Una cuenta predeterminada de bajos privilegios en un proxy LiteLLM puede escalar hasta obtener permisos completos de administrador y ejecutar código en el servidor mediante el encadenamiento de tres vulnerabilidades, según revelaron investigadores de Obsidian Security.

LiteLLM es una pasarela de inteligencia artificial de código abierto ampliamente utilizada que centraliza solicitudes hacia más de 100 proveedores de modelos a través de una única interfaz compatible con OpenAI.

La toma de control de un servidor de este tipo expone todas las claves de acceso de los proveedores almacenadas en él, los secretos utilizados para descifrar credenciales guardadas y cada solicitud y respuesta que transita por la plataforma.

Obsidian calificó la cadena completa de explotación con una puntuación CVSS de 9.9, situándola dentro de la categoría Crítica. BerriAI, responsable del proyecto, incorporó todas las correcciones en LiteLLM v1.83.14-stable, versión que GitHub registra como publicada el 2 de mayo. Actualizar a esta versión o a una posterior elimina la cadena compuesta por tres CVE.

Las tres vulnerabilidades

El primer eslabón corresponde a *CVE-2026-47101*, una vulnerabilidad de evasión de controles de autorización. Cuando un usuario estándar (*internal_user*) genera una clave API virtual, LiteLLM almacena el campo *allowed_routes* proporcionado por el usuario sin verificar si coincide con los permisos asignados a su rol.

En teoría, este parámetro debería limitar las acciones permitidas a la clave. Sin embargo, el proxy también lo interpreta como un mecanismo alternativo de concesión de permisos. Como resultado, un usuario sin privilegios administrativos puede crear una clave con *allowed_routes: [«/«]*, un comodín que habilita el acceso a todas las rutas, incluidas aquellas reservadas para administradores. Esta misma falta de validación también afectaba otros endpoints relacionados con la gestión de claves, motivo por el cual fueron necesarias tres

solicitudes de extracción (*pull requests*) para corregir el problema.

Una vez superado el control de acceso a las rutas, los servicios protegidos quedan expuestos. Varios de estos componentes asumen que la validación ya fue realizada previamente, lo que habilita dos vectores adicionales de ataque.

Uno de ellos es *CVE-2026-47102*, una vulnerabilidad de escalada de privilegios. El endpoint */user/update* permite que un usuario modifique su propio registro, pero no restringe qué campos pueden alterarse. En consecuencia, una actualización que establezca *user_role: «proxy_admin»* es aceptada y almacenada, promoviendo al usuario a administrador completo del proxy. Un *org_admin* puede acceder legítimamente a esta función mediante el flujo previsto por la aplicación; en cambio, un *internal_user* requiere previamente explotar *CVE-2026-47101*.

VulnCheck, entidad responsable de asignar el CVE, otorgó a esta vulnerabilidad una puntuación de 8.7 bajo CVSS 4.0 y de 8.8 bajo CVSS 3.1.

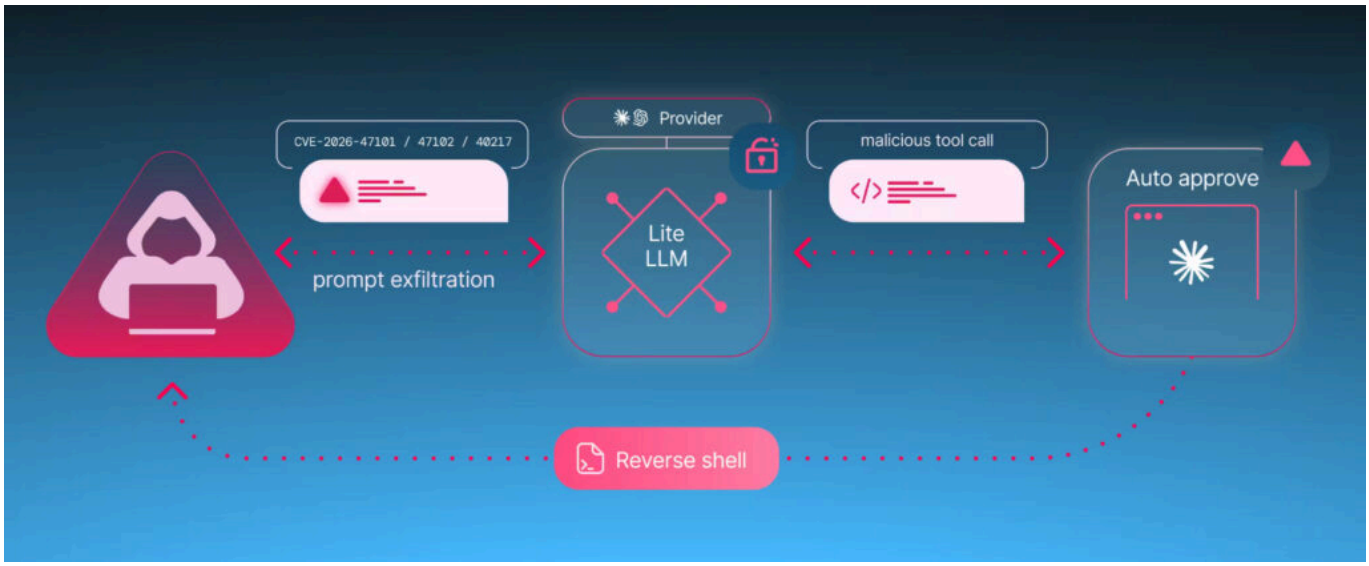
La segunda ruta corresponde a *CVE-2026-40217*, una vulnerabilidad de escape de sandbox presente en el sistema *Custom Code Guardrail*, encargado de compilar y ejecutar código Python suministrado por administradores. Los endpoints de producción ejecutaban dicho código mediante *exec()* sin aplicar filtros sobre el código fuente. Cuando *exec()* recibe un diccionario *globals* sin la clave *builtins*, Python inserta automáticamente el módulo completo de funciones integradas, proporcionando acceso a capacidades como *import*, *open* y *eval*. En estas condiciones, una carga maliciosa simple que invoque *os.system* basta para obtener una shell inversa.

Por otra parte, una ruta independiente asociada al endpoint */guardrails/test_custom_code*, identificada de forma separada por X41 D-Sec, logró evadir una lista de bloqueo basada en expresiones regulares mediante la modificación dinámica de bytecode durante la ejecución. Ambos escenarios terminaban permitiendo la ejecución remota de código en el servidor.

Lo que obtiene un atacante

LiteLLM suele ocupar una posición central dentro de la infraestructura, por lo que el impacto potencial es considerable. Una explotación completa permite acceder a la clave maestra, a la clave de sal utilizada para descifrar credenciales almacenadas y a la URL de conexión de la base de datos. También expone todas las claves configuradas para proveedores como OpenAI, Anthropic, Gemini, Bedrock, Azure y otros.

Las claves almacenadas en archivos de configuración o variables de entorno se encuentran en texto plano. Las que residen en la base de datos están cifradas, pero pueden recuperarse utilizando la clave de sal. Además, toda la información que atraviesa la pasarela —prompts y respuestas— queda accesible. En entornos reales, esto suele incluir información personal identificable (PII), código fuente, tickets internos y credenciales copiadas por usuarios.



Si el proxy también funciona como pasarela de *Model Context Protocol (MCP)* o como puerta de enlace para agentes, los tokens OAuth y las credenciales asociadas a herramientas externas también quedan comprometidos.

Sin embargo, el riesgo más significativo no radica únicamente en la lectura de información, sino en la capacidad de modificarla. Al situarse entre el agente de IA y el modelo, una intrusión permite alterar las respuestas antes de que lleguen a su destino.

Obsidian demostró este escenario utilizando Claude Code a través de un proxy comprometido. No se trata de un caso de *prompt injection*. En lugar de inducir al modelo a comportarse incorrectamente, el atacante aprovecha el mecanismo interno de *callbacks* de LiteLLM, un punto de extensión que se ejecuta con cada solicitud y que no aparece en la interfaz administrativa. Mediante este mecanismo, la respuesta legítima del modelo se sustituye por una llamada falsa a una herramienta y se modifica el contexto de validación para que la acción parezca autorizada.

En la demostración, el desarrollador únicamente escribe la palabra “hello”, y el atacante consigue abrir una shell inversa en la máquina de la víctima.

Independientemente de esta cadena de vulnerabilidades, LiteLLM ya proporciona a un *proxy_admin* una vía legítima para ejecutar código. Su compatibilidad con MCP permite registrar servidores MCP basados en *stdio* que el proxy inicia como procesos locales. Esto constituye una decisión de diseño y no un fallo de seguridad, por lo que las correcciones aplicadas no modifican este comportamiento. En la práctica, obtener privilegios de administrador equivale a obtener capacidad de ejecución de código.

Obsidian logró reproducir una shell inversa mediante este mecanismo en la versión v1.88.0. Además, una vulnerabilidad independiente dentro de la misma infraestructura MCP basada en *stdio*, identificada como *CVE-2026-42271*, permitía crear subprocesos a través de los endpoints de vista previa MCP de LiteLLM. Esta vulnerabilidad fue explotada activamente y añadida al catálogo KEV de CISA a principios de este mes.

Un año complicado para LiteLLM

Este incidente no representa el primer problema grave de LiteLLM durante el año. En marzo, un compromiso de la cadena de suministro introdujo una puerta trasera en dos versiones

distribuidas a través de PyPI. Posteriormente, en abril, una vulnerabilidad crítica de inyección SQL comenzó a ser explotada apenas 36 horas después de hacerse pública.

Obsidian aclara que la cadena descrita en esta ocasión corresponde a una vulnerabilidad divulgada acompañada de una prueba funcional de explotación, y no a evidencia de ataques observados en entornos reales. No obstante, la posición estratégica que ocupa el proxy dentro de las arquitecturas de IA lo convierte en un objetivo especialmente atractivo.

Recomendaciones

La medida más urgente consiste en actualizar a la versión v1.83.14-stable o posterior, primera edición que incorpora todas las correcciones necesarias.

Posteriormente, se recomienda realizar una auditoría exhaustiva. Deben revisarse todas las cuentas con permisos *proxy_admin* y considerar dicho rol como equivalente a acceso de nivel sistema. También es necesario inspeccionar todas las configuraciones de *Custom Code Guardrail* presentes en la instancia.

Conviene verificar especialmente los *callbacks* definidos en *config.yaml* dentro de *litellm_settings.callbacks*, ya que estos no aparecen en la consola de administración y constituyen un lugar ideal para ocultar persistencia tras una ejecución remota de código. Asimismo, debe comprobarse la integridad de todo el código desplegado y no únicamente de los archivos de configuración.

Si existe sospecha de compromiso, resulta imprescindible rotar todas las claves de proveedores, credenciales de bases de datos y cualquier token MCP almacenado.

Un proxy comprometido no solo filtra información sensible. También ocupa una posición privilegiada entre el agente y el modelo, lo que le permite falsificar respuestas y alterar las acciones ejecutadas posteriormente por el sistema. La cadena de ataque descrita se sustenta en una serie de supuestos erróneos de confianza: el control de rutas confiaba en datos proporcionados por el usuario, los componentes internos confiaban en ese control de rutas y,

finalmente, nadie verificaba realmente que dichas suposiciones fueran válidas.