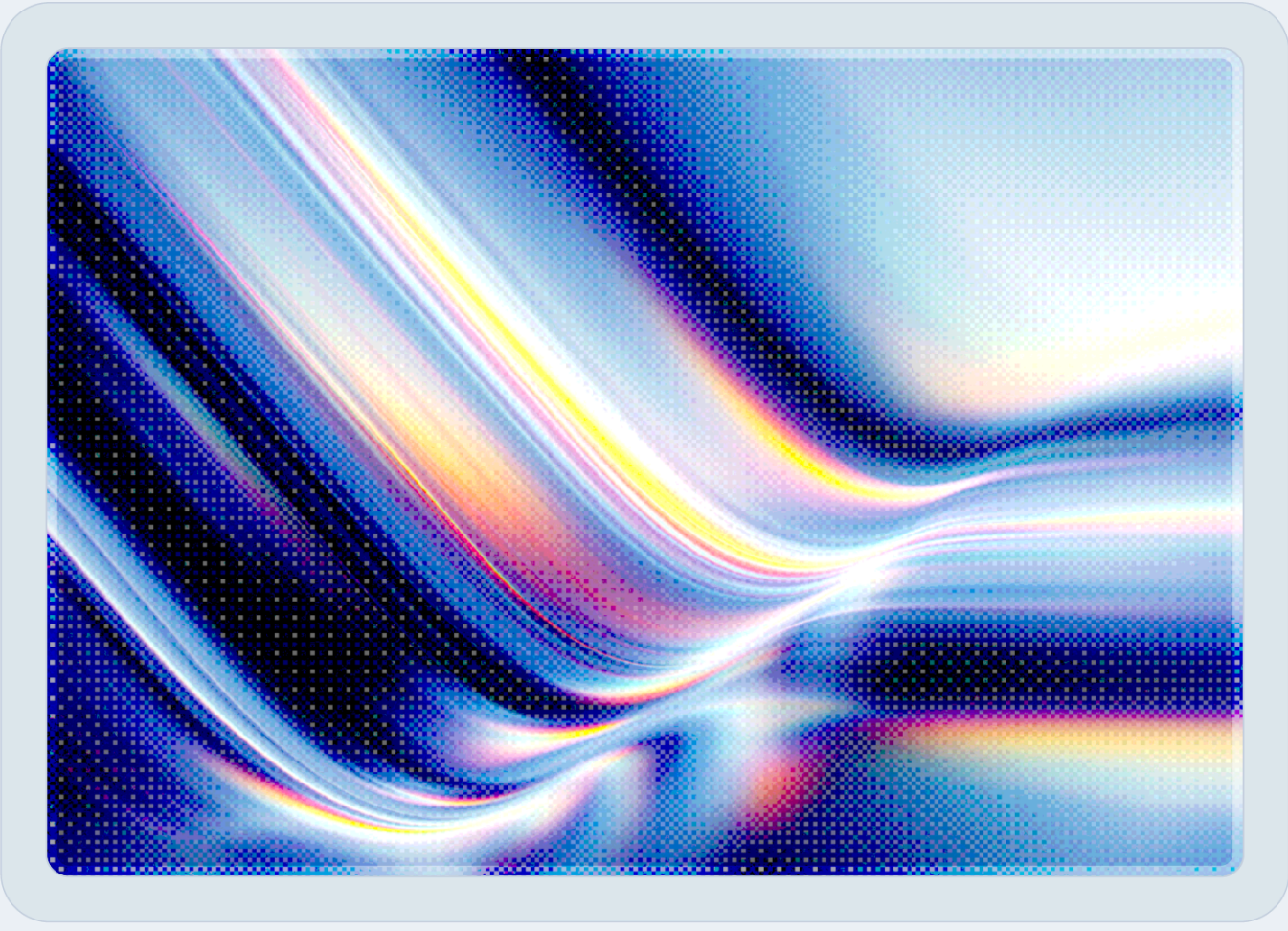
CSAI Foundation | Cloud Security Alliance

# LangGraph Checkpoint RCE Chain in Self-Hosted AI Agents

## Chained SQL Injection and Deserialization Flaws Enable Full Server Takeover

2026-06-15

AI-assisted Rapid Research



CSAI | cloud security alliance®

## Key Takeaways

- Three vulnerabilities discovered by Yarden Porat of Check Point Research and disclosed on June 11, 2026 affect self-hosted LangGraph deployments using SQLite or Redis checkpoint backends; two of the three chain to achieve unauthenticated remote code execution.
- The attack exploits a SQL injection flaw in LangGraph's `_metadata_predicate()` function (CVE-2025-67644, CVSS 7.3) to inject a malicious payload into a checkpoint row, which the framework then deserializes using an unsafe msgpack handler (CVE-2026-28277) that can execute arbitrary Python code.
- The entry point for the chain is the `get_state_history()` API when it accepts user-controlled filter keys without sanitization; applications that expose this endpoint to untrusted callers or pass external input into filter parameters are at highest risk.
- Organizations using the LangSmith managed platform or the PostgreSQL checkpointer are not affected by this chain; the vulnerabilities are specific to the SQLite and Redis persistence backends in self-hosted configurations.
- Patch versions are available: teams should upgrade to `langgraph-checkpoint-sqlite` 3.0.1 or later, `langgraph` 1.0.10 or later, and `@langchain/langgraph-checkpoint-redis` 1.0.2 or later as an immediate priority.
- This is the third RCE-class advisory from the LangGraph checkpoint layer in seven months, a pattern that indicates the persistence layer itself — not just individual implementation defects — is a structural attack surface requiring ongoing architectural scrutiny.

---

## Background

LangGraph is a graph-based AI agent orchestration framework developed by LangChain, Inc. that enables developers to build stateful, multi-step agents capable of sustaining complex workflows across many LLM invocations. Unlike stateless request-response pipelines, LangGraph agents maintain a graph-structured state that evolves as the agent reasons through a task, supports branching and rollback through time-travel queries, and enables human-in-the-loop review at defined checkpoints. This statefulness is the primary feature that distinguishes LangGraph from simpler orchestration approaches

and makes it the framework of choice for production deployments requiring audit trails, resumable workflows, and multi-turn memory [1][7]. By mid-2026, LangGraph has seen significant enterprise adoption for stateful production agentic workloads, with practitioners across regulated industries and complex operational environments increasingly selecting it as a production AI orchestration platform [8].

The mechanism that enables state persistence is the checkpointer. After every node execution in the LangGraph graph — each node corresponding to a step in the agent's reasoning chain — the framework serializes the entire graph state and writes it as a checkpoint to a backing store. These snapshots allow an agent to be suspended and resumed, support branching and rollback via time-travel queries, and provide the audit trail that enterprise compliance requirements demand. LangGraph ships with four checkpointer implementations: an in-memory store for development, and three persistent backends backed by SQLite, PostgreSQL, and Redis respectively [7]. Each backend serializes checkpoint data using the JsonPlusSerializer, which defaults to msgpack encoding with a fallback to an extended JSON format for types that msgpack cannot represent natively.

Self-hosted deployments choosing the SQLite or Redis checkpointers carry a different risk profile than organizations using either the PostgreSQL backend or LangSmith, LangChain's managed cloud platform. SQLite and Redis are commonly chosen for smaller deployments, air-gapped environments, and early-stage production builds where the operational simplicity of a file-based or in-memory database is preferable to a fully managed relational system. Organizations adopting these backends may be operating in configurations that lack the authentication layers and network segmentation often present in managed PostgreSQL deployments — a posture that can increase the practical severity of these vulnerabilities. The vulnerabilities disclosed on June 11, 2026 exploit design properties specific to these two backends — particularly the way the SQLite implementation constructs filter queries — and their practical impact is shaped by the operational posture typical of organizations that select them [1] [2].

---

# Security Analysis

## Three Vulnerabilities, One Kill Chain

Check Point Research identified three vulnerabilities in the LangGraph checkpoint layer and disclosed them simultaneously on June 11, 2026, following a coordinated disclosure process conducted over several months prior to publication. CVE-2025-67644 and CVE-2026-28277 combine into a single attack chain capable of achieving unauthenticated remote code execution on any self-hosted server

running the SQLite checkpointer. CVE-2026-27022 is a parallel injection flaw in the Redis backend that is independently exploitable for data exposure. All three carry patches. The table below summarizes the three CVEs before the discussion that follows treats them individually.

| CVE | CVSS | Component | Impact | Patch Version |
|---|---|---|---|---|
| CVE-2025-67644 | 7.3 | `langgraph-checkpoint-sqlite` | SQL injection; enables fake checkpoint injection | 3.0.1 |
| CVE-2026-28277 | 6.8 | `langgraph` (core) | Unsafe msgpack deserialization; enables RCE | 1.0.10 |
| CVE-2026-27022 | 6.5 | `langgraph-checkpoint-redis` | Redis query injection; enables data exposure | 1.0.2 |

**CVE-2025-67644** is a SQL injection vulnerability in the SQLite checkpoint backend's `_metadata_predicate()` function, present in `langgraph-checkpoint-sqlite` versions prior to 3.0.1. The function constructs SQL WHERE clauses to filter checkpoint rows by metadata fields, using Python f-string formatting to interpolate user-controlled filter dictionary keys directly into the query string without parameterization or escaping. The resulting construction takes the form `json_extract(CAST(metadata AS TEXT), '$.{query_key}') {operator}`, where `query_key` is attacker-supplied text. An attacker who can supply a filter key containing SQL metacharacters can break out of the expected query structure and inject arbitrary SQL, including a `UNION SELECT` statement that fabricates checkpoint rows with attacker-controlled content in every column, including the checkpoint data blob [1][3].

**CVE-2026-28277** is an unsafe deserialization vulnerability in the core `langgraph` checkpoint processing path, affecting versions prior to 1.0.10. When the application retrieves checkpoint rows from the backing store, it deserializes checkpoint data using a custom msgpack extension handler. That handler reconstructs Python objects by evaluating `getattr(importlib.import_module(tup[0]), tup[1])(tup[2])` — importing a module named in the payload, retrieving an attribute from it, and calling that attribute with a payload-

supplied argument. An attacker who controls the contents of a checkpoint row can supply a payload that causes this handler to invoke `os.system()` or a comparable function, achieving arbitrary code execution in the context of the server process [1].

The two vulnerabilities combine into a complete kill chain. CVE-2025-67644 provides write access to the checkpoint store by injecting a fabricated row via SQL UNION injection — without requiring any direct write permissions to the database. CVE-2026-28277 provides the execution trigger when the application subsequently reads back that row and deserializes it. Neither vulnerability is individually sufficient to achieve code execution from an unauthenticated position; chained, they require only that the application expose the `get_state_history()` function to a caller who can supply the filter parameter [1][2]. The workflow of the chain is as follows: the attacker first crafts a msgpack payload encoding a call to an arbitrary Python function; they then submit a filter key containing a UNION SELECT statement that places this payload in a checkpoint data column; the application processes the query result and deserializes the attacker-controlled blob; and the unsafe handler executes the attacker's specified function with their specified argument.

**CVE-2026-27022** affects `@langchain/langgraph-checkpoint-redis` versions prior to 1.0.2. It is a parallel injection flaw in the Redis backend's filter query construction, enabling attackers to manipulate RediSearch query strings in the same structural fashion as CVE-2025-67644 manipulates SQLite queries — through direct interpolation of user-controlled filter keys without sanitization. The Redis variant does not by itself enable the msgpack deserialization chain described above, but it allows an attacker to bypass access controls and retrieve checkpoint data belonging to other users or threads in multi-tenant deployments. It is independently exploitable for cross-tenant data exposure and should not be treated as lower priority simply because it lacks the RCE chain of the SQLite variant [1][2].

## Scope and At-Risk Configurations

The practical scope of exposure depends on deployment architecture. The full RCE chain is exploitable only when three conditions hold simultaneously: the SQLite checkpointer is in use, the `get_state_history()` endpoint or its underlying filter logic is reachable by attacker-controlled input, and the deployed version of `langgraph-checkpoint-sqlite` predates the 3.0.1 patch [1]. Organizations using the LangSmith managed platform are not affected, because checkpoint storage is handled by LangChain's infrastructure rather than a locally managed SQLite instance. Organizations using the PostgreSQL checkpointer are not affected by the SQLite injection chain. All deployments — including those using PostgreSQL — should update the core `langgraph` package to 1.0.10 or later, as the patching requirement for CVE-2026-28277 applies regardless of checkpointer backend.

The vulnerability researcher notes that the attack surface extends beyond explicitly public-facing deployments. Internal applications that pass user-supplied workflow metadata — such as task descriptions, session identifiers, or agent configuration parameters — into checkpoint filter queries without sanitization are also at risk, as are multi-tenant deployments where one tenant's input could be routed into queries operating on another tenant's data store [1][3]. From a security assessment standpoint, the critical question is not whether the `get_state_history()` endpoint is publicly documented, but whether any path through the application allows attacker-supplied strings to reach the filter parameter of that function.

This is the third RCE-class advisory disclosed against the LangGraph ecosystem within seven months. CVE-2025-64439, disclosed in November 2025 and documented in a GitHub security advisory, exploited the JsonPlusSerializer's JSON fallback mode — specifically its support for an unrestricted constructor deserialization path that allowed arbitrary module invocation when loading payloads containing a specially crafted `id` key; that vulnerability was patched in `langgraph-checkpoint` 3.0.0 [4]. The March 2026 CSA research note on LangChain and LangGraph also documented CVE-2025-68664 (CVSS 9.3), a serialization injection flaw in `langchain-core` through which prompt injection could escalate into RCE by exploiting the `lc` marker key in the serialization layer [5]. The pattern is consistent at the category level: each advisory involves deserialization of complex Python objects from data that can be influenced by attacker-controlled inputs. The specific implementation paths differ — CVE-2025-64439 and the current chain exploit the checkpoint store persistence layer, while CVE-2025-68664 targeted the langchain-core serialization layer through prompt injection escalation — but the aggregate pattern across seven months remains notable, and it would be premature to treat any individual patch as having resolved the broader structural exposure.

---

# Recommendations

## Immediate Actions

The first priority is patching. Organizations should immediately verify the installed versions of the affected packages in every environment where LangGraph is deployed and upgrade as indicated. The patched versions are `langgraph-checkpoint-sqlite` 3.0.1 or later for CVE-2025-67644, `langgraph` 1.0.10 or later for CVE-2026-28277, and `@langchain/langgraph-checkpoint-redis` 1.0.2 or later for CVE-2026-27022 [1][3]. Because these components are installed as independent packages and may carry separate version pins in application dependency files, teams

should audit all three explicitly rather than assuming that a general `pip install --upgrade` will address them. For version-constraint conflicts in complex dependency graphs, the LangGraph repository's Security Advisories page should be treated as the authoritative source.

For organizations that cannot complete patch deployment immediately, the most effective interim control is to restrict or remove access to the `get_state_history()` endpoint for any caller that cannot be fully trusted. Restricting or removing access to this endpoint for external-facing and multi-tenant services significantly reduces the attack surface even without a patch — though teams should audit all code paths through which attacker-controlled input might reach the underlying filter logic. Organizations should also audit the earlier CVE-2025-64439 advisory to confirm that `langgraph-checkpoint` 3.0.0 or later is deployed, as environments that have deferred prior checkpoint patches may carry multiple unpatched vulnerabilities simultaneously.

## Short-Term Mitigations

After patching, teams should review how filter parameters reach the checkpoint layer across all code paths in their LangGraph applications. User-supplied metadata, conversation session identifiers, agent configuration keys, and any other externally controlled strings that pass through filter arguments to checkpoint queries should be validated or stripped before reaching the checkpointer. This input review applies even in patched environments: the patches correct the vulnerable query construction, but defense-in-depth security requires that applications not rely on framework-layer sanitization for inputs that arrive from untrusted callers.

Network controls provide a meaningful complementary layer. Self-hosted LangGraph servers not intended to accept public traffic should sit behind network-level controls restricting access to known client addresses. Redis instances used as checkpoint backends should require authentication and should not be exposed on addresses accessible to co-hosted services without enforced trust boundaries between them. SQLite database files should be restricted by filesystem permissions to the application's own process user to prevent reads or modifications from other processes on the same host.

## Strategic Considerations

Three RCE-class advisories in seven months from the same component — the LangGraph checkpoint layer — constitute a pattern that security architects should weigh explicitly when selecting or approving AI agent frameworks for production deployment. The root technical cause across all three advisories is consistent at the category level: a persistence layer that deserializes complex Python objects from a

backing store, using deserialization mechanisms that can be guided to execute arbitrary code when checkpoint content is attacker-controlled. Patching each advisory individually closes the known bypass while leaving the fundamental design in place.

Organizations building or operating self-hosted agentic AI systems should undertake a broader review of where Python object deserialization occurs in their agent stack, what data sources feed those deserialization paths, and whether access controls around those sources are commensurate with the code-execution capability they carry. For new deployments, the choice between LangSmith's managed platform and a self-hosted configuration should weigh the persistent attack surface the checkpoint layer introduces in self-hosted environments against the operational rationale for running infrastructure locally. For existing deployments, the advisory pattern is a strong argument for treating all agent management endpoints — including state history, state modification, and any endpoint that reads from or writes to the checkpoint store — as security-critical surfaces requiring the same authentication and authorization discipline applied to administrative APIs.

---

## CSA Resource Alignment

The vulnerabilities described in this research note map directly to threat categories and controls across several CSA frameworks. Security teams operating within those frameworks will find existing guidance applicable to both the response to this specific incident and the broader structural question it raises about stateful agentic AI deployments.

The CSA MAESTRO framework for agentic AI threat modeling identifies cross-layer attack paths as a primary concern in complex multi-agent systems [6]. The LangGraph vulnerability chain maps closely to the MAESTRO Layer 3 / Layer 4 cross-layer attack pattern. The injection entry point resides in the agent framework's query logic (MAESTRO Layer 3: Agent Frameworks, which covers orchestration and decision-making between agents), while the code execution impact is realized in the persistence and infrastructure layer that backs the checkpoint store (MAESTRO Layer 4: Deployment and Infrastructure, which covers containerized hosting and database persistence). MAESTRO's emphasis on modeling threats across layer boundaries — rather than analyzing each layer independently — is precisely what this attack chain requires and what a single-layer security review would miss. Organizations that have completed a MAESTRO threat model for their LangGraph deployments should revisit their Layer 3-to-Layer 4 threat paths in light of these advisories and confirm that checkpoint persistence is represented in the model with appropriate threat actors and trust assumptions.

The CSA AI Controls Matrix (AICM) provides controls applicable to this class of risk through its Application and Interface Security domain, which addresses input validation and injection resistance in AI application layers, and its Data Security and Privacy domain, which governs the integrity of data persisted by AI systems [9]. The design failure in CVE-2025-67644 — user-controlled input interpolated into a SQL query without parameterization — is an injection vulnerability of precisely the type that AICM application security controls are written to prevent. The deserialization RCE in CVE-2026-28277 falls within AICM controls addressing the integrity of AI system data stores and the validation of data loaded by AI inference paths. Organizations performing AICM gap assessments should include checkpoint query construction and deserialization safety in their reviews of any LangGraph or comparable stateful agent deployment.

The CSA Agentic AI Red Teaming Guide, published in 2025, includes the checkpoint and state persistence layer among the surfaces that red teams should evaluate in agentic AI deployments [10]. This advisory provides additional justification for including checkpoint injection tests — specifically filter parameter fuzzing and crafted deserialization payload attempts — as a standard component of red team exercises against LangGraph-based systems, and for extending that testing to any agent framework that uses a comparable pattern of serializing complex Python objects to an accessible backing store. The recurrence of checkpoint-layer vulnerabilities strengthens the case for making this a standing test category rather than an advisory-triggered one-time exercise.

# References


[1] Check Point Research. "From SQLi to RCE: Exploiting LangGraph's Checkpointer." Check Point Research Blog, June 2026.

[2] The Hacker News. "LangGraph Flaw Chain Exposes Self-Hosted AI Agents to Remote Code Execution." The Hacker News, June 2026.

[3] CybersecurityNews. "Critical Vulnerability Chain in LangGraph Allows Attackers to Gain Full Server Control." CybersecurityNews, June 2026.

[4] GitHub Security Advisory. "RCE in 'json' mode of JsonPlusSerializer." langchain-ai/langgraph, November 2025.

[5] Cloud Security Alliance AI Safety Initiative. "LangChain and LangGraph: Critical Vulnerabilities in AI Orchestration." CSA Labs, March 2026.

[6] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." CSA Blog, February 2025.

[7] LangChain. "Persistence – LangGraph Concepts." LangChain Documentation, 2026.

[8] Gheware DevOps. "LangGraph Multi-Agent Orchestration 2026: Complete Enterprise Guide." Gheware DevOps Blog, 2026.

[9] Cloud Security Alliance. "AI Controls Matrix (AICM)." CSA, 2025.

[10] Cloud Security Alliance. "Agentic AI Red Teaming Guide." CSA, 2025.