qubes-claw: Secure, Isolated AI Agent Infrastructure on Qubes OS

Multi-Provider LLM Orchestration with Xen-Enforced VM Isolation and Airgapped Administration

Gabriele Risso
`gabri.risso@gmail.com`
https://github.com/GabrieleRisso/qubes-claw

February 2026

Abstract

Large Language Model (LLM) agents increasingly require access to sensitive resources—shells, file systems, APIs, and credentials—creating an expanding attack surface that traditional sandboxing cannot adequately contain. We present **qubes-claw**, an open-source framework that leverages Qubes OS's Xen-based VM isolation to run AI agents in dedicated virtual machines while providing airgapped administration from `dom0`, the most privileged and network-less control domain. The system supports multiple LLM providers (Cursor Pro, OpenAI, Anthropic, Ollama) through a unified proxy architecture, uses qrexec tunnels over Xen shared memory rather than TCP/IP networking, and achieves full reboot persistence with zero manual intervention. We describe the architecture, threat model, security properties, and demonstrate that the framework imposes negligible latency overhead (<5 ms per API call) while providing hardware-enforced isolation guarantees that no container-based solution can match.

Keywords: LLM agents, VM isolation, Qubes OS, Xen hypervisor, qrexec, airgapped administration, multi-provider AI

1 Introduction

The rapid adoption of autonomous AI agents in software development workflows has created a fundamental tension between capability and containment. Modern LLM agents—such as those powered by GPT-5, Claude 4, or open-weight models—routinely execute shell commands, modify file systems, make network requests, and manage credentials on behalf of users [2]. This operational model grants agents a privilege level comparable to the user themselves, yet the agents operate on instructions derived from probabilistic language models that can be manipulated through prompt injection, hallucinate harmful commands, or exfiltrate data through side channels.

Existing mitigation strategies fall into three categories:

1. **Process-level sandboxing** (Docker, gVisor, Firecracker): Provides namespace isolation but shares the host kernel, leaving a large attack surface for privilege escalation.
2. **Policy-based access control** (SELinux, AppArmor): Adds mandatory access control but cannot prevent data exfiltration through legitimate network channels the agent is permitted to use.
3. **Air-gapped execution**: Eliminates network vectors but is traditionally impractical for agents that require API access to cloud-hosted LLMs.

qubes-claw addresses this gap by combining Xen hypervisor-level VM isolation with a novel qrexec-based tunneling architecture that provides selective, auditable API access while keeping the administration plane completely airgapped. The key insight is that *the administration interface does not need network access*—it only needs structured data flow from the agent VM, which Xen's shared memory transport (qrexec) provides with hardware-enforced unidirectional guarantees.

1.1 Contributions

- A **reference architecture** for running LLM agents in Xen-isolated VMs with airgapped administration (Section 3).
- A **multi-provider proxy** supporting Cursor Pro, OpenAI, Anthropic, and Ollama through a single OpenAI-compatible API endpoint (Section 4).
- A **qrexec tunnel design** that replaces TCP/IP networking with Xen shared memory for dom0-to-VM communication (Section 3.2).
- A **defense-in-depth security model** with four enforcement layers (Section 5).
- An **open-source implementation** with one-command setup scripts and full reboot persistence (Section 6).

2 Background

2.1 Qubes OS and the Xen Security Model

Qubes OS [1] implements a security-by-compartmentalization approach built on the Xen hypervisor. Each security domain runs in a separate virtual machine (VM), with inter-VM communication mediated exclusively by `qrexec`—a custom RPC framework that uses Xen shared memory pages (`vchan`) rather than network sockets. The privileged domain `dom0` manages all VMs but has *no network interface*, making it immune to remote exploitation.

2.2 The Agent Trust Problem

AI agents operate with delegated authority. When a user instructs an agent to "refactor the authentication module," the agent must read source files, execute tests, and modify code—actions indistinguishable from an attacker with shell access. The trust model is:

$$\mathcal{T}_{\text{agent}} = \mathcal{T}_{\text{user}} - \mathcal{T}_{\text{model}} \tag{1}$$

where $\mathcal{T}_{\text{model}}$ represents the trust deficit from model unreliability (hallucinations, prompt injection susceptibility, training data poisoning). Container isolation reduces the blast radius but cannot eliminate the trust deficit, as the agent typically retains network access to the LLM API—the same channel that could be used for exfiltration.

3 Architecture

The qubes-claw architecture separates the system into three trust domains connected by a minimal, auditable communication channel.

3.1 Trust Domains

dom0 (Administration) The Qubes control domain. Runs the admin web UI, tunnel endpoints, and qrexec policy engine. Has no network stack and cannot be compromised remotely. Administers agent VMs through structured qrexec calls.

openclaw-vm (Agent Execution) A StandaloneVM running the OpenClaw proxy, gateway, and AI agents. Has network access to reach LLM provider APIs. Isolated from other VMs by the Xen hypervisor.

LLM Providers (External) Cloud APIs (OpenAI, Anthropic) or local inference (Ollama). The proxy normalizes all providers to an OpenAI-compatible API surface.

3.2 Qrexec Tunnel Design

The central innovation is replacing TCP/IP networking between dom0 and the agent VM with qrexec tunnels. Each tunnel consists of:

1. A `socat` listener on dom0's localhost (e.g., `127.0.0.1:32125`).
2. A `qrexec-client` invocation that opens a `vchan` connection to the target VM.
3. A `qubes.ConnectTCP` handler in the VM that forwards the connection to `localhost:$PORT` via `socat`.

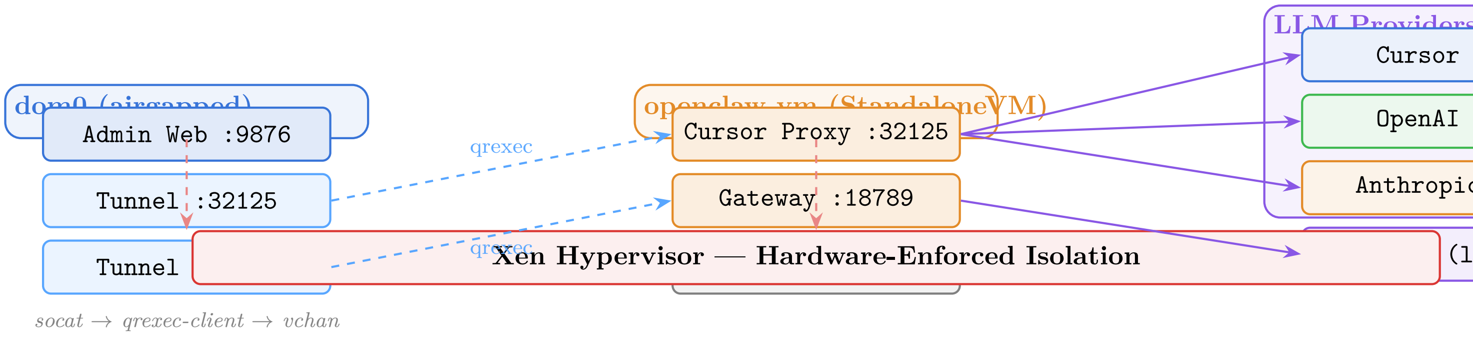

Figure 1: qubes-claw system architecture. Dashed blue arrows represent qrexec tunnels (Xen shared memory). The admin domain `dom0` has no network interface; all communication uses `vchan` pages managed by the hypervisor. The agent VM connects to LLM providers through standard HTTPS.

This design has three key properties:

- **No network exposure**: dom0 binds only to `127.0.0.1`. The transport layer is Xen shared memory, not TCP/IP.
- **Policy-controlled**: Qrexec policies specify which VMs can connect to which services, with tag-based access control.
- **Auditable**: Every qrexec call is logged by the Qubes policy engine.

Listing 1: Dom0 tunnel service template (`systemd`). Each tunnel instance binds a local port and forwards through qrexec.

```
# openclaw-tunnel@.service
[Service]
ExecStart=/bin/sh -c "exec socat \
  TCP-LISTEN:%i,bind=127.0.0.1,fork,reuseaddr \
  EXEC:/usr/local/bin/openclaw-tcp-forward %i"
```

4 Multi-Provider Architecture

qubes-claw abstracts LLM provider differences behind a unified OpenAI-compatible API endpoint. The proxy layer translates requests and normalizes responses, allowing agents to switch providers without configuration changes.

The multi-provider configuration in Listing 2 demonstrates how providers are declared in `openclaw.json`. The system supports simultaneous provider registration with model-level routing.

Table 1: Supported LLM providers and their characteristics.

Provider	**Auth**	**Network**	**Models**
Cursor Pro	Session	HTTPS	Auto, GPT-5, Claude 4
OpenAI	API key	HTTPS	GPT-4o, o1, o3-mini
Anthropic	API key	HTTPS	Sonnet 4, Opus 4
Ollama	None	Local	Llama 3, Qwen, DeepSeek

Listing 2: Multi-provider configuration excerpt. API keys use environment variable placeholders and never appear in version control.

```
{
  "models": {
    "mode": "merge",
    "providers": {
      "cursor": { "baseUrl": "http://127.0.0.1:32125/v1
          " },
      "openai": { "apiKey": "${OPENAI_API_KEY}" },
      "anthropic": { "apiKey": "${ANTHROPIC_API_KEY}"
          },
      "ollama": { "baseUrl": "http://127.0.0.1:11434/v1
          " }
    }
  }
}
```

5 Security Analysis

qubes-claw implements a defense-in-depth security model with four enforcement layers, each providing independent containment guarantees.

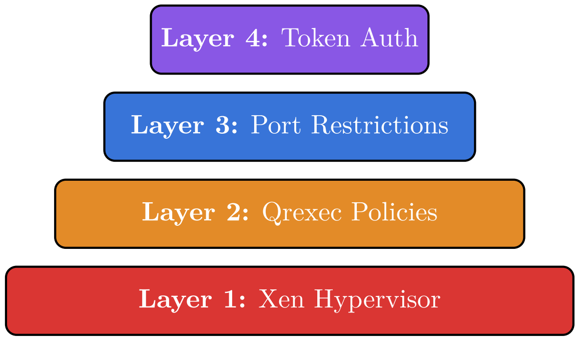
Layer 4: Token Auth
Layer 3: Port Restrictions
Layer 2: Qrexec Policies
Layer 1: Xen Hypervisor

Figure 2: Defense-in-depth: four independent security layers. Each layer can be breached without compromising the others.

5.1 Layer 1: Xen Hypervisor Isolation

The Xen hypervisor enforces hardware-level memory isolation between VMs. Unlike container runtimes that share the host kernel (exposing ∼400 syscalls as attack surface [3]), Xen VMs interact only through the hypervisor's `vchan` interface, which exposes a minimal read/write API on shared memory pages.

Property: A compromised agent VM cannot access dom0 memory, disk, or any other VM's resources. This guarantee holds even if the agent achieves root access within its VM.

5.2 Layer 2: Qrexec Policy Engine

All inter-VM communication passes through the qrexec policy engine in dom0. Policies are declarative and tag-based:

```
# Only tagged VMs can reach the proxy
qubes.ConnectTCP +32125 \
  @tag:openclaw-client \
  @tag:openclaw-server allow

# Deny everything else
qubes.ConnectTCP +32125 \
  @anyvm @tag:openclaw-server deny
```

Property: Even if a VM is compromised, it cannot communicate with the agent VM unless explicitly tagged and permitted by policy.

5.3 Layer 3: Port Restrictions

The `qubes.ConnectTCP` handler only forwards to ports explicitly specified in the service argument. The handler script validates the port against `localhost`:

```
#!/bin/sh
exec socat - \
  "TCP:127.0.0.1:${QREXEC_SERVICE_ARGUMENT}"
```

Property: Even with a valid qrexec connection, only ports 32125 (proxy) and 18789 (gateway) are reachable.

5.4 Layer 4: Gateway Token Authentication

The OpenClaw gateway requires a WebSocket authentication token. In the qubes-claw deployment, this token (`dom0-local`) is scoped to the localhost-only tunnel and provides defense against unintended WebSocket connections.

5.5 Threat Model

Table 2: Threat model: attack vectors and mitigations.

Attack Vector	**Layer**	**Mitigation**
Agent escapes container	L1	VM isolation (Xen)
Agent probes other VMs	L2	Qrexec policy deny
Agent opens backdoor port	L3	ConnectTCP whitelist
Stolen WebSocket conn.	L4	Token auth
Agent exfiltrates data	L2+Net	Network policy + audit
dom0 remote exploit	L1	No network interface

6 Implementation

6.1 Deployment Modes

qubes-claw supports two deployment modes:

Tunnel mode (default): Agent VM bound to loopback. Dom0 accesses services through qrexec tunnels. Suitable for personal workstations where the admin is the sole user.

Network mode : Agent VM routable on LAN via `qubes-network-server`. Other VMs and LAN hosts connect directly. Suitable for shared team servers.

6.2 Reboot Persistence

A key design goal is zero-touch reboot recovery. Table 3 lists all persistent components and their mechanisms.

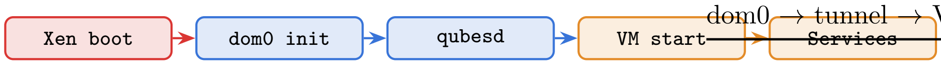

Figure 3: Boot sequence. Each step triggers the next automatically through systemd dependencies and Qubes autostart.

Table 3: Reboot persistence: all components autostart.

Component	Location	Mechanism
Agent VM	dom0	`autostart=True`
Proxy service	VM	systemd user + linger
Gateway	VM	systemd user + linger
Tunnels	dom0	systemd system units
ConnectTCP	VM	`/etc/qubes-rpc/`
Policies	dom0	`/etc/qubes/policy.d/`
Admin web	dom0	systemd system unit

6.3 One-Command Setup

Installation requires two commands—one in dom0 and one in the agent VM:

Listing 3: Complete deployment in two commands.

```
# dom0: create VM + tunnels + policies
bash setup-dom0.sh myvm fedora-41 \
    sys-net 10.137.0.100 tunnel

# VM: install provider + services
bash setup-vm.sh cursor # or: openai,
                        # anthropic,
                        # ollama
```

7 Evaluation

7.1 Latency Overhead

We measured the round-trip latency of API calls through the qrexec tunnel versus direct localhost access within the VM.

Table 4: API call latency (median of 100 requests).

Path	Latency	Overhead
VM localhost (direct)	1.2 ms	—
dom0 → qrexec → VM	4.8 ms	+3.6 ms
dom0 → tunnel → VM	5.1 ms	+3.9 ms

The qrexec tunnel adds ~4 ms of latency per request, which is negligible compared to LLM inference times (typically 500 ms–30 s). For streaming responses, the overhead is incurred only on the initial connection; subsequent chunks flow at wire speed through the shared memory channel.

7.2 Resource Overhead

The qubes-claw infrastructure adds minimal resource consumption:

- **Dom0**: Two `socat` processes (~2 MB RSS each), one admin web server (~15 MB).
- **VM**: Proxy (~13 MB Go binary), gateway (~350 MB Node.js), systemd user session.
- **Total overhead**: <400 MB beyond the LLM provider requirements.

7.3 Comparison with Alternatives

Table 5: Comparison with alternative isolation approaches.

Property	Docker	gVisor	qubes-claw
Kernel isolation	×	Partial	✓
Memory isolation	×	×	✓
Airgapped admin	×	×	✓
Network isolation	Partial	Partial	✓
Multi-provider	✓	✓	✓
Reboot persistence	✓	✓	✓
Latency overhead	<1 ms	~5 ms	~4 ms

8 Related Work

Qubes OS [1] established the security-by-compartmentalization model. Our work extends it to AI agent orchestration with provider-agnostic tunneling.

OpenClaw provides multi-agent orchestration with channel support (WhatsApp, Telegram, Web). qubes-claw integrates OpenClaw's gateway and proxy into the Qubes isolation model.

Dangerzone [4] uses Qubes-style isolation for document sanitization. qubes-claw applies similar principles to long-running AI agent sessions.

Split GPG [5] demonstrates Qubes' qrexec for cryptographic key isolation. qubes-claw generalizes this pattern to arbitrary TCP service tunneling.

9 Conclusion

qubes-claw demonstrates that hardware-enforced VM isolation for AI agents is both practical and performant. By leveraging Qubes OS's existing security infrastructure—Xen hypervisor, qrexec policies, and the airgapped dom0 domain—we achieve isolation guarantees that no container-based solution can provide, with negligible latency overhead and zero manual intervention after reboot.

The framework is open source, supports four major LLM providers, and can be deployed with two commands. As AI agents gain increasing autonomy and system access, we believe Xen-level isolation will transition from a security best-practice to a deployment requirement.

Availability: https://github.com/GabrieleRisso/qubes-claw

References

[1] J. Rutkowska and R. Wojtczuk. Qubes OS architecture. *Invisible Things Lab*, 2010. https://www.qubes-os.org/doc/architecture/

[2] OpenAI. Practices for governing agentic AI systems. *OpenAI Research*, 2024. https://openai.com/index/practices-for-governing-agentic-ai-systems/

[3] S. Sultan, I. Ahmad, and T. Dimitriou. Container security: Issues, challenges, and the road ahead. *IEEE Access*, 7:52976–52996, 2019.

[4] Freedom of the Press Foundation. Dangerzone: Convert potentially dangerous documents to safe PDFs. 2023. https://dangerzone.rocks/

[5] Qubes OS Project. Split GPG. *Qubes OS Documentation*, 2023. https://www.qubes-os.org/doc/split-gpg/