CONFIDENTIAL COMPUTING CONSORTIUM

From Perimeter to Silicon:

Why Agentic AI Demands a New Security Model

A Publication of The Confidential Computing Consortium

November 2026

Lead editor: Raghu Yeluri

Abstract

AI has changed how we think about security, with concerns over how to protect the inference work that they do on sensitive data. Agentic AI systems move beyond passive inference: they plan, execute tool calls, manage credentials, and produce real-world effects. This shift introduces a class of security exposures that traditional perimeter-based controls do not address. When an agent is reasoning and acting, its sensitive state — prompts, retrieved documents, tool outputs, credentials, and model weights in case of local/small models, — resides in memory in cleartext, accessible to any party with privileged access to the host, hypervisor, or accelerator. This could be system administrators, other agents, other tenants on the same host, etc.

This paper describes the threat model for single-agent and multi-agent deployments, identifies the gap left by encryption at rest and in transit, and explains how Confidential Computing (CC) — specifically hardware-rooted Trusted Execution Environments (TEEs) with remote attestation — addresses that gap. It also maps the security domains where CC provides coverage, where coverage is partial, and where it does not apply. The paper concludes with a reference architecture and notes on active standardization work within the Confidential Computing Consortium (CCC).

1. Introduction

Agentic AI systems combine five capabilities that traditional software did not bundle together: multi-step reasoning and planning, persistent memory across sessions, external tool and API access, delegated identity and credentials, and goal-directed execution loops. This combination is qualitatively different from stateless inference. An agent does not merely return an answer; it takes actions that may be irreversible, may involve financial or personal data, and may cascade across other agents or systems. The agents work with many sensitive materials — user prompts, retrieved documents, API keys, OAuth tokens, intermediate plans, model weights.

Confidential Computing protects an agent's runtime binary, system instructions, and model parameters from unauthorized manipulation by infrastructure administrators or co-located tenants. Hardware-rooted remote attestation verifies software integrity, allowing external services and databases to safely delegate transactional authority and confidential access without requiring human-in-the-loop validation. This isolation enables cross-organizational multi-agent collaboration over shared workloads while providing protection for proprietary models and sensitive datasets. Hardware-enforced memory encryption mitigates memory dumping attacks targeting session credentials, tokens, and private payload data during execution. Hardware-backed key generation also enables agentic knowledge isolation, allowing persistent agent knowledge to remain encrypted under keys generated and controlled within the TEE across executions. Upon task completion, the hardware enclave purges volatile memory state, ensuring intermediate execution plans, KV-cache content, and operational artifacts leave no persistent trace on the host.

Confidential Computing is widely available in public clouds and in commercial silicon, with various vendors providing CPU and GPU support. It is becoming widely adopted in sectors ranging from healthcare to blockchain, finance to Adtech and pharmaceutical research to government. There is a strong open source community and software projects from low-level libraries to "drop-in" AI frameworks.

2. Threat Model for Agentic AI

Agentic threats fall into two categories. The behavioral and cognitive threats are unique to agentic AI and have no equivalent in traditional software security. The system and infrastructure threats are amplified versions of known risks — what is new is that they are now triggered autonomously rather than requiring direct user action. Each threat in the tables below is tagged by type: Behavioral, System, or Both, where a threat has elements of each.

2.1 Single-Agent Threats

When agents call external tools, every untrusted input becomes a potential instruction. The threat surface is wider than it is for a stateless model because the agent's responses cause downstream effects.

Threat	Type	Descripton
Direct prompt injection	Behavioral	Malicious instructions embedded in user input that redirect agent behavior.
Indirect prompt injection	Behavioral	Malicious content in external sources — websites, documents, emails, retrieved database records — that the agent processes and acts upon. These hidden instructions embedded in files, emails, or web content redirect agent behavior without user awareness.
Goal hijacking and memory poisoning	Both	An attacker modifies the agent's objective or long-term memory store, altering behavior across multiple sessions.
Jailbreaks via tool chains	Behavioral	Multi-step tool call sequences bypass safety alignment that sign-turn prompts cannot.
Agent binary tampering	System	The agent runtime itself is modified to behave differently than expected.

Threat	Type	Descripton
Orchestration Graph Hijack	System	An attacker manipulates the agent execution graph to reroute output to an unauthorized destination.
Persistent Memory Poisoning	Both	A malicious baseline instruction saved to a vector database is re-fetched and executed/re-executed in later, unrelated sessions.
Credential theft	System	API keys, OAuth tokens, and session credentials are extracted from the agent's memory or environment.

2.2 Multi-Agent and Distributed Threats

Distributed, multi-agent deployments introduce additional attack paths beyond those that apply to a single agent.

Multi-Agent Threats	Type	Descripton
Agent impersonation	System	A rogue or compromised agent presents itself as a legitimate peer, causing other agents to act on false instructions.
Inter-agent trust exploitation	Both	Agents in a pipeline blindly trust outputs from other agents. A single compromised agent can corrupt the entire chain. The trust boundary problem scales quadratically: n agents produce n×(n-1) pairwise trust relationships.
Orchestrator hijacking	System	The component responsible for coordinating agent workflows is manipulated, redirecting the entire pipeline.
Shared-state and memory poisoning	Both	One agent writes malicious data to a shared memory or state store that other agents read and act upon.
Privilege escalation via agent chaining	System	An agent with limited permissions delegates a task to another agent with broader permissions, bypassing access controls.

3. Confidential Computing: Mechanisms and Properties

3.1 Protection Properties

For agentic workloads, Confidential Computing provides three concrete properties:

Runtime confidentiality: agent memory is encrypted while the workload executes. Prompts, plans, tool outputs, and credentials are not visible to the host OS, hypervisor, or CSP operators.

Integrity and anti-tampering: the TEE verifies that the agent binary, MCP servers, tools, etc., and runtime are unmodified when running in a TEE. Runtime injection and modification are blocked; the blast radius of a host compromise is limited.

Verifiable trust: *Agent Trust*: the ability to verify that an agent is the expected agent, running the expected code and model, under the expected policy, using approved services and tools, within an attested execution environment.

Agent Trust is a composite property: no single control establishes it. Hardware-rooted attestation is a necessary foundation but not the complete answer, because it confirms what was true at launch and says nothing about how the agent behaves mid-session. Trust is assembled continuously from evidence:

- **Identity** — the agent has its own unique, verifiable identity, cryptographically linked to its verified execution and credentials.
- **Knowledge Isolation** — persistent memory, credentials, authentication material, and other private state remain protected under keys controlled by the verified agent across executions.
- **Evidence Freshness** — attestation and state checks that are recent, not stale artifacts from an earlier session.
- **Code Integrity** — the agent host binaries and runtime match their signed, expected measurements. Launch-time attestation is only as meaningful as the TCB it covers; Consider small footprint software stacks to minimize Trust Computing base (TCB) to operationally sustain and audit.
- **Model Integrity** — local model weights are verified against signed references and have not been tampered with.
- **Tool Integrity** — the tools and connectors the agent can invoke are approved and unmodified.
- **MCP Server Integrity** — Model Context Protocol servers the agent depends on are themselves attested and trusted.
- **Policy Binding** — the agent is provably operating under the policy set intended for it.
- **Runtime Integrity** — the execution environment remains uncompromised while the agent runs, not only at boot.
- **Human Approval and Revocation** — required approvals are authenticated and cryptographically linked to the relevant action and parameters. Revocation and expiry prevent further unauthorized actions.
- **Execution Evidence** — trustworthy records cryptographically link actions, approvals, policy versions, and outcomes to the execution instance, with defined protections against alteration, replay, and omission.

3.2 Agentic Knowledge Isolation

Confidential Computing provides execution isolation while an agent is running. Agentic systems also require isolation of the agent's persistent knowledge across its lifetime.

Long-term memory requires keys that remain under the control of the agent. Hardware-backed randomness allows a verified agent to generate its own full-entropy root key within the TEE. That root key remains inaccessible in plaintext outside the agent's hardware-isolated trust boundary and can anchor the keys used to protect its persistent memory, credentials, authentication material, and other private state.

Confidential Computing provides mechanisms that can help with this: persistence can be implemented through attestation-gated key recovery. After a restart or relocation, a verified agent can recover its persisted root key only after satisfying the expected attestation and policy checks. A verified parent or verifier can provide the wrapping key needed for recovery, while the agent's root key remains inaccessible to the host and external infrastructure. This preserves continuity without requiring the host or external infrastructure to possess the agent's root key.

This provides **agentic knowledge isolation:** the verified agent retains exclusive control over access to its persistent knowledge across executions. The same property can protect the reference material the agent uses to authenticate the entity it represents, supporting persistent identity, continuity, and accountability over time. This also provides a foundation for cryptographically linked chains of authority and delegation from an entity to its agent and across agent-to-agent interactions.

Confidential Computing can allow Agentic Systems to meet all of these requirements through its ability to provide *runtime confidentiality*, *integrity and anti-tampering* and *knowledge isolation*.

4. Security Domain Coverage

The threats in Section 2 do not fall uniformly within or outside the reach of Confidential Computing. System and infrastructure threats — credential theft, agent binary tampering, agent impersonation, and host-level hijacking of the orchestrator — target the runtime itself, and these are where the properties described in Section 3 apply most directly. Behavioral and cognitive threats — prompt injection, jailbreaks, and poisoned content — operate on the meaning of the agent's inputs, which a TEE processes faithfully whether they are benign or malicious. Many threats span both: a memory store can be tampered with at the host level, which CC prevents, or poisoned through legitimate writes, which it does not. The domains in the table below reflect this split: the domains with full or partial coverage are where system threats are addressed, while the domains with no coverage are where behavioral threats must be handled by other controls.

Confidential Computing addresses some, but not all, of the security domains relevant to agentic AI. The table below summarizes ten domains and the degree of CC coverage for each. Even partial implementation with CC provides significant benefits to an Agentic AI deployment. An incremental approach, prioritising the areas where successful compromises would most affect the system and its users, allows for defense-in-depth security to protect the most sensitive or vulnerable aspects.

Category	Coverage Level	Technical Scope & Impact	What You Must Build/Handle Separately
Infrastructure	Full Coverage	Secures the control plane, attestation, and key brokers. Also fully protects MCP servers and tool registries at the system level.	None at the hardware layer; infrastructure isolation is complete.
Identity & Trust (Agent Identity)	Partial Coverage	The TEE provides a secure, isolated runtime to generate agent identity keys and run attestation.	You must still provide mechanisms to link agent keys to identities, credentials, authority, and lifecycle or revocation state. Also, in the case of an autonomous agent, where the code that will be running in the agent is not known at build time, the attestation can ensure that the initial code is trusted, but cannot provide any trust for the agent behavior.
Model Layer (Signing & Integrity)	Partial Coverage	TEE protects model weights from theft or manipulation during active runtime.	Model signing, supply chain signature verification, and origin trust are separate software operations.
Communication & Gateways (Attested Transport)	Partial Coverage	Provides the raw cryptographic primitives to run end-to-end encrypted, TEE-attested communication channels.	You must explicitly configure, code, and integrate these attested transport protocols in your deployment.
Orchestration (Runtime Policy)	Partial Coverage	Guarantees execution integrity for the agent's planner and orchestration layers, preventing host-level tampering.	You still need to design and enforce the actual software-level runtime policies and behavioral constraints.
Observability & Governance (Audit & Compliance)	Partial Coverage	Enables tamper-evident, TEE-signed logging that cannot be altered or deleted by the host.	You must build the distributed tracing and audit logging infrastructure across your agent mesh.
Supply Chain (Build Integrity)	Partial Coverage	Build measurements and hashes feed directly into the hardware attestation report to block untrusted code.	The security, access controls, and integrity of your CI/CD build pipeline itself remain your responsibility.
Application Layer (Semantic Controls)	No Coverage	Operates strictly at the hardware and system level.	Cannot interpret the meaning of prompts, detect semantic prompt injection, or enforce behavioral safety policies.
Data Layer (Provenance & RAG Security)	Partial Coverage	Protects agent-controlled data and provenance mechanisms; does not establish the truth or semantic safety of external inputs.	CC does not by itself determine the provenance, truthfulness, or semantic safety of information used by a RAG pipeline. It can, however, protect the mechanisms that record and verify provenance, create cryptographic links among information sources, transformations, execution, and outputs, and enable a verified agent to maintain exclusive control over access to its persistent knowledge.
Human Oversight (Governance & HITL)	Partial Coverage	Indirect/supporting coverage	Human oversight, manual overrides, and corporate governance are not provided by Confidential Computing itself. They require organizational processes and software enforcement controls. CC can strengthen those controls by protecting their code, policy state, and credentials within an attested execution environment.

5. Conclusions

Agentic AI systems operate differently than stateless inference models. They hold sensitive material in memory while executing multi-step plans, calling external tools, and coordinating with other agents. The runtime is the attack surface, and existing encryption at rest and in transit do not cover it.

Hardware-based TEEs with remote attestation provide a mechanism for isolating agent workloads and cryptographically verifying their state. This enables a set of properties — runtime confidentiality, integrity, verifiable trust, and agentic knowledge isolation — that are prerequisite for deploying agentic AI in environments where compute is shared, where tenants have different trust levels, or where regulatory requirements demand demonstrable data protection.

Confidential Computing does not directly address semantic attacks, data provenance, or governance, but provides an attestable isolated environment to build such controls. A complete security architecture for agentic AI requires application-layer controls, supply chain practices, and institutional oversight in addition to hardware isolation. The CCC is developing technical blueprints, reference architectures, and open source tooling to support this broader effort, including:

- Gramine: Small TCB runtime
- Veraison: Attestation and associated standards implementations
- dstack: AI Compute provisioning
- SPDM Tools: CC-capable device secure communications

The central design principle is that trust in an agent, its associated runtime, and its dependencies should be measurable and verifiable, not assumed. With attestation evidence composable across CPU and GPU vendors, and with the use of open source tooling, it becomes practical to extend that verifiable trust from silicon through the agent reasoning loop, its persistent knowledge, and all the tools, models and services it interacts with.