CSAI Foundation

Cloud Security Alliance AI Safety Initiative

SGLang CVE-2026-5760: RCE via Poisoned GGUF Model Files

Critical Template Injection in LLM Serving Infrastructure

Unofficial AI-assisted Research

2026-04-22

Key Takeaways

- CVE-2026-5760 carries a CVSS score of 9.8 (Critical) and enables unauthenticated remote code execution on SGLang inference servers through malicious GGUF model files embedded with a Jinja2 server-side template injection (SSTI) payload [1].
- The vulnerability is exploitable without any network authentication: an attacker distributes a trojanized model through a public repository such as Hugging Face, and code executes the moment a victim's server processes a request to the `/v1/rerank` endpoint [2].
- No official patch has been released. SGLang's maintainers did not respond to coordinated disclosure efforts by CERT/CC or CISA outreach prior to publication [1].
- This vulnerability is one of three Critical-severity CVEs (CVSS 9.8) disclosed in SGLang since early 2026, alongside a fourth High-severity finding (CVSS 7.8), indicating systemic security debt in one of the most widely deployed LLM serving frameworks [3].
- Organizations running SGLang should immediately implement network isolation for inference endpoints, establish verified model provenance controls, and replace the vulnerable `jinja2.Environment()` call with `ImmutableSandboxedEnvironment` pending an upstream patch [1][3].

Background

SGLang and Its Role in AI Infrastructure

SGLang (Structured Generation Language) is an open-source, high-performance serving framework for large language models and multimodal AI systems, developed and maintained by the LMSYS organization and released under the Apache 2.0 license [7]. The framework is designed to deliver low-latency, high-throughput LLM inference across configurations ranging from a single GPU to large distributed clusters, and it exposes an OpenAI-compatible API that makes it a drop-in serving backend for many AI applications. As of early 2026, SGLang reportedly runs on over 400,000 GPUs worldwide and generates trillions of tokens daily, with documented production deployments at organizations including xAI, NVIDIA, AMD, LinkedIn, and Cursor [7].

SGLang's breadth of deployment makes it a plausible high-value target for adversaries. The framework supports a wide range of frontier and open-weight models—including Qwen, DeepSeek, Mistral, and Skywork families—and its reranking capability, introduced to support retrieval-augmented generation (RAG) workflows, is the component at the center of CVE-2026-5760 [2]. Because SGLang presents an OpenAI-compatible endpoint, it often sits directly in the critical inference path for production AI applications, meaning a successful compromise of the serving layer can expose not just the host system, but also any connected data stores, internal APIs, or downstream services the inference server is authorized to reach.

The GGUF Model Format and the Metadata Trust Problem

GGUF (GPT-Generated Unified Format) is a binary model file format designed for efficient storage and portability of quantized language models. It is the standard format used by llama.cpp and related tooling, and it is widely distributed through Hugging Face, where as of early 2026 searches of the model hub return hundreds of thousands of GGUF format files [5]. GGUF files store not only model weights but also rich metadata—including tokenizer configuration, generation parameters, and chat templates. The `tokenizer.chat_template` field in particular stores a Jinja2 template string that defines how a model structures conversational prompts, and this field is the vector exploited by CVE-2026-5760.

The security problem with GGUF metadata is a version of the broader trusted-artifact problem in AI supply chains: practitioners often download and load models from public repositories with limited scrutiny of embedded metadata—a pattern analogous to the implicit trust extended to unvetted software packages. Just as a malicious Python package can execute code at install time, a malicious GGUF file can deliver an attack payload at inference time—specifically at the moment the serving framework renders the embedded chat template. Unlike weight-level backdoors that require specific input patterns to activate, this attack executes at the infrastructure level and does not depend on model behavior at all.

Security Analysis

Root Cause: Unsandboxed Jinja2 Template Rendering

The technical root of CVE-2026-5760 is straightforward: SGLang's `getjinjaenv()` function instantiates a standard `jinja2.Environment()` object to render chat templates, rather than the security-hardened `ImmutableSandboxedEnvironment` variant that Jinja2's own documentation

recommends for rendering untrusted content [1]. This distinction is consequential. The default Jinja2 environment permits templates to access Python's object model directly, enabling a template to traverse class hierarchies, locate built-in functions such as `__import__`, and invoke arbitrary operating system commands. Jinja2's sandboxed environment was specifically designed to close these escape paths, but its use is not enforced by default and requires explicit opt-in by the developer.

When SGLang loads a model file, the `tokenizer.chat_template` value from that file is passed directly into the unsandboxed environment as a template to be rendered. The framework includes logic to detect models that use the Qwen3 reranker format—logic that is triggered when a template contains the trigger phrase "The answer can only be 'yes' or 'no'"—and this detection pathway is what an attacker weaponizes [2]. By embedding this trigger phrase alongside a Jinja2 SSTI payload, the attacker ensures that when any request reaches the `/v1/rerank` endpoint, the template renders through the vulnerable code path and executes arbitrary Python on the server. Proof-of-concept code referenced in the CERT/CC advisory targets SGLang version 0.5.9 specifically [1].

The Attack Chain

The exploitation sequence for CVE-2026-5760 follows a pattern that is well-suited for supply chain delivery. An adversary first creates or modifies a GGUF model file so that its `tokenizer.chat_template` field contains a crafted Jinja2 payload that includes the Qwen3 reranker trigger phrase. This file is then published to a model-sharing repository, potentially under a convincing or impersonated model name. Because GGUF files contain both valid model weights and the malicious metadata, they pass casual inspection and basic file-format validation.

A developer or MLOps engineer operating an SGLang-based inference service downloads the model through an automated pipeline or manual pull. The model loads without error. At this point, the payload is dormant: no code has executed yet. The attack activates only when the first request reaches the `/v1/rerank` endpoint, at which point SGLang renders the embedded chat template. The Jinja2 SSTI payload escapes the template context using well-documented Python class hierarchy traversal techniques and executes arbitrary commands in the context of the SGLang service process [2][6]. If the service runs with elevated privileges—as is frequently the case in GPU-accelerated inference deployments due to driver and device access requirements—the consequences of exploitation can include full host compromise, lateral movement into adjacent services, exfiltration of model weights or API keys, and denial of service.

It is worth emphasizing that this attack does not require network-level access to the inference server at the time of exploitation—only that the server has loaded the malicious model and is processing normal inference requests. The supply chain delivery vector effectively pre-positions the attacker inside the

trust boundary before any request is made.

SGLang's Broader Vulnerability Cluster

CVE-2026-5760 is not an isolated finding but the most recent in a series of critical vulnerabilities affecting SGLang. Earlier in 2026, security researchers at Orca Security disclosed three additional CVEs related to unsafe Python pickle deserialization in SGLang's inter-process communication components [3].

CVE-2026-3059 and CVE-2026-3060, both rated CVSS 9.8, affect SGLang's multimodal generation ZMQ broker and encoder parallel disaggregation receiver, respectively. Both components bind their ZMQ sockets to `tcp://*` (all available network interfaces) by default, with no authentication, and immediately call `pickle.loads()` on received payloads. Python's own documentation warns explicitly that pickle deserialization should never be applied to untrusted data, because pickle streams encode object reconstruction instructions that can invoke arbitrary functions—effectively equivalent to remote code execution from a network message [9][3]. Orca Security found over 20 instances of `pickle.loads()` across different SGLang modules, characterizing unsafe pickle deserialization as a systemic pattern in the codebase rather than an isolated mistake [3].

CVE-2026-3989, rated CVSS 7.8, affects a crash dump replay utility that deserializes pickle files without validation. This vector requires local file access or social engineering to plant a malicious `.pkl` file, but represents an additional attack surface for adversaries who have achieved partial access. Unlike CVE-2026-5760, the Orca Security findings did result in an upstream response: SGLang version 0.5.10 addresses the pickle deserialization vulnerabilities [4].

Disclosure and Patch Status

CVE-2026-5760 was reserved on April 7, 2026, and publicly disclosed on April 20, 2026, by researcher Stuart Beck through CERT/CC (VU#915947) [1]. The disclosure timeline reflects a significant challenge: CERT/CC made multiple contact attempts through GitHub Security Advisories and direct email to the SGLang project maintainers, and also sought assistance from CISA, but received no response from the project at any point during the coordination process. As of the disclosure date, no official patch exists for CVE-2026-5760, and no timeline for a fix has been communicated by the SGLang project.

The absence of a vendor response to critical vulnerability coordination is itself a risk signal for organizations evaluating their dependency on SGLang. SGLang's case illustrates a broader risk in the open-source AI/ML ecosystem: frameworks with large deployment footprints may lack the security

response infrastructure that enterprise users require. Practitioners should not assume that high download counts or prominent production users imply adequate security maintenance practices.

Ecosystem Pattern: Template Injection in LLM Serving

CVE-2026-5760 is not the first instance of this vulnerability class in LLM serving infrastructure. CVE-2024-34359, disclosed in 2024 and informally known as "Llama Drama," affected llama-cpp-python through an analogous Jinja2 SSTI vulnerability in its chat template rendering path [2][10]. CVE-2025-61620 subsequently identified Jinja2 template-handling weaknesses in vLLM, though as a denial-of-service condition (CVSS 3.1 Low) rather than a code execution vulnerability; the RCE-class SSTI pattern seen in CVE-2024-34359 and CVE-2026-5760 has not been confirmed in vLLM as of publication. Nevertheless, the recurrence of Jinja2-related vulnerabilities across multiple LLM serving frameworks—whether as code execution vectors or denial-of-service conditions—reflects a broader design failure: the ecosystem treats model metadata, including executable template strings, as trusted content, when in practice it should be treated with the same skepticism applied to user-supplied input.

Recommendations

Immediate Actions

Organizations running SGLang inference servers should take three steps before the end of the current business cycle. First, audit all currently loaded models to determine their provenance: any model sourced from an unverified or community repository should be treated as potentially compromised until provenance can be confirmed. Second, restrict network access to the `/v1/rerank` endpoint at the network boundary—either disabling it entirely if reranking is not in use, or limiting access to known, authenticated clients via firewall rules or an API gateway. Third, review whether any production SGLang deployment exposes ZMQ ports (associated with CVE-2026-3059 and CVE-2026-3060) on external interfaces, and if so, restrict them immediately; version 0.5.10 addresses the pickle deserialization vulnerabilities and should be deployed on any instance where multimodal or disaggregation features are enabled [4].

Short-Term Mitigations

Until an official patch for CVE-2026-5760 is available, development and operations teams should apply code-level mitigations directly to their SGLang deployments. The CERT/CC advisory specifically recommends replacing `jinja2.Environment()` with `jinja2.sandbox.ImmutableSandboxedEnvironment()` in the `getjinjaenv()` function responsible for rendering chat templates [1]. This change prevents templates from accessing Python's object model and closes the SSTI escape path; well-formed templates using standard variable substitution and control flow should continue to function normally, but teams should validate behavior in a non-production environment before promoting the change, as templates relying on sandbox-restricted features may require adjustment.

Model provenance verification should be established as a gate in any CI/CD or MLOps pipeline that introduces new model files. This means verifying cryptographic signatures or checksums from trusted sources, restricting model downloads to specific repositories or organizations with established review processes, and—where feasible—performing static analysis of GGUF metadata (including the `tokenizer.chat_template` field) before a model is deployed to an inference server. Tools such as Socket's AI scanner provide emerging capability for detecting malicious payloads in GGUF and other model formats [8].

Strategic Considerations

The broader lesson of CVE-2026-5760 is that model files should be considered executable artifacts, not passive data. Organizations building AI infrastructure policies should extend the same supply chain controls applied to software dependencies—provenance verification, dependency pinning, artifact signing, and vulnerability scanning—to model files. This is particularly important for organizations whose AI workflows involve automated model pulls from registries, where a compromised upstream artifact can propagate silently across multiple deployments.

The lack of a security response from the SGLang maintainers warrants careful consideration in vendor risk assessments. Organizations with production dependencies on SGLang should evaluate whether their use cases can be migrated to alternative serving frameworks with active security maintenance, or whether they can commit internal resources to maintaining a patched fork. In either case, relying solely on upstream response for critical vulnerabilities is not a sufficient posture for production AI infrastructure without compensating controls and defined escalation procedures.

CSA Resource Alignment

This vulnerability directly engages several CSA frameworks and resources relevant to organizations building or operating AI infrastructure.

CSA's MAESTRO (Multi-layer AI Evaluation and Security Threat Reasoning and Orchestration) threat modeling framework addresses threats at the model infrastructure layer, which is precisely where CVE-2026-5760 operates [11]. The attack exploits a trust relationship between the model file and the serving framework—a threat pattern MAESTRO characterizes as part of the agentic infrastructure attack surface. Security teams applying MAESTRO to their AI deployment architectures should include model-file metadata in their threat model as an untrusted input requiring sanitization.

The CSA AI Controls Matrix (AICM), which extends the Cloud Controls Matrix to AI-specific risks [12], provides control domains directly applicable to this scenario. Supply chain controls within the AICM address provenance verification and artifact integrity for AI components. Inference infrastructure controls address isolation, least-privilege execution, and access control for model-serving endpoints. Organizations using the AICM should map CVE-2026-5760 to controls in the Supply Chain Risk Management and Infrastructure Security domains and assess their current control implementations against this threat.

CSA's Zero Trust guidance applies to the network exposure patterns that amplify the impact of vulnerabilities like CVE-2026-5760. The ZMQ-based vulnerabilities (CVE-2026-3059, CVE-2026-3060) exploit the assumption that internal network access is implicitly trusted: sockets bound to all interfaces with no authentication represent a violation of Zero Trust principles at the network layer. Applying Zero Trust architecture to AI inference infrastructure—requiring authentication and mutual TLS even for internal service communications—would have materially reduced the exploitability of the related pickle deserialization vulnerabilities, by requiring authentication even for internal ZMQ connections.

CSA's STAR (Security Trust Assurance and Risk) program provides a mechanism for AI service providers to publish security posture information relevant to their customers' vendor risk assessments. The absence of a vendor security response for CVE-2026-5760 is directly relevant to STAR assessments for any cloud AI service built on SGLang. Customers and CSA community members are encouraged to request STAR disclosures from AI service providers regarding their SGLang version and patch posture.

More broadly, this vulnerability cluster illustrates risks described in CSA's AI Organizational Responsibilities guidance [13]: organizations adopting open-source AI infrastructure inherit security responsibilities that the upstream project may not fulfill. The guidance recommends establishing clear

internal ownership for the security of third-party AI components, including monitoring vulnerability disclosures and maintaining tested contingency procedures for critical infrastructure components that lack responsive upstream support.

References

[1] CERT/CC. "VU#915947 - SGLang is vulnerable to remote code execution when rendering chat templates from a model file." Carnegie Mellon University CERT Coordination Center, April 20, 2026.

[2] Ravie Lakshmanan. "SGLang CVE-2026-5760 (CVSS 9.8) Enables RCE via Malicious GGUF Model Files." The Hacker News, April 2026.

[3] Orca Security. "Pickle in the Pipeline: Critical RCE Vulnerabilities in SGLang's LLM Serving Framework." Orca Security Research Blog, 2026.

[4] CERT/CC. "VU#665416 - SGLang (sglang) is vulnerable to code execution attacks via unsafe pickle deserialization." Carnegie Mellon University CERT Coordination Center, 2026.

[5] GBHackers. "Malicious GGUF Models Could Trigger Remote Code Execution on SGLang Servers." GBHackers on Security, April 2026.

[6] CyberPress. "Hackers Could Weaponize GGUF Models to Achieve RCE on SGLang Inference Servers." CyberPress, April 2026.

[7] sgl-project. "SGLang: High-Performance Serving Framework for Large Language Models." GitHub, accessed April 2026.

[8] Socket. "Announcing Experimental Malware Scanning for the Hugging Face Platform." Socket Security Blog, accessed April 2026.

[9] Python Software Foundation. "pickle — Python object serialization: Security warning." Python 3 Documentation, accessed April 2026.

[10] National Vulnerability Database. "CVE-2024-34359 Detail." NIST NVD, 2024.

[11] Cloud Security Alliance. "MAESTRO: Multi-Layer AI Security Threat Modeling Framework." Cloud Security Alliance AI Safety Initiative, 2024.

[12] Cloud Security Alliance. "AI Controls Matrix (AICM)." Cloud Security Alliance, accessed April 2026.

[13] Cloud Security Alliance. "AI Organizational Responsibilities." Cloud Security Alliance, accessed April 2026.

