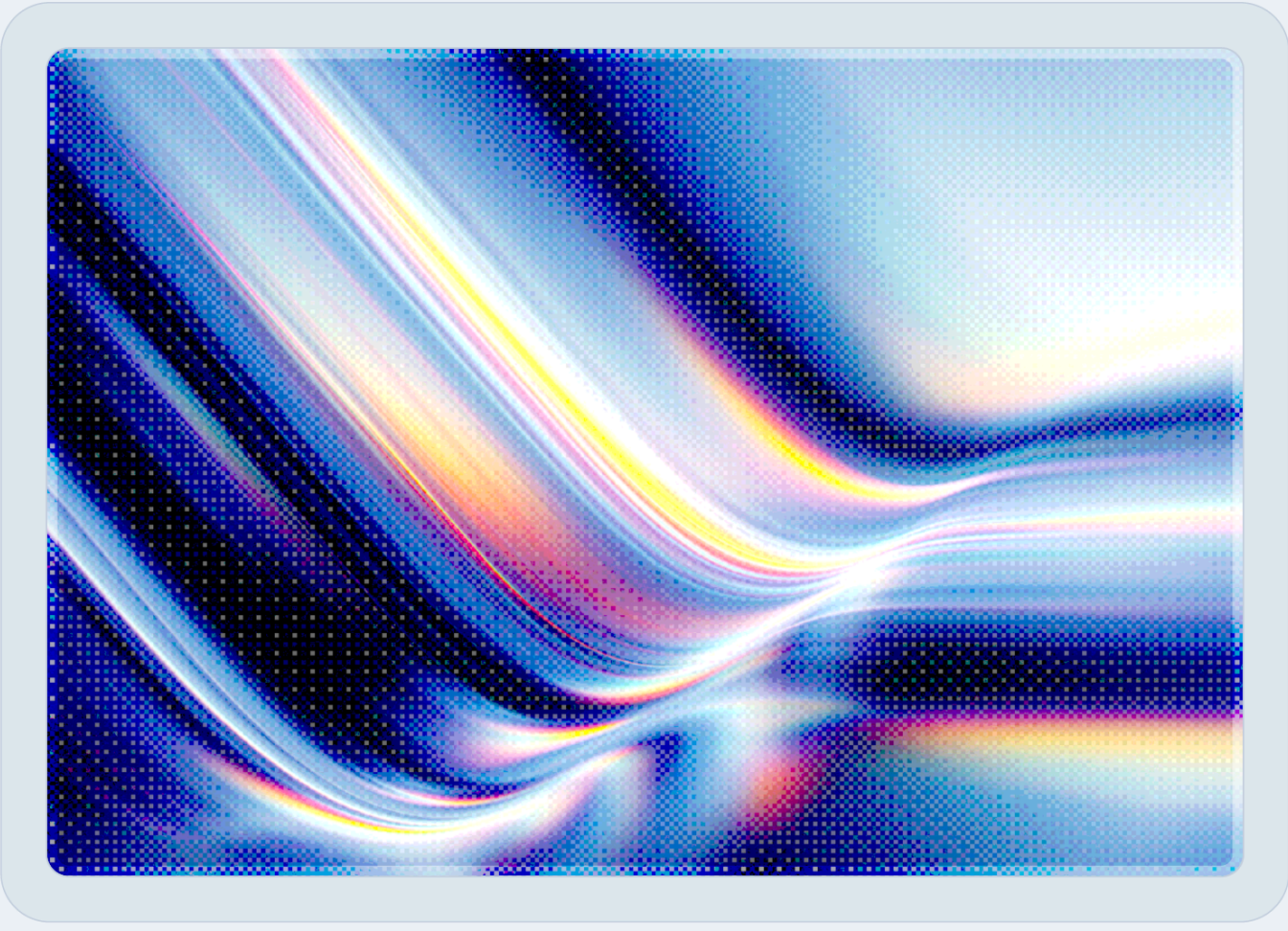
CSAI Foundation | Cloud Security Alliance

LLM Heist: Abusing LiteLLM Callback Hooks Post-Compromise

Traffic Interception, Key Theft, and Tool-Call Injection via a Legitimate Extension Point

2026-08-05

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

Security researcher Wunderwuzzi disclosed a technique, dubbed "LLM Heist," showing that an attacker who already holds administrative access to a LiteLLM AI gateway can weaponize the proxy's own management API and callback system to intercept traffic, steal backend provider credentials, and inject forged tool calls into agent responses — all without deploying malware or triggering a new code-execution vulnerability [1]. The technique reroutes victim traffic to an attacker-controlled LiteLLM instance by changing the `api_base` and `use_litellm_proxy` settings through the legitimate `/model/update` endpoint, after which the attacker's instance can log resolved provider API keys and use built-in callback hooks to rewrite responses in transit [1]. Because the manipulation happens after the LLM has already generated its output, it bypasses prompt-level defenses entirely and can forge tool calls that downstream agents, such as coding assistants with file or shell access, will execute as if they came from the legitimate model [1]. The disclosure lands amid a broader pattern of 2026 LiteLLM security incidents — including a PyPI supply-chain compromise and multiple CVE chains granting remote code execution — that CSA has tracked as evidence that centralized AI gateways have become high-value targets whose compromise can, absent compensating controls, cascade across the provider credentials and downstream agents they serve [2].

Background

LiteLLM is an open-source AI gateway and proxy that unifies access to well over 100 LLM providers behind a single API, a design that lets enterprises apply consistent governance, cost tracking, and routing logic across OpenAI, Anthropic, Google Gemini, AWS Bedrock, and dozens of other backends. That same centralization is what makes the gateway attractive to attackers: a single LiteLLM deployment typically holds the provider API keys for every model an organization uses, along with logs of the prompts and completions flowing through it. Wunderwuzzi's research, published on the Embrace The Red blog in August 2026, examined what an attacker can do with that centralization once they already hold — or have obtained — proxy-admin or master-key access, rather than focusing on a new way to obtain that access in the first place [1].

CSA assesses the starting assumption in the research as realistic rather than hypothetical, given the exposure history of the preceding months. Proxy-admin credentials and the `LITELLM_MASTER_KEY` are plausibly exposed through routes common to other self-hosted secrets — leaked `.env` files and weak or default authentication — in addition to a documented chain of unrelated CVEs. CSA's own research note on CVE-2026-42271 documented a command-injection flaw in LiteLLM's MCP test endpoints that, when combined with the Starlette "BadHost" host-header bypass (CVE-2026-48710), produced a fully unauthenticated remote-code-execution path with a combined CVSS score of 10.0; the U.S. Cybersecurity and Infrastructure Security Agency (CISA) added that chain to its Known Exploited Vulnerabilities catalog after confirming active exploitation [2][3]. Separately, Obsidian Security disclosed a three-CVE privilege-escalation chain (CVE-2026-47101, CVE-2026-47102, and CVE-2026-40217) that lets a default low-privilege LiteLLM user promote themselves to `proxy_admin` and ultimately achieve code execution through a custom-code guardrail [4]. A March 2026 supply-chain compromise, in which threat actor TeamPCP published backdoored LiteLLM packages to PyPI after a compromised CI/CD scanner action in the project's pipeline harvested a maintainer's publish token, further illustrates how administrative control of a gateway instance can arrive through channels that have nothing to do with the gateway's own code [5]. LLM Heist assumes any one of these paths has already succeeded and asks what an attacker does next.

Security Analysis

For analysis purposes, the attack can be grouped into three stages once admin-level access is in hand. First, the attacker uses LiteLLM's legitimate `/model/update` management API to alter routing configuration for a target model: setting `api_base` to point at an attacker-controlled LiteLLM instance and enabling `use_litellm_proxy`, so that requests the victim organization believes are going directly to a provider like OpenAI or Anthropic are instead relayed through infrastructure the attacker controls [1]. Because this change is made through a documented, intended API rather than an exploit, it produces no crash, no anomalous process spawn, and no client-side configuration change — the gateway is doing exactly what an administrator told it to do, which is why detection depends on monitoring the management API itself rather than traditional endpoint signals.

Second, once traffic is rerouted, the victim's outbound requests to the attacker's relay carry the resolved backend provider key in the Authorization header, because LiteLLM must present valid credentials to the real provider on the victim's behalf. The attacker's instance can log or forward those credentials, and depending on configuration such as `store_prompts_in_spend_logs`, it can also retain full prompt and response content for every request that transits it. This gives the attacker not only the ability to make unauthorized inference calls billed to the victim's account but also a live feed of proprietary

prompts, retrieved context, and business data flowing through the gateway — a form of exposure distinct from, and arguably more durable than, a one-time key leak, since the attacker retains standing visibility for as long as the rerouted configuration persists.

Third, and of particular concern for organizations running agentic systems, the attacker's relay can invoke LiteLLM's own callback hook system — specifically `async_post_call_success_hook` and `async_post_call_streaming_iterator_hook` — to modify the model's response before it reaches the victim's application [1]. CSA's review of LiteLLM's configuration model found that callback hooks are a built-in extension point loaded from configuration files and are not surfaced in LiteLLM's admin UI, so they operate largely outside the visibility of operators who assume the gateway is a passive pass-through. An attacker in control of these hooks can inject arbitrary text into a response or, more consequentially, forge a tool call that was never produced by the underlying model. If the requesting client is an autonomous agent — a coding assistant, a customer-service bot with function-calling access, or any system that acts on tool-call output without independent verification — that forged tool call executes with whatever privileges the agent holds. This bypass differs from prompt injection in a specific way: the manipulation occurs after inference, so guardrails and content filters designed to inspect prompts or moderate model-generated text before it is finalized have nothing to inspect, since from the agent's perspective the forged output simply is the model's answer. The source further documents that a sophisticated attacker reverts the rerouted configuration back to its original state once the operation concludes, which removes the most visible artifact of the compromise and means a point-in-time review of current `api_base` settings may show nothing amiss even after the technique has already been used.

The research frames this as a control-plane risk rather than a code-execution bug: nothing in the attack chain requires a new vulnerability in LiteLLM once administrative access exists, and the same pattern would apply in principle to other AI gateway products that expose comparably powerful runtime configuration and callback APIs to anyone holding admin credentials. That generality is consistent with what CSA's research has already observed across the 2026 LiteLLM incident history — the PyPI backdoor, the command-injection-to-RCE chain, and the privilege-escalation-to-RCE chain all converge on the same outcome, a centralized store of provider credentials and a request path with the standing ability to rewrite what an agent sees [2][4][5]. LLM Heist demonstrates that once that convergence point is reached by any means, the gateway's own legitimate feature set is sufficient to complete the compromise.

Recommendations

Immediate Actions

Organizations running LiteLLM or comparable AI gateways in production should treat any change to routing-critical settings as a security event requiring investigation, not a routine operational change. Configure alerting on modifications to `api_base`, `use_litellm_proxy`, and any custom callback registration made through the management API, and route those alerts to a security team rather than only an operations dashboard. Audit current model configurations now for unexpected `api_base` values or unfamiliar proxy chaining, since an attacker who has already executed this technique would have left exactly that artifact behind. Note, however, that a sophisticated attacker may revert `api_base` and proxy settings to their original values once the operation concludes, so a clean present-state configuration does not rule out prior compromise; historical audit-log review, not just point-in-time inspection, is necessary to detect this technique after the fact. Confirm the gateway is patched against the known CVE chains — LiteLLM 1.83.7 or later and Starlette 1.0.1 or later close the BadHost-chained RCE path, and current guardrail and key-management endpoints should be checked against the Obsidian Security advisory for the privilege-escalation chain [3][4].

Short-Term Mitigations

Rotate all provider API keys held by the gateway and store them in an external secrets manager rather than in plaintext configuration, so that a compromised gateway instance cannot itself be the sole record of a credential's value. Restrict provider keys, where the provider supports it, to be usable only from the gateway's approved egress IP addresses, which limits the value of a stolen key even if traffic rerouting succeeds. Enforce network egress restrictions from the gateway host so that outbound connections are limited to known, allow-listed provider endpoints, making an attacker-controlled relay harder to reach in the first place. Forward comprehensive audit logs — including every management API call — to a SIEM with retention sufficient to reconstruct a rerouting timeline, and monitor provider billing dashboards for usage patterns inconsistent with known application traffic, since unauthorized inference volume is one of the more durable signals this technique leaves behind.

Strategic Considerations

Treat the AI gateway as a Tier 0 asset in identity and access architecture: administrative access to it should require the same scrutiny — hardware-backed multi-factor authentication, just-in-time elevation, and separation of duties — applied to domain controllers or secrets vaults, not the lighter controls often

applied to application configuration consoles. Organizations should evaluate whether an internet-exposed management plane is necessary at all; where it is not, the administrative API should sit behind a private network boundary reachable only from a bastion or VPN. Longer term, the underlying gap is that LLM responses are not currently signed or otherwise bound to the inference call that produced them, which is what allows a mid-stream relay to substitute content undetectably; organizations building high-stakes agentic pipelines should track emerging proposals for response signing or provenance attestation and, in the interim, apply independent verification to any tool call before granting it execution privileges, rather than trusting gateway output implicitly.

CSA Resource Alignment

This finding connects most directly to CSA's existing research note on active exploitation of LiteLLM through MCP injection, which documented CVE-2026-42271 and its chaining with the Starlette BadHost bypass to achieve unauthenticated remote code execution against the same gateway product [2]. That prior research established the entry-point risk — how an attacker reaches administrative control of a LiteLLM instance in the first place — while the LLM Heist technique analyzed here describes what that access enables once obtained: traffic interception and forged tool-call injection through the gateway's own callback system. Read together, the two research notes describe a complete kill chain from initial compromise of the gateway to durable, stealthy manipulation of every downstream agent it serves.

Because the tool-call forgery in this technique targets agents that act on model output without independent verification, it also falls within the threat surface CSA's MAESTRO framework for agentic AI threat modeling was built to address [8], particularly at the layers concerned with tool invocation and agent trust boundaries. Organizations mapping this risk to formal controls should reference the AI Controls Matrix (AICM) v1.1, whose identity and access management domain speaks directly to the privileged-access and administrative-boundary failures that make this attack possible, and whose broader control set covers the secrets-management and monitoring gaps identified above [6]. Finally, because the root exposure here is standing administrative trust in a network-reachable control plane, CSA's Zero Trust Guiding Principles — particularly the principles that access is a deliberate act and that breaches should be assumed — provide the architectural frame for treating the gateway's management API as a boundary requiring continuous verification rather than one-time authentication [7].

References

[1] Wunderwuzzi. "LLM Heist: Hijacking LiteLLM for Traffic Interception, Key Theft, and Tool-Call Injection." Embrace The Red, August 3, 2026.

[2] Cloud Security Alliance. "LiteLLM AI Gateway: Active Exploitation via MCP Injection." CSA AI Safety Initiative, June 2026.

[3] The Hacker News. "LiteLLM Flaw CVE-2026-42271 Exploited in the Wild, Chains to Unauthenticated RCE." The Hacker News, June 2026.

[4] Obsidian Security. "Breaking LiteLLM: From Low-Privilege User to Admin and RCE." Obsidian Security Blog, 2026.

[5] Trend Micro. "Your AI Gateway Was a Backdoor: Inside the LiteLLM Supply Chain Compromise." Trend Micro Research, 2026.

[6] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." CSA, June 2026.

[7] Cloud Security Alliance. "Zero Trust Guiding Principles." CSA, July 2023.

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

