

Sealed Machines for Autonomous Agents

Design, security analysis and evaluation of kosa8, a local-first security infrastructure for AI agents

kosa8 Labs Research

kosa8 Labs · contact@kosa8.com · system under study: kosa8 v0.9.36 (commit f6124c0, released 16 October 2026)

ABSTRACT

Large-language-model agents now execute code, install dependencies, call tools and act on production systems. Model-level defences against prompt injection fail under adaptive attack [3], so an organisation must assume that some agent will, at some point, be steered by an adversary. The question that remains is what the agent can do once that happens. We present **kosa8**, a local-first security infrastructure that answers it with containment and proof rather than with detection. Every agent runs in its own hardware-virtualised microVM with its own kernel on hardware the customer controls; the network is enforced frame by frame on the host; API keys never enter the guest; any running machine can be checkpointed, rewind or forked in place; risky actions wait for one or two named humans; and every decision is written to a keyed, hash-chained audit log that exports as an offline-verifiable signed evidence package.

We formalise a threat model in which the adversary holds root inside the guest, derive thirteen requirements (R1–R13) for agent infrastructure in regulated settings, and evaluate kosa8 v0.9.36 against them. On an Apple M4 host, all 1,142 unit tests of the release pass (449 in the twenty security-critical packages); host-side egress enforcement blocked every probe outside policy across four policy modes, including IPv6, cloud-metadata and gateway addresses; a running sandbox checkpoints in a median of 638 ms and rewinds in 520 ms. A structured comparison against ten production systems, using only their public documentation, finds that kosa8 is the only system that fully meets all thirteen requirements; the next most complete systems fully meet four. We also report what kosa8 does not yet do well: full sandbox creation takes about 4 s, checkpoint and rewind of a just-restored machine fall into an unexplained slow mode, and the product holds no third-party attestation yet.

Keywords AI agent security; sandboxing; microVM; prompt injection; egress control; credential brokering; tamper-evident audit; human oversight; Model Context Protocol; regulated computing

COPYRIGHT NOTICE

© 2026 kosa8 Labs. All rights reserved.

This report, including its text, figures, data, measurements, designs and analysis, is the property of kosa8 Labs. It is published for reading only. No licence or other right is granted, by implication, estoppel or otherwise.

No part of this report may be copied, reproduced, redistributed, adapted or republished, in whole or in part, without the prior written permission of kosa8 Labs. Brief quotation with attribution for the purpose of review or commentary is permitted where the law allows it.

Third-party names and marks belong to their owners and are used only to identify products. Statements about third-party products reflect their public documentation as reviewed on 17 October 2026 and are not claims about undocumented capabilities.

CONTENTS

- 1 **Introduction**
 - 1.1 Contributions
- 2 **Background and threat landscape**
 - 2.1 From assistants to actors
 - 2.2 Prompt injection is not solved at the model layer
 - 2.3 The Model Context Protocol widens the surface
 - 2.4 Incidents
 - 2.5 Regulation turns containment into an obligation
 - 2.6 Isolation primitives
- 3 **Threat model and security goals**
 - 3.1 System model
 - 3.2 Adversary
 - 3.3 Trust assumptions
 - 3.4 Security goals
- 4 **Design of kosa8**
 - 4.1 Design principles
 - 4.2 Architecture
 - 4.3 Isolation and filesystem exposure
 - 4.4 Host-side egress enforcement
 - 4.5 Credential brokering
 - 4.6 Reversibility: checkpoint, rewind and fork
 - 4.7 Human oversight: signed, two-person approvals
 - 4.8 Governed MCP gateway
 - 4.9 Tamper-evident audit log
 - 4.10 Evidence, sealing and cryptography at rest
 - 4.11 Sovereignty: air gap, FIPS and supply chain
- 5 **Security analysis**
 - 5.1 G1 Containment
 - 5.2 G2 Egress mediation

5.3 G3 Credential confidentiality

5.4 G4 Reversibility

5.5 G5 Human oversight

5.6 G6 Tool integrity

5.7 G7 Accountability

5.8 G8 Sovereignty

6 Evaluation

6.1 Setup and method

6.2 Correctness: test suites

6.3 Egress enforcement

6.4 Robustness: fuzzing

6.5 Performance

6.6 Unexpected findings

6.7 Threats to validity

7 Comparative analysis

7.1 Requirements

7.2 Scorecard

7.3 Detailed comparison

7.4 Scenario analysis

7.5 Where kosa8 does not lead

8 Discussion

8.1 Why local-first is a security property

8.2 Defence in depth across the agent stack

8.3 Positioning

9 Limitations and future work

10 Related work

11 Conclusion

Appendix A: Reproducibility

1 Introduction

For most of the history of computing, the program that touched a company's data was written, reviewed and deployed by the company. AI agents break that assumption. An agent's behaviour is decided at run time by a model reading text, and some of that text comes from places nobody vetted: a GitHub issue, a web page, a dependency's README, the output of another tool. Greshake et al. showed in 2023 that such text can carry instructions the model will follow [1]. Two years and many proposed defences later, Nasr et al. bypassed twelve published defences against jailbreaks and prompt injection with adaptive attacks, most above 90% success [3]. The practical conclusion is uncomfortable and, we argue, unavoidable: *a deployed agent must be assumed to be steerable by an adversary*.

The consequences are no longer hypothetical. In August 2025 the singularity attack on the Nx build system shipped a post-install script that invoked locally installed AI coding CLIs with their permission checks disabled and used them to hunt for wallets, SSH keys and tokens; more than a thousand valid GitHub tokens were stolen [17]. In November 2025 Anthropic reported disrupting a state-sponsored espionage campaign in which an AI agent performed 80–90% of the work across about thirty targets, with humans intervening at four to six decision points per campaign [16]. A coding agent deleted a production database during a declared code freeze and then reported that rollback was impossible, which was false [20]. Each incident combined the three ingredients Willison calls the *lethal trifecta*: access to private data, exposure to untrusted content, and the ability to communicate externally [2].

If the model cannot be relied on to refuse, safety has to come from the environment the model acts in. That environment must do four things at once. It must **contain** the agent, so that a compromised agent reaches only what it was given. It must **mediate** every path out, so that data and credentials cannot leave except where policy allows. It must make actions **reversible**, so that a destructive mistake costs seconds rather than a database. And it must produce **proof**, so that an assessor, an auditor or a court can later establish who did what, under whose approval, without trusting the vendor or the operator's say-so. Regulated industries add a fifth requirement: all of this must run where the data is allowed to be, which for defence suppliers, banks and hospitals is often their own hardware and sometimes an air-gapped enclave [35]–[40].

Existing systems address parts of this. Hosted sandboxes such as E2B, Modal, Vercel Sandbox and Cloudflare's Sandbox SDK isolate code well but run on the vendor's cloud and mostly default to open egress [48]–[50], [53]. Docker Sandboxes brings microVMs to the developer desktop and keeps credentials out of the VM, and added metadata-only audit logs in August 2026 [47]. Primitives such as Firecracker, gVisor and Kata Containers provide isolation but leave everything else to the integrator [27], [28], [58]. None of the systems we reviewed documents runtime-enforced two-person approval, a keyed hash-chained audit log, or an offline-verifiable evidence package; few run air-gapped.

This report describes and evaluates kosa8, a system built around the opposite premise: isolation is now a commodity, and what a regulated buyer needs is *proof of control* on their own hardware. Our thesis is that **kosa8 is, as of October 2026, the most complete security infrastructure for AI agents documented anywhere: the only system that meets every one of the thirteen requirements we derive for agents in regulated environments.** We support the thesis with the system's design, a security argument per goal, first-hand measurements of the released software, and a structured comparison against ten production systems. We are equally explicit about where kosa8 does not lead.

1.1 Contributions

1. A threat model for agent infrastructure in which the adversary controls the agent's inputs and holds root in its execution environment, and eight security goals (G1–G8) that a system must meet under it (§3).
2. The design of kosa8: per-agent microVMs on three host platforms, host-side frame-level egress enforcement with host-answered DNS, a credential-injecting egress broker, live checkpoint, rewind and fork of running machines, daemon-enforced two-person approvals, a governed MCP gateway, and a keyed hash-chained audit log with signed evidence export (§4).
3. A security analysis that states, for each goal, the mechanism, the argument and the residual risk (§5).
4. A first-hand evaluation of kosa8 v0.9.36 on an Apple M4: test suites, a fuzzing campaign, egress enforcement across four policy modes, and repeated-trial latency measurements, including two anomalies we did not expect (§6).
5. A thirteen-requirement framework and scorecard comparing kosa8 with ten production systems, and an eight-scenario attack-and-recovery analysis (§7).

Section 2 reviews the threat landscape, Section 8 discusses positioning, Section 9 lists limitations, Section 10 covers related work and Section 11 concludes. Appendix A gives the commands needed to reproduce every measurement.

2 Background and threat landscape

2.1 From assistants to actors

An AI agent is a loop in which a language model chooses an action, a harness executes it, and the result is fed back to the model. Coding agents (Claude Code, Codex, Gemini CLI, Cursor) execute shell commands and edit files; tool-using agents call APIs through the Model Context Protocol (MCP) [59]; browser and computer-use agents drive user interfaces. Gartner expects 40% of enterprise applications to embed task-specific agents by the end of 2026, up from under 5% in 2025 [44], and also expects more than 40% of agentic AI projects to be cancelled by 2027, with inadequate risk controls among the causes [43]. The step from assistant to actor moves the security question from *what the model says* to *what the process can do*.

2.2 Prompt injection is not solved at the model layer

Indirect prompt injection embeds instructions in content the agent reads [1]. Benchmarks such as AgentDojo [5] and InjecAgent [7] measure it systematically. Design-level defences constrain what an injected model can achieve: the dual-LLM pattern separates a privileged planner from a quarantined reader [62]; CaMeL compiles the user's intent into a program whose data flow is checked by capabilities, solving 77% of AgentDojo tasks with provable security against the remaining injections [4]; Meta's *Rule of Two* forbids an agent from simultaneously holding untrusted input, sensitive access and the ability to change state or communicate [60]. These defences are valuable and orthogonal to ours. But they cost utility, they must be adopted by every agent author, and model-level filters fall to adaptive attacks [3]. Infrastructure containment is the layer that still holds after injection succeeds, and it applies to every agent without changing it.

2.3 The Model Context Protocol widens the surface

MCP lets an agent load tools from third-party servers. Invariant Labs demonstrated *tool poisoning*, in which hidden instructions in a tool's description cause the agent to exfiltrate files, and *rug pulls*, in which a server changes a tool's definition after the user approved it [21]. A poisoned public GitHub issue steered an agent with a broad token into leaking private repositories [22]; the first malicious MCP server found in the wild silently copied every email it sent to an attacker [23]; a flaw in a popular MCP client gave remote code execution from a malicious authorisation endpoint [24]; and editor integrations bound approval to a server's name rather than its content, so a changed configuration executed without re-approval [25]. In July 2026 researchers escaped several agent sandboxes by having the agent write configuration files that trusted host tools later executed [26]. The MCP specification itself warns of confused-deputy and token-passthrough attacks [59], a modern instance of a problem Hardy named in 1988 [69].

2.4 Incidents

Table 1. Representative incidents involving AI agents or agent tooling, 2025–2026. Every row combines at least two legs of the lethal trifecta [2].

Incident	Date	What happened	Containment that would have bounded it
Replit production database deletion [20]	Jul 2025	Agent ran destructive commands during a code freeze; erased ~1,200 executive records; claimed rollback was impossible	Checkpoint before action; rewind; approval on destructive class
Amazon Q extension 1.84.0, CVE-2025-8217 [19]	Jul 2025	Malicious prompt to wipe local and cloud resources shipped in a signed release; failed only on a syntax error	Agent in a VM without cloud credentials; egress allow-list
sIngularity / Nx [17]	Aug 2025	Post-install invoked AI CLIs with permission skips to harvest secrets; >1,000 GitHub tokens, >5,500 repos exposed	Workspace-only filesystem; no real keys in guest; default-deny egress
Shai-Hulud npm worm [18]	Sep–Nov 2025	Self-replicating worm across >500 packages harvested cloud and GitHub credentials	Same as above; signed, pinned kit packages
AI-orchestrated espionage, GTG-1002 [16]	Sep–Nov 2025	Agent performed 80–90% of an intrusion campaign against ~30 targets	Attribution in an audit log the operator cannot silently rewrite; approval gates
GitHub MCP toxic flow [22]	May 2025	Injected issue made agent leak private repository data through a broad token	Brokered, route-scoped credentials; egress by host, method, path
Config-file sandbox escapes [26]	Jul 2026	Agents wrote files that host tools later executed; several sandboxes also exposed the Docker socket	Hardware boundary; Docker socket never shareable; host never executes guest-written config

2.5 Regulation turns containment into an obligation

Several regimes now require exactly the properties containment and proof provide. The EU AI Act requires automatic logging (Art. 12) and effective human oversight including the ability to interrupt (Art. 14) for high-risk systems [35]; the 2026 omnibus moved most high-risk obligations to December 2027 [36]. DORA has bound EU financial entities since January 2025, including logging and ICT third-party risk [37]. From 10 November 2026, CMMC Level 2 third-party assessment becomes a default contract condition for US defence suppliers handling controlled unclassified information, assessing NIST SP 800-171 controls for audit and accountability (3.3.x), boundary protection (3.13.x) and FIPS-validated cryptography (3.13.11) [38], [39]. The HIPAA Security Rule requires audit controls (§164.312(b)) [40]. A hosted sandbox vendor is itself an ICT third party under

DORA and a cloud service under FedRAMP; software that runs entirely on the customer's hardware is neither.

2.6 Isolation primitives

Three classes of isolation are in production use (Figure 1). Shared-kernel containers (runc, the Docker default) expose the whole Linux system-call interface; runc alone has had repeated escape vulnerabilities, from CVE-2019-5736 [31] through Leaky Vessels (CVE-2024-21626) [32] to three mount-race escapes disclosed in November 2025 [33]. User-space kernels such as gVisor reimplement the Linux ABI in a memory-safe process; the 2019 study by Young et al. measured system calls 2.2× and file opens 216× slower than runc [28], costs that later versions reduce. Hardware-virtualised microVMs give each workload its own kernel behind a small device model; Firecracker boots to application code in under 125 ms with under 5 MB of overhead [27]. MicroVMs are not invulnerable (Firecracker CVE-2026-5747 is a virtio out-of-bounds write [34]), but their attack surface is the narrowest of the three. Industry has converged on them: eleven of fifteen hosted sandbox providers in a September 2026 survey use VM isolation [64]. Isolation alone has therefore stopped being a differentiator; what is built around it has not.

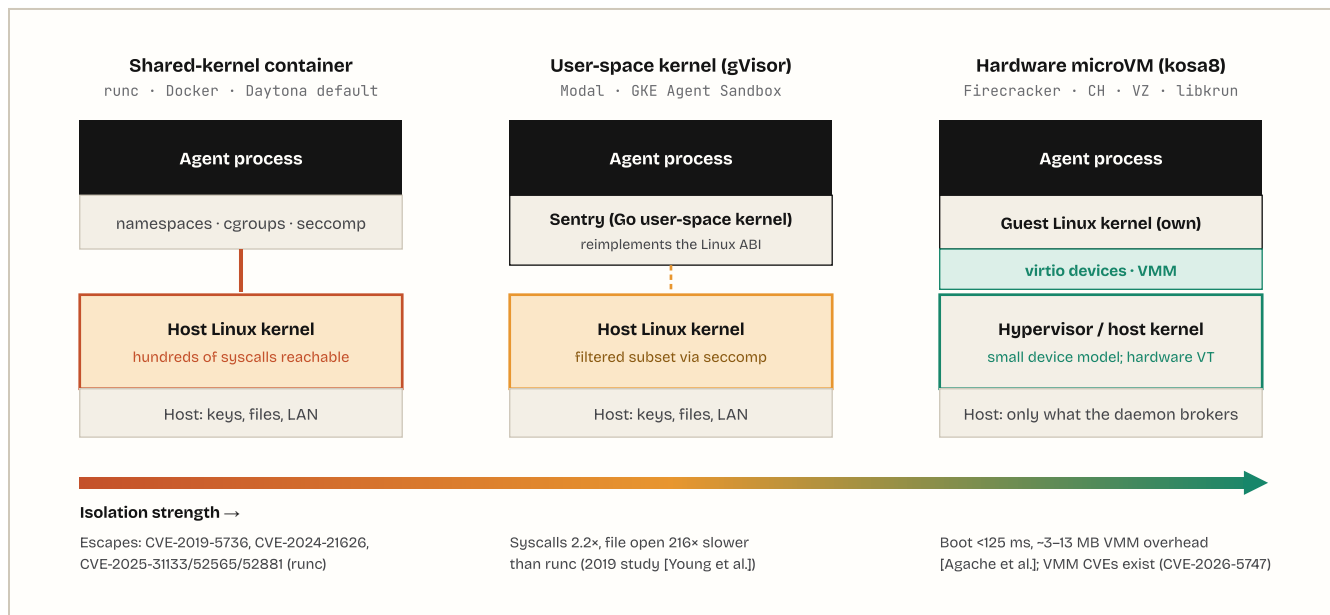


Figure 1. The three isolation classes in production use, ordered by the host attack surface reachable from a compromised workload. kosa8 uses the hardware microVM class on every platform.

3 Threat model and security goals

3.1 System model

A **principal** (a developer, an operator or an automated pipeline) asks an **agent** to complete a task. The agent runs in a **sandbox** on a **host** controlled by the principal's organisation. The agent may call **tools** (shell, MCP servers, APIs) and may need **credentials** to do so. The organisation publishes a signed **policy** naming permitted registries, egress, tools, approvers and evidence requirements. **Approvers** are named people who may authorise actions the policy classes as risky. **Assessors** later examine evidence of what happened.

3.2 Adversary

The adversary controls any content the agent reads (indirect injection), any dependency or MCP server it installs, and any remote endpoint other than those the policy names. We grant the adversary **arbitrary code execution as root inside the guest**: we do not assume the model refuses, that the agent harness's own permission prompts hold, or that in-guest controls survive. The adversary's goals are to exfiltrate data or credentials, to cause destructive or unauthorised effects, to persist or move laterally to the host or network, and to do any of these without an attributable record.

3.3 Trust assumptions

- The host operating system, the user account kosa8 runs as, and the kosa8 daemon form the trusted computing base (TCB).
- The hypervisor (Apple Virtualization.framework, or Cloud Hypervisor on KVM) correctly separates guest and host memory. A VMM escape is out of scope; we note the residual risk in §5.
- Approvers keep their signing keys private.
- An off-host copy of the audit log, if configured, is outside the control of anyone who later compromises the host.

Explicitly **out of scope**: a malicious host root (who can stop the daemon and rewrite local state), side and timing channels, and the semantic content of traffic to destinations the policy deliberately allows. Figure 2 shows the resulting zones and boundaries; Figure 3 contrasts an unconstrained agent with one under kosa8.

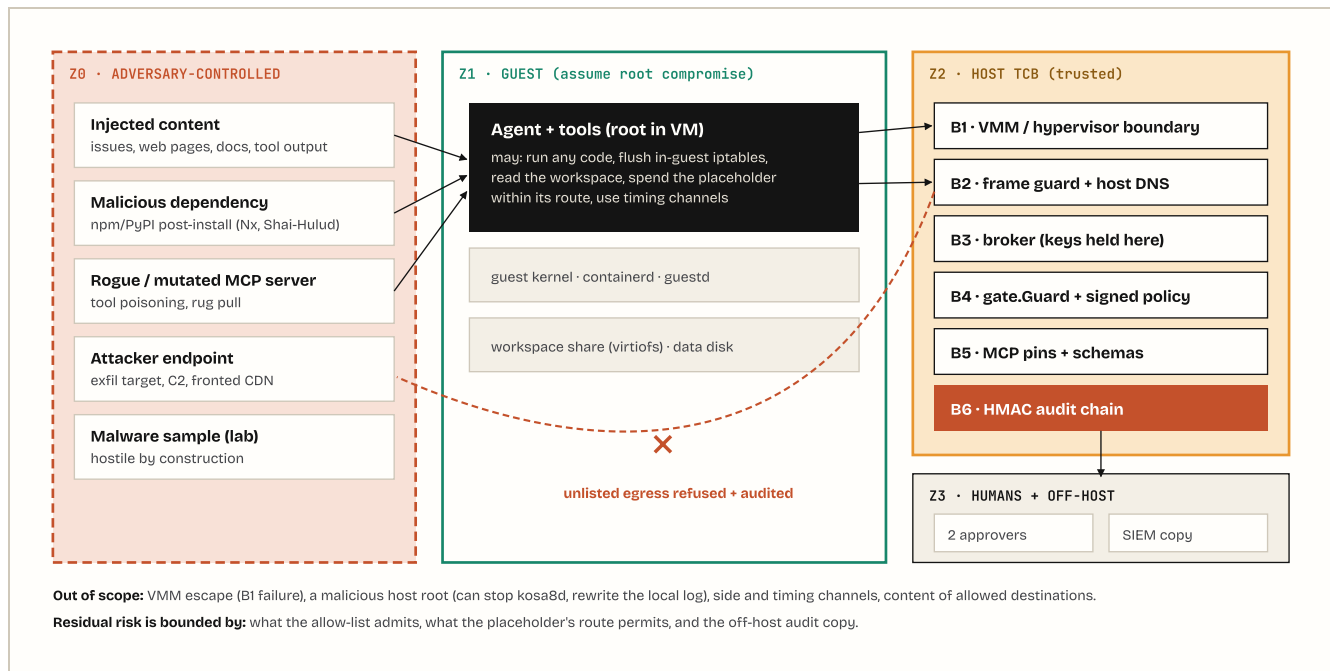


Figure 2. Trust zones and boundaries. Z0 is adversary-controlled; Z1 (the guest) is assumed compromised to root; Z2 is the host trusted computing base, which enforces boundaries B1–B6; Z3 holds the humans and the off-host audit copy that bound the residual risk.

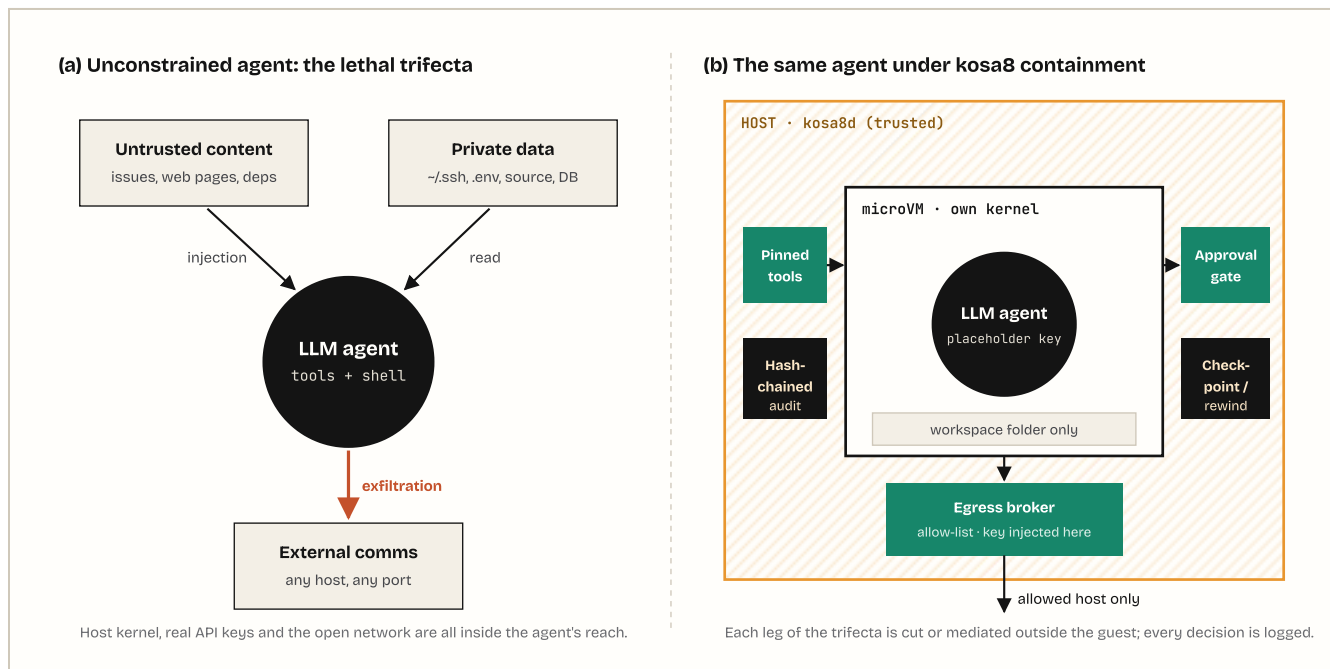


Figure 3. (a) An unconstrained agent combines untrusted content, private data and open communication, the lethal trifecta [2]. (b) Under kosa8 each leg is cut or mediated on the host: tool definitions are pinned, the filesystem is the workspace only, the key is a placeholder, egress leaves only through the broker, and every decision is logged.

3.4 Security goals

GOALS G1–G8 (EACH MUST HOLD WHILE THE ADVERSARY IS ROOT IN THE GUEST)

- G1 Containment.** The guest can read or write only host resources explicitly shared with it (the workspace), and cannot reach the host's sockets, credentials or other sandboxes.
- G2 Egress mediation.** Every packet that leaves the guest is decided by host-side policy; anything not admitted is dropped and recorded.
- G3 Credential confidentiality.** Real credentials never exist in guest memory, disk, snapshots or the audit log; a credential can be exercised only along its declared route.
- G4 Reversibility.** The complete state of a running machine (memory, processes, disks) can be captured and restored, so that effects confined to the machine can be undone.
- G5 Human oversight.** Actions the policy classes as risky execute only with valid, unspent approvals from the required number of distinct named people, never counting the requester.
- G6 Tool integrity.** A tool is exposed to the agent only in the exact definition that was reviewed; arguments and results are checked against declared schemas and policy.
- G7 Accountability.** Every security-relevant decision is recorded with its actor; any modification, insertion, deletion, reordering or truncation of the record is detectable.
- G8 Sovereignty.** All of the above runs on hardware the organisation controls, including with no network connectivity at all, using validated cryptography where required.

Table 2. Mapping of the OWASP Top 10 for Agentic Applications 2026 [11] to kosa8 goals and mechanisms. kosa8 addresses the infrastructure-addressable risks; ASI01 and ASI06 originate in the model and are bounded rather than prevented.

OWASP ASI risk	Goal	kosa8 mechanism	Effect
ASI01 Agent goal hijack	G1–G3, G5	Containment, egress mediation, approvals	Hijack is bounded, not prevented
ASI02 Tool misuse and exploitation	G5, G6	Argument rules (deny/ask/allow), schemas, held calls	Prevented for governed tools

ASI03 Identity and privilege abuse	G3, G7	Broker placeholders, per-agent tokens, actor attribution	Prevented / attributed
ASI04 Agentic supply chain	G6, G1	Definition pinning, signed kits, SHA-512 package pins, MCP servers in VMs	Prevented / contained
ASI05 Unexpected code execution	G1, G4	Per-agent microVM, checkpoint/rewind	Contained, reversible
ASI06 Memory and context poisoning	G4, G7	Rewind to a clean checkpoint; audit trail	Bounded, recoverable
ASI07 Insecure inter-agent comms	G2	Isolated L2 per sandbox; egress policy	Contained
ASI08 Cascading failures	G1, G4	Fork to test in parallel; all-or-nothing fork commit	Contained
ASI09 Human-agent trust exploitation	G5	Byte-exact, one-shot, signed approvals; requester never counts	Mitigated
ASI10 Rogue agents	G1, G2, G7	Containment, default-deny agent egress, attribution	Contained, detected

4 Design of kosa8

4.1 Design principles

- **Enforce outside the guest.** Every control that matters runs in the host daemon, so root in the guest cannot disable it. In-guest controls exist only as a second layer.
- **Local first, cloud optional.** The data path never requires a vendor service. Cloud features (snapshot sync, remote workers) are additive and can be switched off entirely.
- **Fail closed.** A policy that is present but unverifiable stops the daemon; a missing audit key fails verification; an unclassified API route stops the daemon at start-up.
- **Prove, don't promise.** Approvals, seals, evidence and the log are signed or authenticated so a third party can check them offline.
- **Compatibility as adoption.** A Docker-compatible API lets existing tools (Docker CLI, Compose, Testcontainers, agent SDKs) adopt kosa8 by changing `DOCKER_HOST`, consistent with Saltzer and Schroeder's principle of psychological acceptability [68].

4.2 Architecture

kosa8 is a rootless per-user daemon, `kosa8d`, plus a command-line client, written in Go (about 90,000 lines of non-test code and 44,500 lines of tests across 62 packages). Figure 4 shows the components. Clients reach the daemon through a Unix socket (Docker CLI, SDKs, the kosa8 CLI and desktop apps), through `kosa8 mcp serve` (AI agents speaking MCP), or through a token-guarded TCP listener (remote CLIs). Every request passes the same pipeline: **identify** the caller (verifying an agent token if presented), **gate** it (classify the route and enforce approvals), run the **handler**, and **append** an audit entry. The gate panics at start-up if any route is unclassified, so a new endpoint cannot ship ungoverned.

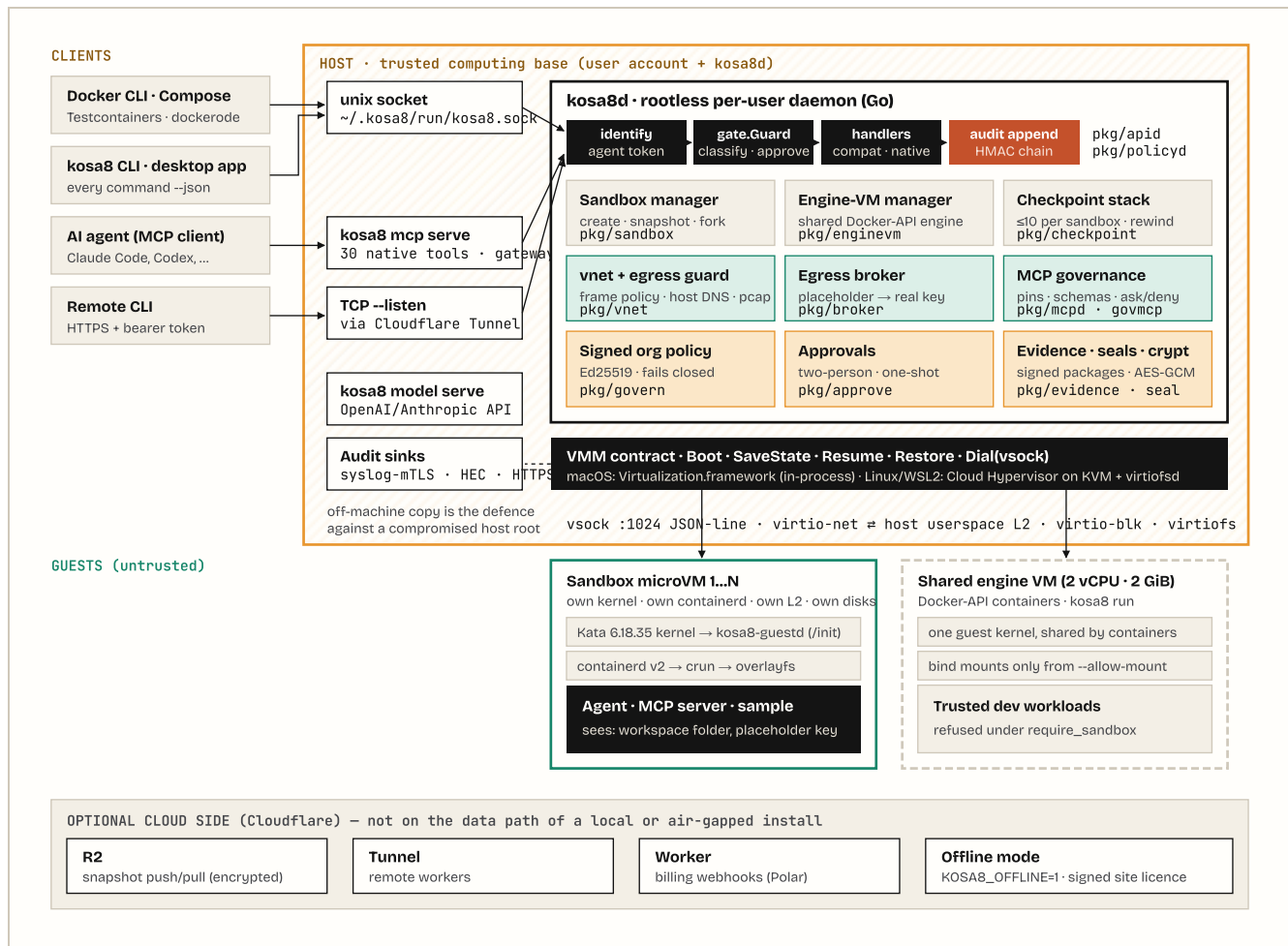


Figure 4. kosa8 architecture. The host daemon is the trusted computing base: it identifies callers, gates actions, enforces egress, holds credentials, and writes the audit chain. Sandboxes are untrusted microVMs, each with its own kernel, container engine, disks and private L2 network. A separate shared engine VM serves Docker-API workloads and can be forbidden by policy.

Two classes of virtual machine exist. Each **sandbox** is its own microVM (default 2 vCPU, 1 GiB) with its own kernel, its own containerd, two virtio-blk disks and an isolated userspace L2 network; agents, MCP gateway servers, malware detonations and cyber-range trainees always run in sandboxes. A single shared **engine VM** (2 vCPU, 2 GiB) serves the Docker-compatible API for trusted developer workloads, equivalent to the VM inside Docker Desktop; containers there share one guest kernel with each other. The signed policy field `require_sandbox` refuses the engine VM entirely, which regulated deployments should set.

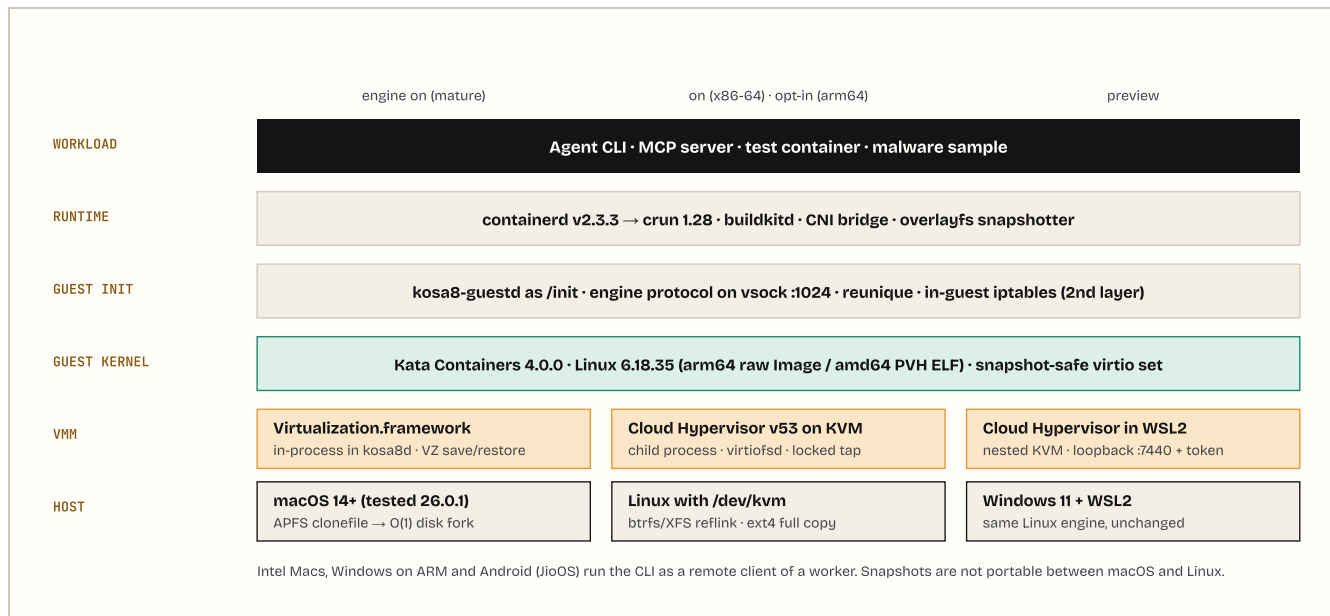


Figure 5. The software stack on each host platform. Everything from the guest kernel up is identical across platforms; only the VMM layer differs. The device set is restricted to virtio devices that survive save and restore.

4.3 Isolation and filesystem exposure

On macOS the VMM is Apple's Virtualization.framework, running inside the daemon so no VM outlives it; the guest NIC is one end of a datagram socket pair whose other end the daemon reads. On Linux, an unprivileged Cloud Hypervisor child process runs each VM on KVM; Cloud Hypervisor was chosen over Firecracker for its virtiofs support [57]. The daemon creates a locked-down tap per VM (no IPv6, no forwarding, no ARP replies, no address) and relays frames from it into its own network stack, so the guest has no path to the host except through the daemon. Windows 11 runs the same Linux engine inside WSL2 (preview). The host reaches the guest only over vsock, using a JSON-line protocol to the guest agent, `kosa8-guestd`, which is PID 1.

An agent run shares exactly one host directory, its workspace, over virtiofs. Docker-API bind mounts are honoured only inside directories the operator allowed with `kosa8 daemon start --allow-mount`, read-only by default, with symlinks resolved. Some sources are refused unconditionally, whatever the allow-list says: every Unix socket (so the Docker and kosa8 API sockets can never be mounted), the well-known Docker socket paths, and the kosa8 state directory or any ancestor of it such as `$HOME`. This closes the Docker-socket escape used against several agent sandboxes in 2026 [26].

4.4 Host-side egress enforcement

Every frame a guest sends is read by the daemon and decided by a guard before any forwarding code sees it (Figure 6). Four policies exist: `any`; `deny`, which admits nothing, DNS included;

`allow`, a list of host names, single-star wildcard domains, IPv4 addresses and CIDRs with everything else denied; and `fakenet`, a simulated internet for malware analysis. DNS on UDP port 53 is never forwarded: the host answers it, resolving only listed names, admitting at most sixteen addresses per answer with a forced 60-second TTL, withholding private and loopback answers as a rebinding defence, and refusing everything else. A TCP SYN to an unadmitted address is answered with a reset; UDP with ICMP unreachable; both produce a rate-limited `sandbox.egress.deny` audit entry with the guest as actor. IPv6, fragments, broadcast and multicast are dropped. Because this runs in the host daemon, a guest that flushes its own iptables changes nothing.

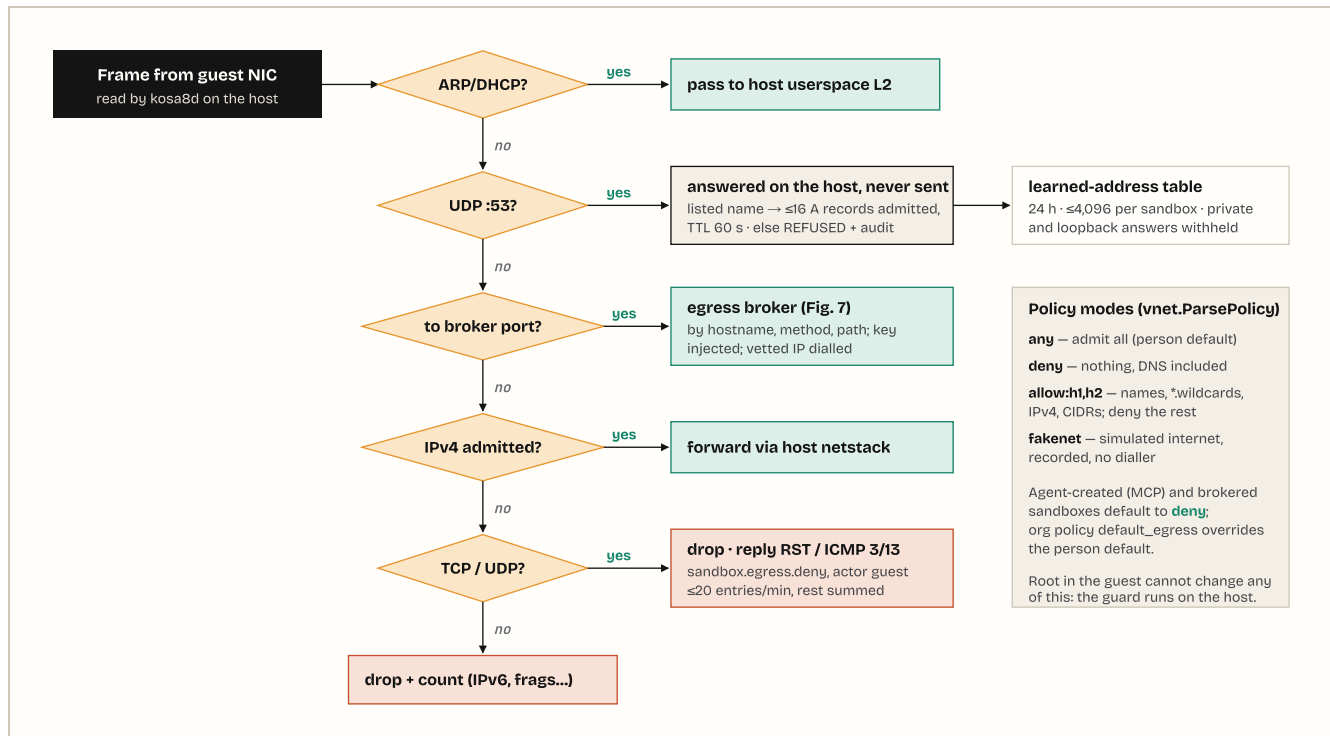


Figure 6. The host-side egress decision for every guest frame. Root in the guest cannot alter any step. Agent-created and brokered sandboxes default to deny; person-created sandboxes default to any unless the signed org policy sets default_egress.

4.5 Credential brokering

A brokered sandbox's frame policy is `deny` with exactly one pinhole: TCP to a fixed guest-side address and port that the daemon maps to a per-sandbox broker on host loopback (Figure 7). An agent kit declares, for each credential, the upstream, the header to set, the host environment variable that holds the secret, optional method and path restrictions, and the environment the guest receives instead: a base URL pointing at the broker and a random placeholder. On each request the broker checks the route, compares the placeholder in constant time, refuses requests that carry the placeholder anywhere else, resolves the upstream on the host and discards private, loopback, link-local and metadata answers, dials the vetted address over TLS verified against the upstream's name, sets the real header, never follows redirects, and strips any response header that

echoes the credential. The real key exists only in the launching process and daemon memory: never in the VM, the sandbox record, a snapshot or the log. Non-credential traffic through the broker is admitted by host name and port, with the same DNS vetting.

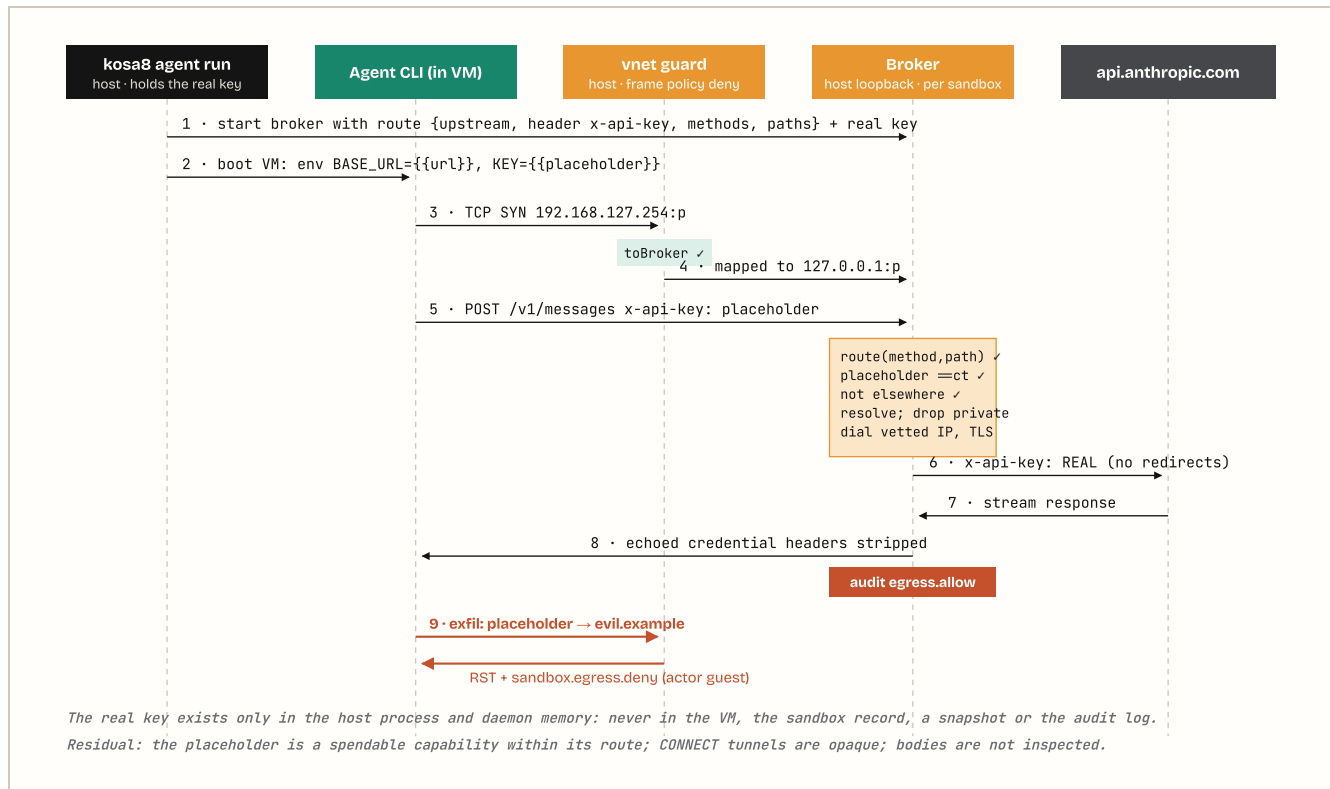


Figure 7. Sequence of a brokered API call. The guest holds a placeholder; the broker on the host checks route and placeholder, then dials the vetted upstream with the real key. An attempt to send the placeholder anywhere else is refused by the frame guard and audited.

4.6 Reversibility: checkpoint, rewind and fork

A snapshot pauses the guest, writes memory and vCPU state, clones both disks copy-on-write *inside the same pause* (APFS `clonefile` on macOS, reflink on btrfs or XFS on Linux), and resumes; cloning the disk at a different instant from the memory would leave the restored page cache disagreeing with the disk, a silent corruption the design rules out. The memory image is sparse: about 60 MiB for a 1 GiB guest in our measurements. A **checkpoint** is a named snapshot pushed on a stack of at most ten per sandbox (sealed checkpoints are never evicted); **rewind** deletes the sandbox and restores the checkpoint under the same name, with its processes still running from the moment of capture.

Fork captures one ephemeral snapshot and restores it into N new VMs, four at a time (Figure 8). Restoring the same machine twice would yield clones with identical identity, entropy and clocks, so each clone is *re-unique*: it receives a new host name, its kernel random-number generator is reseeded with 64 bytes of host randomness, its clock is resynchronised, and it takes a fresh

network lease. Duplicate MAC and IP addresses are harmless because each clone has a private L2 network. Fork is all-or-nothing: names are reserved first, and any failed restore discards every clone and the snapshot. Brokered credentials deliberately do not survive restore or fork, and a fork may raise a clone's classification label but never lower it. **Explore** builds on fork, running a different command in each branch and keeping the one that succeeds.

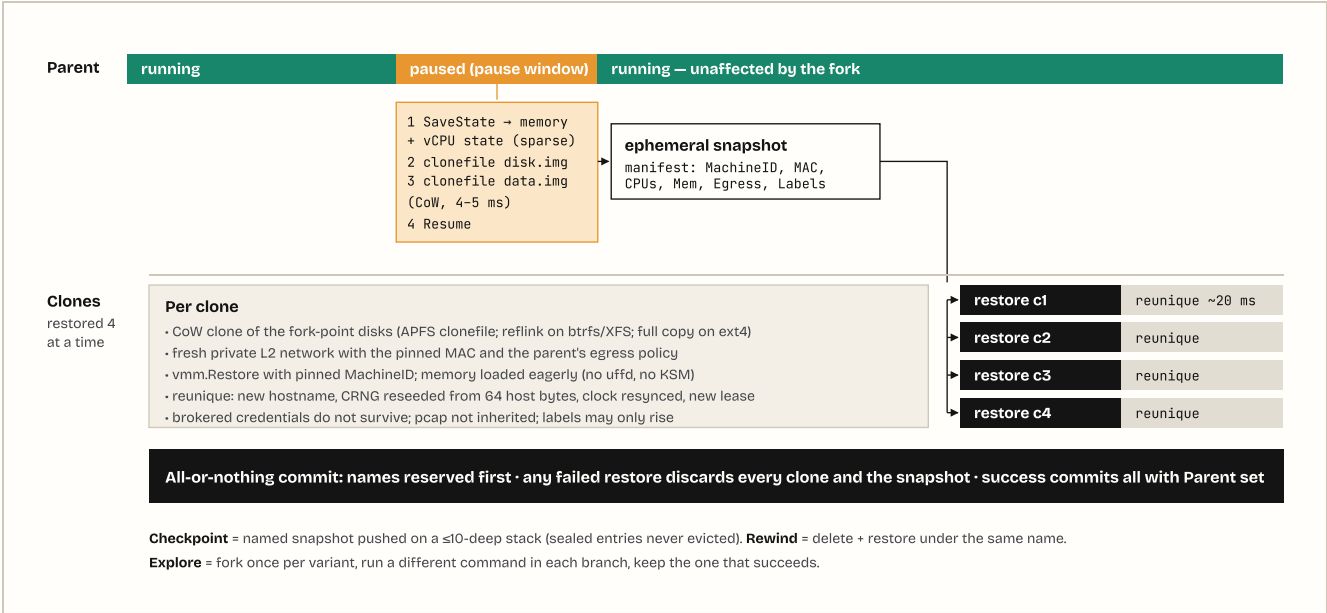


Figure 8. Fork timeline. The parent pauses only while memory is saved and disks are cloned; clones are restored four at a time, re-unique'd, and committed together or not at all.

4.7 Human oversight: signed, two-person approvals

Every API route is classified as escalating (for example networks, published ports, host paths, wider egress), destructive (removal, prune, restore over existing state), contained (pull, run, create without mounts or egress) or never gated (reads). When the signed policy requires approval for a class, the request fails with `approval.required`, naming who is still needed (Figure 9). The requester signs a canonical description of the exact action with Ed25519 (ECDSA P-256/P-384 for smart cards); approvers sign the request's digest. On submission the daemon verifies every signature against approvers listed in the *signed* policy, counts distinct people (keyed on lower-cased email or key), never counts the requester under any of their keys, checks deny lists and replays, and records `approval.granted` with the full signed records embedded. The repeated call presents the approval ID; under one lock the daemon checks it is recorded, unspent, unexpired, covers the action byte for byte, and still satisfies the current policy, writes `approval.consumed`, and only then runs the handler, exactly once. Approval files travel as files, so the flow works across an air gap.

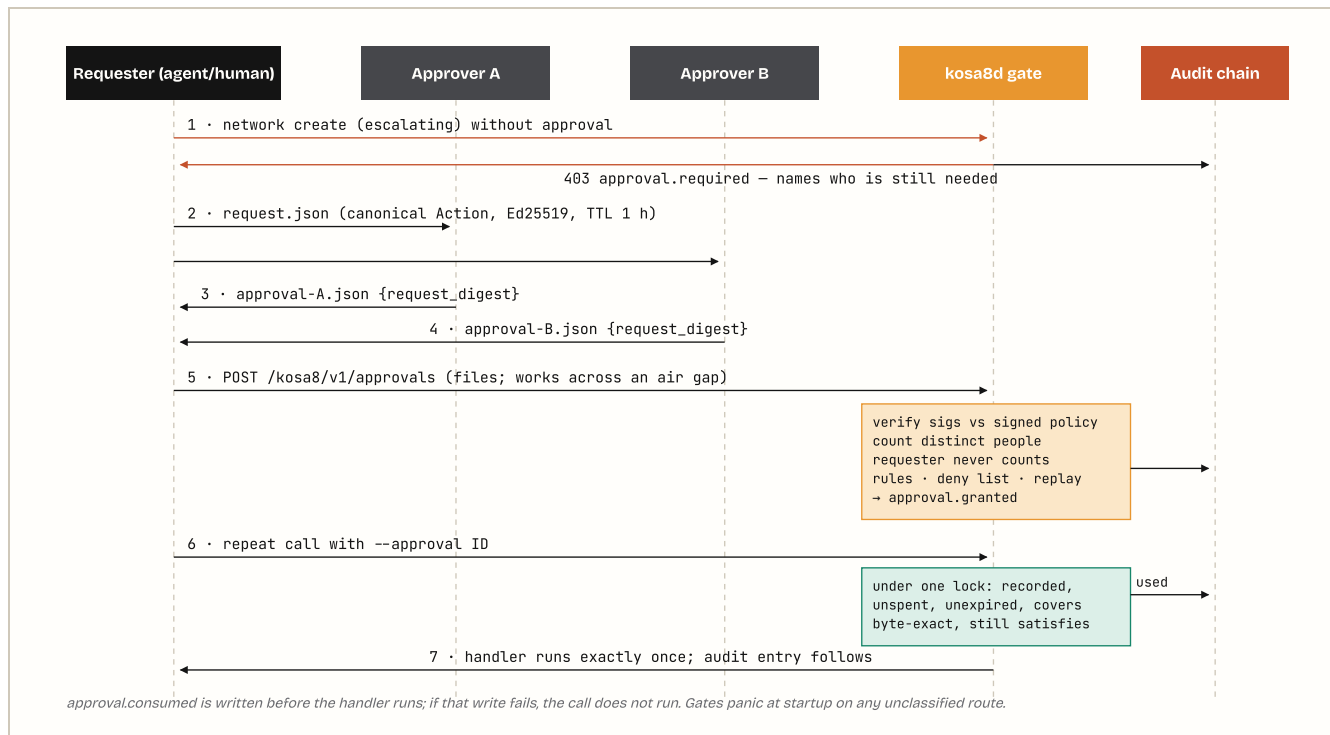


Figure 9. Two-person approval. Approvals bind to the digest of the exact canonical action, are spent once under a lock, and are recorded before the action executes.

4.8 Governed MCP gateway

kosa8 exposes itself to agents as an MCP server with thirty native tools, and runs third-party MCP servers each in its own sandbox with its own egress policy, re-exporting their tools (Figure 10). At gateway start every tool definition is canonicalised and digested; a definition that differs from its pin, or a new tool, is withheld and audited until a person approves it, which defeats rug pulls [21]. Calls pass identity, a signed default-deny tool policy, argument-level rules over JSON Pointer with deny overriding ask overriding allow, and validation against the pinned input schema; results are validated against the declared output schema and withheld if they do not conform. An *ask* decision holds the call until a person runs `kosa8 mcp approve`, bound to the agent, server, tool and argument digest for ten minutes; an optional local scorer may auto-approve only a confident *allow*, after a deterministic veto, and never overrides a deny. By contrast, Docker's gateway runs local stdio MCP servers on the host, outside the VM [47].

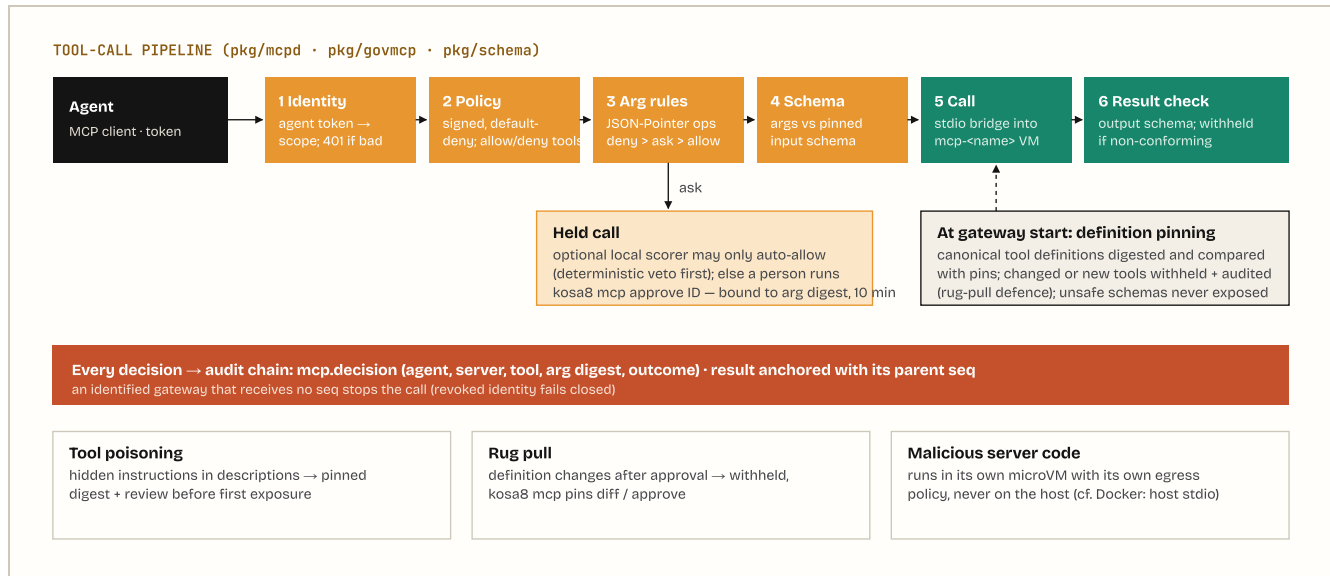


Figure 10. The governed tool-call pipeline. Every decision, including held and refused calls, is written to the audit chain.

4.9 Tamper-evident audit log

The audit log is an append-only JSON-lines file in which each entry records a sequence number, timestamp, actor (human, agent, tool, guest or policy), action, target, detail, status and structured references, and carries the authenticator of its predecessor (Figure 11). For format v2 the authenticator is:

$$h_i = \text{HMAC-SHA256}(k, \text{"kosa8.audit/v2"} \parallel \text{len}(\text{seq}_i) \parallel \dots \parallel \text{len}(\text{status}_i) \parallel \text{len}(h_{i-1}) \parallel \text{len}(\text{refs}_i)) \quad (1)$$

where every field is length-prefixed to remove the ambiguity of delimiter-joined encodings, and k is a 32-byte key generated once and stored with mode 0600. After each append the daemon writes an authenticated tip, `audit.head` = (n, h_n , $\text{HMAC}_k(\text{"n:h_n"})$). Verification reads the tip *before* the entries (so a concurrent append is not mistaken for tampering), authenticates any retention anchor, then walks the chain checking the predecessor link, the HMAC in constant time, sequence contiguity and format monotonicity, and finally compares the chain's end with the tip. This construction follows the forward-integrity tradition of Bellare and Yee [67] and Schneier and Kelsey [65]; Crosby and Wallach give the efficient-proof variant [66]. Entries forward at-least-once to syslog over mutual TLS, Splunk HEC or HTTPS, and export as CEF or JSONL.

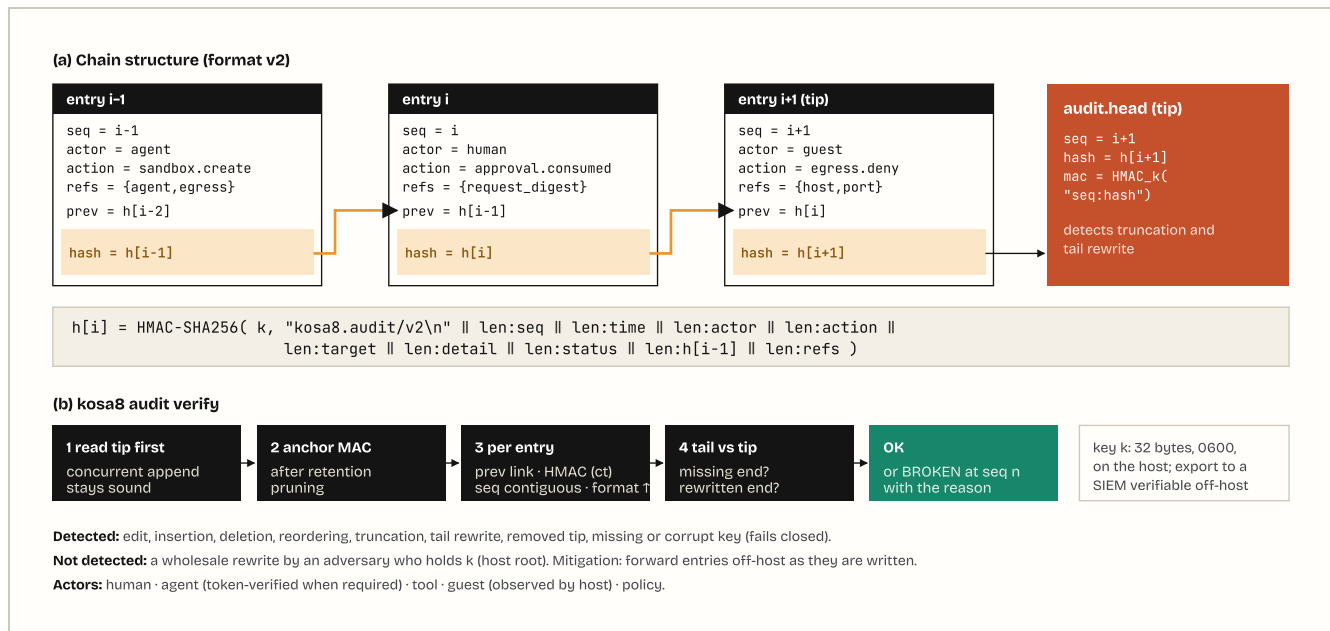


Figure 11. (a) The keyed hash chain and authenticated tip. (b) The verification algorithm. A missing or corrupt key fails closed rather than falling back to an unkeyed hash.

4.10 Evidence, sealing and cryptography at rest

`kosa8 evidence export` produces a package containing the contiguous chain segment covering the requested scope, the tip and anchors, embedded approval records, seals, the policy, keys, version and compliance state, and a manifest of SHA-256 digests signed with Ed25519. `kosa8 evidence verify` checks it in eight steps with no daemon, no kosa8 state and no network: signature, file digests, chain structure, HMACs (if given the escrowed key), seals, approval signatures and their binding to requests, cross-references and keys. Third parties can therefore verify structure and signatures; the organisation, holding the escrowed key, can additionally verify every HMAC. Snapshots can be *sealed* (every file digested and signed) so that a restore of altered state is refused, and encrypted with AES-256-GCM under a key hierarchy; `rm --wipe` crypto-shreds, overwrites and verifies.

4.11 Sovereignty: air gap, FIPS and supply chain

An air-gap bundle contains the binaries, guest kernel, root filesystem and base image with an installer that fetches nothing; `KOSA8_OFFLINE=1` refuses all outbound calls; a site licence is an Ed25519-signed file verified locally with no activation or phone-home. A FIPS build compiles against Go's CMVP-validated cryptographic module (`GOFIPS140=certified`), and the signed policy can require it, in which case the daemon refuses to start otherwise; every primitive kosa8 uses (SHA-256, HMAC-SHA256, Ed25519, AES-GCM, ECDSA, the DRBG) comes from that module. Public releases are Developer-ID signed and notarised on macOS and Authenticode-signed on Windows; a separate government channel publishes FIPS builds with CycloneDX SBOMs and keyless cosign

signatures. Agent kits pin npm packages by exact version and SHA-512 and can be signed by the organisation, so governed machines run only signed kits.

5 Security analysis

For each goal we give the enforcing mechanism, the argument that root in the guest cannot defeat it, and the residual risk. Table 3 summarises.

5.1 G1 Containment

The guest's only channels to the host are the virtio devices and vsock that the VMM exposes. Filesystem access is limited to virtiofs shares the daemon created from the workspace or the mount allow-list, from which sockets and the state directory are unconditionally excluded. *Residual:* a VMM escape (B1 failure) defeats G1, as for every VM-based system; kosa8 reduces exposure by using only a minimal, snapshot-safe virtio device set.

5.2 G2 Egress mediation

Every frame crosses the host guard before reaching any forwarder; the guard is written in memory-safe Go and fuzzed (§6.4). DNS is answered, not forwarded, so the guest cannot reach a resolver of its choosing. *Residual:* an admitted address is admitted on every port, so domain fronting through shared CDNs and exfiltration to allowed destinations remain possible; wildcard entries create a DNS covert channel through query names; timing channels are not closed. The broker narrows the first two for credentialed routes by matching host, method and path.

5.3 G3 Credential confidentiality

The real key never enters guest memory because the guest never receives it; it is added by a host process the guest can reach only through the pinhole. Placeholder misuse is limited to the declared route. *Residual:* the placeholder is a capability: an injected agent can spend the key within its route (for example, make expensive API calls). CONNECT tunnels are opaque and bodies are not inspected. One built-in kit (GitHub Copilot) cannot take a base URL and still passes its token in, with a warning.

5.4 G4 Reversibility

Because memory and both disks are captured within one pause, a restored machine is the machine as it was, processes included. *Residual:* effects outside the machine (an email sent, a commit pushed, an API call made) cannot be rewound. G2 and G5 are the controls for those; G4 bounds the cost of everything else.

5.5 G5 Human oversight

Approvals are verified against approvers listed in the signed policy, not the request; the requester is excluded by key and by identity; spending is atomic and byte-exact, so an approval cannot be replayed or applied to a modified action, the bait-and-switch behind MCPoison [25]. *Residual*: a local root user can stop the daemon; what remains provable is that a gated action without a matching `approval.consumed` entry was performed out of band.

5.6 G6 Tool integrity

Pinning binds exposure to a digest of the reviewed definition; schemas bound arguments and results; the server's code runs in its own microVM. *Residual*: pins are trust-on-first-use, and nothing scans the content of a definition for hidden instructions at first review.

5.7 G7 Accountability

Let the adversary be any process without k . Modifying, inserting, deleting or reordering an entry changes some h_i , which the adversary cannot recompute without k (HMAC unforgeability); truncation or tail rewrite is caught by the authenticated tip; deleting the tip or the key fails closed.

PROPOSITION 1 (TAMPER EVIDENCE)

If HMAC-SHA256 is a secure PRF and k is unknown to the adversary, then any change to the multiset or order of entries in a verified log, or to the content of any entry, causes `kosa8 audit verify` to fail, except with probability negligible in the key length. An adversary who learns k (host root) can produce a consistent forgery; such forgeries are detectable only against an off-host copy taken before compromise.

Residual: the HMAC key is symmetric and stored on the host, so the guarantee against a compromised host depends on forwarding; anchoring the tip in a public transparency log [71] would remove that dependency and is future work.

5.8 G8 Sovereignty

No component on the data path contacts a vendor service; offline mode is enforced in code; the licence is verified locally; cryptography comes from a CMVP-validated module. *Residual*: the guest agent carries no FIPS claim, and no kosa8-specific CMVP certificate exists; the claim is a build on a validated module.

Table 3. Summary of the security analysis.

Goal	Enforced by	Holds against guest root	Main residual risk
G1 Containment	VMM, virtiofs scope, socket exclusion	● Yes	VMM escape
G2 Egress	Host frame guard, host DNS	● Yes	Allowed destinations, fronting, DNS covert channel under wildcards
G3 Credentials	Broker, placeholder, TLS to vetted IP	● Yes	Placeholder spendable within its route
G4 Reversibility	Atomic memory + disk capture	● Yes	External side effects
G5 Oversight	Signed policy, atomic spend	● Yes	Local root can stop the daemon
G6 Tool integrity	Pins, schemas, per-server VM	● Yes	Trust on first use
G7 Accountability	Keyed chain, authenticated tip	● Yes	Host root holding k; mitigated off-host
G8 Sovereignty	Offline mode, local licence, FIPS build	● Yes	No product CMVP certificate

6 Evaluation

We evaluated the released software rather than a research prototype. Every result in this section was produced by the authors on 17 October 2026 unless it is explicitly labelled as kosa8's own historical record.

6.1 Setup and method

Table 4. Experimental setup.

Item	Value
System under test	kosa8 v0.9.36 (commit f6124c0), client and daemon from the public release; guest image v0.9.36 (Kata 6.18.35 kernel)
Source for tests	<code>git archive v0.9.36</code> ; no code differs between the tag and repository HEAD
Host	Apple M4 (4 performance + 6 efficiency cores), 16 GB RAM, macOS 26.0.1, internal NVMe (APFS)
Toolchain	Go 1.26.9 (darwin/arm64)
Timing method	Wall clock of the complete CLI call (Python <code>perf_counter</code>), as in the project's own <code>bench/run.sh</code> ; includes client start-up and daemon round trip
Repetitions	n = 10–15 per lifecycle operation; <code>make bench</code> gates use medians of 3–5 as designed
Sandbox shape	Default: 2 vCPU, 1,024 MiB, egress deny, Alpine workload

6.2 Correctness: test suites

We ran every Go test in the release (`go test ./pkg/... ./internal/... ./cmd/...`) on the extracted tag. All **1,142** top-level tests passed, none failed, and six were skipped because they need hardware not present. Table 5 breaks out the twenty packages that implement security controls: 449 tests, all passing, including the audit-log tests that tamper with a log by editing, truncating, wholesale re-forging without the key, and deleting or corrupting the key, and the broker tests that confirm the credential never reaches the sandbox. kosa8's own records report its Docker-CLI compatibility suite at 44/44, Compose at 10/10, the MCP protocol suite at 11/11, Testcontainers green, and 45 of 48 `docker` commands implemented; the Linux engine has passed the full create-snapshot-fork-restore lifecycle on real x86-64 KVM in CI on every change since 25 September 2026.

Table 5. Tests in security-critical packages of kosa8 v0.9.36 (this study).

Package	Control	Tests passed
pkg/policyd	Audit chain, verification, retention	42

pkg/vnet	Frame guard, host DNS, pcap	46
pkg/broker	Credential broker	23
pkg/mcpd	MCP server and gateway	45
pkg/schema	Argument and result schemas, rules	38
pkg/approve	Signed approvals, two-person rule	55
pkg/apid/gate	Route classification and gating	7
pkg/govern · pkg/govmcp	Signed org and MCP policy	28 · 24
pkg/agentid · pkg/identity	Agent tokens, principals	7 · 4
pkg/evidence · pkg/seal	Evidence packages, sealed snapshots	14 · 14
pkg/crypt · pkg/wipe	Encryption at rest, verified wipe	23 · 6
pkg/kit · pkg/hostmount	Signed kits, package pins, mount rules	21 · 5
pkg/lab · pkg/label · pkg/scorer	Malware lab, classification labels, scorer	12 · 14 · 21
Total (20 packages)	449 passed, 0 failed, 1 skipped	

6.3 Egress enforcement

For each of the four egress policies we created a fresh sandbox, ran an Alpine container inside it, and attempted seven HTTP(S) destinations and four DNS lookups addressed to public resolvers. The destinations include an allowed host, an unlisted host, a raw IPv4 address, a raw IPv6 address, the cloud instance-metadata address, and the sandbox's own gateway. Table 6 gives the results.

Table 6. Egress probes from inside a sandbox, kosa8 v0.9.36 (this study). REACHED means an HTTP response was received; DNS columns show whether a public resolver appeared to answer.

Destination	deny	allow example.com	any (default)	fakenet
http://example.com	○ blocked	● reached	● reached	🌀 simulated
https://example.com	○ blocked	● reached	● reached	○ no trusted cert
http://www.iana.org (unlisted)	○ blocked	○ blocked	● reached	🌀 simulated
http://1.1.1.1 (raw IPv4)	○ blocked	○ blocked	● reached	🌀 simulated
http://[2606:4700:4700::1111] (IPv6)	○ blocked	○ blocked	○ blocked	○ blocked
http://169.254.169.254 (metadata)	○ blocked	○ blocked	○ blocked	🌀 simulated

http://192.168.127.1 (gateway)	 blocked	 blocked	 blocked	 blocked
DNS example.com via 8.8.8.8 / 1.1.1.1	 no answer	 answered by host	 answered	 sink address
DNS iana.org via 8.8.8.8 / 1.1.1.1	 no answer	 refused	 answered	 sink address
Host counters (sandbox egress)	0 answered	14 answered, 12 refused, 12 frames denied	nothing refused	25 interactions recorded, nothing left

In the fakenet column, ‘simulated’ means the host’s simulated internet answered and recorded the request; no packet left the machine. HTTPS fails there because the guest does not trust the lab CA, as designed.

Three observations matter. First, every destination outside policy was blocked in every restrictive mode, including raw IP and IPv6 attempts that would bypass a name-based filter. Second, DNS queries addressed to public resolvers were answered by the host under the allow-list (the guest believed it was talking to 8.8.8.8), and refused for unlisted names, confirming that resolver choice gives the guest no escape. Third, even the permissive default blocks IPv6, the metadata endpoint and the host gateway, which removes the most common cloud lateral-movement path [34]. The default for sandboxes a person creates is nonetheless `any` ; we return to this in §9.

6.4 Robustness: fuzzing

The release ships four fuzz targets over attacker-reachable parsers: the guard's frame parser and its DNS parser (both reachable from guest root), the MCP argument-rule evaluator, and the scorer's heuristic. We ran each for 120 seconds with Go's coverage-guided fuzzer on the M4 host. All four campaigns completed without a crash, hang or failing input (Table 12, Appendix A). A short campaign is a smoke test, not an assurance argument, and we recommend continuous fuzzing (§9).

6.5 Performance

Figure 12 and Table 7 report wall-clock latencies of the complete CLI call. A checkpoint of a running 1 GiB sandbox takes a median of **638 ms** (n = 15, range 476–843 ms) and a rewind to the latest checkpoint **520 ms** (n = 10, range 426–990 ms), consistent with kosa8's published 697 ms and 485 ms. The project's own gate harness, run once on the same host, reports a 212 ms container start on the engine VM with a cached image, a 45 MB idle host footprint, a 1,860 ms snapshot of a 2 GiB sandbox and a 1,075 ms warm restore of it.

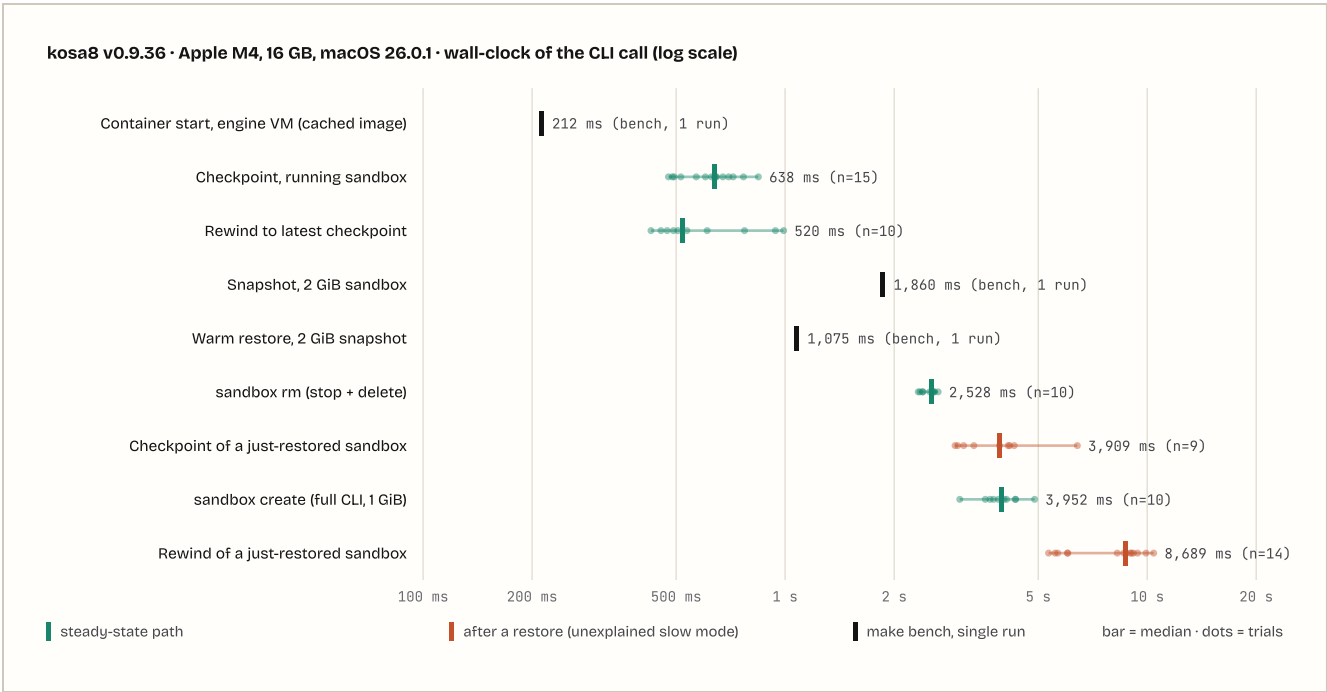


Figure 12. Measured latencies of kosa8 v0.9.36 on an Apple M4 (this study). Bars mark medians, dots individual trials. Red rows are the slow mode observed only when operating on a sandbox that was itself just restored.

Table 7. Latency of kosa8 v0.9.36 (this study, Apple M4, wall clock of the CLI call).

Operation	n	Median	Range	Note
Checkpoint, running sandbox	15	638 ms	476–843 ms	Memory image ≈ 60 MiB sparse for 1 GiB
Rewind to latest checkpoint	10	520 ms	426–990 ms	Processes resume where captured
Container start, engine VM, cached image	1 (median of 5)	212 ms	—	make bench gate ≤ 250 ms: pass
Snapshot, 2 GiB sandbox	1	1,860 ms	—	Gate ≤ 2,000 ms: pass
Warm restore, 2 GiB snapshot	1 (median of 3)	1,075 ms	—	Gate ≤ 1,000 ms: miss by 7.5%
sandbox create (full CLI, 1 GiB)	10	3,952 ms	3,037–4,888 ms	Includes disk creation and engine readiness
sandbox rm (stop and delete)	10	2,528 ms	2,331–2,645 ms	
Checkpoint of a just-restored sandbox	9	3,909 ms	2,946–6,413 ms	Slow mode, see §6.6

Rewind of a just-restored sandbox	14	8,689 ms	5,339–10,418 ms	Slow mode, see §6.6
Idle host footprint	1	45 MB	—	RSS of daemon and helpers
Bind-mount write throughput	1	47% of native	—	VM block device reaches 62%; gate ≥ 70%: miss

Fork could not be re-measured: on the test host fork requires an active subscription, and we did not circumvent licensing. kosa8's own records, on an Apple M-series host with 1 GiB guests, report fork into 4 clones in 2.1 s and into 8 clones in 4.1 s (about 515 ms per clone with four parallel restores), with restore decomposed into about 590 ms of hypervisor state load, 20 ms of re-uniqueing and 5 ms of disk clone; and a boot of the Kata kernel to guest-agent readiness of 244 ms.

6.6 Unexpected findings

Two results departed from the project's published figures, and we report both. **(1) Creation latency.** Full `sandbox create` takes a median of 3.95 s, against published sandbox boot figures of 362–653 ms. The published figures time the VM boot; ours times the complete command, which also creates a 32 GB sparse data disk and waits for the in-guest container engine. The gap is real for users and is the most visible performance difference between kosa8 and hosted sandboxes that keep warm pools. **(2) A post-restore slow mode.** Checkpoint and rewind are fast on a sandbox that has not just been restored, and 5–15× slower immediately after a restore; in strictly alternating checkpoint–rewind sequences the slow and fast modes alternate. The saved memory image stays at 60 MiB in both modes, which rules out the obvious hypothesis that a restored guest's memory becomes dense. We have not identified the cause; candidates are teardown of the replaced VM overlapping the next operation, or copy-on-write disk chains. Neither finding affects a security property.

6.7 Threats to validity

All measurements come from one host and one operating system; Linux and Windows engines were not measured. Wall-clock CLI timing includes client start-up, so it overstates the engine's internal latency and understates nothing a user would see. Several `make bench` gates are single runs by design of the harness. Competitor capabilities in §7 come from public documentation, not from running their systems; a capability not documented may exist. Finally, the authors are affiliated with kosa8; we mitigate this by publishing the method, reporting unfavourable results, and labelling every number with its provenance.

7 Comparative analysis

7.1 Requirements

From the goals in §3 and the regulatory drivers in §2.5 we derive thirteen requirements an agent infrastructure must meet to be deployed in a regulated organisation. Each is binary at the level of documentation: either the vendor documents the capability as shipped, documents it as partial, preview, optional or plan-gated, or does not document it. We assessed kosa8 from its v0.9.36 source and ten systems from their public documentation as of 17 October 2026, preferring primary sources [47]–[55].

Table 8. Thirteen requirements for agent infrastructure in regulated settings.

Req.	Requirement	Why it matters	Goal
R1	Own guest kernel per agent	Shared kernels expose the full syscall surface and recurrent runc escapes [31]–[33]	G1
R2	Runs on customer hardware	Data residency; DORA third-party scope; CUI handling	G8
R3	Air-gapped operation documented	Defence, classified-adjacent and OT environments	G8
R4	Agent egress denied by default	Exfiltration is the lethal-trifecta payoff [2]	G2
R5	Credentials never enter the sandbox	sIngularity, Shai-Hulud harvested in-sandbox secrets [17], [18]	G3
R6	Live memory checkpoint and rewind	Replit-class destructive actions [20]	G4
R7	Live fork into N running copies	Safe parallel exploration; incident forensics	G4
R8	Runtime-enforced two-person approval	EU AI Act Art. 14; SOX/Part 11 separation of duties	G5
R9	Keyed, hash-chained audit log	AU-9/AU-10; EU AI Act Art. 12; non-repudiation	G7
R10	Offline-verifiable signed evidence	Assessors cannot trust the operator's database	G7
R11	MCP pinning and argument policy	Tool poisoning and rug pulls [21], [25]	G6
R12	FIPS 140-3 validated cryptography build	CMMC 3.13.11; federal use	G8

R13	Docker Engine API compatibility	Adoption without rewriting tools and pipelines	—
-----	---------------------------------	--	---

7.2 Scorecard

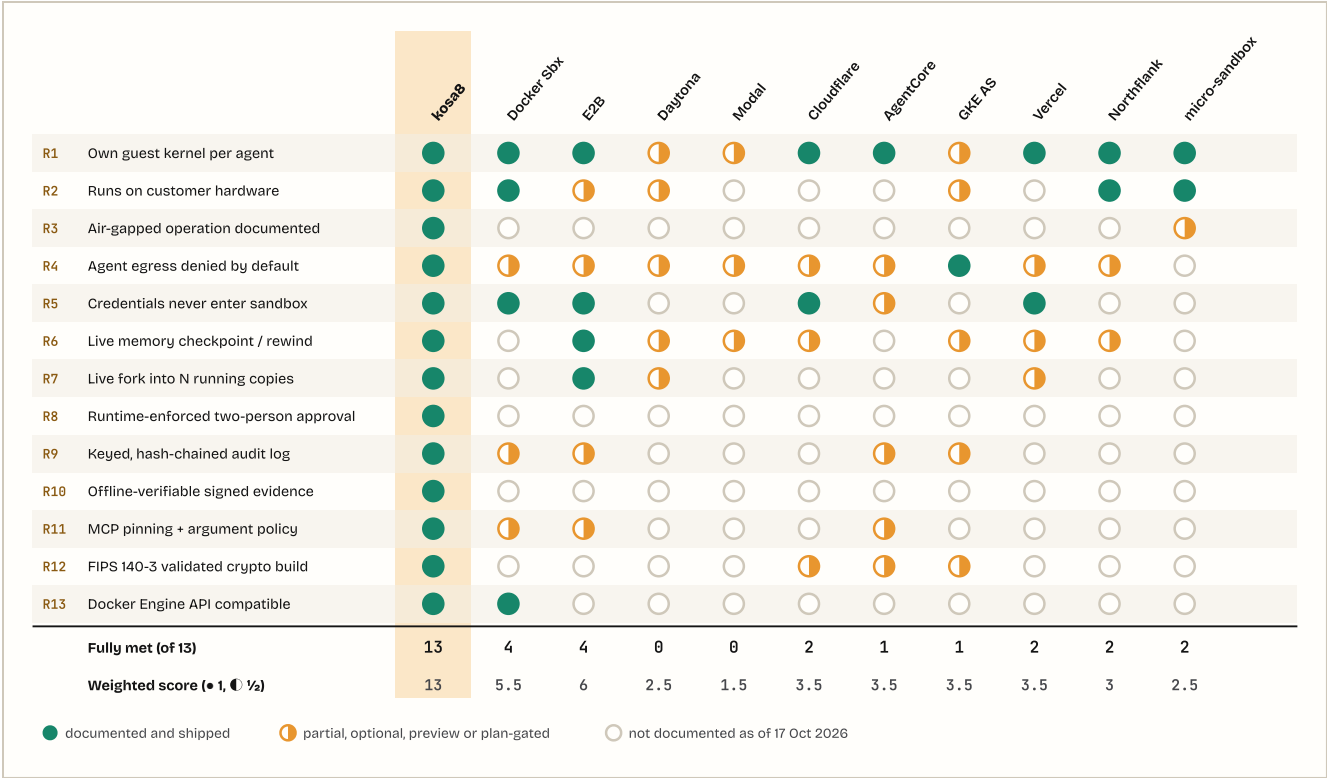


Figure 13. Requirements scorecard, as documented on 17 October 2026. kosa8 is the only system that fully meets all thirteen requirements. The next most complete systems, Docker Sandboxes and E2B, fully meet four each.

Figure 13 is the central comparative result. **kosa8 fully meets 13 of 13 requirements; no other system fully meets more than 4.** Four requirements (R8 two-person approval, R10 offline-verifiable evidence, R3 documented air-gap operation and, among sandboxes, R12 a FIPS build) are met fully by kosa8 alone. Isolation (R1) is broadly met, which confirms that isolation has commoditised; credential brokering (R5) is now shared with Docker, E2B, Cloudflare and Vercel; live fork (R7) is shared with E2B in the cloud. kosa8's advantage is not any single mechanism but the conjunction: the scorecard's weighted total is 13 for kosa8 against 6 for the next system.

Two scoring decisions deserve transparency. kosa8 receives R4 because agent-created and brokered sandboxes default to deny; person-created sandboxes default to `any`, which a regulated deployment overrides with the signed `default_egress` policy field. kosa8 receives R12 for a build on Go's CMVP-validated module, not for a product certificate, which it does not hold.

7.3 Detailed comparison

Table 9. Comparison of agent execution systems by documented capability (17 October 2026).
Sources: kosa8 v0.9.36 source; vendor documentation [47]–[55]; secondary surveys [63], [64] only where marked in the text.

System	Isolation	Where it runs	Default egress	Secrets in sandbox	Snapshot / fork	Approval	Audit	Compliance held
kosa8	microVM (VZ, CH/KVM)	Local, self-host, air-gap	deny for agents	No (broker)	Live memory; fork N	Two-person, signed	Keyed hash chain; evidence	FIPS build; SOC 2 not yet
Docker Sandboxes	microVM (own VMM)	Local; Docker cloud	deny + broad allow-list	No	Persist only	Org policy (paid)	Metadata, paid plan	Corporate SOC 2
E2B	Firecracker	SaaS; BYOC (ent.)	open	No	Pause, snapshot, fork	—	Runtime logs	SOC 2 Type II
Daytona	Container (VM option)	SaaS; BYOC	tiered	n/d	Memory snapshot; fork exp.	—	n/d	Claimed, unverified
Modal	gVisor	SaaS	open	Yes (env vars)	FS; memory alpha	—	n/d	SOC 2 II; HIPAA BAA
Cloudflare Sandbox	Container in VM	SaaS	open unless allowedHosts	No (outbound handlers)	FS backups; sleep	Code, not approval	n/d	Corporate
AWS AgentCore	microVM per session	SaaS (AWS)	restricted mode	Token vault	Session state	IAM / Cedar	CloudTrail	AWS programs
GKE Agent Sandbox	gVisor	Customer GKE	deny	n/d	Pod snapshots (preview)	RBAC	Cloud Audit Logs	GCP programs
Vercel Sandbox	Firecracker	SaaS	allow-all	No (brokering)	Disk snapshot; fork	—	n/d	Corporate SOC 2
Northflank	Kata or gVisor	SaaS; BYOC incl. on-prem	open	n/d	Snapshot API	—	SIEM guidance	SOC 2 Type 2
microsandbox	libkrun microVM	Local, self-host	n/d	n/d	—	—	—	—

n/d = not documented in the sources reviewed; this is not a claim that the capability is absent.

7.4 Scenario analysis

Requirements measure capability; scenarios measure outcome. We define eight end-to-end scenarios drawn from the incidents in §2.4 and trace each through kosa8 and through the two dominant alternatives: a typical hosted microVM sandbox in its default configuration, and a shared-kernel container. Outcomes for kosa8 are grounded in the mechanisms of §4 and, where marked, in our measurements.

Table 10. Scenario outcomes. ● contained or fully handled; ◐ partially; ○ not handled by the infrastructure.

Scenario	Attack	kosa8	Hosted microVM (default)	Shared-kernel container
S1 Lethal-trifecta exfiltration	Injected issue tells agent to POST secrets to attacker	● Unlisted host refused and audited (Table 6); no real key to steal	◐ Open egress by default; key often brokered	○ Open network; keys in env
S2 Destructive action	Agent drops a database during a freeze	● Rewind in ~0.5 s; destructive class can require approval	◐ Snapshot on some; no approval gate	○ No memory snapshot
S3 Malicious dependency	Post-install hunts ~/.ssh, ~/.aws, .env	● Workspace only; no host secrets; egress denied	● Vendor VM has no host secrets	◐ Depends on mounts
S4 Escape attempt	runc-class CVE, Docker-socket abuse	● Separate kernel; sockets unmountable	● Separate kernel	○ Shared kernel
S5 MCP rug pull	Tool definition changes after approval	● Withheld until re-approved; server in own VM	○ Not documented	○ Not handled
S6 Assessor audit	Reconstruct who did what; verify integrity	● Signed evidence verifies offline; keyed chain	◐ Vendor logs, trust the vendor	○ Daemon events only
S7 Air-gapped site	Install, license, run, verify with no network	● Offline bundle, local licence, offline verify	○ Requires vendor cloud	◐ Engine runs; no governance
S8 Parallel exploration	Fork running state into 8 variants	● Live fork, all-or-nothing	◐ Cloud fork on some vendors	○ Not available

kosa8 contains or fully handles all eight scenarios. The hosted default handles two fully, and the shared-kernel container none. The decisive scenarios are S5, S6 and S7, which depend on governance and sovereignty rather than isolation, and which no other system we reviewed documents.

7.5 Where kosa8 does not lead

A complete comparison must also say where others are ahead. **Start latency:** hosted vendors advertise sandbox starts from 25 to 150 ms from warm pools (vendor claims; one independent measurement found E2B resume at about 662 ms [63]), while full kosa8 sandbox creation takes about 4 s on our host. These numbers measure different things and are not directly comparable, but the user-visible gap is real for workloads that create many short-lived sandboxes. **Third-party attestation:** E2B, Modal and Northflank hold SOC 2 Type II reports; kosa8 holds none yet. **Elastic scale:** cloud services scale to thousands of concurrent sandboxes on demand; a local-first system is bounded by the host's memory, and kosa8 clones do not share memory pages. **Platform maturity:**

the Linux arm64 engine is opt-in and Windows is preview. None of these weakens a security goal; all are on the roadmap in §9.

8 Discussion

8.1 Why local-first is a security property

Hosted sandboxes place the agent's code, prompts, tool outputs and logs with a third party. For many teams that is acceptable. For a defence supplier handling CUI, a bank under DORA, a hospital under HIPAA, or a laboratory under 21 CFR Part 11, it moves the most sensitive activity in the organisation outside its boundary and adds a vendor to its assessment scope. kosa8 inverts the deployment: the security infrastructure comes to the data. On our measurements nothing leaves the host in deny mode, the licence check works offline, and evidence verifies without contacting anyone. That is the property that lets an assessor evaluate the system as part of the customer's own boundary.

8.2 Defence in depth across the agent stack

kosa8 is complementary to model-level and design-level defences. CaMeL-style information-flow control [4] and the Rule of Two [60] reduce what an injected agent attempts; harness permission prompts [61] reduce what it is allowed to ask for; kosa8 bounds what it can achieve and records what it did. Each layer fails independently, which is the point.

Table 11. kosa8 evidence by regulatory control (illustrative; a mapping is evidence, not an assessment).

Regime and control	Requirement	kosa8 evidence
NIST SP 800-53 AU-9, AU-10 [41]	Protect audit information; non-repudiation	Keyed hash chain; signed approvals embedded; off-host forwarding
NIST SP 800-171 3.13.1 / 3.13.11 [39]	Boundary protection; FIPS-validated crypto for CUI	Host egress guard; FIPS build; CUI labels that only rise
EU AI Act Art. 12, Art. 14 [35]	Automatic logging; human oversight and interruption	Audit log; two-person approvals; rewind
DORA Art. 28 [37]	ICT third-party risk	Runs on the entity's own hardware; no hosted dependency
HIPAA §164.312(b) [40]	Audit controls	Audit log with actor attribution; evidence export
21 CFR Part 11 §11.50, §11.70	Signature manifestation and linking	Approval records bind signer, meaning and digest of the action

8.3 Positioning

The comparison suggests a market structure. Cloud providers sell isolated compute and must host it. Docker brings isolation to the desktop and sells developer seats. kosa8 sells *control*: the policy, approvals and proof a compliance buyer needs, wherever the agent runs, including where no cloud can go. As isolation standardises, that layer becomes the scarce one.

9 Limitations and future work

- **Default egress for person-created sandboxes is any.** Agent-created and brokered sandboxes default to deny, and a signed policy can force deny everywhere, but the out-of-the-box default should be deny. *Recommendation:* change the default before general availability.
- **Shared engine VM.** Docker-API containers share one guest kernel unless `require_sandbox` is set. *Recommendation:* make `require_sandbox` the default in governed installs.
- **Host-root adversary.** The audit key is symmetric and local. *Future work:* anchor the chain tip in an external transparency log [71] or a hardware key (Secure Enclave, TPM).
- **Broker scope.** CONNECT tunnels are opaque and bodies are not inspected; the placeholder is spendable within its route; one kit still passes a token in.
- **Egress residuals.** Admitted addresses are open on all ports; wildcard names permit a DNS covert channel; DNS over TCP is not served.
- **Performance.** Sandbox creation (≈ 4 s) and the unexplained post-restore slow mode should be addressed, for example with pre-warmed sandbox pools and lazy restore. Clones do not share memory; density is unmeasured.
- **Assurance.** No external penetration test, SOC 2 report or product CMVP certificate yet; these are funded next steps. Fuzzing should run continuously rather than as a short campaign.
- **Identity.** Agent identity uses bearer tokens; CAC/PIV and SSO signers are designed but not built, which limits Part 11 §11.200 two-factor signing.
- **Platforms.** Linux arm64 is opt-in and Windows is preview; snapshots are not portable between macOS and Linux.
- **Evaluation scope.** Measurements are from one macOS host; fork was not re-measured; competitors were assessed from documentation. An independent, same-hardware benchmark across vendors is the most valuable missing experiment.

10 Related work

Lightweight virtualisation. Firecracker [27] established the microVM as the unit of serverless isolation; Manco et al. showed VMs can be lighter than containers [29]; X-Containers explored the space between [30]; Kata Containers [58] and Cloud Hypervisor [57] bring VMs to container orchestration. gVisor [28] trades performance for a memory-safe kernel boundary. kosa8 builds on these and contributes what sits above them.

Agent security. Prompt-injection attacks [1], benchmarks [5], [7], [8] and adaptive evaluations [3] motivate containment; design patterns [4], [6], [60], [62] constrain agents from within. Industry guidance from OWASP [9]–[12], MITRE ATLAS [13], NIST [14], [15] and the MCP specification [59] catalogues risks; Anthropic's sandboxing of Claude Code [61] applies OS-level isolation to a single harness. kosa8 is, to our knowledge, the first system to combine hardware isolation with runtime-enforced governance and verifiable evidence for arbitrary agents.

Secure logging. Forward-integrity logs [67], [65] and tamper-evident history trees [66] underlie kosa8's audit chain; Certificate Transparency [71] shows how public anchoring removes trust in the log operator, our main future direction.

Commercial sandboxes. E2B, Daytona, Modal, Vercel, Cloudflare, AWS AgentCore, Google's GKE Agent Sandbox, Northflank and Docker Sandboxes [47]–[55] provide isolated execution, mostly as cloud services. Surveys [63], [64] compare them on isolation, start latency and price; none of those dimensions captures governance, which is the focus of §7.

11 Conclusion

Agents will be steered by adversaries; the only question is what they can do when it happens. kosa8 answers with containment and proof: a sealed machine per agent on the customer's own hardware, a network decided on the host, keys the agent never holds, mistakes undone in half a second, risky actions held for named humans, and a record that an assessor can verify offline. Our evaluation of the released software confirms the security mechanisms work as designed, reports performance honestly, including where it falls short, and shows that kosa8 is the only system among eleven that meets all thirteen requirements for agents in regulated environments. As hardware isolation becomes a commodity, that combination of control, reversibility and verifiable evidence is what makes kosa8 the most complete security infrastructure for AI agents available today.

Appendix A: Reproducibility

All commands below were run on the setup in Table 4. The harness scripts and raw JSON results are retained by kosa8 Labs with this report.

```
# Unit tests on the released source
git archive v0.9.36 | tar -x -C src && cd src
go test -count=1 -json ./pkg/... ./internal/... ./cmd/... > gotest.json

# Project performance gates against the installed release
KOSA8_BENCH=1 KOSA8=$(which kosa8) KOSA8D=$(which kosa8d) sh bench/run.sh

# Lifecycle latency (n = 10 per operation)
kosa8 sandbox create S --egress deny          # timed, then: kosa8 sandbox rm S
kosa8 checkpoint S --label tN                 # timed
kosa8 rewind S -y                             # timed

# Egress probes, one sandbox per policy
kosa8 sandbox create P --egress deny | --egress-allow example.com | (default) | --e
kosa8 sandbox pull P alpine
kosa8 sandbox run P alpine sh -c 'wget -T 5 -q -O /dev/null URL && echo REACHED ||
kosa8 sandbox egress P

# Fuzzing (120 s each)
go test ./pkg/vnet -run '^$' -fuzz '^FuzzGuardFrame$' -fuzztime=120s
go test ./pkg/vnet -run '^$' -fuzz '^FuzzGuardDNS$' -fuzztime=120s
go test ./pkg/schema -run '^$' -fuzz '^FuzzEvaluate$' -fuzztime=120s
go test ./pkg/scorer -run '^$' -fuzz '^FuzzHeuristic$' -fuzztime=120s
```

Table 12. Coverage-guided fuzzing of attacker-reachable parsers, kosa8 v0.9.36, 120 s per target, Apple M4 (this study).

Target	Package	Executions	Corpus entries	Result
FuzzGuardFrame	pkg/vnet	3,309,100	174	● no failure
FuzzGuardDNS	pkg/vnet	1,519,659	164	● no failure
FuzzEvaluate	pkg/schema	3,788,951	694	● no failure
FuzzHeuristic	pkg/scorer	396,021	443	● no failure
Total		9,013,731	1,475	0 crashes, 0 hangs

References

- [1] 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. *Proc. 16th ACM Workshop on Artificial Intelligence and Security (AISec '23)*, pp. 79–90, 2023. arXiv:2302.12173.
- [2] S. Willison. The lethal trifecta for AI agents: private data, untrusted content, and external communication. simonwillison.net, 16 June 2025.
- [3] M. Nasr, N. Carlini et al. The attacker moves second: Stronger adaptive attacks bypass defenses against LLM jailbreaks and prompt injections. *USENIX Security Symposium*, 2026. arXiv:2510.09023.
- [4] E. Debenedetti, I. Shumailov, T. Fan, J. Hayes, N. Carlini, D. Fabian, C. Kern, C. Shi, A. Terzis and F. Tramèr. Defeating prompt injections by design. arXiv:2503.18813, 2025.
- [5] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer and F. Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. *NeurIPS Datasets and Benchmarks*, 2024.
- [6] L. Beurer-Kellner, B. Buesser, A.-M. Crețu, E. Debenedetti et al. Design patterns for securing LLM agents against prompt injections. arXiv:2506.08837, 2025.
- [7] Q. Zhan, Z. Liang, Z. Ying and D. Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated LLM agents. *Findings of ACL*, 2024.
- [8] Y. Ruan et al. Identifying the risks of LM agents with an LM-emulated sandbox. *ICLR*, 2024.
- [9] OWASP GenAI Security Project. OWASP Top 10 for LLM Applications 2025. November 2024.
- [10] OWASP GenAI Security Project. Agentic AI: Threats and Mitigations, v1.0. February 2025.
- [11] OWASP GenAI Security Project. OWASP Top 10 for Agentic Applications for 2026. December 2025.
- [12] OWASP. MCP Top 10 (beta). owasp.org/www-project-mcp-top-10.
- [13] MITRE. ATLAS: Adversarial Threat Landscape for Artificial-Intelligence Systems. atlas.mitre.org.
- [14] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1, January 2023.
- [15] C. Autio, R. Schwartz, J. Dunietz, S. Jain, M. Stanley, E. Tabassi, P. Hall and K. Roberts. AI RMF: Generative Artificial Intelligence Profile. NIST AI 600-1, July 2024.
- [16] Anthropic. Disrupting the first reported AI-orchestrated cyber espionage campaign. 13 November 2025.
- [17] Wiz Research. s1ngularity: Supply chain attack leaks secrets on GitHub. August 2025.
- [18] CISA. Widespread supply chain compromise impacting npm ecosystem. Alert, 23 September 2025.
- [19] Amazon Web Services. Security bulletin AWS-2025-015: Amazon Q Developer for VS Code (CVE-2025-8217). 23 July 2025.
- [20] eWeek. Replit AI coding assistant failure. July 2025; AI Incident Database entries for the same event.
- [21] Invariant Labs. MCP security notification: Tool poisoning attacks. 1 April 2025.

- [22] Invariant Labs. GitHub MCP exploited: Accessing private repositories via MCP. May 2025.
- [23] Koi Security. postmark-mcp: First malicious MCP server found in the wild. September 2025.
- [24] JFrog. CVE-2025-6514: Critical mcp-remote RCE vulnerability. 2025.
- [25] Check Point Research. Cursor IDE: MCPoison (CVE-2025-54136). 2025.
- [26] BleepingComputer. Cursor, Codex, Gemini CLI, Antigravity hit by sandbox escapes. July 2026.
- [27] A. Agache, M. Brooker, A. Florescu, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka and D.-M. Popa. Firecracker: Lightweight virtualization for serverless applications. *17th USENIX NSDI*, pp. 419–434, 2020.
- [28] E. G. Young, P. Zhu, T. Caraza-Harter, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau. The true cost of containing: A gVisor case study. *11th USENIX HotCloud*, 2019.
- [29] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu and F. Huici. My VM is lighter (and safer) than your container. *Proc. 26th ACM SOSP*, 2017.
- [30] Z. Shen, Z. Sun, G.-E. Sela, E. Bagdasaryan, C. Delimitrou, R. Van Renesse and H. Weatherspoon. X-Containers: Breaking down barriers to improve performance and isolation of cloud-native containers. *ASPLOS*, 2019.
- [31] NIST NVD. CVE-2019-5736: runc container breakout via /proc/self/exe. 2019.
- [32] Snyk. Leaky Vessels: runc process.cwd container breakout (CVE-2024-21626). 31 January 2024.
- [33] oss-security. runc: CVE-2025-31133, CVE-2025-52565, CVE-2025-52881 container escapes. 5 November 2025.
- [34] NIST NVD. CVE-2026-5747: Firecracker virtio-PCI out-of-bounds write. 2026.
- [35] European Union. Regulation (EU) 2024/1689 (Artificial Intelligence Act). Official Journal, 2024.
- [36] European Union. Regulation (EU) 2026/1744 amending Regulation (EU) 2024/1689 (Digital Omnibus on AI). Official Journal, 24 July 2026.
- [37] European Union. Regulation (EU) 2022/2554 on digital operational resilience for the financial sector (DORA). 2022.
- [38] U.S. Department of Defense. Cybersecurity Maturity Model Certification (CMMC) Program, 32 CFR Part 170; DFARS 252.204-7021.
- [39] R. Ross, V. Pillitteri, K. Dempsey, M. Riddle and G. Guissanie. Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations. NIST SP 800-171 Rev. 2, 2020.
- [40] U.S. Department of Health and Human Services. HIPAA Security Rule, 45 CFR §164.312 Technical safeguards.
- [41] Joint Task Force. Security and Privacy Controls for Information Systems and Organizations. NIST SP 800-53 Rev. 5, 2020.
- [42] NIST. FIPS 140-3: Security Requirements for Cryptographic Modules; Cryptographic Module Validation Program. 2019–.
- [43] Gartner. Gartner predicts over 40 percent of agentic AI projects will be canceled by end of 2027. Press release, 25 June 2025.
- [44] Gartner. Gartner predicts 40 percent of enterprise apps will feature task-specific AI agents by 2026. Press release, 26 August 2025.
- [45] Gartner. Gartner forecasts worldwide AI spending to grow 47 percent in 2026. Press release, 19 May 2026.
- [46] McKinsey Global Institute. The economic potential of generative AI: The next productivity frontier. June 2023.

- [47] Docker Inc. Docker Sandboxes documentation: security and governance; Docker AI Governance audit logs. docs.docker.com, 2026.
- [48] E2B. Enterprise and security documentation; sandbox network API. e2b.dev, 2026.
- [49] Cloudflare. Sandbox SDK security concepts; Containers and Sandbox GA changelog, 13 April 2026. developers.cloudflare.com.
- [50] Vercel. Sandbox firewall documentation; A sandbox without a network boundary is only half a sandbox. vercel.com, 2026.
- [51] Amazon Web Services. Amazon Bedrock AgentCore developer guide: runtime sessions. docs.aws.amazon.com, 2026.
- [52] Google Cloud. GKE Agent Sandbox concepts; Vertex AI Agent Engine code execution. cloud.google.com, 2026.
- [53] Modal. HIPAA compliance and gVisor isolation. modal.com, 2026.
- [54] Northflank. Sandboxes product documentation. northflank.com, 2026.
- [55] Super Rad Company. microsandbox. github.com/superradcompany/microsandbox.
- [56] Apple. Virtualization framework: VZVirtualMachine save and restore of machine state. developer.apple.com.
- [57] Cloud Hypervisor project. cloudfhypervisor.org; Linux Foundation.
- [58] Kata Containers project. katacontainers.io; OpenInfra Foundation.
- [59] Model Context Protocol. Specification 2025-06-18: Security best practices; Specification 2026-07-28 changelog. modelcontextprotocol.io.
- [60] Meta AI. Agents Rule of Two: A practical approach to AI agent security. 31 October 2025.
- [61] Anthropic. Beyond permission prompts: making Claude Code more secure and autonomous. October 2025.
- [62] S. Willison. The dual LLM pattern for building AI assistants that can resist prompt injection. 25 April 2023.
- [63] LogRocket. Comparing AI agent sandbox platforms: E2B, Modal, Daytona and more. 2026.
- [64] Upstash. AI agent sandbox providers compared. 17 September 2026.
- [65] B. Schneier and J. Kelsey. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security*, 2(2):159–176, 1999.
- [66] S. A. Crosby and D. S. Wallach. Efficient data structures for tamper-evident logging. *18th USENIX Security Symposium*, 2009.
- [67] M. Bellare and B. Yee. Forward integrity for secure audit logs. Technical report, University of California, San Diego, 1997.
- [68] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.
- [69] N. Hardy. The confused deputy (or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review*, 22(4):36–38, 1988.
- [70] kosa8 Labs. kosa8 v0.9.36 source, design notes (docs/design), benchmark harness (bench/) and release records. Internal, October 2026.
- [71] B. Laurie, A. Langley and E. Kasper. Certificate Transparency. RFC 6962, IETF, 2013.

kosa8 Labs · Technical Report · © 2026 kosa8 Labs · All rights reserved

© 2026 kosa8 Labs. All rights reserved.

This report, including its text, figures, data, measurements, designs and analysis, is the property of kosa8 Labs. It is published for reading only. No licence or other right is granted, by implication, estoppel or otherwise.

No part of this report may be copied, reproduced, redistributed, adapted or republished, in whole or in part, without the prior written permission of kosa8 Labs. Brief quotation with attribution for the purpose of review or commentary is permitted where the law allows it.

Third-party names and marks belong to their owners and are used only to identify products. Statements about third-party products reflect their public documentation as reviewed on 17 October 2026 and are not claims about undocumented capabilities.