INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-20 18:53 UTC

Unpatched SGLang RCE: Malicious GGUF Files Weaponize AI Inference Servers via Jinja2 Template Injection

CVE VULNERABILITY | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CVE-2026-0059
Type	CVE Vulnerability
CVE ID	CVE-2026-5760, CVE-2024-34359, CVE-2025-61620
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	SGLang (open-source LLM serving framework); related: llama_cpp_python (CVE-2024-34359), vLLM (CVE-2025-61620)
Published	2026-04-20T13:14:00
Discovery Source	Rss

Executive Summary

A critical, unpatched remote code execution vulnerability (CVE-2026-5760, CVSS 9.5) in SGLang, an open-source AI inference server framework, allows an attacker to fully compromise a server by loading a malicious AI model file, no credentials required. Any organization running SGLang to serve large language models is exposed until a patch is released; no fix exists as of disclosure. This follows a documented pattern of the same attack class in related AI serving frameworks (CVE-2024-34359 in llama_cpp_python, CVE-2025-61620 in vLLM), indicating systemic risk across the AI inference infrastructure layer.

Technical Analysis

CVE-2026-5760 (CVSS 9.5, CWE-94/CWE-693/CWE-77) is an unauthenticated remote code execution vulnerability in the SGLang LLM serving framework, confirmed via [vendor advisory / CERT/CC / NVD - source to be verified]. The attack vector: a threat actor embeds malicious Jinja2 server-side template injection (SSTI) payloads inside GGUF model file metadata. When an operator loads the poisoned model and the /v1/rerank endpoint processes a request, the Jinja2 templates render without sanitization or sandboxing, executing arbitrary code in the server process context. Authentication is not required; the attack precondition is only that the victim load the weaponized model file, a realistic scenario given the widespread use of community-sourced model repositories. No vendor patch exists; vendor coordination was attempted via CERT/CC [date range -

specify if known]. Related prior art: CVE-2024-34359 (llama_cpp_python, NVD-confirmed, SSTI RCE via Jinja2) and CVE-2025-61620 (vLLM, similar template injection pattern). MITRE ATT&CK coverage: T1190 (Exploit Public-Facing Application), T1059/T1059.006 (Command and Script Interpreter: Python), T1195.001 (Compromise Software Dependencies and Development Tools).

Action Checklist

1. Step 1: Containment, immediately restrict network access to all SGLang inference server instances; block external access to /v1/rerank endpoints at the perimeter firewall or WAF; prevent untrusted or community-sourced GGUF model files from being loaded until a patch is available. Audit all currently loaded models for provenance.
2. Step 2: Detection, review SGLang server logs for anomalous requests to the /v1/rerank endpoint, particularly requests triggering unexpected process execution or outbound network connections; search for Jinja2 template syntax (e.g., '{{', '}}', '{%') in GGUF model metadata fields; monitor host-based logs (auditd on Linux, Sysmon on Windows) for unexpected child process spawning from the SGLang server process.
3. Step 3: Eradication, no vendor patch exists as of disclosure. If reranking is not operationally required, disable or block the /v1/rerank endpoint entirely. If reranking is essential, implement strict model provenance controls (only load models from trusted, internally verified sources with cryptographic integrity checks) and deploy Jinja2 sandboxing at the application layer if source-level modification is feasible. Monitor SGLang's GitHub repository and CERT/CC advisories for patch release.
4. Step 4: Recovery, after applying mitigations, re-verify the integrity of all GGUF model files loaded on affected servers using cryptographic hashes against known-good sources; confirm no unauthorized persistence mechanisms (new cron jobs, modified startup scripts, added users, reverse shells) were established on compromised or exposed hosts; restore service only after model provenance is validated and endpoint restrictions are confirmed active.
5. Step 5: Post-Incident, this vulnerability exposes a control gap in AI/ML infrastructure security: model files are treated as trusted data rather than executable artifacts. Implement formal model vetting procedures analogous to software supply chain controls (NIST SP 800-53 SA-3, NIST AI Risk Management Framework, SSDF); extend SBOM practices to include model provenance; add SGLang and AI serving framework updates to your patch management scope; review whether similar Jinja2 rendering patterns exist in other inference frameworks deployed in your environment (vLLM, llama_cpp_python).

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO and legal/compliance if SGLang server logs confirm exploitation of CVE-2026-5760 (evidence of Jinja2 payload execution, unexpected child processes, or unauthorized outbound connections), if GGUF model files loaded from untrusted sources are confirmed on internet-facing hosts, or if the SGLang inference server processed data subject to regulatory requirements (PII, PHI, financial records) — the latter triggers breach notification assessment under applicable regulations (HIPAA, GDPR, state breach laws).

Recovery Notes	Restore SGLang service only behind the nginx reverse proxy with `/v1/rerank` blocked and model loading restricted to the SHA-256 allowlist established during eradication; do not restore full public network access until a vendor patch for CVE-2026-5760 is available and applied. Monitor `journalctl -u sglang -f` and the auditd/Sysmon process-creation logs continuously for at least 14 days post-restoration, given that Jinja2 RCE exploitation may have enabled staged payloads or scheduled tasks with delayed activation. Verify vLLM and llama_cpp_python instances in the same environment are patched against CVE-2025-61620 and CVE-2024-34359 respectively before restoring any AI inference infrastructure, as the attack pattern is documented across all three frameworks.
Forensic Artifacts	SGLang application stdout/stderr logs (captured via `journalctl -u sglang`) — will contain raw POST body content or error tracebacks from Jinja2 template evaluation if a malicious GGUF was loaded, including the rendered payload string prior to execution. | GGUF model file binary content — `strings .gguf` output revealing Jinja2 template syntax (`{{`, `{%`) embedded in GGUF metadata fields (specifically the `general.name`, `general.description`, or custom KV metadata sections), which is the direct injection vector for CVE-2026-5760. | auditd `execve` syscall records (`/var/log/audit/audit.log`) or Sysmon Event ID 1 (Process Create) logs — will capture subprocess spawning (e.g., `sh -c`, `curl`, `wget`, `python3 -c`) with ParentPID matching the SGLang server process, which is the post-exploitation execution indicator for this Jinja2 RCE class. | Network packet capture (`tcpdump` PCAP on the SGLang host's primary interface) — will contain outbound C2 or reverse shell connection attempts initiated by the Jinja2-rendered payload; filter on connections originating from the SGLang process UID to non-RFC1918 destinations on non-standard ports. | `/tmp`, `/var/tmp`, `/dev/shm` filesystem artifacts — RCE via Jinja2 template injection in AI inference servers (consistent with CVE-2024-34359 llama_cpp_python exploitation pattern) commonly stages dropper scripts or shell payloads in world-writable directories; preserve with `find /tmp /var/tmp /dev/shm -type f -newer /var/log/sglang_start_timestamp -ls` timestamped against the SGLang process start time.

Per-Action IR Details

Step 1: Containment — immediately restrict network access to all SGLang inference server instances; block external access to /v1/rerank endpoints at the perimeter firewall or WAF; prevent untrusted or community-sourced GGUF model files from being loaded until a patch is available. Audit all currently loaded models for provenance.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy

Controls: NIST IR-4 (Incident Handling), NIST SC-7 (Boundary Protection), NIST CM-7 (Least Functionality), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 12.2 — Establish and Maintain a Secure Network Architecture (restrict lateral access to AI inference nodes)

Compensating: Apply iptables rules immediately on each SGLang host to block inbound traffic to the default SGLang port (typically 30000/TCP) from all sources except known internal consumers: `iptables -I INPUT -p tcp --dport 30000 -j DROP` followed by an explicit ACCEPT for trusted source CIDRs. If a WAF is unavailable, use nginx as a reverse proxy with a `deny` directive for the `/v1/rerank` location block. For model provenance audit, run `find /path/to/models -name '*.gguf' -exec sha256sum {} \; > model_inventory_$(date +%F).txt` and compare against known-good hashes from the model's original source repository.

Evidence: Before modifying firewall rules, capture current active network connections from the SGLang server process: `ss -tnp | grep python` (or the SGLang process name) to identify any live reverse shell or C2 connections. Preserve `/var/log/nginx/access.log` or equivalent reverse-proxy logs showing all historical requests to `/v1/rerank`. Dump the process list and open file handles: `ps auxf > proc_snapshot.txt` and `lsof -p > lsof_snapshot.txt`. These will be lost or altered once containment network changes are applied.

Step 2: Detection — review SGLang server logs for anomalous requests to the /v1/rerank endpoint, particularly requests triggering unexpected process execution or outbound network connections; search for Jinja2 template syntax (e.g., '{{', '}}', '{%') in GGUF model metadata fields; monitor host-based logs (auditd on Linux, Sysmon on Windows) for unexpected child process spawning from the SGLang server process.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis

Controls: NIST SI-4 (System Monitoring), NIST AU-2 (Event Logging), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-3 (Content of Audit Records), CIS 8.2 (Collect Audit Logs), MITRE ATT&CK T1059 (Command and Scripting Interpreter) — post-exploitation subprocess execution from SGLang process, MITRE ATT&CK T1190 (Exploit Public-Facing Application) — exploitation via malicious GGUF over /v1/rerank, MITRE ATT&CK T1195.001 (Supply Chain Compromise: Compromise Software Dependencies and Development Tools) — malicious GGUF sourced from untrusted community repositories

Compensating: Deploy Sysmon with a configuration that captures Event ID 1 (Process Create) with ParentImage matching the SGLang Python interpreter path (e.g., `/usr/bin/python3` or the venv python). Filter for child processes such as `sh`, `bash`, `curl`, `wget`, `nc`, or `python3 -c`. Use `auditd` on Linux with a rule targeting execve syscalls from the SGLang process UID: `auditctl -a always,exit -F arch=b64 -S execve -F uid= -k sglang_exec`. To scan GGUF files for embedded Jinja2 template syntax, run: `strings model.gguf | grep -E '\{\{|\{%|%\}|\}\}'` against all loaded model files. For network-based detection without a SIEM, use `tcpdump -i any -w sglang_traffic.pcap port 30000` and inspect with Wireshark, filtering for HTTP POST bodies to `/v1/rerank` containing `%7B%7B` (URL-encoded `{{`) or literal Jinja2 delimiters.

Evidence: Collect SGLang application logs (typically stdout/stderr captured by systemd — retrieve with `journalctl -u sglang --since '7 days ago' > sglang_journal.txt`). Extract all POST requests to `/v1/rerank` from the web/proxy access log: `grep 'POST /v1/rerank' /var/log/nginx/access.log`. Run `strings` against each `.gguf` file in the model directory and redirect output for manual review of metadata fields for Jinja2 syntax. Capture auditd logs (`/var/log/audit/audit.log`) and filter for execve events attributed to the SGLang process UID. On Windows, export Sysmon Event ID 1 logs from `Microsoft-Windows-Sysmon/Operational` filtering on ParentImage containing the SGLang Python path.

Step 3: Eradication — no vendor patch exists as of disclosure; workarounds include disabling or blocking the /v1/rerank endpoint entirely if not operationally required, implementing strict model provenance controls (only load models from trusted, internally verified sources with cryptographic integrity checks), and deploying Jinja2 sandboxing at the application layer if source-level modification is feasible. Monitor SGLang's GitHub repository and CERT/CC advisories for patch release.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-7 (Least Functionality), NIST SI-3 (Malicious Code Protection), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: If `/v1/rerank` is unused, remove it from the SGLang startup configuration or block it at the reverse proxy with `location /v1/rerank { return 403; }` in nginx. For Jinja2 sandboxing without application-layer access: apply a YARA rule to scan GGUF files pre-load — write a rule matching `{7B 7B | 7B 25}` byte sequences within GGUF metadata sections and run it via `yara jinja2_gguf.yar /models/` before any model is loaded. Establish a model allowlist by SHA-256 hash: maintain a plaintext allowlist file and enforce with a wrapper script that checks `sha256sum model.gguf` against the list before invoking SGLang. Subscribe to SGLang GitHub releases via RSS (`https://github.com/sgl-project/sglang/releases.atom`) and CERT/CC alerts at `https://www.kb.cert.org/vuls/` for CVE-2026-5760 patch notification.

Evidence: Before disabling the endpoint or modifying configurations, preserve the current SGLang configuration file (e.g., `config.json` or launch arguments captured from `ps aux | grep sglang > sglang_launch_args.txt`) as evidence of the pre-remediation state. Archive all GGUF files that were loaded at time of detection to an isolated, write-protected evidence store with SHA-256 hashes recorded. If any GGUF file contains Jinja2 syntax identified in Step 2, treat it as a

forensic artifact and do not delete it — quarantine to an air-gapped storage location and document chain of custody per NIST 800-61r3 §3.4 evidence handling guidance.

Step 4: Recovery — after applying mitigations, re-verify the integrity of all GGUF model files loaded on affected servers using cryptographic hashes against known-good sources; confirm no unauthorized persistence mechanisms (new cron jobs, modified startup scripts, added users, reverse shells) were established on compromised or exposed hosts; restore service only after model provenance is validated and endpoint restrictions are confirmed active.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-3 (Configuration Change Control), NIST AU-11 (Audit Record Retention), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 5.3 (Disable Dormant Accounts), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Run a persistence hunt checklist specific to Linux-hosted SGLang servers: (1) `crontab -l -u ` and `cat /etc/cron*/*` to enumerate new cron entries; (2) `diff post_remediation_hashes.txt` and diff against the pre-remediation inventory captured in Step 1. Restore SGLang service under the restricted nginx reverse proxy config with the `/v1/rerank` block active, and validate with `curl -X POST http://localhost:30000/v1/rerank` returning 403.

Evidence: Before restoring service, capture a final system state snapshot: `last -n 50 > last_logins.txt`, `lastb -n 50 > failed_logins.txt`, `find /tmp /var/tmp /dev/shm -type f -ls > tmp_files.txt` (common staging directories for post-exploitation payloads dropped via RCE), and `cat /root/.bash_history /home/*/.bash_history > bash_histories.txt`. These artifacts document whether the Jinja2 RCE was used to stage files or establish persistence before containment was applied, and are required for post-incident analysis under NIST 800-61r3 §4.

Step 5: Post-Incident — this vulnerability exposes a control gap in AI/ML infrastructure security: model files are treated as trusted data rather than executable artifacts. Implement formal model vetting procedures analogous to software supply chain controls (NIST SP 800-161, SSDF); extend SBOM practices to include model provenance; add SGLang and AI serving framework updates to your patch management scope; review whether similar Jinja2 rendering patterns exist in other inference frameworks deployed in your environment (vLLM, llama_cpp_python).

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity (Lessons Learned)

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SI-2 (Flaw Remediation), NIST SA-12 (Supply Chain Protection), NIST RA-3 (Risk Assessment), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Extend your existing software inventory (CIS 2.1) to include all AI model files as tracked artifacts: create a model registry CSV recording filename, SHA-256 hash, source URL, download date, and approving analyst for every GGUF, safetensors, or ONNX file in the environment. Run YARA scans against vLLM's template handling paths and llama_cpp_python's model loader to detect the same Jinja2 injection pattern documented in CVE-2024-34359 and CVE-2025-61620 — write rules matching `{{` and `{%` in model metadata fields. Add a GitHub Actions or cron-based check: `curl -s https://api.github.com/repos/sgl-project/sglang/releases/latest | jq .tag_name` to alert on new SGLang releases. Document the lessons-learned output as a formal after-action report per NIST 800-61r3 §4.1, specifically noting that GGUF model files must be classified as executable artifacts in your organization's data classification policy.

Evidence: Compile the full evidence package from all prior steps into a structured incident record per NIST IR-5 (Incident Monitoring): SGLang access logs, process snapshots, GGUF hash inventories, persistence hunt outputs, and bash history files. Retain for the organization's defined period per NIST AU-11 (Audit Record Retention). Produce an IOC report documenting any malicious GGUF SHA-256 hashes, Jinja2 payload strings found in model metadata, and any C2 IP addresses or domains observed in outbound connections during Step 2 network capture — share internally and consider submission to an ISAC relevant to your sector.

Detection Guidance

Primary detection targets: (1) Network layer, anomalous or unexpected POST requests to /v1/rerank on SGLang server ports; inspect request bodies for Jinja2 template syntax characters ('{{', '}}', '{%', '%}'). (2) Process layer, unexpected child processes spawned from the SGLang Python process (e.g., /bin/sh, curl, wget, python3 subprocesses not part of normal inference workload); use auditd EXECVE rules or Sysmon Event ID 1 targeting the SGLang process parent. (3) Network egress, unexpected outbound connections from inference server hosts, particularly to uncommon external IPs/domains, which may indicate post-exploitation callback or exfiltration. (4) Model file inspection, parse GGUF metadata fields for embedded template syntax prior to loading; GGUF files store metadata as key-value pairs in a defined header structure, making automated scanning feasible. (5) Correlation with llama_cpp_python and vLLM, if those frameworks are also in scope, apply equivalent detection logic given the shared vulnerability class (CVE-2024-34359, CVE-2025-61620). No public IOCs (IPs, hashes, domains) have been attributed to active exploitation of CVE-2026-5760 as of this analysis.

Framework Mappings

MITRE-ATTACK

- **T1059.006** — Python
- **T1190** — Exploit Public-Facing Application
- **T1059** — Command and Scripting Interpreter
- **T1203** — Exploitation for Client Execution
- **T1195.001** — Compromise Software Dependencies and Development Tools

NIST-800-53R5

- **CA-8** — Penetration Testing
- **RA-5** — Vulnerability Monitoring and Scanning
- **SC-7** — Boundary Protection
- **SI-2** — Flaw Remediation
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-10** — Information Input Validation

OWASP-TOP10-2021

- **A03:2021** — Injection

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.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain

SOC2-TSC

- **CC9.2** — Manages risks associated with vendors and business partners

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1059.006	Python	Execution
T1190	Exploit Public-Facing Application	Initial-Access
T1059	Command and Scripting Interpreter	Execution
T1203	Exploitation for Client Execution	Execution
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/04/sglang-cve-2026-5760-cvss-98-enab...	T3
CVE-2024-34359 Detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2024-34359	T1
CVE-2024-34359 Vulnerability Threatening Your Software Supply ...	https://checkmarx.com/blog/llama-drama-critical-vulnerability-cve-2...	T3
CVE-2024-34359 | Critical Vulnerability in llama-cpp-python	https://www.appsecure.security/vulnerability-database/cve-2024-34359/	T3
CVE-2024-34359: llama-cpp-python RCE Vulnerability - SentinelOne	https://www.sentinelone.com/vulnerability-database/cve-2024-34359/	T3
NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-5760, CVE-2024-34359, CVE...	T1

Generated 2026-04-20 18:53 UTC by TJS Security Command Center