Inside NemoClaw

An Architectural Analysis of NVIDIA's Enterprise Security Stack for Autonomous AI Agents

Adnan Tanveer (Addy)
Speedrun AI Labs
Sydney, Australia
speedrunlab.ai

March 17, 2026 | Paper Version 1.0, submitted to arXiv cs.CR

NOTE ON SCOPE: This paper documents the state of NVIDIA NemoClaw and OpenShell as accessed on 17 March 2026, one day after Jensen Huang's GTC 2026 keynote announcement. Both projects are in active development (NemoClaw is labelled 'alpha'). Implementation details, APIs, and configurations described here may change in future releases. The architectural principles (six-layer security decomposition, provider-agnostic inference routing, kernel-level sandbox isolation) remain applicable regardless of version. Readers should verify version compatibility before applying these findings to their own deployments.

Abstract

NVIDIA NemoClaw, announced at GTC 2026 on 16 March by CEO Jensen Huang, is, to our knowledge based on publicly available sources as of March 2026, the first major vendor implementation of enterprise-grade security for OpenClaw autonomous AI agents. This paper presents a detailed architectural analysis derived from direct inspection of the NemoClaw and OpenShell open-source codebases.

We decompose the system into six constituent security layers: filesystem isolation via Linux Landlock LSM, syscall filtering via seccomp BPF, network namespace isolation with veth pairs, an application-layer HTTP CONNECT proxy with per-binary OPA/Rego policy evaluation, optional Layer 7 TLS inspection, and inference routing abstracted behind the OpenAI-compatible API standard. For each layer, we document the implementation details, the dependency chain, and the design decisions that enable the security properties.

We make three contributions: (1) we provide the first detailed public decomposition of the NemoClaw/OpenShell security architecture; (2) we document the complete dependency chain, confirming that the core security stack has zero NVIDIA hardware dependencies; and (3) we analyse the policy lifecycle including hot-reload mechanics and last-known-good behaviour. Our analysis is intended to support enterprises evaluating agent security strategies and researchers working on sandboxing and policy frameworks for agentic AI systems.

Keywords: *agentic AI, autonomous agents, OpenClaw, NemoClaw, OpenShell, sandboxing, Landlock, seccomp, network namespaces, OPA, Rego, enterprise security, NVIDIA, GTC 2026*

1. Introduction

The emergence of autonomous AI agents capable of executing code, accessing file systems, and communicating over networks has created a fundamentally new security challenge for enterprises. Unlike traditional chatbot interfaces where the AI generates text for human review, agentic systems take direct action on behalf of users, with access to sensitive corporate resources, production systems, and external services.

OpenClaw, the open-source agent platform described by NVIDIA CEO Jensen Huang at GTC 2026 as 'the operating system for personal AI' and characterised in his keynote as having surpassed Linux in adoption speed [7], exemplifies both the potential and the risk. Its creator, Peter Steinberger, was subsequently hired by OpenAI, and the platform was acquired. At GTC 2026, Huang announced NemoClaw, an enterprise security stack designed to make OpenClaw deployments trustworthy, scalable, and safe for corporate environments [7].

NemoClaw represents, to our knowledge based on publicly available sources as of March 2026, the first major vendor implementation of comprehensive security controls for an agentic AI platform. While academic research has explored agentic AI security theoretically, including taxonomy papers such as Chhabra et al. [3] and practitioner frameworks like the OWASP Agentic Security Initiative Top 10 [4], NemoClaw and its underlying runtime OpenShell provide the first production-grade open-source implementation combining kernel-level process isolation, policy-driven network control, and inference routing into a single deployable stack.

This paper presents a detailed source code analysis of NemoClaw and OpenShell. Both projects are Apache 2.0 licensed and publicly available. Our contribution is to document the security architecture at a level of detail not available in NVIDIA's existing documentation, providing enterprises and researchers with the information needed to evaluate, extend, and learn from this implementation.

2. Methodology and Evidence Base

This analysis is based on direct inspection of two open-source repositories: the NemoClaw repository (github.com/NVIDIA/NemoClaw) and the OpenShell repository (github.com/NVIDIA/OpenShell), both accessed on 17 March 2026. We examined source code across all crates, configuration files, policy definitions, architecture documentation, install scripts, and test suites. Supplementary information was drawn from NVIDIA's official developer documentation at docs.nvidia.com/nemoclaw/latest/ [9], the OpenShell developer blog post [11], the NemoClaw GTC 2026 press release [7], and Jensen Huang's GTC 2026 keynote delivered on 16 March 2026 [13].

Where we report implementation details (e.g., specific Rust crate dependencies, policy enforcement mechanisms, or compilation flags), these are derived from direct code inspection and are cited to specific source files. Where we report NVIDIA's strategic positioning or adoption claims, these are attributed to NVIDIA's published statements. The analysis reflects the state of both codebases as of the date of access; subsequent releases may introduce changes. All code references use paths relative to the respective repository roots.

3. System Overview

NemoClaw is not a standalone runtime. It is a thin orchestration layer, implemented as an OpenClaw plugin in TypeScript with a Python blueprint runner, that delegates all security enforcement to NVIDIA OpenShell. Understanding the system requires examining both components and their interaction.

3.1 NemoClaw: The Orchestration Layer

NemoClaw consists of a TypeScript plugin (nemoclaw/src/) that registers CLI commands under the 'openclaw nemoclaw' namespace, and a Python blueprint runner (nemoclaw-blueprint/orchestrator/runner.py) that orchestrates sandbox creation through OpenShell CLI commands. The plugin registers an NVIDIA NIM model provider supporting Nemotron 3 Super 120B (131K context window), Nemotron Ultra 253B, Nemotron Super 49B v1.5, and Nemotron 3 Nano 30B. The blueprint system follows a four-stage lifecycle: resolve the artifact from the container registry (ghcr.io/nvidia/nemoclaw-blueprint), verify its SHA-256 digest, plan the deployment resources, and apply through the OpenShell CLI. The plugin stays intentionally thin; orchestration logic lives in the versioned blueprint and evolves on its own release cadence [9].

3.2 OpenShell: The Security Runtime

OpenShell is a Rust-based runtime that provides the actual security enforcement. The codebase is organised into eight crates: openshell-sandbox (process isolation and proxy), openshell-server (gateway and API), openshell-cli (command-line interface), openshell-core (shared types), openshell-policy (policy types), openshell-providers (inference provider discovery), openshell-router (inference routing), and openshell-tui (terminal user interface). The runtime operates inside a Docker container running a k3s Kubernetes cluster (v1.35.2-k3s1), with sandbox pods managed by a custom CRD controller in the agent-sandbox-system namespace [10].

4. Security Architecture: A Six-Layer Decomposition

The NemoClaw/OpenShell security architecture enforces isolation through six distinct layers, each addressing a different category of risk. The layers are applied in a specific sequence during sandbox startup: privilege dropping, filesystem isolation, and syscall filtering occur in the child process pre-exec phase (after fork(), before exec()), while network isolation and proxy-based policy enforcement operate at runtime. This section documents each layer based on direct source code inspection of the repositories accessed on 17 March 2026.

Layer	Mechanism	What It Protects	Applied	Mutable?
1. Filesystem	Landlock LSM	File read/write allowlist	Pre-exec	No (kernel)
2. Syscall	seccomp BPF	Socket domain blocking	Pre-exec	No
3. Network	netns + veth	Forces traffic through proxy	Creation	No
4. Policy (L4)	CONNECT + OPA/Rego	Per-connection binary-bound allow/deny	Runtime	Yes (hot-reload)
5. Policy (L7)	TLS MITM + OPA	HTTP method/path restrictions	Runtime	Yes (hot-reload)
6. Inference	inference.local + router	Credential isolation from agents	Runtime	Yes (refresh)

Table 1: Six-layer security decomposition of the NemoClaw/OpenShell stack. All information derived from source code inspection of the openshell-sandbox crate.

4.1 Filesystem Isolation via Landlock LSM

Filesystem access is restricted using Linux Landlock, a security module available since kernel 5.13. The implementation (crates/openshell-sandbox/src/sandbox/linux/landlock.rs) creates a Landlock ruleset at ABI V2 that grants AccessFs::from_read permissions to read-only paths and AccessFs::from_all permissions to read-write paths. All other filesystem paths are denied by default. The ruleset is applied via restrict_self(), which is irrevocable for the calling process and all its descendants.

The NemoClaw default policy (nemoclaw-blueprint/policies/openclaw-sandbox.yaml) permits read-only access to /usr, /lib, /proc, /dev/urandom, /app, /etc, and /var/log, with read-write access limited to /sandbox, /tmp, and /dev/null. When TLS termination is active, /etc/openshell-tls is automatically added to the read-only list. A compatibility mode (best_effort vs hard_requirement) determines whether the sandbox continues without filesystem isolation or aborts if Landlock is unavailable on the host kernel.

4.2 Syscall Filtering via seccomp BPF

The seccomp filter (crates/openshell-sandbox/src/sandbox/linux/seccomp.rs) targets SYS_socket, inspecting argument 0 (the address family domain). It blocks AF_NETLINK, AF_PACKET, AF_BLUETOOTH, and AF_VSOCK unconditionally. In Block mode, AF_INET and AF_INET6 are additionally blocked. In Proxy mode, AF_INET/AF_INET6 are allowed because the sandboxed process must connect to the HTTP CONNECT proxy over the veth pair. The filter uses prctl(PR_SET_NO_NEW_PRIVS) before application, preventing privilege escalation. Three network modes determine enforcement: Block (all sockets denied), Proxy (inet allowed, routed through proxy), and Allow (no seccomp filter applied).

4.3 Network Namespace Isolation

Network isolation is implemented through Linux network namespaces (crates/openshell-sandbox/src/sandbox/linux/netns.rs). A new namespace is created with a UUID-derived name (sandbox-{8-char-uuid}) and a veth pair on a private subnet (10.200.0.0/24). The host side (10.200.0.1) runs the HTTP CONNECT proxy; the sandbox side (10.200.0.2) has a default route to the host IP. This topology ensures the sandboxed process can only reach the proxy, never the internet directly.

Iptables bypass detection rules are installed inside the namespace: seven rules in order allow proxy traffic, allow loopback, allow established connections, then log and reject TCP bypass attempts, then log and reject UDP bypass attempts including DNS. A background /dev/kmsg monitor (bypass_monitor.rs) parses iptables LOG output to extract the destination, port, protocol, and UID of offending processes, providing forensic-grade attribution. If namespace creation fails, startup fails (fail-closed behaviour), ensuring either full isolation is active or the sandbox does not run.

4.4 Layer 4 Policy: HTTP CONNECT Proxy with OPA

The HTTP CONNECT proxy (crates/openshell-sandbox/src/proxy.rs) is the central enforcement point. For each CONNECT request, the proxy resolves TCP peer identity via /proc, computes SHA-256 hashes for trust-on-first-use (TOFU) verification via BinaryIdentityCache, walks the ancestor chain via /proc/{pid}/status PPid fields, and builds a NetworkInput combining binary identity with the target host and port.

The OPA engine (crates/openshell-sandbox/src/opa.rs) uses the regorus crate, a pure-Rust Rego evaluator with no external OPA daemon dependency. Policy rules are compiled into the binary via include_str!('sandbox-policy.rego'). The engine evaluates network_action rules returning 'allow' (with matched policy name) or 'deny' (with human-readable reason). After OPA allows a connection, SSRF protection validates resolved DNS addresses are not internal (RFC 1918, loopback, link-local, IPv4-mapped IPv6), preventing hostname-based attacks against internal infrastructure.

4.5 Layer 7 Inspection

For endpoints with protocol-aware inspection, the proxy performs TLS man-in-the-middle using an ephemeral CA generated at startup via the rcgen crate (l7/tls.rs). Leaf certificates are generated per-hostname and cached in a CertCache. Inside the decrypted tunnel, HTTP requests are parsed (supporting both Content-Length and chunked Transfer-Encoding) and evaluated against per-request OPA rules restricting HTTP methods and URL paths. Access presets (e.g., 'read-only' expanding to GET/HEAD/OPTIONS, 'full' allowing all methods) simplify policy authoring [10].

4.6 Inference Routing and Credential Isolation

When the agent sends a request to the virtual host inference.local, the proxy bypasses OPA and intercepts the connection. It TLS-terminates the client, detects inference API patterns (POST /v1/chat/completions, POST /v1/messages, GET /v1/models), strips credential headers (Authorization, x-api-key), and forwards via openshell-router. The router rewrites the Authorization header with the route's API key and replaces the 'model' field in the JSON body with the route's configured model value.

This design ensures agents never see real API keys; credentials exist only in the supervisor process, outside the sandboxed environment. Response streaming uses chunked transfer encoding for low time-to-first-byte. The handler loops for HTTP keep-alive. Route configuration is refreshed periodically from the gateway without restarting the sandbox.

5. Inference Provider Architecture

NemoClaw ships four inference profiles in blueprint.yaml: 'default' (NVIDIA cloud at integrate.api.nvidia.com/v1), 'ncp' (NVIDIA Cloud Partner with dynamic endpoint), 'nim-local' (self-hosted NIM container using the OpenAI provider type), and 'vllm' (local vLLM at localhost:8000/v1 with a dummy credential). The onboard wizard (nemoclaw/src/commands/onboard.ts) additionally supports experimental Ollama (at host.openshell.internal:11434/v1) and custom endpoints.

OpenShell's provider crate ships discovery modules for Anthropic, Claude, Codex, GitHub, GitLab, NVIDIA NIM, OpenAI, OpenCode, Outlook, and a generic provider. Each implements a ProviderPlugin trait performing credential discovery via environment variables. The NVIDIA provider (nvidia.rs) is purely an API key discovery mechanism; it contains no hardware detection, no CUDA calls, and no driver checks. The system architecture diagram (architecture/system-architecture.md) explicitly shows LM Studio and vLLM as supported self-hosted backends alongside cloud providers. The security properties documented in Sections 4.1 through 4.6 are independent of which inference provider is selected.

6. Dependency Analysis

The OpenShell Rust workspace (Cargo.toml) depends on standard open-source crates: tokio (async), tonic/prost (gRPC), axum/hyper (HTTP), rustls (TLS, notably not OpenSSL), clap (CLI), nix/libc (Unix), regorus (OPA/Rego), russh (SSH), sqlx (database), and kube (Kubernetes). Platform-specific dependencies (landlock v0.4, seccompiler v0.5) are gated behind cfg(target_os = "linux"). On non-Linux platforms, the sandbox module logs a warning and continues without kernel-level isolation.

A recursive search (grep -ri) of the entire OpenShell repository for terms including 'cuda', 'nvidia-smi', 'nvml', 'libcuda', 'nvidia-driver', 'gpu_device', 'cuda_runtime', 'nvcuda', and 'libnvidia' across all Rust, TOML, YAML, shell, and Python files returned exactly two results, both in test code: an e2e GPU test (e2e/python/test_sandbox_gpu.py, marked @pytest.mark.gpu) validating optional GPU passthrough, and a unit test (crates/openshell-cli/src/run.rs) verifying the CLI does not request GPU for images like 'cuda-toolkit:latest'. No core crate references any NVIDIA hardware dependency.

6.1 GPU Support as Optional Feature

GPU support in OpenShell exists as an opt-in feature triggered by the --gpu flag when creating a sandbox. Per the OpenShell README: 'OpenShell can pass host GPUs into sandboxes for local inference, fine-tuning, or any GPU workload.' This feature requires NVIDIA drivers and the NVIDIA Container Toolkit on the host. The core sandboxing, policy enforcement, inference routing, and all security features documented in Sections 4.1 through 4.6 operate without any GPU hardware or NVIDIA-specific software.

6.2 Platform Support

OpenShell's official support matrix (docs/reference/support-matrix.md) lists: Linux x86_64 and aarch64 (Supported), macOS Apple Silicon via Docker Desktop (Supported), and Windows WSL 2 (Experimental). Prebuilt binaries ship for x86_64-unknown-linux-musl, aarch64-unknown-linux-musl, and aarch64-apple-darwin. On macOS, Linux kernel security modules run inside Docker Desktop's Linux VM, as documented: 'On macOS, these kernel modules run inside the Docker Desktop Linux VM, not on the host kernel.'

7. Policy Lifecycle and Hot-Reload

Static domains (filesystem_policy, landlock, process) are applied once in pre-exec and are immutable. The gateway rejects updates to static fields and rejects network mode changes. Dynamic domains (network_policies) are hot-reloadable: a background task polls every 10 seconds (configurable via OPENSHELL_POLICY_POLL_INTERVAL_SECS), builds a new regorus engine through full validation, and atomically swaps it behind a Mutex. If validation fails, the previous engine is untouched (last-known-good behaviour).

Policy revisions use monotonically increasing versions, deterministic SHA-256 hashing (sorted map keys to avoid protobuf serialisation non-determinism), and four statuses: pending, loaded, failed, superseded. Idempotent updates are supported: if the deterministic hash matches the latest revision, the gateway returns the existing version without creating a new revision.

8. Related Work

Chhabra et al. [3] provide a taxonomy of agentic AI threats. OWASP [4] catalogues the top 10 agent security risks. IBM [5] frames agents as 'digital insiders' requiring sequestration. CELLMATE [6] proposes HTTP-level browser agent sandboxing with agent sitemaps. Microsoft's failure mode taxonomy identifies agent compromise, workflow manipulation, and memory poisoning. These are primarily theoretical or attack-focused.

NemoClaw/OpenShell represents, to our knowledge as of March 2026, the first production-grade open-source implementation combining kernel isolation, application-layer policy enforcement, and inference routing for autonomous agents. This paper bridges theory and practice by documenting how one such system is actually built.

9. Discussion

Fail-closed network isolation. If namespace creation fails, the sandbox refuses to start, ensuring the six-layer guarantee is all-or-nothing rather than silently degraded.

Binary identity binding. TOFU SHA-256 verification of every binary and its ancestor chain enables per-binary policies (e.g., 'only git may access github.com'), preventing data exfiltration through otherwise-allowed endpoints.

Credential isolation via inference.local. Even a fully compromised agent cannot extract real API keys, as credentials exist only in the supervisor process outside the sandbox boundary.

Provider-agnostic inference. By abstracting behind the OpenAI-compatible API standard, NVIDIA decoupled the security stack from any specific backend, with significant deployment flexibility implications.

9.1 Limitations

This analysis has limitations: we have not performed adversarial sandbox escape testing; we have not benchmarked policy evaluation overhead; we have not compared security properties on macOS via Docker Desktop versus native Linux; and both codebases are in active development (NemoClaw is labelled 'alpha'). Implementation details may change in subsequent releases.

10. Future Work

We identify three directions: (1) experimental validation of security properties on macOS via Docker Desktop on Apple Silicon, including stability under sustained agent workloads; (2) adversarial evaluation of the sandbox boundary through proxy, filesystem, and inference routing layers; (3) performance benchmarking of policy evaluation overhead across inference providers and model sizes. These form the basis of a planned follow-up paper.

11. Conclusion

We have presented a detailed architectural analysis of NVIDIA NemoClaw and OpenShell, decomposing the enterprise agent security stack into six layers based on direct source code inspection. The architecture combines kernel-level isolation (Landlock LSM, seccomp BPF, network namespaces) with application-layer policy enforcement (OPA/Rego with binary identity binding) and protocol-aware

inspection (TLS MITM with per-request evaluation). The inference routing layer is explicitly provider-agnostic. GPU support is optional.

As autonomous agents become prevalent in enterprise environments, these security patterns provide a reference architecture for the emerging field of agentic AI security. Our goal with this publication is to contribute to the community's understanding of practical agent security implementation. Speedrun AI Labs believes in creating more value than we extract. If you are evaluating enterprise agent security, we hope this analysis saves you the reverse-engineering time it cost us.

Acknowledgements

The author thanks NVIDIA for open-sourcing NemoClaw and OpenShell under the Apache 2.0 licence, enabling the independent analysis presented in this paper. Thanks also to Peter Steinberger for creating OpenClaw, and to the broader open-source community contributing to agentic AI security tooling.

References

[1] NVIDIA Corporation. NemoClaw: OpenClaw Plugin for OpenShell. GitHub, March 2026. github.com/NVIDIA/NemoClaw

[2] NVIDIA Corporation. OpenShell. GitHub, 2025-2026. github.com/NVIDIA/OpenShell

[3] Chhabra, A. et al. "Agentic AI Security: Threats, Defenses, Evaluation, and Open Challenges." arXiv:2510.23883, October 2025.

[4] OWASP Foundation. OWASP Agentic Security Initiative Top 10, 2026.

[5] IBM. "Agentic AI Security Guide." IBM Think, February 2026.

[6] Meng, L. "CELLMATE: Sandboxing Browser AI Agents." UC San Diego, arXiv:2512.12594, 2025.

[7] NVIDIA Corporation. "NVIDIA Announces NemoClaw for the OpenClaw Community." GlobeNewswire, 16 March 2026.

[8] Steinberger, P. OpenClaw. openclaw.ai, 2026.

[9] NVIDIA Corporation. NemoClaw Developer Guide. docs.nvidia.com/nemoclaw/latest/, March 2026.

[10] NVIDIA Corporation. OpenShell Architecture Documentation. github.com/NVIDIA/OpenShell/architecture/, 2025-2026.

[11] NVIDIA Corporation. "Run Autonomous, Self-Evolving Agents More Safely with NVIDIA OpenShell." NVIDIA Developer Blog, 2026.

[12] NVIDIA Corporation. "Secure Long Running AI Agents with OpenShell on DGX Station." build.nvidia.com, 2026.

[13] Huang, J. NVIDIA GTC 2026 Keynote. San Jose, CA, 16 March 2026.