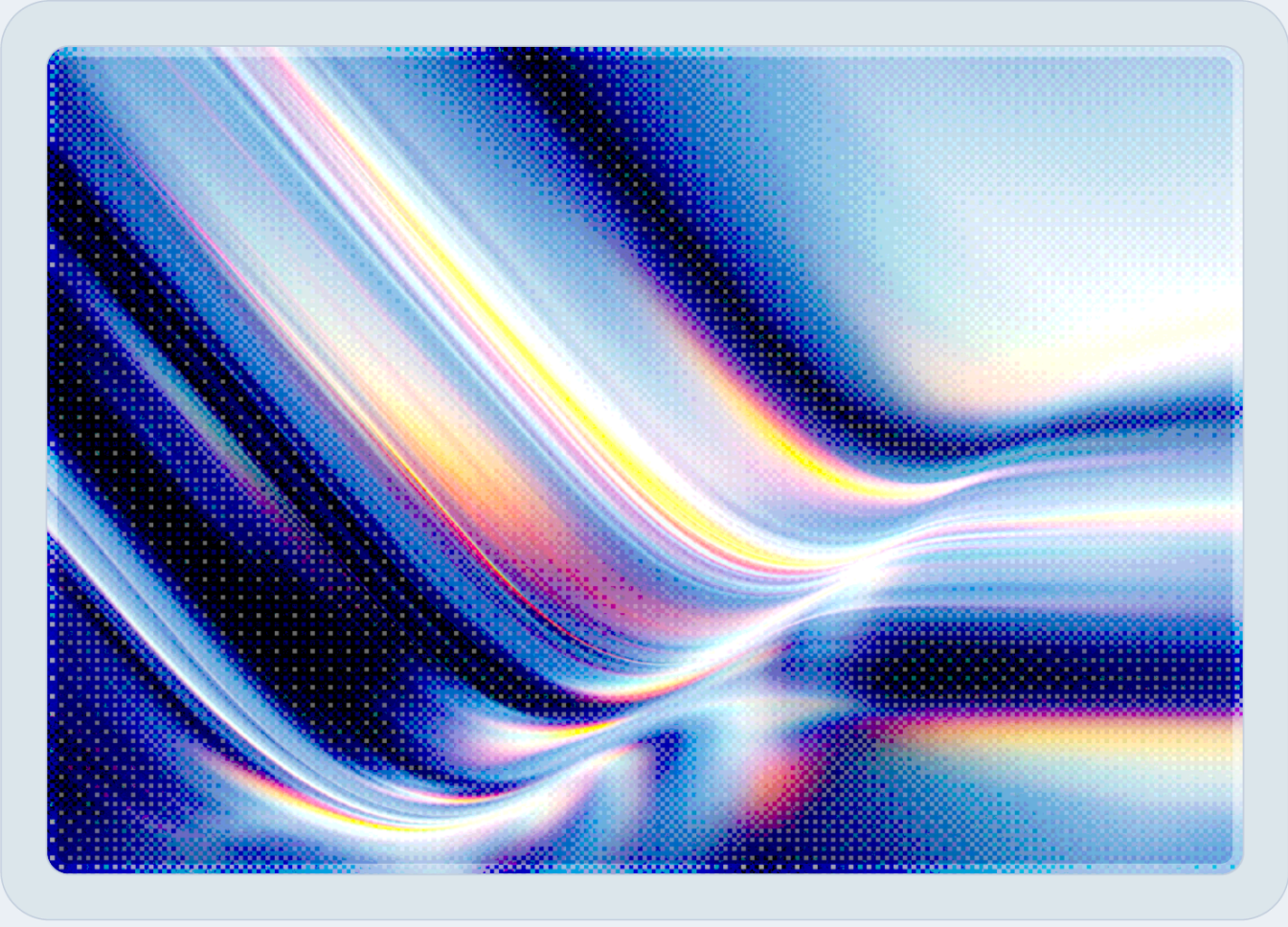
CSAI Foundation | Cloud Security Alliance

Bleeding Llama: Unauthenticated Memory Leak in Ollama

CVE-2026-7482 Exposes 300,000 AI Inference Servers to Heap Data Exfiltration

2026-05-11

AI-assisted Rapid Research



CSAI

cloud security alliance®

Key Takeaways

- Cyera Research disclosed CVE-2026-7482, a heap out-of-bounds read in Ollama's GGUF model loader, assigned a CVSS score of 9.1 [1][3]. An unauthenticated attacker can send a specially crafted HTTP request to any reachable Ollama server to leak process memory containing API keys, system prompts, conversation history, and environment variables.
- Approximately 300,000 Ollama servers are publicly reachable on the internet, and the platform's REST API offers no authentication by default—meaning network exposure alone is sufficient for exploitation without any credential theft or privilege escalation.
- The attack requires only two sequential HTTP requests: a POST to `/api/create` to trigger the out-of-bounds read during GGUF quantization, followed by a POST to `/api/push` to exfiltrate the harvested heap data to an attacker-controlled model registry.
- The vulnerability was patched in Ollama version 0.17.1, released February 25, 2026. Organizations must upgrade immediately and, critically, place all Ollama deployments behind an authentication proxy regardless of whether they have applied the patch.
- Separately, two unpatched vulnerabilities (CVE-2026-42248, CVE-2026-42249) in Ollama's Windows auto-update mechanism enable persistent code execution via a tampered update server—a distinct but compounding risk for Windows-based AI inference environments.

Background

Ollama is an open-source framework that enables developers and enterprises to run large language models locally, treating model download, serving, and API access as a single integrated runtime. With over 170,000 GitHub stars and more than 100 million Docker Hub pulls as of early 2026, Ollama has become one of the most widely deployed self-hosted AI inference platforms in the world [1][4]. Its appeal lies in operational simplicity: a single daemon process manages model lifecycle, exposes a REST API on port 11434, and handles requests from any connected client. That same simplicity, however, creates a tension with the security expectations of enterprise deployments. Ollama binds by default to localhost, but many operators consciously or inadvertently expose it on all network interfaces, and the API carries no built-in authentication mechanism [5].

The GGUF (GPT-Generated Unified Format) file format has become the widely adopted serialization standard for quantized LLM weights in local inference ecosystems. When a user creates a model from a GGUF file via Ollama's `/api/create` endpoint, the server parses the file's tensor metadata, loads weight data into memory, and optionally performs additional quantization. This pipeline involves reading tensor offsets and sizes declared in the GGUF header and then seeking to the corresponding positions in the file to load the actual weight data. The vulnerability examined in this note lives at the boundary between the declared metadata and the actual bytes on disk.

Researchers at Cyera reported the flaw to Ollama's security team on February 2, 2026. Ollama acknowledged the report and issued a fix on February 25, shipping the corrected code in version 0.17.1. Cyera published their full technical disclosure in early May 2026 under the name "Bleeding Llama" – evoking the Heartbleed vulnerability, which similarly allowed remote attackers to read sensitive memory from a network-accessible service without authentication [1]. CERT/CC catalogued the issue under vulnerability note VU#518910 [11].

Security Analysis

The Root Cause: Unsafe Memory Access During GGUF Quantization

The vulnerability originates in Ollama's use of Go's `unsafe` package within the model file parsing and quantization pipeline. Specifically, the `WriteTo()` function in `fs/ggml/gguf.go` and the quantization routines in `server/quantization.go` construct a read boundary based on the tensor offset and size values declared in the GGUF file's metadata headers [1][2]. These declared dimensions are not validated against the actual length of the supplied file before the read operation begins. An attacker who controls the GGUF file can therefore declare a tensor shape far larger than the real data—for instance, setting a single tensor's declared size to several gigabytes when the file contains only kilobytes of actual weight data. When Ollama's quantization code attempts to read the declared volume of data, it reads past the end of the allocated buffer and into adjacent heap memory belonging to the Ollama process [3].

This class of flaw—a heap out-of-bounds read driven by attacker-supplied size metadata—represents a classic memory safety violation. Go's type system ordinarily prevents such accesses, but the explicit invocation of the `unsafe` package suspends those protections in the affected code path. The result is that an attacker-controlled integer in a GGUF file header becomes the effective upper bound on how much Ollama process memory is read during model creation [1].

Attack Chain and Exfiltration Path

Exploitation of CVE-2026-7482 requires no authentication, no existing session, and no prior knowledge of the server beyond its IP address and port. An attacker issues an HTTP POST request to the `/api/create` endpoint with a JSON body referencing or embedding the crafted GGUF file. Ollama accepts the request, begins model creation, and triggers the out-of-bounds read during quantization [2]. The heap data read during this operation is incorporated into the model artifact that Ollama creates. The attacker then issues a second HTTP POST to the `/api/push` endpoint, instructing Ollama to upload the newly created model to an external registry under the attacker's control. The exfiltrated heap memory travels to the attacker's registry as part of the model artifact, completing a two-step cycle from crafted file upload to data exfiltration [1][6].

Whatever resided on the Ollama process heap at the time of the read is subject to disclosure. This includes system prompts from all concurrently loaded models, fragments of chat messages from other users sharing the same Ollama instance, environment variables (which in many deployments carry OpenAI API keys, Anthropic API tokens, AWS credentials, and database connection strings), and any code or document content that has passed through the inference engine [2][7]. In multi-tenant deployments—such as shared development servers or containerized inference services where multiple users or applications share a single Ollama instance—the blast radius can extend to the conversations and secrets of other active users or applications sharing the same instance at the time of the attack.

Scale of Exposure

The risk surface for this vulnerability is substantially magnified by Ollama's deployment patterns. Cyera's researchers identified approximately 300,000 publicly internet-accessible Ollama servers using internet-scanning tools [4][7]. This figure is higher than the 175,000 exposed instances identified by independent research across 130 countries in January 2026 [10], a difference that likely reflects continued growth in Ollama adoption over the intervening months – and a trajectory in the wrong direction. Ollama provides no authentication layer by default, so any instance reachable over the network —whether intentionally exposed or misconfigured—is fully exploitable without credential theft. The platform's documentation frames it as a local development tool, but enterprise adoption has outpaced security hardening guidance, and many production deployments listen on all network interfaces rather than localhost alone [5].

SecurityWeek assessed that CVE-2026-7482 "could expose 300,000 Ollama deployments to information theft," characterizing the combination of wide internet exposure and no default authentication as a critical force multiplier for the underlying memory safety flaw [4]. The runZero

research team published tooling to identify affected Ollama instances across enterprise networks, recommending that security teams treat any Ollama instance running a version prior to 0.17.1 as actively compromised until patched and rotated [8].

Windows Auto-Update Vulnerabilities (CVE-2026-42248, CVE-2026-42249)

Separately from the memory leak, researchers at Striga published findings in May 2026 detailing two unpatched vulnerabilities in Ollama's Windows auto-update mechanism that compound risk for Windows-based AI inference deployments [9]. CVE-2026-42249 (CVSS 7.7) describes a path traversal flaw in the Windows updater's staging directory logic: the updater constructs the local path for the installer's staging directory directly from HTTP response headers without sanitizing user-controlled path components, permitting an attacker who can influence update traffic to write files to arbitrary locations on the filesystem [9][12]. CVE-2026-42248 describes a missing signature verification step: Ollama's Windows update verification routine unconditionally returns success, performing no digital signature or trust chain validation on downloaded update executables before staging or executing them [9].

When chained, these two weaknesses allow an attacker positioned to intercept or serve update responses—through a compromised network path, a malicious Wi-Fi access point, or a tampered DNS resolution—to deliver an arbitrary executable that Ollama stages in the Windows Startup folder, achieving persistence that survives reboots [9]. Striga disclosed these vulnerabilities to Ollama on January 27, 2026, and published findings 90 days later after no patch was released. As of this writing, CVE-2026-42248 and CVE-2026-42249 remain unpatched [9][12].

Recommendations

Immediate Actions

The most urgent priority for all organizations running Ollama is to upgrade to version 0.17.1 or later, which contains the fix for CVE-2026-7482. Upgrade alone is not sufficient, however: because Ollama provides no native authentication, any network-accessible instance remains open to a wide range of unauthenticated API abuse even after patching. Organizations must place all Ollama deployments behind an authenticating reverse proxy or API gateway that enforces identity verification before any request reaches the Ollama daemon. This control is necessary irrespective of patch status and should be treated as a baseline security requirement rather than an optional hardening step [1][4][5].

Organizations should also conduct an immediate inventory of all Ollama instances in their environment—including developer laptops, shared build servers, containerized inference services, and AI application back-ends—to identify any that are reachable from networks broader than intended. Instances bound to `0.0.0.0` rather than `127.0.0.1` or a private interface should be reconfigured immediately. Network-level controls including firewall rules and security group policies should restrict access to port 11434 to explicitly authorized source addresses [8].

For Windows environments, organizations should disable Ollama's automatic update feature in application settings until CVE-2026-42248 and CVE-2026-42249 are patched. This setting prevents the background update process from fetching and staging executables that have not been verified [9].

Short-Term Mitigations

Following emergency patching and network isolation, organizations should rotate all credentials that may have been present as environment variables or process-accessible secrets on any Ollama instance that was reachable before the patch was applied. This includes API keys for cloud AI providers, object storage credentials, database connection strings, and any application-layer secrets loaded in the Ollama runtime environment. The conservative assumption for any pre-patch, network-exposed instance is that process memory was accessible to an unauthenticated attacker [2][7].

Based on the two-step attack chain described in the Security Analysis section, security teams should review Ollama access logs for anomalous patterns indicative of exploitation—specifically, model creation requests referencing unexpected GGUF sources followed closely by model push requests to external registries. While not every such sequence is malicious, any push request to a registry outside of organizational control from a production inference server warrants investigation. Organizations that lack comprehensive API access logging for their Ollama deployments should instrument this capability as a prerequisite to meaningful detection coverage.

For multi-tenant inference environments in which multiple users or applications share a single Ollama instance, organizations should evaluate whether the consolidation architecture remains appropriate given the demonstrated heap cross-contamination risk. Until runtime memory isolation between tenant workloads can be guaranteed, isolation at the process or container level—one Ollama instance per trust boundary—is the defensible posture.

Strategic Considerations

Bleeding Llama exemplifies a risk pattern that has appeared across multiple AI inference and orchestration frameworks: platforms designed primarily for developer convenience, often assuming a trusted local environment, are being adopted at enterprise scale without commensurate security hardening. Ollama's lack of authentication is not a bug—it is a documented design choice [5]—and it means that network reachability is the only access control standing between an attacker and full API access. Organizations deploying AI inference platforms should evaluate each platform's security architecture against the same criteria they would apply to any externally-facing service: authentication, authorization, TLS, audit logging, and input validation must be present by default, not optional hardening.

AI inference servers represent a high-value target class because they sit at the intersection of proprietary model assets, sensitive user data, and runtime secrets. A successful compromise of an inference server can yield not only the data that flows through it but also the secrets that grant access to upstream AI provider APIs, downstream data stores, and cloud infrastructure. Security teams should classify AI inference infrastructure within their asset inventory at a sensitivity tier commensurate with the data those systems process — consistent with asset classification guidance in CSA's AI Organizational Responsibilities framework [16] — and apply corresponding protective and detective controls.

CSA Resource Alignment

The vulnerabilities described in this note connect directly to several domains within CSA's AI and cloud security frameworks. The AI Controls Matrix (AICM), a superset of the Cloud Controls Matrix (CCM) that addresses AI-specific risk controls, provides relevant guidance in its infrastructure security and access control domains [13]. AICM controls governing authentication requirements for AI API endpoints, input validation for model loading pipelines, and secrets management for AI runtime environments address precisely the gaps that CVE-2026-7482 exploits. Organizations using AICM as a governance baseline should review their control implementations for AI inference services against the relevant infrastructure and access control domains.

CSA's LLM Threats Taxonomy classifies "Sensitive Data Disclosure" among the primary threat categories for large language model deployments, with infrastructure-layer memory exposure listed as a representative threat scenario [14]. Bleeding Llama is a direct instantiation of this threat category, demonstrating how a memory safety flaw in the AI inference layer can yield disclosure of conversation history, in-context prompts, and operational secrets without any interaction with the model itself.

Organizations using the CSA taxonomy to build AI threat models should extend those models to include the inference server's operating environment as an explicit threat surface, not just the model weights and training pipeline.

The MAESTRO (Machine learning and AI Security Threat and Risk Ontology) framework developed within the CSA AI Safety Initiative addresses AI system threats across the full stack, including the infrastructure layer where Ollama operates. The Bleeding Llama attack traverses MAESTRO's model serving layer (the GGUF loader and quantization pipeline) and infrastructure layer (the process heap and environment variables), illustrating the cross-layer nature of AI inference vulnerabilities. Security architects using MAESTRO for threat modeling should ensure that the model serving and quantization subsystems are represented as explicit attack surfaces, with memory safety and input validation requirements enumerated for each.

CSA's Zero Trust guidance is directly applicable to the structural risk this vulnerability class represents. Ollama's no-authentication-by-default posture is inconsistent with Zero Trust principles, which require that all service-to-service communication be authenticated regardless of network position. Deploying an authenticating API gateway in front of Ollama—as recommended in the Immediate Actions section above—is precisely the Zero Trust perimeter control that converts an implicitly trusted internal service into one that verifies every caller [15].

Finally, CSA's AI Organizational Responsibilities publications address the governance obligations of organizations that deploy AI infrastructure, including requirements for asset inventory, vulnerability management, and incident response planning for AI systems. The scale of unpatched and exposed Ollama deployments described in this note reflects a gap between the rate at which AI inference infrastructure is adopted and the rate at which organizational security governance is extended to cover it. CSA's AI Organizational Responsibilities guidance provides a structured framework for closing that gap [16].

References

[1] Cyera Research. "Bleeding Llama: Critical Unauthenticated Memory Leak in Ollama." Cyera, May 2026.

[2] The Hacker News. "Ollama Out-of-Bounds Read Vulnerability Allows Remote Process Memory Leak." The Hacker News, May 2026.

[3] cvefeed.io. "CVE-2026-7482 – Ollama heap out-of-bounds read in GGUF tensor parsing leaks server process memory to unauthenticated remote attackers." CVE Feed, 2026.

[4] SecurityWeek. "Critical Bug Could Expose 300,000 Ollama Deployments to Information Theft." SecurityWeek, May 2026.

[5] CSO Online. "Ollama vulnerability highlights danger of AI frameworks with unrestricted access." CSO Online, May 2026.

[6] CyberSecurityNews. "Critical Ollama Memory Leak Vulnerability Exposes 300,000 Servers Globally." Cyber Security News, May 2026.

[7] Cybernews. "Major AI platform Ollama critically leaking: 300,000 servers exposed to hackers." Cybernews, May 2026.

[8] runZero. "Ollama vulnerability CVE-2026-7482: Find impacted assets." runZero Research, May 2026.

[9] Striga. "Ollama Updates Itself Into Persistent RCE on Windows." Striga AI Security Research, May 2026.

[10] The Hacker News. "Researchers Find 175,000 Publicly Exposed Ollama AI Servers Across 130 Countries." The Hacker News, January 2026.

[11] CERT/CC. "VU#518910 – Ollama GGUF Quantization Remote Memory Leak." Carnegie Mellon University CERT Coordination Center, 2026.

[12] Help Net Security. "Unpatched flaws turn Ollama's auto-updater into a persistent RCE vector, researchers say." Help Net Security, May 2026.

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

[14] Cloud Security Alliance. "Large Language Model (LLM) Threats Taxonomy." Cloud Security Alliance, 2024.

[15] Cloud Security Alliance. "Zero Trust Advancement Center." Cloud Security Alliance, 2025.

[16] Cloud Security Alliance. "AI Organizational Responsibilities." Cloud Security Alliance, 2024.