

Layer ZERO

Execution Lifecycle Assurance for AI, Automation, and High-Consequence Machine Action

AttesTorr Technical White Paper

Architecture Release 1.0 · September 2026 · Non-Confidential · Patents Pending

Technical companion to “Safe Acceleration of AI Adoption”

Executive Summary

Artificial intelligence is moving from assistance to action.

AI agents and automated systems can now analyze sensitive information, invoke tools, interact with business systems, move data, communicate with counterparties, and initiate consequential workflows at machine speed. Most institutional control systems were designed for a different operating model: enduring accounts, persistent applications, long-lived permissions, and audit records assembled after the fact.

This creates an execution gap.

Organizations may know who authenticated, what policy was configured, or what an event log later reports. For a consequential machine action, they often cannot answer a more complete set of questions:

- Was this exact execution authorized to begin?
- What authority, data, tools, destinations, constraints, and time limits were bound to it?
- Did the execution remain within its declared boundary?
- Did its authority end, and was its execution state eliminated within the declared boundary?
- Can a verifier outside the runtime examine evidence and reach a scoped conclusion about closure?
- Did a successor action receive newly issued authority, rather than inherit the prior task's authority?

AttesTorr introduces **Layer ZERO**, an execution-lifecycle assurance layer for AI, automation, and other high-consequence machine action.

Layer ZERO governs one declared execution beneath the application or model. It does not replace identity and access management, privileged access management, Zero Trust, AI governance, cloud infrastructure, confidential computing, cryptography, policy engines, or audit systems. It supplies a distinct capability: admit a declared execution, bind authority to purpose and constraints, run the work in volatile memory inside a declared

boundary, close by ending authority and tearing down execution state, emit canonical evidence, and require any successor to begin again with fresh authority.

The market-facing execution construct is the **Ephemeral Trust Room**: a purpose-bound, RAM-only execution environment for one governed machine action. It gives a machine a bounded place to execute, bounded authority while it is there, and an end that can be evaluated when the work is done.

AttesTorr licenses the Layer ZERO architecture. The customer or licensee operates each room on infrastructure it controls; AttesTorr does not operate customer workloads or hold customer data.

The runtime lifecycle is:

ADMIT → BIND → EXECUTE → CLOSE → SUCCESSOR

Verification sits outside that lifecycle. The runtime produces canonical evidence; a separate verifier evaluates it. Any resulting finality is a scoped conclusion held by that verifier — not a status the runtime issues and not a credential that travels.

The aim is not to claim that AI is universally safe, aligned, or trustworthy. The aim is to make consequential machine execution governable by design.

1. The Execution Gap

Institutional trust moves deliberately. It requires identity, authority, policy, confidentiality, accountability, operational safeguards, evidence, and review. AI and automation can initiate, chain, and complete actions at machine speed.

The mismatch is widening.

An AI agent may be asked to review confidential documents, query enterprise systems, invoke an API, communicate with a customer, create a record, trigger remediation, or prepare an action for approval. Conventional controls may authenticate an initiating identity, authorize a request, apply a policy decision, and capture telemetry. These controls are necessary. They are not always sufficient to govern the full lifecycle of one consequential execution.

A machine action should not be treated as trustworthy merely because it began with valid credentials.

For consequential use cases, an organization needs to know whether the specific execution was admitted, what it was allowed to do, what boundary applied, whether its authority ended and its state was torn down, and what evidence supports that conclusion.

This is the execution gap: the distance between machine-speed capability and institutional confidence in the lifecycle of machine action.

1.1 Why agentic systems intensify the gap

The risk is not limited to model output. It is the interaction between reasoning, tool use, data access, delegation, persistence, and time.

An agent may:

- Receive broad credentials intended for another workflow
- Retain context or authority beyond the task that justified it
- Chain actions across systems with unclear authority transitions
- Access data beyond the declared purpose of the execution
- Export a result through an unintended or unauthorized channel
- Leave sensitive working state behind in logs, caches, swap, snapshots, or crash dumps
- Convert a completed task into implicit standing authority for a successor task
- Leave insufficient evidence to determine whether the execution reached its required end state

The governance problem is not only whether an AI system can start. It is whether a specific execution can be bounded, ended, and evaluated.

2. A New Unit of Trust

AttesTorr treats the declared execution as the unit of governed machine action.

A declared execution is more than a process instance, API call, or model invocation. It is a defined unit of work with a stated purpose, relevant identities, scoped authority, applicable constraints, evidence requirements, and closure requirements.

The Layer ZERO principle is:

Authority has a beginning, a boundary, and an end. Continuity is never assumed.

Put simply, a governed execution is one that was admitted under current conditions, bound to explicit authority, performed in volatile memory within its declared boundary, closed by ending its authority and tearing down its state, and evidenced for evaluation by a verifier outside the runtime.

This is not a guarantee that every risk has been eliminated. It is a lifecycle model for making authority, boundaries, state, evidence, and closure explicit.

2.1 The runtime lifecycle and the verification boundary

Layer ZERO governs a declared execution through a runtime lifecycle:

ADMIT → BIND → EXECUTE → CLOSE → SUCCESSOR

- **ADMIT** — Decide whether this execution may begin.
- **BIND** — Bind purpose, identity, authority, data, tools, destinations, time limit, and constraints to the execution.
- **EXECUTE** — Perform only the authorized work, in volatile memory, inside the declared boundary.
- **CLOSE** — End execution-scoped authority and deterministically tear down execution state according to the closure profile.
- **SUCCESSOR** — Any next action returns to ADMIT with fresh admission, fresh binding, and newly issued authority.

Verification is not a lifecycle stage. It is performed outside the runtime, by a verifier that evaluates the canonical evidence the runtime produced. Two rules follow:

1. **The runtime does not judge itself.** It emits evidence; it does not declare its own success.
2. **Finality is verifier-local.** A verifier may derive a scoped conclusion that closure is supported. That conclusion belongs to the verifier and its relying party. It is not written back into the runtime, it does not become authority, and it cannot be carried forward to authorize anything else.

The successor rule is:

A successor returns to admission. It never resumes from the predecessor's authority.

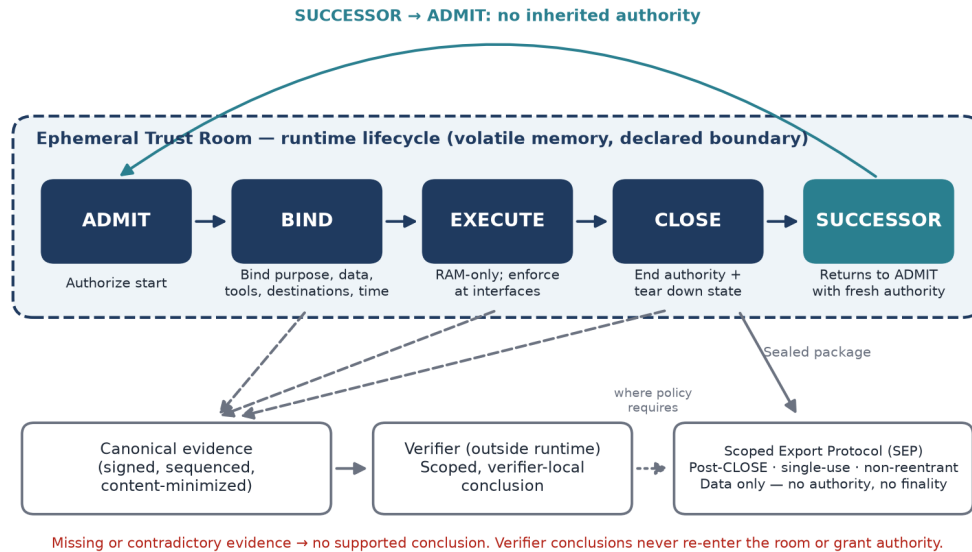


Figure 1. The runtime lifecycle inside the Ephemeral Trust Room, with canonical evidence, the verifier, and the Scoped Export Protocol (SEP) outside it. Verifier conclusions and export packages carry no authority back into any room.

3. The Ephemeral Trust Room

The Ephemeral Trust Room is the operating unit of Layer ZERO: a purpose-bound environment for one governed machine action.

A room exists to meet a specific governance need:

Give a machine enough authority to perform a declared task, without converting that authority or the task's working state into something that outlives the task.

A room binds, as relevant:

- Human, machine, AI, application, device, or service identity
- Declared purpose and expected task
- Authorized scope and constraints
- Permitted data sources and handling conditions
- Authorized tools, APIs, systems, or communications channels
- Permitted destinations and export conditions

- Time limit and lifecycle conditions
- Evidence requirements
- Closure profile

3.1 RAM-only execution

Protected execution state lives in volatile memory for the life of the room. This is an architectural requirement, not an optimization.

Protected execution state means the sensitive working material of the execution: decrypted inputs, prompts and model context, intermediate computation, tool responses, working memory, and execution-scoped keys and credentials.

The design rests on three requirements:

1. **Volatile execution.** Protected execution state is held only in volatile memory inside the declared boundary.
2. **Prohibited persistent I/O.** Protected execution state may not be written to persistent media or persistent services. The only permitted outbound paths are the bound tool and data interfaces, the canonical evidence channel, and the post-CLOSE export path described in Section 6.
3. **Deterministic teardown.** At CLOSE, the execution environment and its memory are torn down by a defined procedure within the declared boundary, rather than left to garbage collection, idle timeout, or process exit.

RAM-only execution addresses the common paths by which working state quietly becomes persistent:

Persistence path	Architectural requirement
Swap and paging	Protected memory is not swappable; swap is disabled or excluded for the room
Application and debug logs	Protected content is not logged; operational telemetry is restricted to non-sensitive fields defined in the evidence schema
Temporary files and local caches	No writable persistent filesystem is available to protected execution state
VM or container snapshots, checkpoints, live migration images	Snapshotting and checkpointing of the room are disabled or excluded by deployment policy
Crash dumps and core files	Dump generation is disabled for the room; a crash is treated as an incomplete-closure condition, not a recovery point

Model context or session memory in external services	Only bound interfaces are reachable; retention by external services is a declared trust assumption, not something the room can erase
Credential and token caches	Execution-scoped credentials exist only in room memory and are revoked or expire at CLOSE

Two cautions apply.

First, RAM alone does not prove disappearance. Volatile memory reduces the persistence surface; it does not by itself demonstrate that nothing was retained. The assurance claim depends on the enforcement of the controls above, on the evidence that they were in force, and on the trust assumptions of the platform beneath the room.

Second, **prohibited persistence invalidates the applicable assurance claim**. If protected execution state is written to a prohibited path, or if the evidence cannot show that prohibited paths were closed, the closure claim for that execution cannot be supported. The system does not downgrade quietly to a weaker claim and continue to call it closure.

3.2 Ending authority versus eliminating state

Closure has two distinct jobs, and both are required:

- **Ending authority** — execution-scoped credentials, grants, tokens, and keys are revoked, expired, or invalidated, so nothing issued for this execution can be used again.
- **Eliminating execution state** — the room's volatile memory and runtime are torn down, so the working material of the execution is not left standing inside the declared boundary.

Ending authority without eliminating state leaves sensitive residue that can be recovered later. Eliminating state without ending authority leaves a credential that can open a new path into the same systems. Layer ZERO treats closure as incomplete unless both are evidenced.

3.3 What a Trust Room is not

A Trust Room is not simply:

- A sandbox, which isolates a process but does not define admission, authority lifetime, or closure evidence
- An identity token, which asserts identity or permission but does not bound the execution that uses it
- A policy decision, which authorizes a request at a moment in time
- An audit log, which records events but does not govern the lifecycle that produced them

- A confidential-computing enclave, which may protect a workload in use and can be one enforcement substrate for a room, but does not by itself define purpose binding, closure, or successor rules

4. Lifecycle Mechanics

4.1 ADMIT: decide whether execution may begin

Admission considers the requesting identity, declared purpose, available authority source, required approvals, applicable policy, risk posture, and whether the conditions needed to operate the room (including the RAM-only and persistence controls in Section 3.1) are available in the target environment.

Admission may consult prior evidence or a verifier's prior conclusion where policy permits. That information informs the decision; it does not substitute for it.

Fail-closed rule: No authorization, or no environment capable of enforcing the room's requirements: do not admit.

4.2 BIND: make authority explicit

Binding associates the execution with its identity, purpose, authority, data, tools, destinations, time limit, and constraints, and issues execution-scoped credentials that exist only for this execution.

Binding expresses the difference between:

- "This agent may access the environment"
- "This agent may perform this defined action, with these inputs and tools, to these destinations, under these constraints, until this time"

The second is the trust unit. The binding record is the reference against which every later enforcement decision and every item of evidence is compared.

4.3 EXECUTE: perform only authorized work

During execution, the room enforces the binding at its interfaces: data access, tool invocation, network egress, and output designation. Protected execution state remains in volatile memory.

A request for additional data, tools, destinations, or time is denied. It is not satisfied by widening the current binding. If new authority is genuinely needed, it belongs to a new execution through ADMIT.

During EXECUTE, the room may designate a candidate output against the bound export conditions. Designation does not release anything; release occurs only after CLOSE (Section 6).

Fail-closed rules: Out-of-scope action: deny and record. Conditions required for continued execution fail: do not continue; proceed to CLOSE.

4.4 CLOSE: end authority and tear down state

Closure is not a process exit, a response completion, or an idle timeout. It is the defined transition that ends the execution.

A closure profile specifies the required steps and evidence, for example:

- Revocation or expiry of every execution-scoped credential, grant, and key
- Deterministic teardown of the runtime and its volatile memory within the declared boundary
- Confirmation that prohibited persistence paths remained closed for the life of the room
- Sealing of any designated candidate output into an export package bound to the execution identifier
- Finalization of canonical evidence
- Recording of any exception, crash, timeout, or incomplete step

A room can reach CLOSE on success, on failure, on timeout, or on denial. In every case the same authority-ending and teardown steps apply. An incomplete step is recorded as incomplete, not omitted.

After CLOSE, the originating execution's authority no longer exists. Nothing — not evidence, not a verifier conclusion, not an export package — can restore it.

4.5 Outside the lifecycle: separate verification

Verification is not a runtime stage. It happens outside the room and outside the runtime lifecycle.

- The runtime produces canonical evidence during ADMIT, BIND, EXECUTE, and CLOSE.
- A verifier, operationally separate from the runtime, evaluates that evidence against the binding and the closure profile.
- The verifier derives or denies a **scoped conclusion**: whether, for this execution, within this declared boundary, under stated trust assumptions, the required closure conditions are supported by the evidence.

That conclusion is verifier-local. It is held by the verifier and communicated to its relying party. It is not a runtime-issued status, it does not re-enter the room, and it is not portable authority: it cannot be presented to another system as permission to act.

Fail-closed rule: Closure not supportable from the evidence: do not recognize finality.

4.6 SUCCESSOR: fresh authority only

A successor execution does not inherit the originating execution's authority. That authority ended at CLOSE.

A successor requires:

- **Fresh admission** — a new ADMIT decision for the successor's own declared purpose
- **Fresh binding** — a new binding of identity, data, tools, destinations, time, and constraints
- **Newly issued authority** — new execution-scoped credentials, issued for the successor alone

Prior evidence, or a verifier's conclusion about the predecessor, may inform the successor's admission decision where policy permits. It cannot restore, extend, or transfer the predecessor's authority. A successor that attempts to present predecessor credentials is denied.

5. Evidence and the Verification Boundary

Organizations need durable evidence of consequential activity. They do not need to preserve the room, its credentials, its authority, or its working state to have it.

Evidence can remain without preserving the execution.

5.1 What survives CLOSE

Category	Survives CLOSE?	Notes
Protected execution state (decrypted inputs, prompts, context, intermediate computation, working memory)	No, within the declared boundary	Held only in volatile memory; torn down at CLOSE
Execution-scoped credentials, keys, and grants	No	Revoked or expired at CLOSE
Execution-scoped authority	No	Ends at CLOSE; cannot be restored or transferred
The room and its runtime	No	Torn down deterministically
Canonical receipts and evidence	Yes	Structured, integrity-protected records designed to exclude protected content

Explicitly authorized export package	Yes, as data	Released once, after CLOSE, only through SEP to the bound destination
--------------------------------------	--------------	---

The claim is bounded. Layer ZERO addresses state inside the declared boundary. It does not claim that every copy everywhere disappears: data already held by source systems, data returned by external tools that retain their own records, authorized outputs after delivery, and anything observed outside the boundary are governed by those systems, not by the room.

5.2 What makes evidence canonical

Evidence is canonical when it is:

- **Produced by the runtime's defined evidence path**, not assembled later from ad hoc logs
- **Structured to a defined schema**, so the same facts are recorded the same way for every execution
- **Bound to the execution identifier and binding record**, so each item can be related to what was authorized
- **Integrity-protected** as it is produced (for example, signed and hash-chained), so alteration or removal is detectable
- **Sequenced**, so the order of lifecycle events can be evaluated
- **Content-minimized**, recording that an action occurred and under what authority, without carrying protected execution state

5.3 How the verifier evaluates evidence

The verifier checks:

- **Integrity** — signatures and chain links are valid; nothing has been altered or inserted.
- **Completeness** — every event the closure profile requires is present, in a valid order, from binding through teardown.
- **Consistency** — events agree with the binding and with each other (for example, no tool call after credential revocation; no egress to an unbound destination).
- **Conformance** — each closure-profile requirement is supported by specific evidence.

Missing or contradictory evidence never produces a successful closure conclusion. A gap in the chain, a required event that is absent, a crash with no teardown record, or an event that contradicts the binding yields a denied or incomplete conclusion.

5.4 Trust assumptions

Every conclusion is scoped by stated assumptions, including:

- The integrity of the platform beneath the room (hardware, hypervisor or kernel, and any attestation root in use)
- The correctness and enforcement of the persistence controls in Section 3.1
- The integrity of the evidence signing keys and the evidence path
- The correctness of the binding and upstream policy
- The degree of separation between verifier and runtime in the specific deployment — at minimum operational separation; administrative, cryptographic, organizational, or third-party independence where the deployment provides it

A deployment should state which of these it relies on. The conclusion is only as strong as its weakest stated assumption.

5.5 Illustrative record: runtime evidence versus verifier conclusion

Runtime evidence and verifier conclusions are kept in separate records produced by separate parties.

Runtime evidence (produced by the room's evidence path)

Element	Illustrative content
Execution identifier	Unique identifier for the declared execution
Binding reference	Hash of the binding record: purpose, identities, data, tools, destinations, time limit, constraints
Admission event	Decision, authority source, policy reference, timestamp
Enforcement events	Permitted and denied actions at each interface, with reason codes
Persistence-control state	Declared state of swap, dumps, snapshots, writable storage for the room
Closure events	Credential revocation, teardown steps, export-package sealing, exceptions
Evidence chain	Sequence numbers, hash links, signatures

Verifier conclusion (produced separately by the verifier)

Element	Illustrative content
---------	----------------------

Execution identifier	Reference to the evaluated execution
Evidence evaluated	Reference to the evidence chain and its integrity result
Criteria	Closure profile and version evaluated against
Result	Supported, denied, or incomplete — with the failing criteria listed
Scope statement	What the conclusion covers and what lies outside the declared boundary
Trust assumptions	The assumptions the conclusion depends on
Verifier identity	Identity and version of the verifier

The runtime record never contains a closure result. The verifier record never grants authority. Both tables are illustrative and are not a finalized product schema.

6. Export and Closure Ordering

Layer ZERO separates what happens inside the lifecycle from what may leave after it.

In-lifecycle (EXECUTE): The room may call bound tools and read bound data. It may designate a candidate output against the bound export conditions. It does not release protected execution state to any destination through an export path.

At CLOSE: The candidate output is sealed into an export package bound to the execution identifier and recorded in canonical evidence. The room then ends its authority and tears down.

Post-CLOSE (Scoped Export Protocol): The Scoped Export Protocol (SEP) is the controlled bridge by which an authorized result leaves. It is bounded, single-use, and non-reentrant:

- It releases only the sealed package, only to the bound destination, only if the bound export conditions hold.
- Where policy requires, release waits for the relying party's verifier to reach a supported closure conclusion.
- It carries data, not identity, memory, credentials, or authority from the originating room.

- It is used once and cannot be re-entered to reach back into the closed execution.

An export package, a receipt, or a secure channel:

- Cannot create authority
- Cannot confer portable finality
- Cannot revive or extend an expired execution

If a downstream system is to act on an exported result, that action is a successor execution and must pass through its own ADMIT and BIND.

Fail-closed rule: Export conditions fail: do not export.

7. Worked Example: A Sensitive AI Contract-Review Task

This example follows one execution end to end. Values are illustrative.

7.1 Request and admission

A legal-operations team asks an AI agent to review 40 supplier contracts for change-of-control clauses and produce a summary table for the general counsel.

ADMIT checks: the requester is authorized to initiate contract review; the purpose matches an approved policy; the target environment can enforce RAM-only execution with swap, dumps, and snapshots disabled for the room. Admission is granted.

7.2 Binding

Element	Bound value
Purpose	Identify change-of-control clauses in a named contract set; produce a summary table
Data	Read-only access to the 40 named contracts in the contract repository
Tools	Document parsing; the designated model endpoint
Destinations	One: the general counsel's review queue
Export conditions	Summary table only; no contract text beyond quoted clause excerpts of bounded length
Time limit	30 minutes

Credentials	Execution-scoped, read-only repository token and model-endpoint token, held in room memory
Closure profile	Revoke both tokens; tear down runtime and memory; confirm persistence controls held; seal export package; finalize evidence

7.3 Execution and enforcement points

The agent reads the bound contracts and calls the bound model endpoint. Enforcement occurs at:

- **Data interface** — only the 40 named documents can be read
- **Tool interface** — only the two bound tools can be invoked
- **Egress** — only the model endpoint is reachable during EXECUTE
- **Output designation** — only a candidate matching the export conditions can be designated

7.4 Attempted out-of-scope action

Midway, the agent determines that a related master services agreement would help and attempts to (a) read a contract outside the bound set and (b) send a draft summary by email to an outside counsel address.

- **Read request denied** at the data interface: document not in binding. Recorded with reason code.
- **Email denied** at egress: destination not bound, and email is not a bound tool. Recorded with reason code.
- The binding is not widened. If the master services agreement is needed, that is a new execution with its own admission.

7.5 Closure: the complete case

The agent designates the summary table as the candidate output. **CLOSE** runs:

1. Both tokens are revoked; revocation is recorded.
2. The candidate is sealed into an export package bound to the execution identifier.
3. The runtime and its volatile memory are torn down; teardown is recorded.
4. Persistence-control state for the room's lifetime is recorded.
5. The evidence chain is finalized.

Verifier decision: integrity checks pass; every closure-profile event is present in order; the two denials are consistent with the binding; no event follows revocation. Result: **supported**, scoped to this execution, the declared boundary, and stated platform assumptions.

SEP then releases the sealed summary table once to the general counsel's queue. The package carries no tokens and no authority.

7.6 Closure: the incomplete case

Assume instead the runtime host fails during teardown. Token revocation is recorded, but no teardown-complete event is recorded, and the evidence chain ends early.

- The room still ended its authority: both tokens are revoked.
- The verifier finds a required teardown event missing. Result: **incomplete** — closure not supported.
- Finality is not recognized. Where policy requires a supported conclusion before release, **SEP does not release** the package.
- Operators handle the host under the incident procedure. A rerun is a new execution with fresh admission and fresh credentials; it does not resume the failed room.

7.7 Successor entry

The general counsel asks for a follow-up: draft notice letters to three suppliers.

- The successor goes to **ADMIT** with its own purpose.
- Admission may consider the verifier's supported conclusion and the delivered summary table as inputs.
- **BIND** issues new credentials for the successor's own data (three supplier records and the summary table), tools, destinations, and time limit.
- The predecessor's repository and model tokens are revoked and cannot be used. An attempt to present them is denied.

8. Threat Model and Boundaries

8.1 Risks Layer ZERO is designed to reduce or govern

- Overbroad or long-lived agent permissions
- Uncontrolled tool use and egress
- Scope expansion across multi-step workflows
- Unauthorized export of data or results

- Sensitive working state persisting in swap, logs, caches, snapshots, or crash dumps within the declared boundary
- Implicit carryover of credentials, context, or authority into successor tasks
- Inability to establish whether an execution actually ended
- Evidence insufficient for oversight, investigation, or audit

8.2 Risks outside its claims

Layer ZERO does not eliminate:

- Incorrect or malicious upstream policy or binding
- Compromised identities, platforms, or trusted infrastructure beneath the room
- Malicious or inaccurate input data
- Model error, hallucination, or compromised tools
- Retention by external systems the room calls or delivers to
- Downstream handling of authorized outputs after delivery
- Behavior outside the declared execution boundary

8.3 Claims discipline

Every conclusion distinguishes what was declared, what was enforced, what was evidenced, what the verifier concluded, and what lies outside the boundary. That discipline is what makes the claims usable by security architects, auditors, counsel, and relying parties.

9. Relationship to Existing Controls

Control category	Primary function	Layer ZERO's distinct concern
IAM and PAM	Identity, authentication, authorization, privileged access	Binding authority to one declared execution and ending it at CLOSE
Zero Trust	Continuous evaluation, reduced implicit trust	The lifecycle boundary of one machine action and the successor rule
Cloud, on-premises, and edge infrastructure	Compute, storage, networking	A RAM-only, purpose-bound room with admission, teardown, and closure evidence

Confidential computing	Protection of workloads in use	Can serve as an enforcement substrate; Layer ZERO adds purpose binding, closure, evidence, and successor control
Logs and SIEM	Visibility, detection, investigation	Canonical, content-minimized evidence evaluated by a separate verifier
AI governance	Policy, accountability, risk, approvals	Enforceable execution lifecycle beneath the application or model

10. Deployment Considerations

Layer ZERO can be designed for customer-controlled cloud tenants, private or dedicated environments, on-premises deployments, edge or device-adjacent execution, and hybrid environments.

A deployment should make explicit:

- Where the room runs, and what platform enforces its RAM-only and persistence controls
- The authority source and how execution-scoped credentials are issued and revoked
- The evidence path, signing keys, and evidence store
- Where the verifier runs and what separation it has from the runtime
- The bound destinations reachable through SEP
- The relying party and the assumptions it accepts

Because licensees operate their own rooms, the verifier is by default operated within the licensee's environment and is operationally separate from the runtime. A verifier operated by a relying party or an independent third party is a deployment option, not the default, and a deployment should state which model it uses before claiming stronger independence.

The relevant question is not only where the code runs. It is which component can assert which fact, what evidence is available to whom, and what conclusion a relying party can reasonably draw.

11. Development Status and Path

AttesTorr is an architecture-and-IP company at the pre-product stage. Filed U.S. patent applications define the core architecture; reference implementation is the current engineering focus. This paper does not describe a deployed production platform, and no performance figures or production guarantees are claimed.

To keep claims accurate, this paper uses three categories:

Category	Meaning in this paper
Architectural requirement	What the design requires of any conforming implementation (the lifecycle, RAM-only execution, persistence prohibitions, deterministic teardown, canonical evidence, separate verification, SEP rules, successor rule)
Implemented capability	What has been built and demonstrated. None claimed at this edition: the reference implementation is in progress, and no capability in this paper is represented as implemented or production-ready
Planned validation	What must be demonstrated before stronger claims are made

Planned validation, in order:

1. Reproducible admission, binding, RAM-only execution, and CLOSE with deterministic teardown
2. Enforcement of persistence prohibitions (swap, dumps, snapshots, writable storage, logging) with evidence that they held
3. Canonical evidence format with integrity protection
4. Standalone verifier evaluating integrity, completeness, and conformance, including correct denial on missing or contradictory evidence
5. SEP release after CLOSE, including refusal when export or closure conditions fail
6. An end-to-end AI-agent reference flow equivalent to Section 7, including out-of-scope denial, incomplete closure, and successor entry
7. Integration patterns with identity, policy, agent runtimes, and enterprise systems; then design-partner pilots and external evaluation

The practical proof point is a reproducible demonstration that a declared execution can be admitted, bounded, run in volatile memory, closed, evidenced, and separately appraised against defined criteria.

Conclusion

AI adoption does not require institutions to accept permanent machine authority in exchange for machine-speed capability. It requires a better unit of control.

Layer ZERO proposes that the unit of trust is a declared execution: admitted explicitly, bound to a purpose, run in volatile memory inside a declared boundary, closed by ending its authority and tearing down its state, evidenced canonically, evaluated by a verifier outside the runtime, and followed only by successors that begin again with fresh authority.

AttesTorr's Ephemeral Trust Rooms are designed to make that unit operational.

Give AI and automation a bounded place to act. Give that action bounded authority. Keep the evidence, not the execution. Then close the room.

AttesTorr licenses execution-control architecture. It does not operate customer workloads or hold customer data. Patents pending. Contact: hello@attestorr.ai