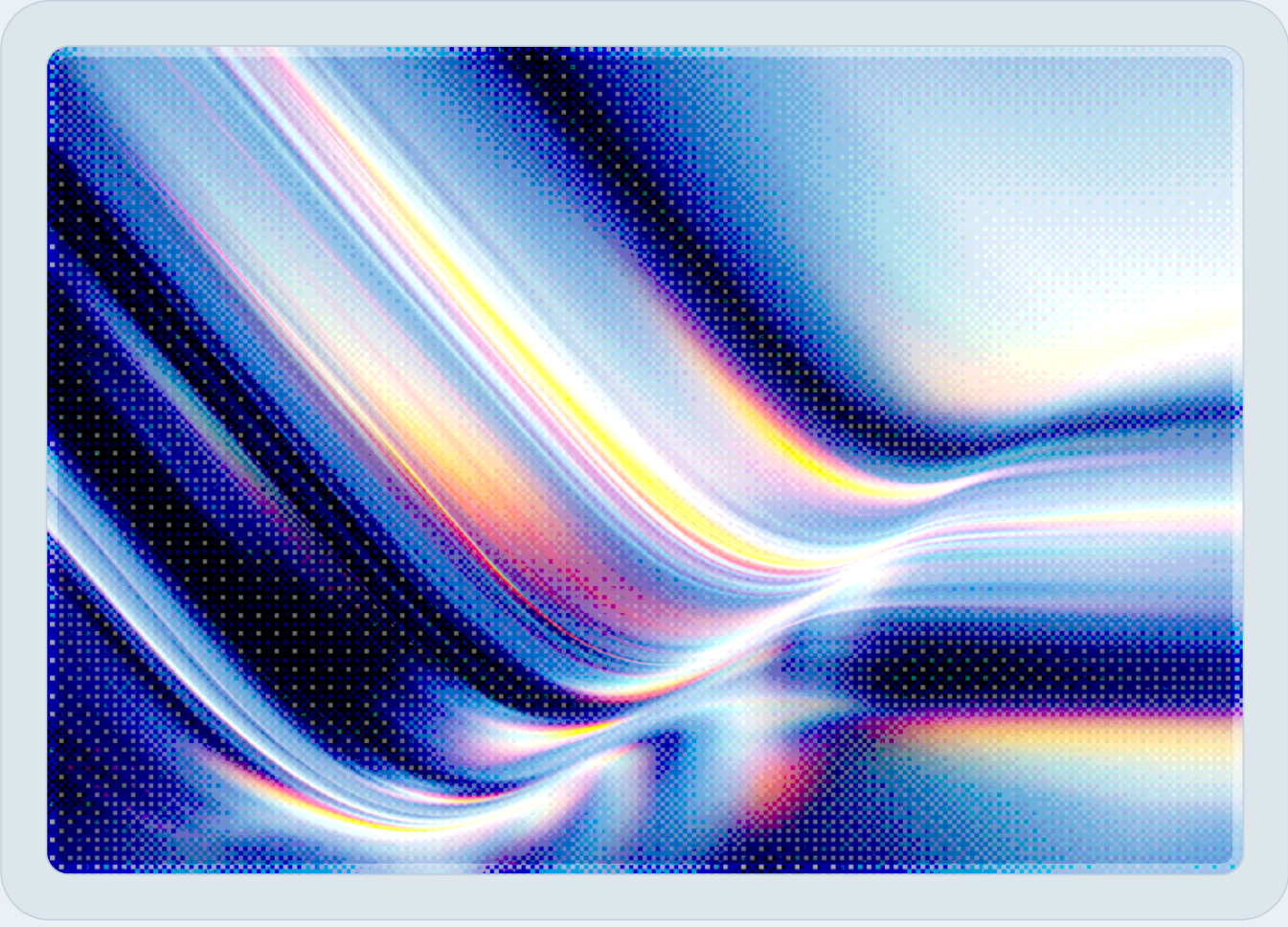
CSAI Foundation | Cloud Security Alliance

DNS Rebinding in NemoClaw Enables Persistent Model Poisoning

2026-08-27

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

A vulnerability disclosed on August 25, 2026 shows that a single visit to a malicious webpage can silently and persistently corrupt the local AI model behind NVIDIA's NemoClaw agent deployment stack, no phishing, credential theft, or malware installation required. Tracked as CVE-2026-65105 and rated High severity (CVSS 3.1 base score 8.1) [9], the flaw combines an insecure network default with a decades-old browser attack technique called DNS rebinding to reach an unauthenticated local API and rewrite the chat template that governs how every future message is presented to the model [1][2][3]. Because the injected instructions live in the template layer rather than in a single conversation, they persist across reboots, survive any system prompt the agent later supplies, and are invisible to anyone inspecting the visible conversation or logs. Researchers at Oasis Security, now being acquired by Cyera in a deal valued at roughly $1 billion [10], disclosed the flaw to NVIDIA's Product Security Incident Response Team (PSIRT) before publication; NVIDIA has shipped a fix for macOS and Linux, but Windows and WSL deployments remained exposed at disclosure time, with only a warning added rather than a structural remediation [1][3]. The episode illustrates a pattern this Initiative has flagged before in the context of MCP server exposure: network defaults chosen to ease local connectivity on developer machines running agentic AI create a browser-reachable attack surface that traditional sandboxing does not close [4].

Background

NemoClaw is NVIDIA's open-source reference stack for deploying AI coding agents, including OpenClaw-based agents, inside NVIDIA's OpenShell sandbox environment. It gives developers filesystem, network, and process isolation for agent workloads while supporting multiple inference backends, among them local inference via Ollama, which lets a model run on the developer's own GPU rather than sending prompts to a cloud API [2][3]. To let a sandboxed, containerized agent reach the Ollama service running on the host machine, NemoClaw starts Ollama with `OLLAMA_HOST=0.0.0.0:11434`, binding the API to every network interface on the machine rather than restricting it to the loopback address. Ollama's API does not require authentication on that port, and non-loopback bindings additionally cause the service to skip its Host-header validation, a check specifically designed to stop browsers from reaching local services. That validation exists in upstream Ollama specifically because of an earlier, publicly disclosed DNS-rebinding vulnerability, CVE-2024-

28224, which Ollama patched in v0.1.29 in March 2024; NemoClaw's non-loopback binding does not merely omit a protection but actively reintroduces the exposure that fix was meant to close [1]. The only remaining defense is a CORS policy, which was not built to withstand the browser trick that this research exploited [1][2][3].

That trick, DNS rebinding, is a long-known browser weakness that has resurfaced repeatedly against local development servers and IoT devices bound to permissive addresses. An attacker registers a domain and initially points it at a server they control; once a victim's browser loads the attacker's page and passes any origin checks tied to that domain, the attacker's DNS server changes the domain's resolution to `127.0.0.1`. The browser continues to treat subsequent requests from that page as same-origin, because same-origin policy keys off the domain name rather than the IP address it currently resolves to, so JavaScript on the page can now issue requests that land on the victim's own machine [1][2][3]. Because NemoClaw's Ollama instance skips Host validation on its exposed interface and its CORS check passes once the rebound origin matches the expected host, the rebinding page gains full, unauthenticated access to the local Ollama API purely because the victim opened a tab.

Security Analysis

The attack chain researchers demonstrated proceeds in four stages. First, the victim, running NemoClaw with the vulnerable Ollama configuration, visits an attacker-controlled webpage; nothing about the page needs to look suspicious, since the exploit runs entirely in background JavaScript. Second, the page's domain rebinds from the attacker's server to `127.0.0.1` after the initial page load, at which point the browser's outbound requests reach the victim's local Ollama service while the browser still considers them same-origin. Third, the attacker's script calls the unauthenticated `/api/show` endpoint to retrieve the target model's existing chat template, the Go `text/template` definition that controls how conversation turns, including system prompts, are rendered into the text the model actually receives at inference time. Fourth, the script calls `/api/create` to re-upload a modified version of that template with attacker-controlled text appended, so the poisoned template is now the one Ollama serves for every subsequent request against that model [1][2][3].

This is a materially different failure mode from ordinary prompt injection, where malicious text arrives inside a single session and can, in principle, be filtered or overridden by a well-designed system prompt. Template-level poisoning sits underneath the system-prompt layer: the template is applied to every message at inference time regardless of what system prompt a client supplies afterward, so the injected instructions persist across conversations, across reboots, and across any client that talks to that Ollama instance, all while remaining invisible to anyone reviewing the conversation transcript or the agent's own logged system prompt [2][3]. Researchers noted that instructions embedded this way could plausibly

direct an agent to recommend insecure code, suppress security warnings it would otherwise raise, or exfiltrate data, though the published research focused on demonstrating the poisoning mechanism itself rather than cataloguing every downstream misuse an attacker might attempt [2].

The same unauthenticated access also enables a set of less subtle abuses: enumerating installed models, retrieving the machine's hostname and any system prompts already cached on the host, deleting installed models, pulling arbitrarily large models to exhaust local disk space, and forcing sign-outs from linked ollama.com accounts [1][2]. In this Initiative's assessment, the severity here comes less from any single capability and more from what a NemoClaw-hosted agent typically has permission to touch. Agentic coding tools of this kind are routinely granted access to source control, package registries, and cloud credentials so they can act on a developer's behalf; a poisoned model retains all of that access while now operating under attacker-supplied instructions that neither the developer nor downstream reviewers of committed code have any obvious way to detect [1][2][3]. OpenShell's sandboxing isolates the agent's filesystem and process space, but it does not appear to address, and the published research does not indicate it was intended to address, an external attacker reaching a network-exposed host service through the victim's own browser, which is exactly the gap this vulnerability exploits.

NVIDIA's remediation covers macOS and Linux but, as of publication, does not extend to Windows or WSL. NemoClaw v0.0.35 corrected the binding behavior on macOS and Linux, and a later build, v0.0.106, refuses to start unless it can confirm Ollama is bound to loopback only. That startup check, however, does not reach the code path used on Windows and WSL, where v0.0.34 added only a warning rather than enforcing loopback binding, leaving those platforms exposed at the time of public disclosure on August 25, 2026 [1][3]. No active exploitation had been reported as of that date, and NVIDIA has since published a support advisory pointing customers to the fix [3]; the Recommendations section below details the interim mitigations organizations running NemoClaw on Windows or WSL should apply until an enforced fix ships for those platforms.

Recommendations

Immediate Actions

Organizations running NemoClaw should confirm they are on v0.0.35 or later on macOS and Linux hosts, and on Windows or WSL should not rely on the warning added in v0.0.34; instead, manually reconfigure Ollama to bind to `127.0.0.1` only, or place it behind a token-authenticated reverse proxy, until NVIDIA ships an enforced fix for those platforms. Security teams should also inspect the output of `/api/show` on developer machines against the vendor-shipped baseline template to check for signs of prior tampering, since a poisoned template would otherwise go unnoticed indefinitely.

Short-Term Mitigations

Host-based firewall rules should block inbound connections to port 11434, and browser-originated requests to local inference ports specifically, on any machine running local model inference. Teams should implement chat-template integrity verification, hashing the deployed template against a known-good baseline as part of routine endpoint checks, since Host-header and CORS validation alone have proven insufficient against DNS rebinding. The asset-inventory and allowlisting discipline this Initiative has recommended for MCP servers applies equally here: local inference endpoints exposed by agent deployment wrappers like NemoClaw should be tracked and reviewed with the same rigor as any other network-exposed AI component [4].

Strategic Considerations

This disclosure is best understood as one instance of a broader pattern rather than an isolated NVIDIA defect. Defaults chosen to simplify container-to-host connectivity, binding a service to all interfaces so a container can reach it, recur across the agentic AI tooling ecosystem, and browser-based techniques like DNS rebinding turn any such default into a remotely reachable vulnerability without the attacker ever touching the victim's network directly. Security architects should incorporate localhost-exposure and DNS-rebinding scenarios into standard threat modeling for agentic AI deployments rather than treating them as generic web application issues out of scope for AI risk assessments, and should recognize that sandboxing an agent's execution environment does not substitute for authenticating and network-isolating the inference service that agent depends on.

CSA Resource Alignment

This incident sits squarely within the network-exposure and poisoning attack dimensions the Cloud Security Alliance has already documented for agentic AI infrastructure. Security researchers have termed this general class of browser-reachable, permissive-network-binding bypass a "0.0.0.0-day" pattern, a term coined by Oligo Security's 2024 research into browser-to-localhost attacks [8], and NemoClaw's exposure is a direct, real-world instance of that pattern applied to a local inference server rather than to a browser-facing web application. CSA's MCP Attack Surface: Tool Poisoning and IDE Auto-Execution [4] documents a structurally related failure in the same developer-tooling ecosystem: leading IDEs auto-execute MCP servers with developer-level OS privileges and no process isolation, letting a poisoned tool description redirect agent behavior without the user's awareness. Organizations

that have already inventoried MCP configuration risk per that paper's guidance should extend the same review discipline, treating any network-exposed local service as a supply-chain asset, to local inference ports like NemoClaw's.

CSA's Miasma and IronWorm: Self-Replicating Worms Targeting AI Credentials [5] documents a related consequence of the same underlying trust model: self-replicating supply-chain worms that emerged from the npm ecosystem in mid-2026 explicitly targeted developer AI coding tool credentials, exploiting the same broad access, source control, package registries, cloud credentials, that a poisoned NemoClaw agent would also retain. Both cases illustrate that any technique compromising an AI coding agent, whether a supply-chain worm harvesting its credentials or, as here, a poisoned inference template redirecting its behavior, inherits the full scope of developer-level trust and access the agent was granted.

Where CSA has not yet published research specific to local inference server exposure, the standing frameworks provide the applicable structure. MAESTRO (Agentic AI Threat Modeling) [6] should be used to model the local inference layer as a distinct threat surface, since it sits below the agent's own orchestration logic and is not addressed by sandboxing the agent process itself. The AI Controls Matrix (AICM) v1.1 [7] provides the applicable network security and identity and access management control objectives for restricting inference-service exposure and enforcing authentication on locally hosted AI components, and should be the default control framework organizations map this remediation work against rather than relying on CCM alone.

References

[1] The Hacker News. "A Malicious Webpage Could Poison Your Local AI Model Behind NVIDIA NemoClaw." The Hacker News, August 25, 2026.

[2] Cyera Research (Oasis Security). "Drive-By Agent Hijacking: One Website Visit, Persistent Model Poisoning." Cyera, August 25, 2026.

[3] CSO Online. "NemoClaw's AI can be poisoned through a browser tab." CSO Online, August 25, 2026.

[4] Cloud Security Alliance. "MCP Attack Surface: Tool Poisoning and IDE Auto-Execution." Cloud Security Alliance, July 1, 2026.

[5] Cloud Security Alliance. "Miasma and IronWorm: Self-Replicating Worms Targeting AI Credentials." Cloud Security Alliance, June 2026.

[6] Cloud Security Alliance. "MAESTRO: Agentic AI Threat Modeling Framework." Cloud Security Alliance, February 6, 2025.

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

[8] Oligo Security. "0.0.0.0 Day: Exploiting Localhost APIs From the Browser." Oligo Security, August 2024.

[9] Rapid7. "CVE-2026-65105." Rapid7 Vulnerability & Exploit Database, 2026.

[10] TechCrunch. "Cyera agrees to acquire Oasis Security for $1B to safeguard proliferating AI agents." TechCrunch, July 28, 2026.

