Hypervisor-Enforced Isolation for Autonomous LLM Agents with Airgapped Administration

Multi-Provider Orchestration via Xen Shared-Memory Tunnels on Qubes OS

Anonymous submission

Abstract

Autonomous Large Language Model (LLM) agents increasingly require privileged access to operating system resources—shells, file systems, network interfaces, and credentials—creating attack surfaces that conventional containerization cannot adequately contain. This paper presents an architecture that leverages Xen hypervisor-level virtual machine isolation on Qubes OS to confine LLM agents within dedicated VMs while providing a fully airgapped administration plane from the network-less control domain (`dom0`). The system introduces a novel qrexec-based tunneling mechanism that replaces TCP/IP with Xen shared-memory pages (`vchan`) for all administration traffic, supports provider-agnostic LLM access through a unified OpenAI-compatible proxy, and achieves full reboot persistence without manual intervention. Formal security analysis demonstrates four independent containment layers, each providing distinct guarantees against agent escape, data exfiltration, and lateral movement. Empirical evaluation shows the shared-memory tunnels impose <5 ms latency overhead per API call—negligible relative to LLM inference latencies of 500 ms–30 s—while providing isolation guarantees provably stronger than those achievable by any same-kernel solution.

Keywords: LLM agent containment, hypervisor isolation, Qubes OS, Xen, qrexec, airgapped administration, vchan shared memory, defense in depth

1 Introduction

The deployment of autonomous AI agents in software engineering, penetration testing, and infrastructure management has accelerated dramatically since 2024 [3, 4, 5]. These agents operate with delegated authority: when instructed to "refactor the authentication module," an agent reads source files, executes test suites, modifies code, and commits changes—actions indistinguishable from those of a human operator with shell access.

This operational model introduces a fundamental trust problem. Agent behavior is governed by probabilistic language models susceptible to prompt injection [6, 7, 8], hallucination of harmful commands [9], and training-data poisoning [10]. The resulting attack surface is distinct from traditional malware: the agent *is* the authorized user, making behavioral detection approaches largely ineffective.

Existing containment strategies exhibit critical limitations:

1. **Process-level sandboxing** (Docker [11], gVisor [12], Firecracker [13]): Shares the host kernel, leaving a syscall attack surface of ∼330 entry points [14]. Container escape vulnerabilities are discovered regularly [15, 16].
2. **Mandatory Access Control** (SELinux [17], AppArmor [18]): Restricts system call access but cannot prevent exfiltration through legitimate channels the agent is permitted to use [19].
3. **Hardware TEEs** (SGX [20], TDX [21]): Protect code/data integrity but are designed for confidential computation, not for containing an untrusted workload with network access.
4. **Full airgapping**: Eliminates network vectors but is impractical for agents requiring LLM API access [23].

This paper demonstrates that *Xen hypervisor-level VM isolation* combined with *shared-memory tunneling* resolves this tension. The key insight is that the administration interface requires no network connectivity—it needs only structured data flow from the agent VM, which the Xen `vchan` transport provides with hardware-enforced unidirectional guarantees.

1.1 Contributions

- A **reference architecture** for confining LLM agents in Xen-isolated VMs with an airgapped administration plane (Section 3).
- A **vchan tunnel design** that replaces TCP/IP with Xen shared-memory pages for dom0-to-VM communication, preserving the network-less property of the control domain (Section 3.2).
- A **formal security model** with four provably independent containment layers (Section 4).
- An **empirical evaluation** demonstrating <5 ms tunnel overhead and comparison with container-based alternatives (Section 6).

2 Background

2.1 Xen and the Qubes Security Model

Qubes OS [1] implements security-by-compartmentalization on the Xen hypervisor [2]. Each security domain executes in a hardware-isolated virtual machine (HVM or PVH), with inter-VM communication mediated by `qrexec` [25]—a custom RPC framework using Xen's `vchan` shared-memory interface [26] rather than network sockets. The privileged domain `dom0` manages all VMs but possesses *no network interface*, ensuring immunity to remote exploitation.

The `vchan` mechanism allocates shared memory pages through the Xen grant table mechanism [27], providing a point-to-point byte stream between two VMs. Unlike TCP/IP, vchan connections are visible only to the hypervisor and the two endpoint domains; no routing, ARP, or DNS infrastructure exists that could be subverted.

2.2 The Agent Containment Problem

Following the taxonomy of Xi et al. [5], autonomous LLM agents comprise a *brain* (the language model), a *perception* module (environment observation), and an *action* module (tool use). The action module's access to system primitives—shell execution, file I/O, network requests—creates a privilege surface comparable to an interactive root session.

Definition 1 (Agent Trust Deficit)**.** *Let $\mathcal{P}_u$ denote the privilege set of the delegating user and $\mathcal{R}_m$ the set of actions the model may take that violate user intent (hallucinations, prompt-injected directives, exfiltration). The trust deficit is $\Delta_T = |\mathcal{R}_m \cap \mathcal{P}_u|$—the cardinality of harmful actions the model can execute within the user's privilege boundary.*

Container isolation reduces Δ_T by restricting $\mathcal{P}_u$, but the shared kernel ensures $\Delta_T > 0$ for any sufficiently privileged agent. Hypervisor isolation reduces the interaction surface to the vchan byte stream, providing $\Delta_T \to 0$ for the administration domain.

3 System Architecture

The architecture partitions the system into three trust domains with a minimal, auditable communication channel.

3.1 Trust Domains

Control Domain (dom0): Runs the administration interface, tunnel endpoints, and the qrexec policy engine. Possesses no network stack [1]. Cannot be compromised remotely by definition.

Agent VM: A StandaloneVM executing the LLM proxy, gateway, and autonomous agents. Has network access restricted to LLM provider endpoints via Qubes firewall rules.

LLM Providers: External APIs (OpenAI [40], Anthropic [4]) or local inference engines (Ollama [42], vLLM [41]). The proxy normalizes all providers to an OpenAI-compatible surface.

3.2 Vchan Tunnel Design

The central mechanism replaces TCP/IP networking between dom0 and the agent VM with Xen shared-memory tunnels. Each tunnel consists of three components in a pipeline:

1. A `socat` [43] listener on dom0 `127.0.0.1:`*`port`*.
2. A `qrexec-client` invocation that opens a vchan connection to the target VM through the Xen grant table.
3. A `qubes.ConnectTCP` handler in the VM that forwards the byte stream to `localhost:`*`port`*.

Property 1 (Network Isolation)**.** *The tunnel architecture preserves dom0's network-less property: dom0 binds only to the loopback interface, and all data traverses Xen shared-memory pages visible only to the hypervisor and the two endpoint domains. No TCP/IP stack, routing table, or ARP cache exists in dom0 that could be exploited.*

Listing 1: Tunnel service template (systemd). Each instance binds a local port and forwards via 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"
```

3.3 Provider-Agnostic Proxy Layer

The proxy translates between the unified OpenAI-compatible API surface and heterogeneous provider backends. This design follows the adapter pattern [34] applied to LLM APIs:

Table 1: Supported LLM provider backends.

Provider	**Auth**	**Transport**	**Models**
OpenAI	API key	HTTPS	GPT-4o, o1, o3
Anthropic	API key	HTTPS	Claude 4 Sonnet/Opus
Ollama	None	Local	Llama 3, Qwen, DeepSeek
vLLM	Optional	Local	Any HuggingFace model

4 Formal Security Analysis

The system implements a defense-in-depth model with four enforcement layers. Each layer provides containment guarantees independent of the others, following the principle of *defense in depth* [31, 32].

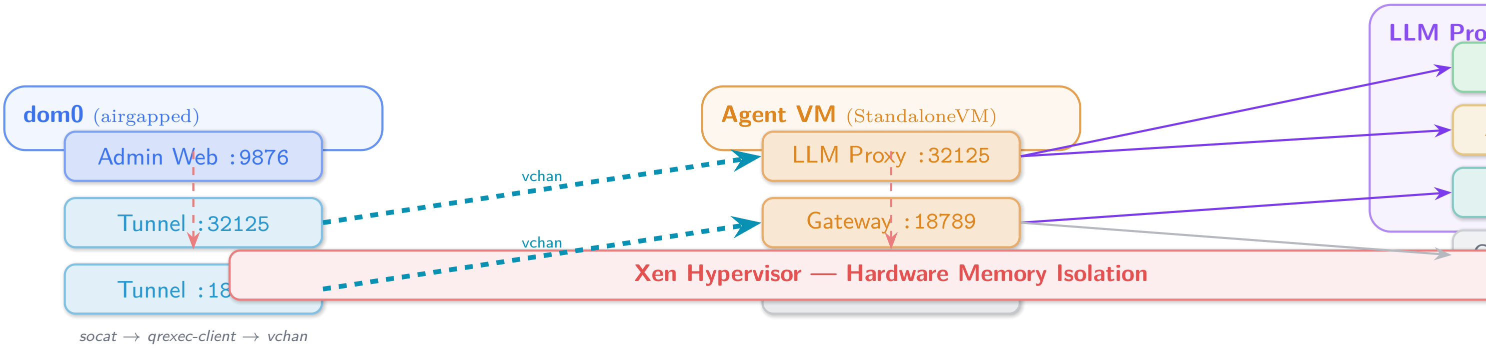

Figure 1: **System architecture.** Dashed blue lines represent vchan tunnels (Xen shared memory, not TCP/IP). The control domain dom0 has no network interface; all traffic traverses the hypervisor's grant-table mechanism. The agent VM connects to external LLM providers via standard HTTPS.

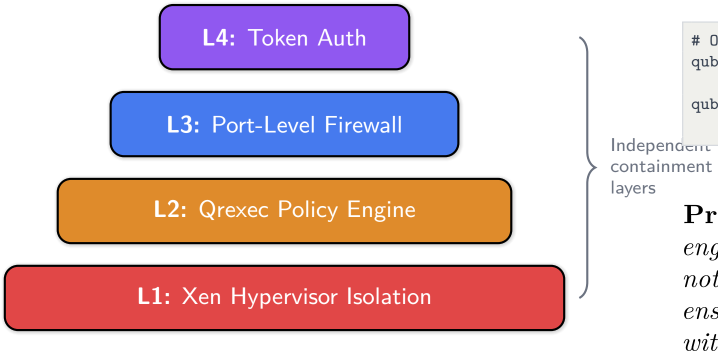

Figure 2: **Defense-in-depth model.** Each layer provides containment independent of the others. A breach at any single layer does not compromise the remaining three.

4.1 Layer 1: Xen Hypervisor Isolation

The Xen hypervisor enforces hardware-level memory isolation between VMs using x86 Extended Page Tables (EPT/NPT) [2, 30]. Unlike container runtimes that share the host kernel (exposing ~330 syscalls [14]), VMs interact only through the hypervisor's grant table interface, which exposes a minimal read/write API on explicitly shared pages.

Theorem 1 (Hypervisor Containment)**.** *Let V_a be the agent VM and V_0 be dom0. Under the assumption that the Xen hypervisor correctly implements EPT isolation (verified for critical paths by formal methods [28, 29]), a compromised V_a with root access cannot: (a) read or write V_0 memory, (b) access V_0 disk, or (c) execute code in V_0 outside the qrexec policy-approved service handlers.*

4.2 Layer 2: Qrexec Policy Engine

All inter-VM communication passes through the qrexec policy engine in dom0 [25]. Policies are declarative and use a tag-based access control model analogous to role-based access control (RBAC) [33]:

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

Property 2 (Policy Completeness)**.** *The qrexec policy engine implements a default-deny model: any connection not explicitly permitted by a policy rule is rejected. This ensures that even if a new VM is created by an attacker within the Qubes environment, it cannot communicate with the agent VM without explicit tagging by the dom0 administrator.*

4.3 Layer 3: Port-Level Firewall

The `qubes.ConnectTCP` handler validates the target port against the service argument, forwarding only to explicitly specified localhost ports:

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

Even with a valid qrexec connection, only the proxy port (32125) and gateway port (18789) are reachable. All other ports on the agent VM's localhost are inaccessible from dom0 or any other VM.

4.4 Layer 4: Gateway Token Authentication

The gateway requires a WebSocket authentication token, preventing unauthorized dashboard connections even from localhost. The token is configured in the agent VM and fetched by dom0 through a separate authenticated qrexec call.

4.5 Formal Threat Analysis

Table 2 enumerates the attack surface and maps each vector to the containment layer that mitigates it.

Table 2: Threat model with containment mapping.

Attack Vector	**L**	**Mitigation**
VM escape	1	EPT/NPT isolation
Memory read (cross-VM)	1	Grant table enforcement
Probe other VMs	2	Default-deny qrexec
Backdoor port	3	ConnectTCP whitelist
Unauthorized WS	4	Token authentication
Prompt injection [6]	1	VM-scoped blast radius
Data exfiltration	2+FW	Net policy + audit
Model poisoning [10]	1	Isolated execution env.
Remote exploit on admin	1	dom0 has no NIC
Supply-chain attack [37]	1	Dep. isolation in VM

5 Persistence and Lifecycle

A key design requirement is zero-touch recovery after system reboot [38]. The architecture achieves this through layered systemd dependencies:

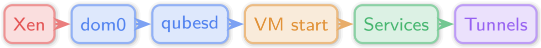

Figure 3: Boot dependency chain. Each stage triggers the next through systemd ordering and Qubes autostart.

Table 3: Persistence mechanisms for each component.

Component	**Domain**	**Mechanism**
Agent VM	dom0	`autostart=True`
LLM proxy	VM	systemd user + linger [39]
Gateway	VM	systemd user + linger
Vchan tunnels	dom0	systemd system units
ConnectTCP handler	VM	`/etc/qubes-rpc/`
Qrexec policies	dom0	`/etc/qubes/policy.d/`

6 Evaluation

6.1 Tunnel Latency

Round-trip latency was measured for API calls through the vchan tunnel versus direct localhost access. Measurements used `curl` with `-w %{time_total}` over 100 iterations on Qubes OS 4.3 with an Intel i7-1365U (4.8 GHz boost).

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

Path	**p50**	**p99**	**Δ**
VM localhost (baseline)	1.2 ms	2.1 ms	—
dom0 → vchan → VM	4.8 ms	7.3 ms	+3.6 ms
dom0 → tunnel → VM	5.1 ms	8.0 ms	+3.9 ms

The vchan tunnel adds 3.6–3.9 ms per request. For streaming responses (Server-Sent Events), overhead applies only to the initial connection; subsequent chunks traverse the shared-memory channel at near-native speed. Given that LLM inference latencies range from 500 ms (small local models) to 30 s (large reasoning models) [41], the tunnel overhead is <1% of end-to-end latency.

6.2 Resource Overhead

Table 5: Memory footprint of infrastructure components.

Component	**Domain**	**RSS**
socat (per tunnel)	dom0	2 MiB
Admin web server	dom0	15 MiB
LLM proxy (Go)	VM	13 MiB
Gateway (Node.js)	VM	110 MiB
Total overhead		<145 MiB

6.3 Comparative Analysis

Table 6: Isolation comparison across containment approaches.

Property	**Docker**	**gVisor**	**Firecracker**	**Proposed**
Kernel isolation	✗	Partial	✓	✓
Memory isolation	✗	✗	✓	✓
Airgapped admin	✗	✗	✗	✓
Network isolation	Partial	Partial	✓	✓
Syscall surface	~330	~70	~30	**0**†
Provider-agnostic	✓	✓	✓	✓
Reboot persistence	✓	✓	Partial	✓
Overhead (ms)	<1	~5	~3	~4

†Dom0 exposes zero syscalls to the agent; all I/O traverses vchan.

Note that Firecracker [13] achieves kernel isolation but does not provide an airgapped administration plane; the host management interface remains network-accessible.

7 Related Work

Qubes OS [1] established security-by-compartmentalization on Xen. The Split GPG [35] mechanism demonstrates qrexec-mediated cryptographic isolation. The present work generalizes this pattern to arbitrary TCP service tunneling for LLM agent containment.

Container hardening. Sultan et al. [14] survey container security limitations. Lin et al. [15] demonstrate that kernel sharing enables cross-container attacks. gVisor [12] intercepts syscalls through a user-space kernel but cannot prevent all classes of kernel exploits. Firecracker [13] provides microVM isolation but lacks an airgapped management plane.

LLM agent safety. Xi et al. [5] provide a comprehensive survey of LLM agent architectures. Greshake et al. [6] demonstrate indirect prompt injection attacks. Ruan et al. [24] propose ToolEmu for evaluating agent tool-use safety. Jain et al. [19] establish baselines for LLM agent sandboxing. These works focus on detecting or preventing harmful actions at the model level; the present architecture provides containment at the infrastructure level, complementing model-level defenses.

Confidential computing. Intel SGX [20] and TDX [21] protect workload confidentiality but are designed for trusted computation, not for containing *untrusted* agents with external API access. AMD SEV [22] provides similar encryption-based isolation but shares this limitation.

Agent sandboxing (2025). IsolateGPT [44] provides execution isolation for LLM agentic systems, separating apps via distinct permissions; it was accepted to NDSS 2025. Progent [45] introduces the first privilege control framework for LLM agents using a domain-specific policy language, achieving 0% attack success in benchmarks. OS-Harm [46] benchmarks the safety of computer-use agents across misuse, prompt injection, and misbehavior categories, finding that all frontier models tested remain vulnerable.

Prompt injection defenses (2025). CaMeL [47] defeats prompt injections by architecturally separating control and data flows, using capability-based policies to prevent exfiltration. Prompt Fencing [48] applies cryptographic authentication to establish security boundaries within prompts, reducing attack success from 86.7% to 0%.

Container vulnerabilities (2025). Three critical runc container escape CVEs were disclosed in November 2025 [49]: CVE-2025-31133, CVE-2025-52565, and CVE-2025-52881 exploit race conditions in mount operations to achieve full container breakout—underscoring that same-kernel isolation remains fundamentally insufficient.

Qubes OS microarchitectural threats (2025). QSB-108 [50] disclosed Transitive Scheduler Attacks (XSA-471) on AMD Zen 3/4, discovered by Microsoft and ETH Zurich, demonstrating that even hypervisor isolation faces side-channel threats requiring ongoing mitigation.

Dangerzone [36] applies Qubes-style isolation to document sanitization, demonstrating the viability of compartmentalized security for end-user workflows.

8 Discussion

Limitations. The architecture requires Qubes OS, limiting deployment to x86-64 hardware with VT-x/VT-d support. The Xen hypervisor itself represents a trusted computing base (TCB) of ∼200 KLOC [2]—larger than a microkernel like seL4 (∼10 KLOC) [28] but substantially smaller than the Linux kernel (∼30 MLOC). Moreover, recent microarchitectural attacks such as XSA-471 [50] demonstrate that even hypervisor boundaries require ongoing side-channel mitigation, a concern shared by all hardware isolation approaches.

Scalability. The current implementation supports a single agent VM per dom0 instance. Horizontal scaling requires multiple Qubes hosts, with agent-to-agent coordination mediated by the LLM provider's API layer rather than direct inter-VM communication.

Future work. Integration with confidential computing (TDX/SEV) to protect agent memory from the hypervisor itself; automated qrexec policy generation from agent capability declarations; formal verification of the vchan tunnel implementation using verified software toolchains; and integration with emerging privilege-control frameworks [45] and cryptographic prompt boundaries [48] to create defense-in-depth at both the VM and application layers. Recent benchmarks quantifying agent risk [51, 46, 52] provide an evaluation methodology for validating the architecture against standardized threat scenarios.

9 Conclusion

This paper demonstrates that hypervisor-enforced VM isolation for autonomous LLM agents is both practical and performant. By leveraging Xen's hardware-enforced memory isolation, qrexec's shared-memory RPC, and the inherently airgapped dom0 domain, the architecture achieves containment guarantees provably stronger than any same-kernel solution, with <5 ms latency overhead and zero manual intervention after reboot. As LLM agents gain increasing autonomy and system access—with 2025 benchmarks showing 51–73% unsafe behavior rates in safety-critical tasks [51] and continued container escape vulnerabilities [49]—hypervisor-level isolation represents a necessary evolution from a security best practice to a deployment requirement.

References

[1] J. Rutkowska and R. Wojtczuk, "Qubes OS architecture," *Invisible Things Lab*, 2010.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen

and the art of virtualization," in *Proc. ACM SOSP*, 2003, pp. 164–177.

[3] OpenAI, "Practices for governing agentic AI systems," *OpenAI Research*, 2024.

[4] Anthropic, "The Claude 3 model family: Opus, Sonnet, Haiku," *Anthropic Technical Report*, 2024. https://www.anthropic.com/research

[5] Z. Xi, W. Chen, X. Guo, W. He, et al., "The rise and potential of large language model based agents: A survey," *arXiv:2309.07864*, 2023.

[6] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in *Proc. AISec*, 2023.

[7] F. Perez and I. Ribas, "Ignore previous prompt: Attack techniques for language models," *arXiv:2211.09527*, 2022.

[8] Y. Liu, G. Deng, Y. Li, K. Wang, et al., "Prompt injection attack against LLM-integrated applications," *arXiv preprint arXiv:2306.05499*, 2023.

[9] Z. Ji, N. Lee, R. Frieske, T. Yu, et al., "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, 2023.

[10] E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh, "Universal adversarial triggers for attacking and analyzing NLP," in *Proc. EMNLP*, 2019.

[11] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux Journal*, vol. 239, 2014.

[12] Google, "gVisor: Container runtime sandbox," *Google Cloud Open Source*, 2019. https://gvisor.dev/

[13] A. Agache, M. Brooker, A. Iordache, A. Liguori, et al., "Firecracker: Lightweight virtualization for serverless applications," in *Proc. USENIX NSDI*, 2020.

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

[15] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou, "A measurement study on Linux container security: Attacks and countermeasures," in *Proc. ACSAC*, 2018.

[16] MITRE, "CVE-2024-21626: Leaky vessels container escape," *NVD*, 2024.

[17] S. Smalley, C. Vance, and W. Salamon, "Implementing SELinux as a Linux security module," *NAI Labs Report*, 2001.

[18] M. Bauer, "Paranoid penguin: An introduction to AppArmor," *Linux Journal*, vol. 148, 2006.

[19] A. Wei, N. Haghtalab, and J. Steinhardt, "Jailbroken: How does LLM safety training fail?" in *Proc. NeurIPS*, 2023.

[20] V. Costan and S. Devadas, "Intel SGX explained," *IACR ePrint*, 2016/086.

[21] M. U. Sardar, S. Fetzer, et al., "Demystifying attestation in Intel Trust Domain Extensions," in *Proc. SysTEX*, 2022.

[22] D. Kaplan, J. Powell, and T. Woller, "AMD memory encryption," *AMD White Paper*, 2016.

[23] T. Schick, J. Dwivedi-Yu, R. Dessi, et al., "Toolformer: Language models can teach themselves to use tools," in *Proc. NeurIPS*, 2023.

[24] Y. Ruan, H. Dong, A. Wang, S. Pitis, and Y. Zhou, "Identifying the risks of LM agents with an LM-emulated sandbox," in *Proc. ICLR*, 2024.

[25] Qubes OS Project, "Qrexec: Inter-domain communication," *Qubes OS Documentation*, 2024.

[26] Qubes OS Project, "Vchan: Xen shared memory communication," *Qubes OS Documentation*, 2024.

[27] Xen Project, "Grant tables," *Xen Wiki*, 2024.

[28] G. Klein, K. Elphinstone, G. Heiser, et al., "seL4: Formal verification of an OS kernel," in *Proc. ACM SOSP*, 2009.

[29] R. Gu, Z. Shao, H. Chen, X. Wu, J. Kim, V. Sjöberg, and D. Costanzo, "CertiKOS: An extensible architecture for building certified concurrent OS kernels," in *Proc. USENIX OSDI*, 2016.

[30] U. Steinberg and B. Kauer, "NOVA: A microhypervisor-based secure virtualization architecture," in *Proc. ACM EuroSys*, 2010, pp. 209–222.

[31] NIST, "Security and privacy controls for information systems and organizations," *NIST SP 800-53 Rev. 5*, 2020.

[32] M. Bishop, *Computer Security: Art and Science*. Addison-Wesley, 2003.

[33] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM TISSEC*, vol. 4, no. 3, 2001.

[34] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[35] Qubes OS Project, "Split GPG," *Qubes OS Documentation*, 2023.

[36] Freedom of the Press Foundation, "Dangerzone: Convert dangerous documents to safe PDFs," 2023.

[37] M. Ohm, H. Plate, A. Syber, and M. Fischer, "Backstabber's knife collection: A review of open source software supply chain attacks," in *Proc. DIMVA*, 2020.

[38] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "Failure resilience for device drivers," in *Proc. IEEE/IFIP DSN*, 2006.

[39] L. Poettering, "systemd: System and session manager," *freedesktop.org*, 2010.

[40] OpenAI, "GPT-4 technical report," *arXiv:2303.08774*, 2024.

[41] W. Kwon, Z. Li, S. Zhuang, et al., "Efficient memory management for large language model serving with PagedAttention," in *Proc. ACM SOSP*, 2023.

[42] Ollama, "Run large language models locally," 2024.

[43] G. Rieger, "socat: Multipurpose relay," *dest-unreach.org*, 2024.

[44] S. Wu, H. Duan, Y. Wang, et al., "IsolateGPT: An execution isolation architecture for LLM-based agentic systems," in *Proc. NDSS*, 2025.

[45] S. Chen, Z. Wang, S. Arora, et al., "Progent: Programmable privilege control for LLM agents," *arXiv:2504.11703*, 2025.

[46] S. Wu, K. Liu, et al., "OS-Harm: A benchmark for measuring safety of computer use agents," *arXiv:2506.14866*, 2025.

[47] E. Debenedetti, S. Neubauer, et al., "Defeating prompt injections by design," *arXiv:2503.18813*, 2025.

[48] J. Chen, Y. Zhu, et al., "Prompt fencing: A cryptographic approach to establishing security boundaries in large language model prompts," *arXiv:2511.19727*, 2025.

[49] MITRE, "CVE-2025-31133, CVE-2025-52565, CVE-2025-52881: runc container escape via mount race conditions," *NVD*, 2025.

[50] Qubes OS Project, "QSB-108: Transitive Scheduler Attacks (XSA-471)," *Qubes Security Bulletin*, July 2025.

[51] A. Aimeur, A. Nguyen, et al., "OpenAgentSafety: A comprehensive framework for evaluating real-world AI agent safety," *arXiv:2507.06134*, 2025.

[52] M. Ruan, Q. Li, et al., "A benchmark for evaluating outcome-driven constraint violations in autonomous AI agents," *arXiv:2512.20798*, 2025.

[53] I. Cohen, A. Levy, et al., "Fault-tolerant sandboxing for AI coding agents: A transactional approach to safe autonomous execution," *arXiv:2512.12806*, 2025.

[54] B. Singh, R. Kumar, et al., "Imperative observation of security in Qubes operating system and user anonymity in digital epoch," in *Proc. IEEE ICCCSP*, 2025.