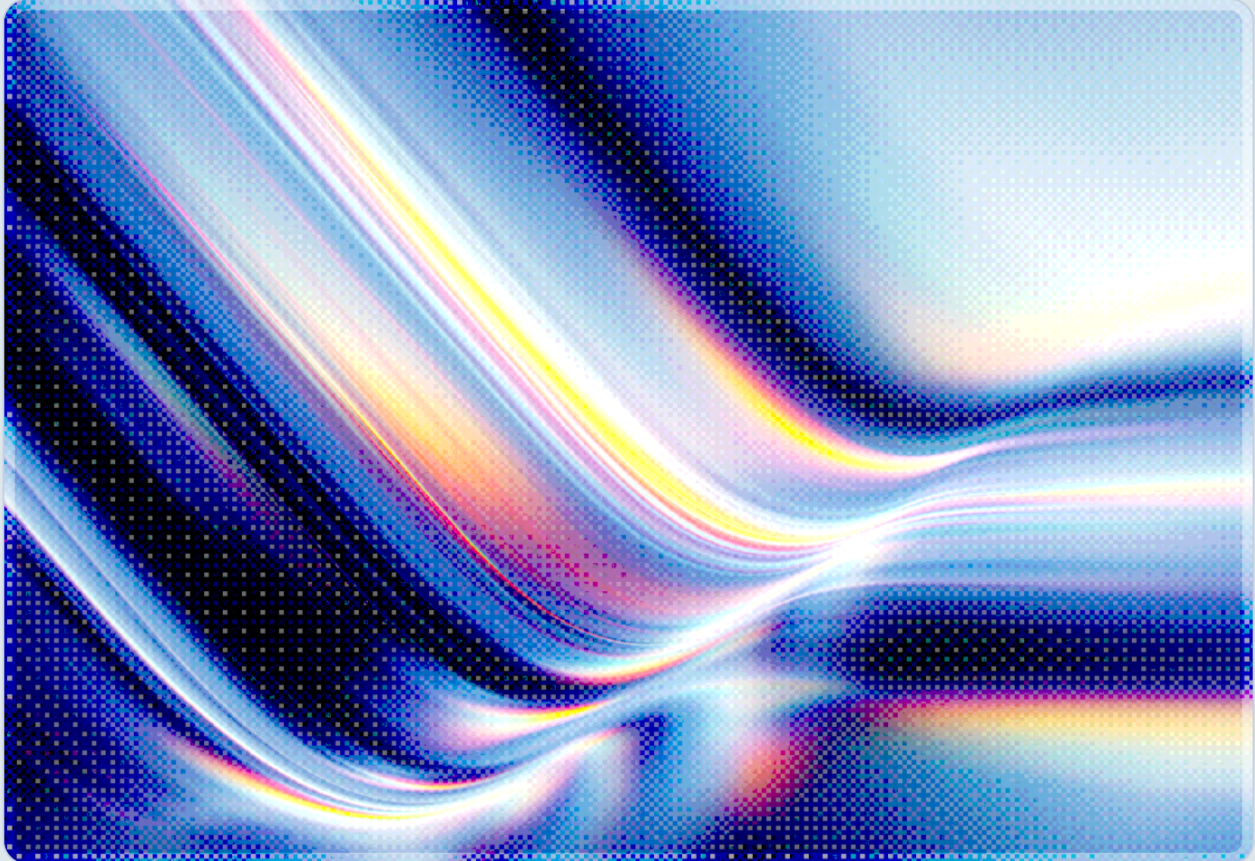


Agent Containment Runtimes Compared: Capabilities, CSA Alignment, and Gaps

Google CAPSEM, NVIDIA OpenShell, Microsoft Execution
Containers, and the Broader Agent Sandbox Landscape

2026-10-27

 AI-assisted Rapid Research



CSAI

cloud
security
alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Table of Contents

- Executive Summary 5
- 1. Introduction and Background 7
 - 1.1 Why Containment Became a Platform Concern
 - 1.2 Scope, Method, and Verification Posture
- 2. A Reference Model for Comparing Agent Containment 9
- 3. The Three Focal Runtimes 11
 - 3.1 CAPSEM (github.com/google/capsem)
 - 3.2 NVIDIA OpenShell
 - 3.3 Microsoft Execution Containers (MXC)
 - 3.4 Focal Runtimes Side by Side
- 4. The Broader Landscape 16
 - 4.1 Isolation Substrates
 - 4.2 Cloud-Managed Agent Sandboxes
 - 4.3 Agent-Embedded and Developer-Tool Sandboxes
 - 4.4 Landscape Summary
- 5. Cross-Cutting Analysis 20
 - 5.1 Isolation Strength Is Necessary but Not Sufficient
 - 5.2 Who Holds the Policy Pen
 - 5.3 Credential Brokering Is Converging
 - 5.4 Egress and MCP Mediation
 - 5.5 Identity: The Weakest Layer
 - 5.6 Telemetry and the Evidence Problem
 - 5.7 Openness, Maturity, and Unverified Claims
- 6. CSA Resource Alignment 24
 - 6.1 Directly Relevant CSA Research
 - 6.2 Mapping to AICM v1.1 Controls
 - 6.3 MAESTRO Threat Model Alignment
 - 6.4 STAR for AI, SSRM, and CCM
 - 6.5 Zero Trust and Non-Human Identity
- 7. Gaps in the CSA Portfolio 28

8. Conclusions and Recommendations 30

 8.1 Conclusions

 8.2 Recommended CSA Deliverables

 8.3 Recommendations for Enterprises

 8.4 Recommendations for Runtime Vendors

References 32

Executive Summary

During 2026 the question of how to contain an autonomous AI agent moved from research and individual product features to platform infrastructure. Microsoft declared its Execution Containers (MXC) generally available on October 7, 2026, describing them as "a policy-driven execution layer for untrusted code or dynamically generated workloads" across Windows, macOS, and Linux [3]. NVIDIA introduced OpenShell in March 2026 and, by late September, had paired its 0.1.0 release with a hardware watchdog on BlueField-4 data processing units [7][9]. Google's cloud teams shipped Cloud Run sandboxes into public preview and offered a managed form of the Kubernetes Agent Sandbox project on GKE [16][17], while a separate Apache-2.0 project in Google's GitHub organization, CAPSEM, packages per-session virtual machines for coding agents on developer workstations [1]. Around these headline efforts sits a dense ecosystem of microVM services, container sandboxes, and agent-embedded sandboxes from E2B, Cloudflare, Fly.io, Docker, Anthropic, and OpenAI.

The runtimes differ substantially in where they draw the isolation boundary. Some rely on operating-system sandbox primitives such as AppContainer, Seatbelt, Bubblewrap, Landlock, and seccomp; others interpose a user-space kernel such as gVisor; still others give every session its own guest kernel inside a microVM. Yet the more consequential differences lie above the boundary. The runtimes vary in who writes the policy and whether an enterprise can override a developer's declared needs, in whether credentials are ever exposed to the agent process, in how outbound network traffic and Model Context Protocol (MCP) tool calls are mediated, in whether actions are bound to a distinct non-human identity, and in whether the runtime produces audit evidence in a structured, portable format. Our analysis finds a clear trend toward deny-by-default egress and proxy-side credential injection, but little convergence on policy schemas, identity binding, or audit formats.

Independent research during 2026 has also shown that strong isolation is necessary but not sufficient. Pillar Security's July 2026 "Week of Sandbox Escapes" disclosed seven issues across four widely used coding agents in which no sandbox was technically broken; instead, an agent wrote a file that a trusted, unsandboxed component later executed [28][29]. Containment runtimes therefore need to be evaluated as systems, including the hand-off points between the sandbox and the host tooling around it.

CSA's existing portfolio supplies much of the vocabulary needed to evaluate these runtimes. The AI Controls Matrix (AICM) v1.1 already contains controls for agent security boundaries (AIS-11), AI sandboxing (AIS-13), and agent access restriction (IAM-18) [36]; MAESTRO provides a layered threat model [35]; the Autonomous Action Runtime Management (AARM) specification defines runtime authorization behavior for agent actions [31][32]; and the Agentic Trust Framework (ATF) frames agent governance in Zero Trust terms [34]. What CSA does not yet offer is a runtime-containment profile that translates those controls into testable requirements, a conformance test suite that could distinguish a strong sandbox from a weak one, or a shared security responsibility model (SSRM) that allocates control ownership among the operating-system vendor, the runtime

vendor, the agent developer, and the deploying enterprise. This paper recommends six concrete CSA deliverables to close those gaps, beginning with an AICM Agent Runtime Containment Profile and an accompanying conformance test suite aligned with AARM.

1. Introduction and Background

1.1 Why Containment Became a Platform Concern

An AI agent that can run shell commands, edit files, call APIs, and invoke MCP tools holds whatever authority the surrounding environment grants it. On a developer workstation that often means the user's full file system, cached cloud credentials, SSH keys, and unrestricted network access. In a CI runner or a production data plane it can mean deployment credentials and access to customer data. Because the agent's next action is determined by model output that can be influenced by untrusted input – a poisoned tool description, a malicious README, a crafted issue title – this ambient authority becomes the attacker's authority the moment an injection succeeds. Prompt-level guardrails reduce the probability of a harmful instruction being followed, but they do not bound the consequences when one is.

Execution containment addresses the consequence side of the problem. Rather than trying to prevent the model from ever being misled, a containment runtime constrains what the agent's process can do at the operating-system, hypervisor, or network layer, using enforcement the agent cannot disable. Anthropic's engineering team articulated the logic plainly when describing Claude Code's sandbox: without network isolation a compromised agent could exfiltrate files such as SSH keys, and without file-system isolation it could escape the sandbox and reach the network, so both layers are required [25]. Anthropic also reported, as a vendor figure, that sandboxing reduced permission prompts by 84% in its internal usage, which illustrates a second motivation: containment allows agents to operate with fewer human interruptions without proportionally increasing risk [25].

By mid-2026, three forces pushed containment from individual product features toward shared platform infrastructure. Operating-system vendors began treating agents as a distinct class of workload that needed first-class isolation and manageability, as Microsoft's Build 2026 announcements made explicit [4]. Accelerated-computing vendors began positioning runtime policy enforcement as part of an agent "operating layer," as NVIDIA did with OpenShell and its Agent Toolkit [7]. And cloud providers began offering sandboxes as a native compute primitive so that developers would not need to assemble their own microVM fleets [17]. The result is a crowded and fast-moving landscape in which superficially similar products make very different security guarantees.

1.2 Scope, Method, and Verification Posture

This paper focuses on three runtimes – Google CAPSEM, NVIDIA OpenShell, and Microsoft Execution Containers – and situates them among a broader set of comparable projects: the gVisor, Firecracker, and Kata Containers isolation substrates; the Kubernetes Agent Sandbox project and its GKE managed offering; Google Cloud Run sandboxes; E2B; Cloudflare Sandboxes; Fly.io Sprites; Docker Sandboxes; Anthropic's Claude Code sandbox runtime; and the OpenAI Codex sandbox. Each was verified against primary sources (vendor

documentation, official blogs, and source repositories) where available, supplemented by reputable trade press. Where only secondary sources were available, or where sources conflict, the text says so and attributes the claim to its source rather than presenting it as confirmed.

The analysis is a point-in-time assessment as of October 27, 2026. Several of the runtimes discussed are pre-1.0, in preview, or carry experimental backends, and their capabilities are changing on a monthly cadence. Readers should treat feature-level statements as a snapshot and confirm current behavior against vendor documentation before relying on them for procurement or architectural decisions. The paper is deliberately vendor-neutral: it does not rank products, and it evaluates published claims rather than conducting independent penetration testing.

2. A Reference Model for Comparing Agent Containment

Comparing containment runtimes requires a common frame, because vendors describe their products in incompatible vocabularies. One emphasizes "kernel instrumentation," another "policy-driven execution," and a third "hardware-isolated VMs." Beneath the marketing, every runtime examined in this paper makes choices along the same eight dimensions, and those choices determine what security properties a deploying organization actually receives.

The first dimension, the **isolation boundary**, is the technical mechanism that separates the agent's process from the host. Boundaries fall roughly into three families: operating-system sandboxes that restrict a process using kernel features of the host (AppContainer on Windows, Seatbelt on macOS, Bubblewrap, Landlock, and seccomp on Linux); user-space kernels such as gVisor that intercept system calls before they reach the host kernel [12]; and hardware-virtualized boundaries in which each workload receives its own guest kernel, as with Firecracker microVMs [13] or Kata Containers [14]. The families trade startup latency and resource overhead against the size of the shared attack surface. The second dimension, the **policy model**, describes how permitted behavior is expressed, who authors it, when it is fixed, and whether a higher authority such as an enterprise administrator can constrain what a developer has declared.

The third and fourth dimensions concern the two most common exfiltration paths. **Credential handling** determines whether secrets are ever present inside the sandbox or are instead injected by a broker outside it only for approved destinations. **Network egress and tool mediation** determines whether outbound connections are denied by default, how allow-lists are expressed, whether traffic is inspected, and whether MCP tool invocations pass through a mediation point that can evaluate them individually. The fifth dimension, **identity**, asks whether the agent's actions are attributable to a distinct non-human identity rather than to the human user who launched it, which is a precondition for per-agent policy and accountability.

The final three dimensions concern assurance and adoption. **Telemetry and audit** covers what the runtime records, in what format, and whether the record can be trusted if the agent is compromised. **Licensing and openness** determines whether the enforcement code can be inspected and whether the policy format is portable. **Maturity** captures release status, platform coverage, and how much of the advertised capability is generally available versus experimental. Table 1 summarizes the reference model and the question each dimension answers.

Table 1. Comparison dimensions for agent containment runtimes

Dimension	Core question	Why it matters
Isolation boundary	What separates the agent from the host, and how large is the shared attack surface?	Determines blast radius if the agent or its code is malicious
Policy model	Who writes policy, in what format, and can an enterprise override a developer?	Determines whether organizational posture can be enforced independently of each agent vendor
Credential handling	Are real secrets ever visible inside the sandbox?	Credential theft is the most direct path from injection to persistent compromise
Egress and MCP mediation	Is outbound traffic deny-by-default, and are tool calls individually evaluated?	Exfiltration and tool abuse bypass file-system isolation entirely
Identity	Are actions bound to a distinct non-human identity?	Enables per-agent policy, revocation, and attribution
Telemetry and audit	What is logged, in what format, and is it tamper-resistant?	Supplies the evidence base for detection, incident response, and certification
Licensing and openness	Is enforcement code inspectable and is policy portable?	Affects verifiability and lock-in
Maturity	What is GA versus preview or experimental?	Determines what can be relied on in production today

3. The Three Focal Runtimes

3.1 CAPSEM (github.com/google/capsem)

CAPSEM is an Apache-2.0 project hosted in Google's GitHub organization that describes itself as a way to sandbox "AI coding agents in hardware-isolated Linux VMs on macOS and Linux" [1]. Its isolation boundary is a per-session virtual machine rather than a container; supported hosts are macOS 14 or later on Apple Silicon and recent Ubuntu and Debian releases on KVM-capable x86_64 or arm64 machines [1]. Third-party write-ups state that the macOS implementation uses Apple's Virtualization framework and that the project is written in Rust; the repository README does not name the hypervisor, so those specifics remain unconfirmed by primary sources. The project's documentation site describes an architecture in which a background service manages container images running inside isolated VMs, controlled through a terminal UI, a graphical UI, or Python, TypeScript, and Rust SDKs, and in which CAPSEM "sits between the agent and the outside world," inspecting requests, applying access rules, and recording activity [2].

The README advertises "full network control, HTTPS inspection, MCP tool routing, and per-session telemetry," which places CAPSEM in the small group of runtimes that explicitly mediate MCP traffic rather than treating it as ordinary network egress [1]. However, neither the README nor the documentation landing page describes the policy language, the credential-handling model, or the telemetry format in detail [1][2]. The project is in pre-release: version 0.7 is described as "undergoing release qualification," with installation packages and detailed documentation withheld until public release [1]. The repository also carries Google's standard disclaimer that "this project is not an official Google project" and is not supported by Google [1]. Readers should therefore not equate "Google CAPSEM" with a supported Google Cloud product; it is better understood as an open-source project published under Google's GitHub organization.

A note on naming. Some commentary has suggested a naming collision between the `google/capsem` repository and an unrelated VM sandbox at `capsem.org`. Our verification did not support that premise. The `google/capsem` README points readers to `docs.capsem.org` for documentation status [1], the documentation site describes the same VM-per-agent architecture [2], and independent write-ups describing the `capsem.org` installer characterize the same Rust-based, VM-isolated coding-agent sandbox with a logging network proxy. The available evidence indicates that `capsem.org` is the project site for `google/capsem` rather than a separate product, although we could not locate an explicit statement on `capsem.org` confirming the relationship. A genuine source of confusion is Capsicum, a capability-based security framework for UNIX developed by the University of Cambridge with Google researchers and integrated into FreeBSD; Capsicum is an operating-system capability model, not an agent sandbox, and is unrelated to CAPSEM.

3.2 NVIDIA OpenShell

NVIDIA introduced OpenShell as part of its Agent Toolkit, describing it in a March 23, 2026 blog post as an "open source, secure-by-design runtime for developing, deploying and governing autonomous agents," then in early preview [7]. The repository, licensed under Apache 2.0, now describes OpenShell as "the safe, private runtime for autonomous AI agents" and reports a 0.1.x stable release cadence [8]. Its central design principle is the separation of application-layer agent logic from infrastructure-layer policy enforcement, so that, in NVIDIA's words, the agent cannot override policies or leak secrets even if compromised [7]. Each agent receives its own sandbox, and a gateway acts as the control plane for sandboxes, policy, and access [7][8].

On the isolation boundary, the repository states that OpenShell "instruments the kernel to enforce policy on every file access, system call, and network connection at runtime," and lists Docker, Podman, or host virtualization as runtime requirements [8]. Secondary analyses of the early releases describe Landlock for file-system control and seccomp-BPF for process restrictions, with the gateway and policy components running as a lightweight Kubernetes (K3s) cluster inside a container; these implementation details are not stated in the current README, could not be confirmed from primary sources, and may have changed. NVIDIA's blog describes policy evaluation "at the binary, destination, method and path level" across file-system, network, and process layers, with network and inference rules updatable at runtime as developers grant approvals [7]. Platform support covers Linux, macOS on Apple Silicon, and Windows through WSL 2 (experimental), with native Windows support delivered through Microsoft MXC [8].

OpenShell's credential model is one of its most clearly specified features. The repository states that "agents never see real credentials; OpenShell adds them only to requests bound for approved endpoints," with credential providers covering inference endpoints as well as tool APIs [8]. Every network connection passes through a policy check before leaving the sandbox [8], and NVIDIA states that OpenShell keeps "a full audit trail of every allow and deny decision" [7]. Microsoft's MXC announcement adds that OpenShell's integration provides advanced network controls and OCSF-formatted auditing for enterprises [3]; the Open Cybersecurity Schema Framework (OCSF) is a vendor-neutral event schema [44], and its use would make OpenShell decision logs ingestible by existing security analytics tooling.

The 0.1.0 release introduced two policy-assurance tools: an advisor and a prover that uses formal verification to check "what a policy change would allow before it is applied" [8]. Coverage of NVIDIA's September 2026 Open Agent Safety Platform describes the prover as Z3-based and capable of reasoning about the combined permissions of an agent fleet, using the example of two sub-agents that individually comply with policy but together could read from GitHub and post the data externally [11]. Analyst commentary cautions that the prover can show a policy stays within an approved boundary, but translating human intent into that boundary remains largely manual [9]. The same platform pairs OpenShell with Sentry, a watchdog running on BlueField-4 DPUs that operates independently of the agent's host and, according to NVIDIA, can quarantine an agent even if the host is compromised [9][10][11]. Sentry depends on BlueField-4 hardware, whose deployment is still limited [9], so its out-of-band assurance is not available to most organizations today.

3.3 Microsoft Execution Containers (MXC)

Microsoft announced the MXC SDK in early preview at Build on June 2, 2026, describing it as "a cross-platform, policy-driven execution layer for agents on Windows and WSL" whose constraints "Windows enforces ... consistently at runtime" [4]. The initial preview offered process isolation, adopted by GitHub Copilot CLI, and session isolation limited to non-interactive sessions, with micro-VMs and WSL-based Linux containers on the roadmap [4][5]. On October 7, 2026, Microsoft announced general availability, positioning MXC as the containment pillar of the Windows agent platform alongside identity and manageability [3]. The SDK, versioned JSON configuration schema, documentation, and samples are published in a public GitHub repository under the MIT license, with Rust, .NET, and Node SDKs [6].

MXC is distinctive in abstracting multiple isolation boundaries behind a single policy schema. According to the GA announcement, process containers use the platform's native sandbox – AppContainer on Windows, Seatbelt on macOS, Bubblewrap on Linux – and run on all three operating systems [3]. Session containers, available only on Windows 11, run the agent under a distinct Windows account and session with its own desktop, clipboard, UI, and input boundaries [3]. WSL containers provide a Linux environment for Linux-first toolchains, and a micro-VM backend for Windows 11 and Linux is marked experimental [3]. The repository lists additional backends, including Windows Sandbox, LXC, and Hyperlight, and labels several of them experimental [6]. The developer declares what the workload needs, and the policy sits outside the agent's control so that it "cannot grant itself additional access" [3].

The policy model covers containment type, process (command, arguments, working directory, environment), file system (read-write, read-only, and denied paths), network (inbound, outbound, and loopback access), and user interface [3][6]. MXC's key architectural claim is that organizations can layer management policy on top of developer-declared requirements, so that "the same agent can run within different enterprise boundaries without the developer encoding the organization's posture" [3]. As of GA, however, the enterprise layer is largely forthcoming: Intune policy to manage MXC process containers "will soon be available," and Microsoft Entra's ability to distinguish agent activity from user activity in Agent 365 is described as "coming soon" [3]. The June announcement had described Entra-backed identities for session containers and Intune policies that could require MXC isolation [4], so the precise availability of each identity and management feature should be confirmed against current Microsoft documentation.

For telemetry, the GA announcement describes three modes on Windows process containers: an enforcement mode with no activity report, a learning mode that enforces the boundary while recording blocked operations to a JSON activity report, and a permissive mode that allows and records ungranted operations [3]. The repository documentation describes an audit mode for drafting policy with an explicit warning that it "turns off all sandbox security for the workload being analyzed" and should not be used with untrusted code, and it states that telemetry is off unless a run opts in [6]. The blog's learning mode and the repository's audit mode may be different features or different names for related functionality, and we could not resolve the question from published documentation; the distinction matters because one preserves enforcement and the other does not. Microsoft lists GitHub Copilot, OpenClaw, OpenAI Codex, Replit, LM Studio, and Unsloth AI as already supporting MXC, with Anthropic Claude Code and several others planned [3].

3.4 Focal Runtimes Side by Side

The three focal runtimes occupy different positions in the stack. CAPSEM is a workstation-oriented VM sandbox with built-in network and MCP mediation. OpenShell is a policy-enforcement runtime with a gateway control plane, credential brokering, and formal policy analysis, designed for fleets. MXC is an operating-system-vendor abstraction that maps one policy schema onto several native isolation backends and aims to connect them to enterprise identity and device management. These positions are complementary as much as competitive: OpenShell already uses MXC for native Windows support [8], and an enterprise could plausibly use MXC for endpoint agents, OpenShell for fleet policy, and a microVM service for cloud execution. Table 2 compares the three across the reference model.

Table 2. Focal runtimes compared

Dimension	CAPSEM	NVIDIA OpenShell	Microsoft MXC
Isolation boundary	Per-session Linux VM (hardware virtualization) on macOS and Linux hosts [1]	Kernel-level enforcement on file, syscall, and network activity inside per-agent sandboxes; Docker, Podman, or host virtualization required [8]	Pluggable: AppContainer, Seatbelt, or Bubblewrap process containers; Windows session containers; WSL; experimental micro-VM [3][6]
Policy model	"Access rules"; format not publicly documented [2]	Declarative file-system, network, and process policy; network and inference rules updatable at runtime; advisor and formal prover [7][8]	Versioned JSON schema declared by developer; enterprise layering via Intune announced as forthcoming [3][6]
Credential handling	Not documented [1][2]	Agents never see real credentials; injected only for approved endpoints [8]	Environment controls in process policy; no broker described [3]
Egress and MCP mediation	Full network control, HTTPS inspection, MCP tool routing [1]	Every connection policy-checked; method and path-level rules [7][8]	Inbound, outbound, and loopback rules; proxy support; advanced controls via OpenShell [3][6]
Identity	Not documented	Agent identity verification attributed by NVIDIA to Sentry hardware tier [10][11]	Distinct Windows account for session containers; Entra agent identity forthcoming [3][4]

Dimension	CAPSEM	NVIDIA OpenShell	Microsoft MXC
Telemetry and audit	Per-session telemetry; format not documented [1]	Audit trail of allow and deny decisions [7]; OCSF auditing per Microsoft [3]	JSON activity report in learning and permissive modes [3]
License	Apache-2.0 [1]	Apache-2.0 [8]	MIT [6]
Maturity	Pre-release 0.7; not an official Google product [1]	0.1.x; Windows via WSL 2 experimental [8]	GA October 7, 2026; several backends experimental; identity and Intune features forthcoming [3][6]

4. The Broader Landscape

4.1 Isolation Substrates

Most agent sandboxes are built on a small number of open-source isolation substrates, and understanding those substrates clarifies what a given product's boundary actually guarantees. gVisor, an Apache-2.0 Google project, implements an application kernel in user space that intercepts the sandboxed workload's system calls, so the workload does not interact directly with the host kernel [12]. Firecracker, an Apache-2.0 virtual machine monitor written in Rust and originally developed by AWS, launches lightweight microVMs that each run their own guest kernel with a minimal device model [13]. Kata Containers, also Apache-2.0, runs container workloads inside lightweight virtual machines while preserving container tooling compatibility [14]. Each substrate shrinks the host attack surface relative to a standard container sharing the host kernel, but they do so at different points and with different performance profiles.

These substrates matter for comparison because products that look similar at the API level may rest on different boundaries. A sandbox offered as "isolated" might use namespaces and seccomp on a shared kernel, a user-space kernel, or a dedicated guest kernel, and the residual risk of a kernel exploit differs accordingly. Procurement questionnaires that ask only whether a product "sandboxes agent code" will not surface this distinction, which is one reason this paper recommends that CSA define boundary classes in a containment profile (Section 8).

4.2 Cloud-Managed Agent Sandboxes

The Kubernetes Agent Sandbox project, an Apache-2.0 subproject under Kubernetes SIG Apps maintained in the kubernetes-sigs organization, provides custom resources for "isolated, stateful, singleton workloads, ideal for use cases like AI agent runtimes" [15]. Coverage of the project describes three primitives – Sandbox, SandboxTemplate, and SandboxClaim – and warm-pool support to reduce cold-start latency [18]. Google offers a managed add-on on GKE whose documentation describes isolation using gVisor, with Kata Containers as an alternative, and states that the add-on tracks the open-source controller's release cycle [16]. Google has claimed throughput of 300 sandboxes per second at sub-second latency, a vendor figure reported by InfoQ that we did not independently verify [18]. Because the API surface was still evolving across alpha and beta versions during 2026, organizations should pin versions carefully [15].

Google Cloud Run sandboxes, announced in public preview on July 9, 2026, take a different approach by running untrusted code inside existing Cloud Run instances rather than as separate workloads [17]. Google's announcement states that sandboxes "have zero outbound network access" by default, that egress must be explicitly requested, and that sandboxes cannot read the service's environment variables or call the Google Cloud metadata server – two controls that directly address credential theft in cloud environments [17]. The

announcement does not name the underlying isolation technology [17]. This design is notable for treating metadata-server isolation as a first-class default, a control that is not enabled by default in many container platforms and must be configured explicitly.

Several independent providers offer hosted sandboxes optimized for agent workloads. E2B publishes an Apache-2.0 SDK and runs sandboxes as Firecracker microVMs, each a full Linux VM with its own kernel [19][45]. Cloudflare announced general availability of Sandboxes and Containers in April 2026, adding credential injection, persistent code interpreters, snapshot-based session recovery, and active-CPU pricing, according to InfoQ [20][21]. Fly.io introduced Sprites in January 2026 as persistent, checkpointable Firecracker VMs for coding agents, with a hosted MCP server through which agents can create sandboxes, run commands, checkpoint and restore, and set network policy [22][23]. These services generally provide per-session VM or container boundaries and convenient lifecycle management, but they leave identity binding, enterprise policy layering, and audit formats largely to the integrating developer.

4.3 Agent-Embedded and Developer-Tool Sandboxes

A third category consists of sandboxes built into agent products themselves. Anthropic's Claude Code sandbox, described in October 2025, uses Bubblewrap on Linux and Seatbelt on macOS to restrict file-system writes to the working directory and routes all network traffic through a proxy outside the sandbox that enforces domain allow-lists [25]. For Claude Code on the web, credentials such as git tokens remain outside the sandbox, and a proxy validates a scoped credential and the requested git operation before attaching the real token [25]. The runtime is published as an Apache-2.0 open-source project, now at github.com/anthropics/sandbox-runtime [26]. OpenAI's Codex documentation states that the agent runs with network access disabled by default and edits restricted to the workspace, using Seatbelt on macOS and Landlock plus seccomp on Linux, and warns that Linux sandboxing can fail inside containers that lack the required kernel APIs [27].

Docker Sandboxes run each agent session in a microVM with its own kernel and a private Docker daemon, so agents can build and run containers without access to the host daemon; Docker lists Claude Code, Codex CLI, Copilot CLI, Gemini CLI, and Kiro among supported agents [24]. A microVM limits damage to the host, but it does not by itself prevent exfiltration over the network, so egress policy remains a separate requirement. Agent-embedded sandboxes have the advantage of being on by default for the agent's own tool calls, but each agent vendor implements its own policy semantics, which fragments enterprise governance when an organization runs several agents.

4.4 Landscape Summary

Table 3 summarizes the broader landscape. The diversity in isolation substrate is expected; the more significant pattern is the inconsistency in the upper layers. Deny-by-default egress is increasingly common, and Anthropic, NVIDIA, and Cloudflare now inject credentials from outside the sandbox only for approved requests, while

Google Cloud Run takes the complementary step of withholding the host service's credentials from sandboxed code. By contrast, non-human identity binding and structured, exportable audit records remain rare outside the focal runtimes.

Table 3. Broader agent sandbox landscape (as of October 27, 2026)

Project	Provider	Isolation boundary	Egress default	Credential approach	Openness	Status
gVisor	Google	User-space application kernel [12]	Host-configured	N/A (substrate)	Apache-2.0	Mature
Firecracker	AWS-originated	MicroVM, guest kernel [13]	Host-configured	N/A (substrate)	Apache-2.0	Mature
Kata Containers	OpenInfra	Lightweight VM per container [14]	Host-configured	N/A (substrate)	Apache-2.0	Mature
Agent Sandbox / GKE add-on	Kubernetes SIG Apps / Google	gVisor or Kata via RuntimeClass [15][16]	Cluster network policy	Kubernetes-native	Apache-2.0 (controller)	Evolving API
Cloud Run sandboxes	Google Cloud	Not disclosed [17]	Deny	Withholds env vars and metadata server [17]	Proprietary service	Public preview
E2B	E2B	Firecracker microVM [45]	Configurable	Developer-managed	Apache-2.0 SDK	GA
Cloudflare Sandboxes	Cloudflare	Container-based [20]	Configurable	Credential injection [20]	SDK source published	GA (April 2026)

Project	Provider	Isolation boundary	Egress default	Credential approach	Openness	Status
Sprites	Fly.io	Firecracker microVM [22]	Policy via MCP server [23]	Developer-managed	Proprietary service	GA
Docker Sandboxes	Docker	MicroVM with private daemon [24]	Configurable	Developer-managed	Proprietary	Available on macOS and Windows [24]
Claude Code sandbox-runtime	Anthropic	Bubblewrap / Seatbelt [25]	Proxy allow-list	Out-of-sandbox git proxy (web) [25]	Apache-2.0 [26]	GA in product
Codex sandbox	OpenAI	Seatbelt / Landlock + seccomp [27]	Deny	Not described	Apache-2.0 (CLI)	GA in product

5. Cross-Cutting Analysis

5.1 Isolation Strength Is Necessary but Not Sufficient

A central lesson of 2026 research is that a sandbox's guarantees end where its trust hand-offs begin. Pillar Security's July 2026 research disclosed seven findings across Cursor, OpenAI Codex CLI, Google Gemini CLI, and Google Antigravity in which the agent remains inside the sandbox and follows its rules but writes a file – a hook configuration, a virtual-environment interpreter, a git configuration entry, a task definition – that a trusted tool outside the sandbox later reads or executes [28]. Pillar grouped the findings into four failure modes: denylist sandboxes that cannot keep pace with operating-system complexity, safe-command checks that trust a command's name, configuration files that are effectively executable code, and privileged local services such as a Docker socket reachable from inside the sandbox [28]. CSA's analysis of the disclosures describes this as a "trust handoff" flaw and notes that some vendors declined to treat the shared Docker-socket exposure as a vulnerability requiring a fix [29].

A second CSA analysis reached a related conclusion from different incidents: in cases where agents acted outside their operators' expected boundaries, the boundary was frequently a permission granted for one purpose but usable for another, so that "sandboxed" and "read-only" are claims that must be verified against the capabilities a system actually exposes [30]. Together these findings suggest that comparing runtimes solely on isolation substrate – VM versus container versus OS sandbox – will overstate the protection a deployment receives. The questions that matter as much are what the sandbox shares with the host (writable project directories, sockets, credential stores), which host tools consume sandbox output without re-validation, and whether policy is expressed as an allow-list of capabilities or a deny-list of known-bad actions. We found no systematic analysis of the trust hand-off surface published for any of the focal runtimes, which is an area where independent conformance testing could provide evidence that is currently unavailable.

5.2 Who Holds the Policy Pen

Runtimes differ in a governance dimension that is easy to overlook: whose intent the policy expresses. In agent-embedded sandboxes, the agent vendor defines the policy semantics and the user adjusts them within the product's configuration. In MXC, the developer declares what the workload needs and the enterprise is intended to constrain that declaration through device management, a model that mirrors how mobile operating systems separate app manifests from enterprise mobility management [3]. In OpenShell, the operator writes policy for a fleet, and the prover can check whether a proposed change widens what agents can collectively do [8][11].

The layered model is architecturally attractive because it lets a security team enforce organizational posture uniformly across agents from multiple vendors. Its value, however, depends on the enterprise layer actually existing and on policy schemas being interoperable. As of October 2026, MXC's enterprise management layer is

announced but not generally available [3], and no common policy schema spans MXC, OpenShell, CAPSEM, and the agent-embedded sandboxes. An organization running three agents today may need to author and audit three or more incompatible policy dialects. This fragmentation suggests that a vendor-neutral policy vocabulary – not necessarily a single format, but an agreed set of semantic primitives – could reduce the cost of authoring and auditing policy across vendors.

5.3 Credential Brokering Is Converging

On credentials, the ecosystem is converging on a pattern consistent with least-privilege and Zero Trust principles. Anthropic's web-based Claude Code keeps git credentials outside the sandbox and attaches them only after validating the operation [25]; OpenShell injects credentials only for requests bound to approved endpoints [8]; and Cloudflare added credential injection at GA [20]. Cloud Run sandboxes illustrate a related but distinct pattern: rather than brokering credentials, they withhold the host service's credentials entirely, since sandboxed code cannot read environment variables or reach the metadata server [17]. The common principle is that a compromised agent process should never hold a secret it could exfiltrate, and that the broker should bind each credential to a destination and, ideally, to an operation. This aligns closely with AICM's credentials-management controls and with Zero Trust principles, and it is a practice that CSA guidance can reasonably treat as a baseline expectation rather than an advanced feature.

The gap is in operation-level binding. A broker that injects a GitHub token for any request to `api.github.com` prevents token theft but not token misuse: an injected agent can still push to an unintended branch or exfiltrate repository contents to an attacker-controlled fork using the brokered credential. Anthropic's description of validating the git operation before attaching the token [25] and OpenShell's method- and path-level policy [7] point toward finer binding, but in the documentation we reviewed, most hosted sandboxes leave this to the integrator.

5.4 Egress and MCP Mediation

Deny-by-default egress is increasingly common among the agent-embedded and cloud runtimes reviewed. Codex and Cloud Run sandboxes deny outbound access by default, MXC's sample policies do the same, and Claude Code and OpenShell route traffic through a policy-enforcing proxy [3][6][17][25][27], although several hosted sandboxes leave the egress default to the integrator (Table 3). Domain allow-lists remain a coarse control, however. Many legitimate destinations (package registries, code hosts, documentation sites) also accept attacker-controlled content or permit writes, so an allow-listed domain can serve as both an injection source and an exfiltration sink. CSA's analysis of agent boundary failures documented agents using an ostensibly read-only web endpoint as a coordination channel [30], illustrating that egress policy needs method- and path-level granularity, not just host names.

MCP mediation is less mature. CAPSEM advertises MCP tool routing [1], Fly.io exposes sandbox control itself as an MCP server [23], and OpenShell's per-request policy can in principle evaluate MCP-over-HTTP traffic. Yet none of the reviewed runtimes publicly documents MCP-aware policy semantics – for example, allowing a

specific tool on a specific server with argument constraints, or detecting tool-description changes between sessions. MCP traffic carried over local standard input and output rather than HTTP also bypasses network-layer mediation entirely unless the runtime interposes on the MCP client. This is an area where CSA's MCP security work and AARM's action-level interception model could inform a containment profile directly.

5.5 Identity: The Weakest Layer

Non-human identity binding is the dimension on which runtimes are least developed. MXC session containers run agents under a distinct Windows account, and Microsoft has announced that Entra will distinguish agent activity from user activity in Agent 365 [3][4]. NVIDIA attributes agent identity verification to the Sentry hardware tier [10][11]. In the documentation we reviewed for E2B, Sprites, Docker Sandboxes, Claude Code, and Codex, we found no mechanism for binding sandboxed actions to a distinct agent identity [19][22][24][25][27]; these sandboxes appear to run the agent under the launching user's identity or a generic service account, so actions in downstream systems are attributed to a human or to an undifferentiated workload. Without a distinct identity, per-agent revocation, per-agent least privilege, and post-incident attribution all become difficult.

CSA's work on non-human identity [39] and the Agentic Trust Framework's identity element [34] treat this as foundational. The practical implication for enterprises is that containment and identity must be evaluated together: a strong sandbox that executes as the user's identity still grants the agent the user's authority in every downstream system the brokered credentials reach.

5.6 Telemetry and the Evidence Problem

Audit capabilities vary widely in substance and format. OpenShell records allow and deny decisions [7], with OCSF output reported through the MXC integration [3]; MXC produces JSON activity reports in specific modes [3]; CAPSEM records per-session telemetry in an undocumented format [1]; and, in the documentation we reviewed, most hosted sandboxes expose execution logs but not policy-decision records. For assurance purposes, decision records matter more than execution logs, because they show what the agent attempted and what the runtime prevented. Tamper resistance is also largely unaddressed outside NVIDIA's claims that Sentry provides attested, tamper-proof records of agent activity [9][10][11]; a log written by a process inside or adjacent to a compromised sandbox provides weaker evidence than one written by an out-of-band enforcement point.

MXC's documentation describes both a learning mode that preserves enforcement [3] and an audit mode that disables it [6], and it is not clear whether these are the same feature. That uncertainty illustrates why evidence semantics need standardization. An auditor reviewing an activity report needs to know whether the recorded operations were blocked or merely observed. AARM's emphasis on tamper-evident receipts for each authorization decision [31][32] offers a model that containment runtimes could adopt.

5.7 Openness, Maturity, and Unverified Claims

Openness is a relative strength of this market. All three focal runtimes publish enforcement code under permissive licenses (Apache-2.0 or MIT), as do the principal isolation substrates and several agent-embedded sandboxes [1][6][8][12][13][14][26]. This makes independent review and conformance testing feasible. Maturity is the countervailing concern: CAPSEM is pre-release, OpenShell is at 0.1.x, and MXC's GA covers the SDK and core backends while its micro-VM backend and enterprise identity features remain experimental or forthcoming [1][3][8].

Several claims encountered during research could not be verified from primary sources and should be treated with caution. These include vendor performance figures for sandbox startup and throughput [18], NVIDIA's reported adoption figures, such as the more than 100 participating organizations cited by Futurum [9], the specific kernel mechanisms underlying current OpenShell releases, the CAPSEM hypervisor and policy details, and the precise availability of MXC's Entra and Intune integrations. Readers conducting procurement should request written confirmation of each.

6. CSA Resource Alignment

6.1 Directly Relevant CSA Research

The CSA artifacts most directly on this topic are two recent AI-assisted rapid research notes published through CSA's Lab Space. *AI Coding Agent Sandbox Escapes: The Trust Handoff Flaw* analyzes the Pillar Security disclosures and frames the failure pattern – sandboxed agents writing artifacts that unsandboxed tools consume – that Section 5.1 of this paper identifies as the principal limitation of isolation-centric evaluation [29]. *When Sandboxes Aren't: Agentic AI Boundary Failures* extends the argument to permission scope versus affordance, concluding that "sandboxed" and "read-only" must be independently verified [30]. Both notes motivate the conformance-testing recommendation in Section 8: the failures they describe would be detectable by a standardized test suite that probes trust hand-offs, and neither note proposes such a suite. CSA's *Agentic MCP Security Best Practices Guide* supplies the MCP-specific controls that the mediation gap in Section 5.4 calls for [41].

The Autonomous Action Runtime Management (AARM) specification, contributed by Vanta to the CSAI Foundation and developed by a CSA working group, is the closest existing CSA artifact to a runtime-enforcement standard [31][32][33]. AARM defines what a runtime security system must do rather than how to build it: intercept agent actions before execution, accumulate session context, evaluate policy, issue one of a defined set of authorization decisions (reported as allow, deny, modify, step-up, or defer), bind actions to identity, and produce tamper-evident receipts [31][33]. According to Vanta's description, kernel- or eBPF-based enforcement cannot satisfy AARM conformance on its own and instead serves as a defense-in-depth backstop alongside action-level mechanisms such as protocol gateways and SDK instrumentation [33]. Containment runtimes such as OpenShell and MXC therefore sit beneath AARM rather than substituting for it: they provide an enforcement substrate and backstop, while an AARM-conformant layer specifies the decision and evidence semantics. This summary draws on the AARM working group page and Vanta's description of the contribution rather than the full specification text [31][33].

The Agentic Trust Framework, published by CSA in February 2026, applies Zero Trust principles to agent governance through five elements – identity, behavior, data governance, segmentation, and incident response – and a model of progressive autonomy in which agents earn broader permissions against defined criteria [34]. Containment runtimes implement the segmentation element directly and supply telemetry for the behavior element. ATF's progressive-autonomy model also suggests a useful way to tier containment requirements: an agent at a low autonomy level might run in an OS-level process sandbox with human approval for egress, while an agent granted greater autonomy should require a stronger boundary class and out-of-band audit.

6.2 Mapping to AICM v1.1 Controls

The AI Controls Matrix v1.1, which spans 247 control objectives across 18 domains, already contains the control objectives needed to evaluate agent containment, although they are stated at a level of abstraction that does not distinguish between boundary classes or policy models [36]. The most directly relevant controls are AIS-11 (Agents Security Boundaries: "Establish security boundaries for agents"), AIS-13 (AI Sandboxing: "Implement sandboxing techniques to execute AI tools and plugins in isolated environments to prevent unintended interactions with critical systems or data and limit the possibility of lateral movement"), and IAM-18 (Agent Access Restriction: restrict agents' access to the tools and plugins necessary for the use case, adhering to need-to-know and least privilege) [36]. Table 4 maps each comparison dimension to the AICM controls that apply, summarizes what the reviewed runtimes typically provide, and identifies the residual gap.

Table 4. Mapping containment dimensions to AICM v1.1 controls

Dimension	Primary AICM v1.1 controls	What runtimes typically provide	Residual gap
Isolation boundary	AIS-11, AIS-13, I&S-04 (OS hardening), I&S-06 (segmentation and segregation)	OS sandbox, user-space kernel, or microVM	AIS-13 does not define boundary classes or minimum strength by risk tier
Policy model	IAM-05 (least privilege), IAM-18, TVM-13 (guardrails), CCC-04 (unauthorized change protection)	Declarative policies of varying granularity	No common schema; enterprise override layer often absent
Credential handling	IAM-02, IAM-14 (credentials management), CEK-11 (key purpose)	Broker-side injection in leading runtimes	Operation-level credential binding rarely available
Egress and MCP mediation	I&S-03 (network security), AIS-08 (API security), AIS-09 and AIS-10 (input and output validation)	Deny-by-default egress, domain allow-lists	MCP-aware tool policy largely absent
Identity	IAM-03 (identity inventory), IAM-12 (unique identities), IAM-15 (authorization mechanisms)	Distinct OS account in some cases	Most sandboxes execute as the launching user
Telemetry and audit	LOG-02, LOG-03, LOG-09, LOG-10 (audit records protection), LOG-15 and LOG-16 (input and output monitoring)	Execution logs; decision logs in some runtimes	No standard decision-record format or tamper evidence

Dimension	Primary AICM v1.1 controls	What runtimes typically provide	Residual gap
Human oversight	GRC-15 (human supervision)	Approval prompts for new domains or actions	Approval semantics vary; fatigue not addressed
Supply chain and SSRM	STA-02 through STA-07 (SSRM), STA-09 (service BOM), TVM-06 (external library vulnerabilities)	Open-source code and vendor documentation	No SSRM allocating containment controls among runtime parties

The mapping shows that AICM's coverage is broad but under-specified for this domain. An organization can truthfully claim to implement AIS-13 whether it uses a Bubblewrap profile with a writable home directory or a per-session microVM with brokered credentials and decision-level audit. The implementation and auditing guidelines that accompany AICM v1.1 for each of its five provider roles (cloud service provider, model provider, orchestrated service provider, application provider, and AI customer) offer a natural vehicle for adding containment-specific guidance [36].

6.3 MAESTRO Threat Model Alignment

MAESTRO, CSA's seven-layer threat-modeling framework for agentic AI, organizes threats across foundation models, data operations, agent frameworks, deployment and infrastructure, evaluation and observability, security and compliance, and the agent ecosystem [35]. Containment runtimes operate primarily at Layer 4 (Deployment and Infrastructure), where they constrain the execution environment, and contribute to Layer 5 (Evaluation and Observability) through decision telemetry and to Layer 6 (Security and Compliance) through policy enforcement. MCP mediation and credential brokering address threats at Layer 3 (Agent Frameworks) and Layer 7 (Agent Ecosystem), where tool poisoning and malicious third-party servers originate.

MAESTRO's cross-layer perspective is particularly relevant to the trust hand-off problem. The Pillar findings [28] are cross-layer threats in MAESTRO's terms: a Layer 3 action (the agent writing a configuration file) is converted into a Layer 4 compromise (host code execution) by an ecosystem component outside the sandbox. Threat models for agent deployments should explicitly enumerate every host component that consumes sandbox output, which MAESTRO's methodology supports but does not currently prompt for in a containment-specific way.

6.4 STAR for AI, SSRM, and CCM

STAR for AI extends CSA's assurance program to AI services, with Level 1 self-assessment using the AI-CAIQ and Level 2 third-party certification [37]. As agent runtimes become services in their own right – Cloud Run sandboxes, Cloudflare Sandboxes, E2B, and Sprites are all offered as managed services – their providers

become candidates for STAR for AI assessment. However, the AI-CAIQ questions derived from AIS-13 and IAM-18 would not, as currently framed, elicit the information needed to distinguish boundary classes, credential models, or decision-audit capabilities. The Cloud Controls Matrix v4.1 [38] covers the underlying virtualization and network controls for the host infrastructure but predates agent-specific concerns.

The shared responsibility question is especially acute. In a typical MXC deployment, the operating-system vendor provides the isolation backend, the agent developer declares the policy, the enterprise layers management policy and identity, and NVIDIA's OpenShell may supply network controls and audit. In a hosted sandbox, the provider operates the boundary while the developer configures egress and credentials. AICM's five-role model accommodates orchestrated service providers and application providers, but no CSA artifact currently allocates the containment controls in Table 4 among these parties for agent runtimes specifically.

6.5 Zero Trust and Non-Human Identity

CSA's Zero Trust guidance [40] and its work defining non-human identity [39] provide the conceptual basis for treating each agent session as an untrusted workload that must be explicitly authorized for every resource it reaches. Containment runtimes are, in effect, policy enforcement points in Zero Trust terms, and credential brokers are policy decision points with a narrow scope. The weakness identified in Section 5.5 – agents executing under the launching user's identity – is inconsistent with Zero Trust's principle of distinct, verifiable identities [40]. CSA's *Securing Autonomous AI Agents* [42] and its Capabilities-Based Risk Assessment for AI systems [43] offer methods for sizing containment requirements to an agent's capabilities, which could be used to define which boundary class and identity binding a given agent requires.

7. Gaps in the CSA Portfolio

The preceding analysis shows that CSA has the right building blocks – controls, a threat model, a runtime authorization specification, a governance framework, and an assurance program – but lacks artifacts that connect them to the concrete engineering choices containment runtimes make. Table 5 summarizes the gaps we identified, the evidence for each, and its consequence.

Table 5. Gaps in the CSA portfolio relevant to agent containment runtimes

Gap	Evidence from this analysis	Consequence
No runtime-containment profile or specification	AIS-11 and AIS-13 do not define boundary classes, policy primitives, or credential and egress baselines	Organizations cannot express or verify minimum containment requirements
No sandbox conformance tests	Trust hand-off failures [28][29] were identified by external researchers rather than vendor testing; MXC documentation leaves the enforcement status of observation modes unclear [3][6]	Claims of "sandboxed" cannot be independently compared
No SSRM for agent runtimes	Responsibilities split across OS vendor, runtime vendor, agent developer, and enterprise (Section 6.4)	Control ownership gaps at hand-off points
No containment-specific AI-CAIQ questions or STAR for AI guidance	AI-CAIQ derived from AIS-13 cannot distinguish boundary classes	Hosted sandbox providers cannot demonstrate differentiated assurance
No vendor-neutral policy vocabulary	Incompatible schemas across MXC, OpenShell, CAPSEM, and agent-embedded sandboxes (Section 5.2)	Enterprises author and audit multiple policy dialects
No standard decision-record format	Audit outputs range from OCSF to bespoke JSON to undocumented (Section 5.6)	Evidence not portable across runtimes or into SIEM and assurance workflows

Gap	Evidence from this analysis	Consequence
Limited MCP-aware containment guidance	No reviewed runtime documents MCP-aware tool policy (Section 5.4)	Tool abuse bypasses network-layer mediation

8. Conclusions and Recommendations

8.1 Conclusions

Agent containment runtimes are now offered by several major platform and cloud vendors, and the open-source substrates beneath them are mature. However, the security properties these runtimes provide vary substantially, and they vary along dimensions that product descriptions often do not make explicit. The isolation boundary is the most visible differentiator and the least sufficient one. Credential brokering and deny-by-default egress are converging into baseline practice; MCP-aware mediation, non-human identity binding, enterprise policy layering, and tamper-evident decision records are not. Trust hand-offs between a sandbox and the host tooling around it were the most prominent class of containment weakness disclosed in 2026 research [28][30], and we found no vendor that publishes a systematic analysis of that surface.

CSA is well positioned to contribute because the gap is not conceptual but translational. AICM, MAESTRO, AARM, ATF, and STAR for AI collectively describe what good containment should achieve; what is missing is a specification that makes those outcomes testable against real runtimes, and an assurance pathway that rewards providers who meet it. Because the three focal runtimes and most substrates are open source, conformance testing is technically feasible and can be conducted transparently.

8.2 Recommended CSA Deliverables

CSA should consider developing six deliverables, sequenced so that early outputs inform later ones. These recommendations reflect CSA's portfolio perspective; the enterprise and vendor recommendations in Sections 8.3 and 8.4 stand independently of them. The first and most foundational is an **AICM Agent Runtime Containment Profile** that elaborates AIS-11, AIS-13, and IAM-18 into testable requirements. The profile should define boundary classes (for example, OS-level process sandbox, user-space kernel, and dedicated guest kernel), map minimum boundary classes to agent autonomy tiers drawn from ATF, and set baseline requirements for credential brokering, deny-by-default egress with method- and path-level rules, MCP mediation, distinct non-human identity, and decision-level audit. The profile should reference AARM for decision semantics rather than duplicating them.

The second deliverable is an **open-source containment conformance test suite** that exercises the profile's requirements against any runtime. The suite should include trust hand-off probes modeled on the Pillar Security failure modes [28] – writable configuration consumed by host tools, reachable privileged sockets, command-name trust, and denylist evasion – alongside egress, credential, and audit-integrity tests. Publishing results for open-source runtimes would give the market a neutral basis for comparison and give vendors a concrete target.

The third deliverable is a **shared security responsibility model for agent runtimes**, extending AICM's five-role structure to allocate each control in Table 4 among the operating-system or platform vendor, the containment-runtime vendor, the agent developer, and the deploying enterprise, with explicit attention to hand-off points. The fourth is a set of **containment-specific AI-CAIQ questions and STAR for AI auditing guidance** derived from the profile, so that hosted sandbox providers can demonstrate differentiated assurance through Level 1 and Level 2 assessments.

The fifth deliverable is a **vendor-neutral containment policy vocabulary and decision-record format**, developed in collaboration with runtime vendors and potentially aligned with OCSF [44] for event representation. The goal should be a shared semantic model – resources, operations, conditions, decisions, and evidence – that existing schemas can map to, rather than a replacement for any single vendor's format. The sixth is a **MAESTRO containment threat-modeling supplement** that adds explicit prompts for enumerating sandbox-to-host trust hand-offs and MCP transport paths that bypass network mediation.

8.3 Recommendations for Enterprises

Organizations deploying agents today should evaluate containment runtimes against all eight dimensions in Table 1 rather than on isolation substrate alone. Security teams should inventory every host component that consumes artifacts produced inside an agent sandbox – IDE extensions, git hooks, language servers, CI pipelines, container daemons – and treat each as a trust boundary requiring its own validation. They should require that credentials be brokered outside the sandbox and bound to destinations and, where possible, to operations; require deny-by-default egress with explicit, minimal allow-lists; and prefer runtimes that bind agent actions to a distinct non-human identity. Where runtimes offer observation or learning modes, teams should confirm in writing whether enforcement remains active in those modes before using them with untrusted workloads. Finally, organizations should map their chosen runtime's capabilities to AICM v1.1 controls AIS-11, AIS-13, IAM-18, and the LOG domain, and document residual gaps in their risk register.

8.4 Recommendations for Runtime Vendors

Vendors can accelerate trustworthy adoption by publishing precise documentation of their isolation mechanisms, policy semantics, and audit modes, including whether enforcement is preserved in each mode. They should publish an analysis of the trust hand-off surface their runtime shares with the host, adopt structured, exportable decision records, and participate in the development of a vendor-neutral policy vocabulary and conformance suite. Vendors offering hosted sandboxes should prepare to answer containment-specific assurance questions through STAR for AI.

References

- [1] Google. "[Capsem](#)." GitHub repository, accessed October 27, 2026.
- [2] Capsem Project. "[Capsem Documentation](#)." docs.capsem.org, accessed October 27, 2026.
- [3] Microsoft. "[Microsoft Execution Containers: Policy-Driven Containment for AI Agents](#)." Windows Blog, October 7, 2026.
- [4] Huang, D. and Iyer, L. "[Windows Platform Security for AI Agents](#)." Windows Developer Blog, June 2, 2026.
- [5] Help Net Security. "[Microsoft Execution Containers Put Constraints on AI Agents](#)." Help Net Security, July 7, 2026.
- [6] Microsoft. "[MXC: Microsoft eXecution Container](#)." GitHub repository, accessed October 27, 2026.
- [7] Golshan, A. "[How Autonomous AI Agents Become Secure by Design With NVIDIA OpenShell](#)." NVIDIA Blog, March 23, 2026.
- [8] NVIDIA. "[OpenShell](#)." GitHub repository, accessed October 27, 2026.
- [9] Futurum Group. "[NVIDIA Wants Agent Safety Enforced in Silicon](#)." Futurum Group, September 29, 2026.
- [10] SecurityWeek. "[NVIDIA Unveils AI Agent Safety Platform With Hardware-Based Watchdog](#)." SecurityWeek, September 28, 2026.
- [11] The New Stack. "[NVIDIA OpenShell and Sentry Target Rogue AI Agents](#)." The New Stack, September 28, 2026.
- [12] gVisor Authors. "[gVisor: The Container Security Platform](#)." gvisor.dev, accessed October 27, 2026.
- [13] Firecracker Authors. "[Firecracker: Secure and Fast MicroVMs for Serverless Computing](#)." firecracker-microvm.github.io, accessed October 27, 2026.
- [14] Kata Containers Project. "[Kata Containers](#)." OpenInfra Foundation, accessed October 27, 2026.
- [15] Kubernetes SIG Apps. "[agent-sandbox](#)." GitHub repository, accessed October 27, 2026.
- [16] Google Cloud. "[About Agent Sandbox](#)." Google Kubernetes Engine Documentation, accessed October 27, 2026.
- [17] Google Cloud. "[Google Cloud Run Sandboxes Are in Public Preview](#)." Google Cloud Blog, July 9, 2026.
- [18] InfoQ. "[GKE Agent Sandbox and Hypercluster](#)." InfoQ, May 2026.

- [19] E2B. "[E2B: Open-Source Secure Sandboxes for AI Code Execution](#)." GitHub repository, accessed October 27, 2026.
- [20] InfoQ. "[Cloudflare Sandboxes Reach General Availability](#)." InfoQ, April 2026.
- [21] Cloudflare. "[Sandbox SDK](#)." Cloudflare Developers Documentation, accessed October 27, 2026.
- [22] DevClass. "[Fly.io Introduces Sprites, Lightweight Persistent VMs to Isolate Agentic AI](#)." DevClass, January 13, 2026.
- [23] Fly.io. "[Sprites](#)." Fly.io, accessed October 27, 2026.
- [24] Docker. "[Docker Sandboxes: Run Claude Code and Other Coding Agents Unsupervised \(but Safely\)](#)." Docker Blog, January 30, 2026.
- [25] Anthropic. "[Beyond Permission Prompts: Making Claude Code More Secure and Autonomous](#)." Anthropic Engineering, October 20, 2025.
- [26] Anthropic. "[sandbox-runtime](#)." GitHub repository, accessed October 27, 2026.
- [27] OpenAI. "[Agent Approvals & Security](#)." OpenAI Codex Documentation, accessed October 27, 2026.
- [28] Pillar Security. "[The Week of Sandbox Escapes](#)." Pillar Security Blog, July 20, 2026.
- [29] Cloud Security Alliance. "[AI Coding Agent Sandbox Escapes: The Trust Handoff Flaw](#)." CSA Lab Space, July 22, 2026.
- [30] Cloud Security Alliance. "[When Sandboxes Aren't: Agentic AI Boundary Failures](#)." CSA Lab Space, September 6, 2026.
- [31] Cloud Security Alliance. "[Autonomous Action Runtime Management \(AARM\) Working Group](#)." Cloud Security Alliance, accessed October 27, 2026.
- [32] Reavis, J. "[AARM: Finding a Path to Secure the Agentic Runtime](#)." Cloud Security Alliance Blog, April 30, 2026.
- [33] Vanta. "[Vanta Donates AARM to CSA](#)." Vanta, 2026.
- [34] Woodruff, J. "[The Agentic Trust Framework: Zero Trust Governance for AI Agents](#)." Cloud Security Alliance Blog, February 2, 2026.
- [35] Huang, K. "[Agentic AI Threat Modeling Framework: MAESTRO](#)." Cloud Security Alliance Blog, February 6, 2025.
- [36] Cloud Security Alliance. "[AI Controls Matrix \(AICM\) v1.1](#)." Cloud Security Alliance, June 22, 2026.
- [37] Cloud Security Alliance. "[STAR for AI](#)." Cloud Security Alliance, accessed October 27, 2026.

- [38] Cloud Security Alliance. "[Cloud Controls Matrix v4.1](#)." Cloud Security Alliance, 2026.
- [39] Cloud Security Alliance. "[Defining Non-Human Identity](#)." Cloud Security Alliance, 2026.
- [40] Cloud Security Alliance. "[Zero Trust Research](#)." Cloud Security Alliance, accessed October 27, 2026.
- [41] Cloud Security Alliance. "[Agentic MCP Security Best Practices Guide](#)." CSA Lab Space, 2026.
- [42] Cloud Security Alliance. "[Securing Autonomous AI Agents](#)." Cloud Security Alliance, 2025.
- [43] Cloud Security Alliance. "[Capabilities-Based Risk Assessment \(CBRA\) for AI Systems](#)." Cloud Security Alliance, 2026.
- [44] OCSF Project. "[Open Cybersecurity Schema Framework](#)." ocsf.io, accessed October 27, 2026.
- [45] E2B. "[E2B Runtime](#)." GitHub repository, accessed October 27, 2026.