CSAI Foundation

Cloud Security Alliance AI Safety Initiative

Agentic Universe: April 2026

Definitions, Taxonomy, and Security Considerations for AI Agents in Enterprise Environments

Unofficial AI-assisted Research

2026-04-01

Table of Contents

Executive Summary

The term "AI agent" has become one of the most overused and least precisely defined in enterprise technology. In April 2026, the same label is applied to customer service chatbots that follow scripted decision trees, autonomous trading systems that execute multi-million-dollar positions without human approval, retrieval-augmented research assistants that search and synthesize corporate knowledge bases, multi-agent orchestration platforms that coordinate dozens of specialized sub-agents across enterprise workflows, and embedded copilots that write code, draft emails, and schedule meetings within productivity applications. These systems share the label "agent" but differ in their architectures, autonomy levels, communication patterns, trust requirements, and — critically — their threat models. Treating them as a single category for security purposes is an error that organizations can no longer afford.

This paper provides a structured framework for understanding the diversity of AI agents operating in modern enterprises. It establishes precise definitions that distinguish agents from large language models, APIs, robotic process automation, and traditional software automation. It presents a taxonomy of eleven agent types classified along four axes: autonomy level, persistence model, tool access scope, and communication pattern. For each type, it describes the core architecture, operational model, representative platforms, and — most importantly — the distinct threat model and attack surface that security teams must address.

The central argument is that agent diversity is the fundamental security challenge. An organization that deploys a single security policy across all agent types will either over-constrain simple agents, creating friction that drives shadow deployment, or under-constrain autonomous agents, leaving critical exposure unaddressed. Effective agent security requires policies, controls, and evaluation criteria that are calibrated to the specific type of agent, its deployment context, its autonomy level, and the sensitivity of the systems and data it can access.

Most enterprises already have dozens of agents operating in their environments — many embedded within SaaS platforms or introduced through developer tools — without formal classification, identity governance, or runtime monitoring. This creates a condition where autonomous systems are acting on sensitive data and enterprise systems without being explicitly recognized as security-relevant entities. The exposure is not hypothetical; it is the current state.

To address this, the paper introduces a five-step operational model for agent security: (1) **Identify** agents using structured attributes, (2) **Classify** them across multiple dimensions using the taxonomy, (3) **Apply** type-specific control playbooks, (4) **Monitor** behavior at runtime through agent-specific telemetry, and (5) **Assure** performance continuously through evaluation and certification. This Identify-Classify-Control-

Monitor–Assure cycle provides the operational backbone for the CSA Agent Security Architecture described in Appendix C, and maps to the CSA AI Controls Matrix, MAESTRO threat modeling layers, and Zero Trust principles.

1. Introduction: The Rise of the Agentic Control Plane

Eighteen months ago, the enterprise AI conversation centered on foundation models: how to select them, how to deploy them, how to prevent them from hallucinating in customer-facing applications, and how to protect the sensitive data that flowed through their prompts and completions. The model was the unit of analysis, the unit of governance, and the unit of security concern.

That era has ended. The model is now a component — important but insufficient as a unit of governance — embedded within a larger system that includes persistent memory, tool access, planning capabilities, identity credentials, and the ability to take actions in the world. The shift from model to agent is not merely an evolution in capability; it is a change in the fundamental security paradigm. A model generates text. An agent acts. A model's failure mode is a wrong answer. An agent's failure mode is a wrong action — one that may be irreversible, financially consequential, or legally significant.

The speed of this transition has caught many organizations by surprise. Gartner projects that by the end of 2026, forty percent of enterprise applications will integrate with task-specific AI agents, up from fewer than five percent in 2025 — though vendor forecasts at this stage carry significant uncertainty [1]. The CSA Agentic Identity Survey found that forty percent of organizations already have agents operating in production, with an additional thirty-one percent running active pilots, and more than seventy percent expect to manage dozens to hundreds of deployed agents within twelve months [2]. The Model Context Protocol (MCP), which provides a standardized interface for agents to access external tools, has grown from its initial release by Anthropic in November 2024 to over 10,000 active community-built servers and 97 million monthly SDK downloads as of March 2026, with the protocol donated to the Linux Foundation's newly formed Agentic AI Foundation in December 2025 alongside founding members including AWS, Anthropic, Google, Microsoft, and OpenAI [3]. Google's Agent-to-Agent (A2A) protocol, launched in April 2025, has attracted over 50 technology partners including Salesforce, SAP, and Atlassian [4]. Visa, Mastercard, Google, Stripe, and PayPal have all launched agent payment protocols enabling AI agents to autonomously initiate financial transactions [5].

This proliferation has created what the Cloud Security Alliance terms the **agentic control plane** — the orchestration, identity, authorization, monitoring, and governance infrastructure through which autonomous AI agents are deployed, managed, and held accountable [6]. Securing this control plane is the defining enterprise security challenge of 2026, and the first step in securing it is understanding what, precisely, we are securing. That is the purpose of this paper.

2. Defining AI Agents

2.1 The Problem of Definition

The term "AI agent" suffers from definitional overload. A Google search returns definitions ranging from "any software that uses AI" to "a fully autonomous system that pursues goals independently." Vendors apply the label to products that range from rule-based chatbots to autonomous multi-step research systems. This imprecision is not merely an academic inconvenience — it creates concrete security problems. When an organization's policy states that "all AI agents must be governed under our AI security framework," the practical meaning of that policy depends entirely on what qualifies as an agent. If the definition is too broad, every AI-enhanced application in the enterprise falls under agent governance, creating unsustainable compliance overhead. If it is too narrow, autonomous systems that should be governed escape oversight because they do not match the definition.

This paper proposes three layered definitions that serve different purposes depending on the context.

2.2 Three Layered Definitions

The Functional Definition (Minimal)

An AI agent is a software system that uses a foundation model to interpret instructions, make decisions, and take actions to achieve a specified goal, with the ability to operate across multiple steps without requiring human input at each step.

This minimal definition captures the essential characteristic that distinguishes an agent from a standalone model query: multi-step autonomous operation. A system that receives a prompt, generates a single response, and returns it to a human for the next instruction is not an agent under this definition — it is a model. A system that receives a goal, decomposes it into subtasks, executes those subtasks through tool invocations, evaluates intermediate results, and adjusts its approach based on those evaluations is an agent.

The Architectural Definition (Technical)

An AI agent is a software system comprising five core components: (1) a foundation model that provides reasoning and language capabilities, (2) a memory system that maintains context across interactions, including both short-term working memory and potentially long-term persistent storage, (3) a tool interface that enables the agent to interact with external systems, APIs, databases, or other agents, (4) a planning or reasoning loop that determines which actions to take and in what sequence, and (5) an identity and authorization layer that governs what the agent is permitted to do and on whose behalf it operates.

This architectural definition enables security architects to reason about agents in terms of their attack surface. Each component introduces distinct vulnerabilities: the model is susceptible to prompt injection, the memory to poisoning, the tool interface to abuse and over-permissioning, the planning loop to goal hijacking, and the identity layer to credential theft and privilege escalation.

The Operational Definition (Governance)

An AI agent is a goal-directed, context-aware, tool-using entity that operates with a degree of autonomy on behalf of a human or organizational principal, maintaining state across interactions and capable of taking actions that have consequences beyond generating text.

This operational definition is designed for governance and risk management contexts. It emphasizes three properties that determine governance requirements: the agent acts on behalf of a principal (creating a delegation relationship with accountability implications), it maintains state (creating persistence and memory security requirements), and it takes consequential actions (creating a risk surface that extends beyond information exposure to include operational, financial, and legal consequences).

2.3 What an Agent Is Not

Precise definition requires not only specifying what qualifies but also what does not. The following distinctions are essential for organizations developing agent governance policies.

An agent is not a large language model. An LLM is a statistical model that generates text based on input prompts. It has no memory between calls, no tool access, no planning capability, and no identity. An LLM becomes a component of an agent when it is embedded in an architecture that adds these capabilities. The security posture of a standalone LLM API call is fundamentally different from the security posture of an agent that uses that same LLM within a tool-augmented, memory-persistent, multi-step execution loop.

An agent is not an API. An API provides a defined interface for requesting a specific function. Its behavior is bounded and specified by a contract — the set of valid inputs, outputs, and error conditions is documented and stable. An agent may call APIs, but its behavior is probabilistic, contextual, and potentially unbounded — it may decide which APIs to call, in what order, with what parameters, based on its interpretation of a goal and the intermediate results it receives.

An agent is not robotic process automation (RPA). RPA follows predefined scripts to automate repetitive tasks. Its behavior is fully deterministic and specified at design time. It does not interpret instructions, adapt to novel situations, or make decisions about which actions to take. An agent's behavior is emergent from the interaction between its model, its context, and its available tools — it cannot be fully specified at design time, which is precisely why it requires a different security model.

An agent is not a SaaS feature with AI. Many SaaS applications have added AI-powered features — smart search, document summarization, automated tagging, predictive analytics. These features use AI models but do not constitute agents because they lack autonomous multi-step operation, tool access beyond the application's own functionality, and the ability to take actions across system boundaries. When a SaaS vendor adds a feature that does cross these boundaries — a copilot that can send emails, schedule meetings, modify records, and invoke external services — it has deployed an embedded agent, and the security considerations change accordingly.

2.4 Autonomy as a Spectrum

A critical dimension of any agent definition is its level of autonomy — the degree to which it can operate without human intervention. This paper uses a four-level autonomy model that aligns with observed enterprise deployment patterns.

Level	Label	Description	Human Role	Example
L0	No autonomy	Human makes all decisions; AI provides information only	Decision-maker	Search-enhanced Q&A
L1	Limited autonomy	AI performs tasks with human approval at each action	Approver	Code review copilot with commit approval
L2	Conditional autonomy	AI acts independently for defined low-risk operations; escalates for high-risk	Exception handler	Customer service agent with refund limits
L3	High autonomy	AI operates independently for critical actions within defined boundaries	Monitor / override	Autonomous trading system, SOC triage agent

Security controls must be calibrated to the autonomy level. An L0 system requires controls focused on information accuracy. An L3 system requires controls focused on action authorization, behavioral monitoring, circuit breakers, and kill switches. Applying L3 controls to an L0 system creates unnecessary friction; applying L0 controls to an L3 system creates unacceptable risk.

2.5 Agency vs. Orchestration

A distinction that matters for security architecture is the difference between an agent and an orchestrator. An agent executes tasks directly — it invokes tools, processes data, generates outputs, and takes actions. An orchestrator coordinates other agents — it decomposes goals into subtasks, assigns those subtasks to specialized agents, aggregates results, and manages the overall workflow. Some systems are both: an orchestrator agent that coordinates sub-agents while also performing its own direct tasks.

The security implications differ. Compromising an agent affects the tasks it performs. Compromising an orchestrator affects all agents it coordinates — it is a force multiplier for the attacker. Orchestrator security is therefore a higher-priority concern than individual agent security, and orchestrators require stronger isolation, more rigorous authentication, and more comprehensive monitoring than the agents they manage. This is directly analogous to the distinction between compromising a workload and compromising the Kubernetes control plane that manages all workloads.

3. Taxonomy of AI Agents

The following taxonomy classifies AI agents along four primary axes: autonomy level, persistence model (stateless vs. stateful), tool access scope, and communication pattern. Eleven agent types are defined, each with distinct operational characteristics and security profiles.

A note on composability: In production environments, agents frequently span multiple taxonomy categories. A single agent may simultaneously be tool-augmented, MCP-enabled, and retrieval-augmented, while operating as part of a multi-agent system. The taxonomy should be applied as a multi-label classification system, not a single-label one. An agent's composite threat model is the union of threats from all applicable categories. When classifying a real-world agent for governance purposes, identify all applicable types and layer the corresponding threat models and controls.

3.1 Conversational Agents

Conversational agents interact with humans through natural language dialogue. They maintain short-term context within a conversation but typically do not persist state across sessions or invoke external tools. Their primary function is information retrieval, question answering, and guided interaction.

Characteristic	Description
Autonomy	L0–L1
Persistence	Session-scoped (stateless across sessions)
Tool access	None or limited (search, knowledge base lookup)
Communication	Human-to-agent, natural language
Typical use cases	Customer support chatbots, FAQ systems, internal help desks
Example platforms	Custom GPTs, Dialogflow CX, Amazon Lex, Zendesk AI

Conversational agents represent the simplest end of the agent spectrum. Their security profile is dominated by prompt injection risk (particularly indirect injection via retrieved content), data leakage through conversation, and the potential for social engineering through manipulated agent responses.

3.2 Task Execution Agents

Task execution agents perform specific, bounded tasks when instructed by a user or system trigger. They may invoke tools and APIs to complete their assigned task but operate within a narrow scope and typically require human initiation for each task instance.

Characteristic	Description
Autonomy	L1–L2
Persistence	Task-scoped (state maintained during task, discarded after)
Tool access	Moderate (specific tools for the assigned task domain)
Communication	Human-to-agent instruction, structured output
Typical use cases	Data analysis, report generation, code generation, document processing
Example platforms	OpenAI Assistants API, Claude with tool use, Amazon Bedrock Agents

Task execution agents introduce tool abuse as a primary risk. Because they invoke external systems — databases, APIs, file systems — an attacker who can influence the agent's task interpretation can redirect tool invocations to unintended targets, extract data through tool queries, or modify records through tool writes. The bounded scope limits blast radius, but tool-level authorization must be enforced.

3.3 Autonomous Agents

Autonomous agents pursue goals over extended time horizons with minimal human intervention. They decompose complex objectives into subtasks, execute those subtasks through tool invocations and information gathering, evaluate progress, and adjust their approach based on intermediate results. They maintain persistent state — long-term memory, accumulated context, learned preferences — across sessions.

Characteristic	Description
Autonomy	L2–L3
Persistence	Persistent (long-term memory, cross-session state)
Tool access	Broad (multiple tools, potentially dynamically discovered)

Characteristic	Description
Communication	Goal-directed, self-initiated actions
Typical use cases	Research automation, long-running data analysis, autonomous cybersecurity operations
Example platforms	AutoGPT, OpenClaw (open-source agentic AI platform, 250K+ GitHub stars), Devin (Cognition), Claude Code

Autonomous agents present the broadest threat surface of any single-agent type. Their persistent memory creates a poisoning vector that can influence behavior across sessions. Their broad tool access creates over-permissioning risks. Their goal-directed autonomy creates goal hijacking risks where an attacker manipulates the agent's objective function. Their long-running nature means that a subtle compromise may persist undetected for extended periods, accumulating influence over the agent's behavior through memory manipulation. The lesson of Knight Capital — which lost $440 million in 45 minutes due to an automated system operating without adequate circuit breakers — applies directionally: autonomous systems with access to financial markets can cause significant damage before human oversight detects the deviation [7]. The Knight Capital failure was caused by deterministic software, not probabilistic AI, but the underlying principle — that speed of autonomous execution requires proportionally robust safeguards — applies with even greater force to agents whose behavior is non-deterministic.

3.4 Multi-Agent Systems

Multi-agent systems comprise multiple agents that collaborate to achieve a shared objective. Individual agents may specialize in different functions — research, analysis, writing, code generation, quality assurance — and coordinate through defined communication channels. The emergent behavior of the system may differ from the behavior of any individual agent.

Characteristic	Description
Autonomy	L1–L3 (varies by agent within the system)
Persistence	System-level state shared across agents
Tool access	Distributed across specialized agents
Communication	Agent-to-agent, structured messages, shared context

Characteristic	Description
Typical use cases	Complex research workflows, software development pipelines, business process automation
Example platforms	CrewAI, AutoGen, LangGraph multi-agent, OpenAI Swarm

Multi-agent systems introduce risks that do not exist in single-agent architectures. **Cascading failures** – where an error or compromise in one agent propagates through the system via inter-agent communication – can amplify the impact of a single-point failure across the entire workflow (OWASP ASI08). **Agent impersonation** – where a malicious agent inserts itself into the communication channel between legitimate agents – can redirect the entire system's behavior. **Trust boundary violations** across agents with different permission scopes can create privilege escalation paths. The security of a multi-agent system is not the sum of the security of its individual agents; it is a distinct property that must be evaluated at the system level.

3.5 Orchestrator / Supervisor Agents

Orchestrator agents manage and coordinate other agents without directly performing the tasks themselves. They decompose high-level objectives, assign tasks to specialized agents, monitor progress, handle exceptions, and aggregate results. In hierarchical architectures, supervisor agents oversee other orchestrators, creating multi-level coordination structures.

Characteristic	Description
Autonomy	L2–L3
Persistence	Persistent (maintains workflow state and agent registry)
Tool access	Meta-tools (agent spawning, task assignment, result aggregation)
Communication	Agent-to-agent orchestration, workflow management
Typical use cases	Enterprise workflow automation, CI/CD pipeline orchestration, incident response coordination
Example platforms	LangGraph orchestrators, Semantic Kernel planners, custom enterprise orchestrators

As discussed in Section 2.5, orchestrator compromise is the highest-impact attack vector in multi-agent architectures. An attacker who controls the orchestrator controls the instructions, permissions, and data flows for every agent in the system. OWASP's Agent Goal Hijack (ASI01) category applies with particular force to orchestrators, as hijacking the orchestrator's goal redirects the entire agent population. Orchestrators should be treated as control plane components, not data plane components, with corresponding isolation, authentication, and monitoring requirements.

3.6 Tool-Augmented Agents

Tool-augmented agents are distinguished by their primary reliance on external tool invocations to accomplish tasks. While most agents use some tools, tool-augmented agents define their capability primarily through their tool portfolio — the set of external systems, APIs, and functions they can invoke.

Characteristic	Description
Autonomy	L1–L3
Persistence	Varies (often session-scoped with tool state managed externally)
Tool access	Extensive (the defining characteristic — large tool portfolios)
Communication	Model-to-tool invocation via function calling, MCP, or custom protocols
Typical use cases	System administration, database management, DevOps automation, security operations
Example platforms	Any agent framework with MCP integration, OpenAI function calling, Claude tool use

Tool-augmented agents concentrate risk at the tool interface. **Tool poisoning** — where an attacker manipulates tool descriptions to cause the agent to misuse a tool — is a primary threat. **Over-permissioning** — granting the agent access to tools beyond what its task requires — expands the blast radius of any compromise. **Confused deputy attacks** — where the agent is tricked into using its legitimate tool access to perform unauthorized actions on behalf of an attacker — exploit the trust that tool providers place in the agent's identity. The Model Context Protocol, which has become the dominant tool integration standard, carries known security weaknesses including pre-authentication RCE (CVE-2025-6514), tool description poisoning, rug-pull attacks, and session hijacking [3][8].

3.7 Retrieval-Augmented Agents (RAG Agents)

RAG agents combine a foundation model with a retrieval system that searches knowledge bases, document stores, or databases to provide grounded, source-backed responses. They may operate as part of a larger agent architecture or as standalone systems.

Characteristic	Description
Autonomy	L0–L2
Persistence	Knowledge base is persistent; agent may be session-scoped
Tool access	Retrieval-focused (vector search, document APIs, database queries)
Communication	User query → retrieval → model synthesis → grounded response
Typical use cases	Enterprise knowledge management, legal research, compliance Q&A, financial analysis
Example platforms	LangChain RAG chains, Amazon Bedrock Knowledge Bases, Azure AI Search + OpenAI

RAG agents introduce a distinct attack surface at the retrieval boundary. **Retrieval boundary violations** occur when vector similarity search returns documents the user is not authorized to see — because the search operates on mathematical similarity, not access control lists. **Indirect prompt injection via retrieved content** allows adversarial text embedded in source documents to hijack the agent's behavior when those documents are retrieved and included in the prompt. **Embedding inversion** enables attackers to reverse-engineer information about source documents from the vector embeddings stored in the vector database. **Data poisoning through document ingestion** allows attackers to corrupt the knowledge base by inserting adversarial content into documents that will later be retrieved and trusted.

3.8 MCP-Enabled Agents

MCP-enabled agents use the Model Context Protocol to dynamically discover and connect to tool servers, gaining capabilities at runtime that were not hardcoded at development time. This dynamic capability acquisition is both the defining advantage and the defining risk of MCP-enabled agents.

Characteristic	Description
Autonomy	L1–L3

Characteristic	Description
Persistence	Varies; MCP connections may be session-scoped or persistent
Tool access	Dynamic (tools discovered at runtime via MCP server connections)
Communication	MCP client-server protocol (stdio, SSE, or streamable HTTP transport)
Typical use cases	Extensible development environments, enterprise integration hubs, general-purpose agents
Example platforms	OpenClaw, Claude Desktop, Cursor, Windsurf, Cline, custom MCP clients

MCP-enabled agents inherit all tool-augmented agent risks and add protocol-specific vulnerabilities. MCP's design prioritizes developer ergonomics: connecting a new server is as simple as adding a configuration entry, authentication is optional in many implementations (38 to 41 percent of MCP servers lack authentication according to security assessments), and tool descriptions are free-form text that the agent's model interprets without validation [17]. This design created an enormous adoption surface — over 10,000 active community-built MCP servers — but also an enormous attack surface. Between January and February 2026, security researchers filed over 30 CVEs targeting MCP servers, clients, and infrastructure. Critical vulnerabilities include CVE-2025-6514 (CVSS 9.6, pre-authentication RCE in mcp-remote affecting 437,000+ downloads), CVE-2025-68143/68144/68145 (critical path traversal and argument injection in Anthropic's own mcp-server-git), and CVE-2026-25536 (cross-client data leak in the MCP TypeScript SDK) [8][17]. A VirusTotal survey of nearly 18,000 MCP server projects found that over 8 percent showed signs of intentional malice [17]. The CSAI program has identified six distinct MCP threat categories: pre-authentication remote code execution, tool poisoning through hidden instructions, rug-pull attacks that modify tool definitions post-approval, session hijacking through predictable identifiers, supply chain compromise through malicious server packages, and cross-server data exfiltration [3][8].

3.9 Embedded Agents (Within SaaS Platforms)

Embedded agents operate within the context of a SaaS application, extending the application's native functionality with AI-powered capabilities. They inherit the application's identity context, data access permissions, and interaction model.

Characteristic	Description
Autonomy	L1–L2 (constrained by the host platform)

Characteristic	Description
Persistence	Platform-managed (persists within the SaaS context)
Tool access	Platform-scoped (tools are the SaaS application's own features and integrations)
Communication	Natural language within the SaaS UI, API calls within the platform
Typical use cases	Email triage, meeting scheduling, document drafting, CRM record management, code assistance
Example platforms	Microsoft 365 Copilot, Google Workspace Gemini, Salesforce Einstein Copilot, GitHub Copilot

Embedded agents present a distinct governance challenge: they are often deployed by the SaaS vendor without the explicit decision or oversight of the enterprise security team. When Microsoft adds Copilot capabilities to Outlook, or Salesforce adds Einstein Copilot to its CRM, the enterprise inherits an agent with access to sensitive data and the ability to take actions — regardless of whether the enterprise has an agent governance program. The security model depends heavily on the SaaS vendor's implementation: what data the agent can access, what actions it can take, how its permissions relate to the user's permissions, and what logging and monitoring capabilities the enterprise can access.

3.10 Edge / Device Agents

Edge agents execute on local devices — laptops, mobile phones, IoT devices, vehicles, industrial equipment — rather than in cloud environments. They provide low-latency, privacy-preserving AI capabilities that function without continuous network connectivity.

Characteristic	Description
Autonomy	L1–L3 (often higher autonomy due to connectivity constraints)
Persistence	Local device storage
Tool access	Device-local (sensors, actuators, local APIs, local file systems)
Communication	Local interfaces, intermittent cloud synchronization
Typical use cases	On-device assistants, autonomous vehicle systems, industrial automation, offline-capable field agents

Characteristic	Description
Example platforms	Apple Intelligence, on-device LLMs (Llama, Phi, Gemma), NVIDIA Jetson agents

Edge agents introduce a physical attack surface that cloud-hosted agents do not face. Device compromise — through physical access, firmware exploitation, or sideloading — can give an attacker direct access to the agent's model, memory, and tool interfaces. Model extraction is a heightened risk for on-device models where the weights are physically accessible. The intermittent connectivity model means that behavioral monitoring and policy updates may be delayed, creating windows of unsupervised operation.

3.11 Human-in-the-Loop Agents

Human-in-the-loop (HITL) agents are architecturally designed to require human approval at defined checkpoints in their execution. Unlike agents that happen to operate at L1 autonomy, HITL agents have approval gates built into their execution loop as a structural feature.

Characteristic	Description
Autonomy	L1 (by design)
Persistence	Varies (task- or session-scoped)
Tool access	Full tool access, gated by human approval at defined checkpoints
Communication	Agent proposes actions, human approves/modifies/rejects
Typical use cases	High-stakes financial operations, regulated process automation, safety-critical systems
Example platforms	Any framework with approval workflows (LangGraph interrupt, custom approval gates)

HITL agents shift the primary threat model from autonomous action risks to approval workflow risks. **Approval fatigue** — where human reviewers become desensitized to approval requests and approve actions without genuine scrutiny — is a well-documented human factors risk. **Context manipulation** — where the agent presents information to the human in a way that makes a malicious action appear benign — exploits the trust humans place in the agent's summary of what it intends to do. **Approval bypass** — where the agent finds a way to execute an action without triggering the approval gate — targets the approval mechanism itself.

4. Agent Architectures and Operational Models

4.1 Core Components

Every agent, regardless of type, is composed of variations on five core architectural components. Understanding these components enables security architects to identify attack surfaces systematically.

The Model. The foundation model provides the agent's reasoning, language understanding, and decision-making capabilities. It may be a cloud-hosted API (GPT-4, Claude, Gemini), an open-weight model deployed on-premises (Llama, Mistral), or a specialized fine-tuned model. The model's architecture has direct security implications: cloud-hosted models provide no visibility into weights but offer contractual assurances; open-weight models enable inspection but can be fine-tuned to remove safety guardrails. The model is the component most susceptible to prompt injection attacks.

Memory. Agents maintain context through multiple memory tiers. **Short-term memory** (the context window) holds the current conversation and recent tool outputs, bounded by the model's context length. **Working memory** stores structured intermediate results during multi-step tasks. **Long-term memory** persists information across sessions using vector databases (Pinecone, Weaviate, pgvector, Chroma), key-value stores, or structured databases. **Episodic memory** records past interactions and outcomes for future reference. Each tier introduces distinct security concerns: short-term memory is vulnerable to injection through the current context; long-term memory is vulnerable to persistent poisoning that influences future behavior across sessions. The MINJA attack (presented at NeurIPS 2025) demonstrated that attackers can inject malicious records into an agent's memory through query-only interaction, achieving over 95 percent injection success rates [15]. The MemoryGraft technique (December 2025) showed that implanting fake "successful experiences" into agent memory via benign-looking content such as README files could poison 47.9 percent of memories retrieved during subsequent operations [16].

Tools. Tools are the interfaces through which agents interact with external systems. They may be defined through function calling schemas (OpenAI, Anthropic), MCP server connections, REST API integrations, code execution environments, or custom interfaces. The tool portfolio determines the agent's capability envelope — what it can do — and therefore its potential blast radius when compromised. Tool security encompasses authentication (does the agent have proper credentials for the tool?), authorization (is the agent permitted to use this tool for this purpose?), input validation (are the agent's tool inputs sanitized?), and output validation (are tool responses verified before the agent acts on them?).

The Planning / Reasoning Loop. The planning loop determines how the agent decides what to do. Multiple patterns exist, each with distinct security properties:

ReAct (Reason + Act) interleaves reasoning and action in a tight loop: the model generates a thought about what to do, executes an action, observes the result, and repeats. This pattern is transparent (each step is logged) but can be manipulated through adversarial observations that redirect the reasoning chain [9].

Plan-and-Execute separates planning from execution: the model first generates a complete plan, then a separate execution engine carries out each step. This pattern enables plan review before execution but creates a risk if the planning step is compromised — a poisoned plan directs all subsequent actions [10].

Tool Chaining links multiple tool invocations in sequence, where the output of one tool becomes the input to the next. This pattern creates data flow paths that are difficult to monitor and may propagate tainted data from a compromised tool through the entire chain.

Stateful Workflow Graphs treat agent execution as a state machine with typed state, conditional edges, and explicit checkpoint/resume semantics. This pattern, popularized by LangGraph and increasingly adopted in production, is distinct from both ReAct and plan-and-execute. The developer defines a graph of nodes (LLM calls, tool calls, human approval gates, routing logic) and the framework manages state persistence, replay, and branching. The security implications are significant: state serialization and deserialization at checkpoint boundaries introduces a target for poisoning; checkpoint stores that persist workflow state can be tampered with to alter execution paths; and subgraph isolation between different workflow branches requires explicit enforcement to prevent state leakage across branches.

Event-Driven Agents respond to external triggers (webhooks, message queue events, file system changes) rather than operating in a continuous loop. This pattern creates a threat surface at the event ingestion point — a malicious event can trigger unintended agent behavior.

Identity and Authorization. The identity layer determines who the agent is and what it is permitted to do. Current implementations range from no identity at all (agents operate with the credentials of whatever system process runs them) to OAuth token delegation, to emerging approaches based on SPIFFE/SPIRE workload identity. The CSA Agentic Identity Survey found that 44 percent of organizations rely on static API keys as their primary agent authentication method [2]. The identity layer appears to be one of the least mature components in many agent architectures based on the survey data cited above, and its weaknesses contribute to the privilege escalation and unauthorized access risks documented in Section 7.

4.2 Deterministic vs. Probabilistic Behavior

A property that fundamentally distinguishes agents from traditional software is non-determinism. The same input to an agent may produce different actions depending on the model's sampling parameters, the current context window contents, the order of tool responses, and the state of the agent's memory. This non-determinism has direct security implications.

Traditional software security testing relies on deterministic behavior: a test that passes once is expected to pass in the same conditions. Agent behavior cannot be verified this way. An agent that behaves safely in 999 out of 1,000 executions may take a dangerous action in the 1,001st — not because of a bug, but because the model's probabilistic sampling happened to select a different token sequence. This means that agent security requires **runtime behavioral monitoring** in addition to pre-deployment testing, because pre-deployment testing cannot exhaustively characterize the agent's behavior across all possible execution paths.

A related but distinct risk is **upstream model versioning**: when a model provider ships a new version (GPT-4 to GPT-4-turbo, Claude version updates), agent behavior can change without any modification to the agent's code, tools, or configuration. This has caused production incidents at multiple enterprises and represents a form of supply chain risk at the model layer that is separate from the inherent sampling randomness discussed above.

Approaches to managing non-determinism include structured output enforcement (constraining the model to output valid JSON or specific action schemas), constitutional AI techniques (encoding behavioral rules that the model evaluates its own outputs against), tool validation layers (verifying that tool invocations are within permitted parameters before execution), and temperature/sampling controls (reducing randomness at the cost of response diversity). None of these fully eliminates non-determinism, which is why observability and runtime monitoring are essential rather than optional components of agent security architecture.

5. Communication Models and Protocols

5.1 How Agents Interact

Agents communicate across four primary interaction patterns, each with distinct trust boundary characteristics.

Human-to-Agent communication occurs through natural language interfaces (chat, voice, email), structured forms, or API calls. The trust boundary is between the human user and the agent. The primary risks are prompt injection (the user — or content the user provides — manipulates the agent) and social engineering (the agent manipulates the user through misleading responses).

Agent-to-Tool communication occurs through function calling, API invocation, or protocol-mediated interactions (MCP). The trust boundary is between the agent and the external system. The primary risks are tool abuse, confused deputy attacks, and data exfiltration through tool queries.

Agent-to-Agent communication occurs through structured messages, shared memory, or protocol-mediated channels. The trust boundary is between agents that may have different permission levels, different owners, or different trust profiles. The primary risks are agent impersonation, privilege escalation through delegation, and cascading failure propagation.

System-to-Agent communication occurs through event triggers, scheduled invocations, or API calls from enterprise systems. The trust boundary is between the triggering system and the agent. The primary risks are malicious event injection and unauthorized agent invocation.

5.2 Emerging Protocols

Model Context Protocol (MCP). Developed by Anthropic and open-sourced in November 2024, MCP defines a client-server protocol for connecting AI agents to external tool servers. The protocol specifies five primitives: **tools** (functions the agent can invoke), **resources** (data the agent can read), **prompts** (templates for structured interactions), **sampling** (allowing servers to request LLM completions from the client — a security-significant capability since a malicious server can effectively execute prompt injection through the client's model), and **roots** (file system access declarations from clients to servers). Transport mechanisms include stdio (for local processes) and streamable HTTP; the earlier Server-Sent Events (SSE) transport was deprecated in the March 2026 spec revision. MCP's adoption has been rapid — supported by OpenClaw, Claude Desktop, Cursor, Windsurf, Cline, and dozens of other agent platforms — but its security model, while improving, has significant gaps. An OAuth 2.1-based authorization framework (MCP Auth) was

added in early 2026 but remains optional, and most community servers do not implement it. Tool descriptions are unverified free-form text; and there is no mandatory message signing or integrity protection [3][8].

Google Agent-to-Agent (A2A) Protocol. Announced in April 2025, A2A provides a protocol for communication between agents built on different frameworks. The protocol uses **Agent Cards** (JSON metadata describing an agent's capabilities and authentication requirements), **Tasks** (units of work with defined lifecycles), and **Channels** (communication streams supporting text, structured data, and streaming). A2A is designed to complement MCP (which connects agents to tools) by connecting agents to each other. Over 50 technology partners have adopted A2A, including Salesforce, SAP, Atlassian, and ServiceNow [4].

OpenAI Function Calling. OpenAI's tool use protocol defines tools through JSON Schema specifications that describe function names, parameters, and descriptions. The model generates structured JSON function call requests, which the calling application executes and returns results to the model. Parallel tool calling enables multiple function invocations in a single model turn. This protocol is the most widely adopted tool-use mechanism but operates within a single API call context rather than as a persistent server protocol.

Payment Protocols. Visa's Trusted Agent Protocol (October 2025), Mastercard's Agent Pay (April 2025), Google's Agent Payments Protocol AP2 (September 2025), and Stripe's Machine Payments Protocol (March 2026) address the specific challenge of AI agents initiating financial transactions. These protocols must solve authentication (proving the agent acts on behalf of an authorized user), authorization (enforcing spending limits and merchant restrictions), and fraud detection (distinguishing legitimate agent transactions from malicious activity) [5].

5.3 Trust Boundaries in Communication

Every communication channel between an agent and another entity (human, tool, agent, or system) represents a trust boundary. The security architecture for agent communication must address three questions at each boundary:

Authentication: How does each party verify the identity of the other? Current gaps include agents that cannot verify MCP server identity, MCP servers that cannot verify agent identity, and agents that accept messages from other agents without verifiable credentials.

Authorization: What is each party permitted to do across this boundary? Current gaps include tool access that is not scoped to the current task, delegation chains that do not narrow permissions at each hop, and agents that inherit their orchestrator's full permission set rather than receiving minimum necessary scope.

Context propagation risks: What information crosses this boundary, and could it be used to manipulate the receiving party? Context propagation is the mechanism through which indirect prompt injection operates: adversarial content in a tool response, a retrieved document, or an inter-agent message enters the agent's context and influences its subsequent behavior.

6. Enterprise Integration Patterns

6.1 How Agents Are Deployed in Enterprises

Enterprise agent deployments follow several distinct patterns, each with different security, governance, and observability requirements.

Internal Productivity Agents assist employees with tasks such as document drafting, data analysis, code generation, meeting summarization, and email management. They typically operate as embedded agents within productivity suites (Microsoft 365 Copilot, Google Workspace Gemini) or as standalone tools (GitHub Copilot, Claude Code, Cursor). The primary security concern is data leakage — these agents have access to the employee's documents, emails, calendars, and files, and sensitive content can flow through the agent to model providers. The shadow AI problem — employees using unauthorized AI tools on sensitive data — is concentrated in this category.

Customer-Facing Agents interact directly with customers through chat interfaces, voice channels, or application integrations. They handle customer inquiries, process transactions, provide personalized recommendations, and route complex issues to human agents. The primary security concerns are data privacy (customer PII exposure), response integrity (ensuring the agent does not provide inaccurate or harmful information), and adversarial manipulation (customers or attackers probing the agent for unauthorized functionality).

Security Automation Agents perform cybersecurity functions including alert triage, threat hunting, vulnerability scanning, and incident response. These agents require access to security telemetry, internal network data, and potentially active response capabilities (blocking IPs, isolating endpoints, revoking credentials). The bidirectional risk is acute: a compromised security agent with containment capabilities could be weaponized to disrupt the organization it was deployed to protect [6].

Developer Copilots assist software engineers with code generation, debugging, code review, and documentation. They operate within development environments (VS Code, JetBrains, terminal) and have access to source code repositories, build systems, and deployment pipelines. The primary security concerns are insecure code generation, supply chain risk (suggesting malicious dependencies), and unauthorized access to source code.

Cross-System Orchestration Agents coordinate workflows that span multiple enterprise systems — ERP, CRM, HR, finance, IT service management. These agents require broad integration access and often operate with elevated permissions across multiple trust domains. They are the enterprise equivalent of orchestrator agents (Section 3.5) and present the highest blast radius in an enterprise context.

6.2 Identity Integration Challenges

Agent deployment in enterprises surfaces fundamental challenges in identity and access management that existing IAM systems were not designed to handle.

Agent identity representation. Most enterprise IAM systems model identities as either human users or service accounts. Agents are neither — they act on behalf of humans but persist beyond sessions, they use tools like service accounts but make probabilistic decisions, and they may operate under delegated authority that requires tracing back to a human principal. The CSA Agent Identity Governance Framework defines five identity types — copilot, autonomous, orchestrator, ephemeral sub-agent, and agent-as-a-service — each requiring distinct lifecycle management [2][6]. Standards development is accelerating: the IETF published draft-klrc-aiagent-auth-00 in March 2026, a 26-page framework called AIMS (Agent Identity Management System) that composes WIMSE, SPIFFE, and OAuth 2.0 for agent credentials [21]. The OpenID Foundation published a whitepaper on Identity Management for Agentic AI in October 2025 [22]. NIST launched its AI Agent Standards Initiative in February 2026 with three pillars — agent standards development, open source protocol maintenance, and research in agent security and identity [23]. Despite this activity, the standards remain early-stage: no single identity framework has achieved production-grade maturity for agent-to-agent authentication scenarios where no human participates in the consent flow.

Permission scoping for dynamic behavior. Traditional permission models assign static roles or policies to identities. Agents require dynamic permissions that change based on the current task, the data being accessed, and the autonomy level in effect. A customer service agent that is authorized to process refunds up to $50 should not need — and should not have — the same permissions when answering a product information question.

Delegation chains. When an orchestrator agent delegates a task to a sub-agent, the sub-agent must receive credentials sufficient to perform the task but no broader than what the task requires. Current implementations frequently fail this requirement — sub-agents inherit the orchestrator's full permission set, creating privilege escalation paths that OWASP classifies as ASI03 (Identity and Privilege Abuse) [11].

6.3 Observability Gaps

Enterprise observability systems — SIEM, APM, log aggregation — were designed to monitor infrastructure and application behavior, not agent decision-making. The critical gap is at the reasoning layer: existing tools can capture that an agent made an API call, but they cannot capture why the agent made that call, what context informed the decision, whether the action was consistent with the agent's assigned goal, or whether the agent's reasoning was influenced by injected content.

Addressing this gap requires agent-specific telemetry that captures, at minimum: the prompt or instruction that initiated the agent's action, the reasoning trace (chain-of-thought or planning output) that led to the decision, the tools invoked and the parameters used, the results returned by each tool, the agent's assessment of those results, and whether any human approval was requested or obtained. The CSA AI Risk Observatory defines a standardized agentic telemetry event format that captures these elements [6].

7. Threat Models by Agent Type

This section maps the eleven agent types defined in Section 3 to their primary threat models, demonstrating why different agent types require fundamentally different security controls.

7.1 Threat Model Summary Matrix

Agent Type	Primary Threats	Unique Vulnerabilities	OWASP ASI Mapping
Conversational	Prompt injection, data leakage, social engineering	Content-based manipulation of responses	ASI01, ASI06, ASI09
Task Execution	Tool abuse, over-permissioning, confused deputy	Unauthorized tool invocations	ASI02, ASI03
Autonomous	Goal hijacking, memory poisoning, behavioral drift	Persistent compromise across sessions	ASI01, ASI06, ASI10
Multi-Agent	Cascade failures, agent impersonation, trust boundary violations	Emergent misbehavior, privilege escalation across agents	ASI07, ASI08, ASI03
Orchestrator	Control plane compromise, goal redirection for all sub-agents	Single point of failure for entire agent population	ASI01, ASI08, ASI10
Tool-Augmented	Tool poisoning, rug-pull attacks, credential theft via tools	Dynamic capability acquisition from untrusted sources	ASI02, ASI04
RAG	Retrieval boundary violations, indirect injection via documents, embedding inversion	Unauthorized data access through semantic search	ASI06, ASI01

Agent Type	Primary Threats	Unique Vulnerabilities	OWASP ASI Mapping
MCP-Enabled	Protocol-level attacks, server impersonation, session hijacking	Pre-auth RCE, tool description poisoning, supply chain compromise	ASI02, ASI04, ASI05
Embedded (SaaS)	Shadow deployment, inherited over-permissioning, data flow opacity	Vendor-controlled security model, limited enterprise visibility	ASI03, ASI09
Edge/Device	Physical attack surface, model extraction, offline operation windows	Device compromise, firmware exploitation, delayed monitoring	ASI10, ASI05
Human-in-the-Loop	Approval fatigue, context manipulation, approval bypass	Human factors exploitation, false sense of security	ASI09, ASI01

7.2 Cross-Cutting Threats

Several threat categories apply across all or most agent types:

Prompt injection affects any agent that processes natural language, though its severity varies dramatically by agent type. For a conversational agent, prompt injection may cause an inappropriate response. For an autonomous agent with tool access, prompt injection may cause unauthorized actions with real-world consequences. The EchoLeak vulnerability (CVE-2025-32711, CVSS 9.3) demonstrated this concretely: a crafted email sent to a Microsoft 365 Copilot user's inbox contained hidden instructions that the agent ingested during summarization, enabling it to extract sensitive data from OneDrive, SharePoint, and Teams and exfiltrate it through a trusted Microsoft domain — the first confirmed zero-click prompt injection exploit in a production enterprise AI system [18]. Johann Rehberger's "Month of AI Bugs" campaign in August 2025 disclosed over two dozen prompt injection vulnerabilities across every major agentic AI coding assistant, including ChatGPT, Claude Code, GitHub Copilot, Cursor, and Devin [19]. Prompt injection appeared in 73 percent of production AI deployments assessed during security audits in 2025 [20].

Supply chain risks (OWASP ASI04) affect any agent that relies on external components — models, tools, plugins, MCP servers, knowledge bases, or training data. The ClawHavoc campaign demonstrated this risk at scale: between 341 and 1,467 malicious skills on ClawHub combined prompt injection with traditional malware, affecting thousands of OpenClaw deployments [12].

Tool output injection is distinct from tool description poisoning and represents one of the most practical agent attack paths. A legitimate external service — a web scraping tool, a database query server, any MCP server returning user-generated content — may return output containing prompt injection payloads. When this output is placed into the agent's context window, it can hijack subsequent reasoning and actions. Unlike tool description poisoning, which requires compromising the tool registry, tool output injection exploits normal tool operation with adversarial data in the environment.

Data exfiltration via context affects any agent that processes sensitive data. The mechanism varies: conversational agents may leak data through responses, tool-augmented agents through tool queries, RAG agents through retrieval, and multi-agent systems through inter-agent messages. The common thread is that the agent's context window becomes a channel through which sensitive data can move in unintended directions.

Non-human identity compromise affects any agent that holds credentials. With 44 percent of organizations relying on static API keys for agent authentication and 78 percent lacking documented policies for agent identity lifecycle management [2], the identity layer is the weakest point in most agent deployments.

7.3 Why One Policy Fails

The threat model matrix demonstrates why a single "AI agent security policy" fails in practice. Consider a policy that requires "human approval for all agent actions." This policy is:

- **Appropriate** for HITL agents in high-stakes financial operations (it is their design pattern)
- **Impractical** for autonomous agents performing high-frequency operations (the approval latency eliminates the agent's value)
- **Meaningless** for embedded SaaS agents (the enterprise cannot add approval gates to Microsoft Copilot's architecture)
- **Insufficient** for orchestrator agents (approving each orchestrator action does not address the risk of orchestrator compromise)
- **Irrelevant** for edge agents (they may operate offline without connectivity to an approval system)

Effective agent security requires a policy framework that defines controls at the intersection of agent type, autonomy level, deployment context, and data sensitivity. The taxonomy in Section 3, combined with the threat models in this section, provides the analytical foundation for building such a framework.

8. Security Control Implications

8.1 From One-Size-Fits-All to Agent-Type-Specific Controls

The findings in Sections 3 through 7 converge on a single principle: agent security controls must be calibrated to the specific agent type, its autonomy level, its tool access scope, and its deployment context. This section maps the implications to enterprise security frameworks.

8.2 Control Granularity

Traditional application security operates at three levels: network controls (can the application communicate?), identity controls (who is using the application?), and data controls (what data can the application access?). Agent security requires three additional levels of granularity:

Task-level controls govern what the agent is currently doing. An agent authorized for customer support should not be able to execute database administration commands, even if the underlying tool access would technically permit it. Task-level controls require understanding the agent's current objective and evaluating each action against that objective.

Tool-level controls govern which tools the agent can invoke and with what parameters. This goes beyond traditional API authorization — it requires understanding the semantic intent of each tool invocation and whether it is consistent with the agent's assigned task. Tool-level controls should enforce least privilege, just-in-time access, and mandatory validation of tool outputs before the agent acts on them.

Context-level controls govern what information enters and exits the agent's context window. This includes input filtering (scanning prompts and tool responses for injection attempts), output filtering (scanning agent outputs for sensitive data before delivery), and context isolation (preventing sensitive data from one task from leaking into a different task through shared memory).

8.3 Guardrail and Sandboxing Architectures

The controls described above require concrete implementation mechanisms. Two architectural patterns are essential for production agent security: guardrails and sandboxes.

Guardrail architectures enforce policy on agent inputs, outputs, and actions through interception layers positioned at defined points in the agent execution loop. Structural guardrails (JSON schema validation, allowlisted tool sets, parameter range checks) provide deterministic enforcement that an agent cannot

bypass through prompt manipulation. Model-based guardrails (content classifiers, PII detectors, intent analyzers) provide probabilistic enforcement that can catch semantic violations but are themselves susceptible to adversarial bypass. The most robust implementations layer both: structural guardrails as hard boundaries and model-based guardrails as defense-in-depth. Guardrails may be deployed inline (intercepting every input and output in the critical path), as sidecars (evaluating in parallel with the agent and halting execution if policy violations are detected), or as external policy engines (where the agent must request authorization from an out-of-process policy service before taking high-risk actions). The out-of-process pattern — used by NVIDIA NemoClaw's policy engine — is the most secure because a compromised agent process cannot override the policy enforcement.

Sandboxing architectures limit the blast radius of a compromised agent by restricting its access to system resources. The technology choice depends on the agent type and deployment context. Firecracker microVMs provide the strongest isolation (dedicated kernel per workload, approximately 125ms boot time, less than 5 MiB memory overhead) and are appropriate for agents executing untrusted code. gVisor provides user-space kernel interception with lower overhead and is suitable for compute-heavy agents with limited I/O. Hardened Docker containers provide minimal overhead but share the host kernel, creating escape risk for agents processing untrusted input. For MCP servers — which are frequently community-built packages of unknown provenance — container isolation with restricted network egress, read-only filesystem mounts, and resource limits (CPU, memory, execution time) should be considered a minimum deployment standard. Google's Kubernetes Agent Sandbox (open-sourced November 2025) provides an emerging pattern for isolated, stateful agent workloads in Kubernetes environments. Cost and execution time circuit breakers (maximum tokens per task, maximum tool calls per execution, maximum wall-clock time) provide an additional safeguard against runaway agents that is both a security and an operational control.

8.4 Alignment with CSA Frameworks

The CSA AI Controls Matrix (AICM) provides 243 control objectives across 18 security domains [6]. The CSAI program has assessed each control for agentic fitness, classifying controls as fully applicable, applicable with modification, or missing entirely for agentic scenarios. The primary gaps — and the areas where new controls are needed — fall into six categories:

1. **Agent identity and lifecycle management** — controls for the five agent identity types, credential management, and decommissioning
2. **Runtime behavioral governance** — drift detection, goal alignment monitoring, autonomous action boundaries
3. **Inter-agent communication security** — message integrity, replay prevention, trust establishment
4. **Tool and capability governance** — access approval, permission scoping, integrity verification
5. **Orchestration layer security** — multi-agent coordination, cascading failure prevention

6. **Human-agent interaction controls** – escalation triggers, override mechanisms, informed consent

The MAESTRO framework provides the threat modeling structure. Its seven layers – Foundation Model, Data and Knowledge Management, Agent Framework, Agent Orchestration, Tool and Resource Integration, Deployment and Infrastructure, and Ecosystem and External Interactions – map directly to the architectural components described in Section 4.1 and provide a structured approach to identifying which layer-specific threats apply to each agent type [13].

Zero Trust principles apply to agent architectures with particular force. Never trust, always verify applies to every agent communication – agents should not trust tool outputs, retrieved content, inter-agent messages, or even their own memory without verification. Least privilege access requires that agent permissions be scoped to the current task and automatically revoked when the task completes. Continuous monitoring and verification requires behavioral telemetry that can detect deviation from expected agent behavior in real time.

9. Toward Agent-Specific Evaluation and Assurance

9.1 Why Evaluation Must Be Agent-Type-Specific

The diversity of agent types documented in this paper demands evaluation frameworks that are calibrated to the specific risks of each type. Evaluating a conversational agent for goal hijacking resistance (relevant to autonomous agents) or testing a RAG agent for delegation safety (relevant to multi-agent systems) wastes evaluation resources on irrelevant threats while potentially missing the threats that matter most.

9.2 Evaluation Dimensions by Agent Type

An effective evaluation framework should assess agents across multiple dimensions, weighted according to the agent's type and deployment context:

Tool-use safety evaluates whether the agent respects permission boundaries, validates tool outputs before acting, and avoids invoking tools that exceed its authorized scope. This dimension is most critical for tool-augmented agents, MCP-enabled agents, and autonomous agents — and least relevant for conversational agents with no tool access.

Delegation safety evaluates the agent's behavior when delegating to or receiving tasks from other agents. Does it propagate excessive privileges? Does it verify the identity and authority of delegating agents? Does it refuse delegated tasks that exceed its authorization? This dimension is critical for multi-agent systems and orchestrators — and irrelevant for standalone agents.

Goal persistence measures the agent's resistance to goal hijacking under adversarial pressure. Can an attacker redirect the agent's objective through prompt injection, manipulated tool outputs, or poisoned memory? This dimension is critical for autonomous agents and orchestrators — and less relevant for task execution agents that operate under close human supervision.

Autonomy calibration evaluates whether the agent appropriately escalates to humans when confidence is low or stakes are high. Does it recognize the limits of its competence? Does it halt when encountering situations outside its designed parameters? This dimension is relevant across all agent types but is most critical for L2 and L3 autonomy agents operating in high-stakes domains.

Behavioral consistency evaluates whether the agent's actions remain within expected parameters over time. Does the agent exhibit drift — gradual changes in behavior due to memory accumulation, context pollution, or model updates? This dimension is critical for persistent agents with long-term memory and less relevant for stateless, session-scoped agents.

9.3 Certification Models

The evaluation dimensions above can be formalized into certification schemes that provide standardized assurance about an agent's security properties. The CSA STAR for AI program, extended with agentic-specific assessment criteria, provides one such model. Certification levels should correspond to autonomy levels: agents operating at L3 autonomy with broad tool access require more rigorous evaluation than L1 agents with human approval gates.

Three certification approaches are emerging:

Pre-deployment evaluation assesses the agent against a benchmark suite before it is permitted to operate. This provides a baseline assurance but does not address behavioral drift during operation.

Continuous runtime monitoring evaluates the agent's behavior in production against defined behavioral baselines. Deviations trigger alerts, investigation, and potentially automatic containment. This approach addresses the non-determinism challenge (Section 4.2) but requires significant instrumentation investment.

Periodic re-certification combines pre-deployment evaluation with scheduled reassessment, triggered either by time intervals or by events such as model updates, tool access changes, or detected anomalies. This approach balances thoroughness with operational feasibility.

The most robust assurance model combines all three: pre-deployment certification as a gate, continuous monitoring as a guardrail, and periodic re-certification as a verification mechanism.

10. Conclusion: Why Agent Diversity Breaks Traditional Security Models

The traditional enterprise security model was designed for a world of deterministic software, static identities, bounded permissions, and human-initiated actions. AI agents challenge each of these assumptions to varying degrees. Their behavior is probabilistic. Their identities are still immature. Their permissions must be dynamic and context-aware. Their actions are self-initiated and may have irreversible consequences.

But the more fundamental challenge is not that agents are different from traditional software — it is that agents are different from each other. A conversational chatbot and an autonomous trading agent share the label "AI agent" but share almost nothing in terms of their architecture, autonomy, tool access, persistence, communication patterns, or threat model. An MCP-enabled development agent and an embedded SaaS copilot both invoke external tools but through completely different protocols, trust models, and governance mechanisms. A human-in-the-loop financial operations agent and an edge device agent both take consequential actions but in radically different deployment contexts with radically different monitoring capabilities.

This diversity is the central challenge. Organizations that recognize it and build agent-type-specific governance will be positioned to deploy agents safely and capture their considerable operational benefits. Organizations that treat all agents as a homogeneous category will either constrain beneficial agents with inappropriate controls or leave dangerous agents inadequately governed — and in most cases, they will do both simultaneously.

The framework presented in this paper — precise definitions, structured taxonomy, architectural analysis, protocol mapping, and type-specific threat models — provides the analytical foundation for agent-type-specific governance. It is designed to be operationalized through the CSA AI Controls Matrix and its agentic extensions, the MAESTRO threat modeling framework, and Zero Trust architectural principles. The financial services industry's experience with the 2026 transition to agentic operations, documented in CSA's concurrent State of Cloud and AI for Financial Services report, demonstrates both the urgency and the scale of this challenge: 62 percent of surveyed financial institutions already deploy AI agents, and 85 percent anticipate autonomous AI-initiated financial transactions in the near term [14].

The agentic universe is already here. The question is not whether enterprises will populate it with diverse agent types — they already have. The question is whether they will govern that diversity with the precision it demands. The path forward is the Identify-Classify-Control-Monitor-Assure cycle introduced in this paper and operationalized through the CSA Agent Security Architecture: identify every agent in the environment,

classify it across the taxonomy dimensions, apply the type-specific control playbook, monitor its behavior at runtime, and continuously assure its performance through evaluation. Organizations that begin this cycle now — starting with inventory and classification — will build the governance foundation that scales as agent populations grow.

Appendix A: Agent Identification Framework

A.1 Purpose

The diversity of agent types in enterprise environments makes it impractical to apply uniform security controls. Before an organization can secure an agent, it must first identify and classify it. This framework provides a structured method for security teams to identify the characteristics of an agent, classify it within the taxonomy presented in this paper, and map it to appropriate security controls and evaluation criteria.

A.2 Agent Identification Worksheet

Security teams should evaluate each agent across the following attributes:

Attribute	Key Question	Possible Values	Security Relevance
Autonomy Level	Can the agent act without human approval?	L0, L1, L2, L3	Determines need for approval gates, circuit breakers
Persistence Model	Does the agent retain memory across sessions?	Stateless, Session, Persistent	Indicates memory poisoning and long-term drift risk
Tool Access Scope	What systems can the agent act on?	None, Limited, Broad, Dynamic	Defines blast radius and abuse potential
Communication Pattern	Does it interact with other agents or systems?	Human-only, Tool-based, Agent-to-Agent	Introduces trust boundary and cascade risks
Identity Model	What credentials does the agent use?	Static API key, OAuth token, Workload identity	Determines impersonation and privilege escalation risks
Trigger Mechanism	What initiates the agent?	Human, Event-driven, Scheduled, Autonomous	Impacts attack surface (e.g., zero-click via event triggers)

Attribute	Key Question	Possible Values	Security Relevance
Deployment Context	Where does it operate?	SaaS-embedded, Cloud, On-premises, Edge	Affects observability, isolation, and control enforcement

A.3 Classification Output

Agents should be classified as multi-label entities, not assigned to a single type. A production agent may simultaneously be MCP-enabled, tool-augmented, retrieval-augmented, and part of a multi-agent system. The resulting composite classification determines the applicable threat models (the union of threats from all matching types), the required control sets, and the evaluation criteria for assurance and certification.

Appendix B: Agent Security Playbooks by Type

B.1 Purpose

Each agent type introduces distinct risks that require targeted controls, not generic AI policies. The following baseline security playbooks are aligned to the taxonomy in Section 3.

B.2 Conversational Agents

Primary risks: Prompt injection, data leakage, social engineering. **Core controls:** Input filtering and sanitization, retrieval content validation, output redaction for PII and sensitive data, context window isolation between sessions. **Monitoring:** Response anomaly detection, prompt injection pattern indicators.

B.3 Task Execution Agents

Primary risks: Tool misuse, over-permissioning, confused deputy attacks. **Core controls:** Task-scoped authorization, tool allowlisting, parameter validation, output verification before execution. **Monitoring:** Unexpected tool invocation patterns, data access anomalies relative to task scope.

B.4 Autonomous Agents

Primary risks: Goal hijacking, memory poisoning, behavioral drift. **Core controls:** Goal integrity validation, memory validation and segmentation, circuit breakers (time, cost, action count), policy enforcement layer outside agent runtime. **Monitoring:** Deviation from goal trajectory, long-term behavioral drift, anomalous tool usage patterns.

B.5 Multi-Agent Systems

Primary risks: Cascading failures, agent impersonation, privilege escalation across trust boundaries. **Core controls:** Signed agent-to-agent communication, trust boundary enforcement, least-privilege delegation with scope narrowing at each hop, isolation between agent roles. **Monitoring:** Cross-agent anomaly detection, task propagation analysis, delegation chain audit.

B.6 Orchestrator / Supervisor Agents

Primary risks: Control plane compromise, system-wide goal hijacking. **Core controls:** Strong workload identity, strict isolation from execution-layer agents, policy-based task validation before delegation, comprehensive audit logging of all delegation actions. **Monitoring:** Task assignment anomalies, unauthorized agent spawning, scope escalation detection.

B.7 Tool-Augmented and MCP-Enabled Agents

Primary risks: Tool poisoning, supply chain compromise, session hijacking, tool output injection. **Core controls:** Tool integrity verification before use, MCP server allowlisting and vetting, mandatory authentication for all server connections, sandboxed execution environments with network egress controls. **Monitoring:** New tool connections, unexpected capability expansion, tool output anomaly detection.

B.8 RAG Agents

Primary risks: Retrieval boundary violations, indirect injection via retrieved content, embedding inversion. **Core controls:** Access control enforcement at the retrieval layer (document-level permissions, not just user-level), source document integrity validation, output scanning for data from unauthorized sources, vector database access restrictions. **Monitoring:** Retrieval pattern anomalies, cross-boundary data access, injection indicators in retrieved content.

B.9 Embedded SaaS Agents

Primary risks: Shadow deployment, inherited over-permissioning, data flow opacity. **Core controls:** Vendor security assessment including agent capabilities, enterprise admin controls for agent features (enable/disable per user group), data loss prevention integration, vendor-provided audit logging. **Monitoring:** Data access patterns, cross-application data flow, feature activation tracking.

B.10 Edge / Device Agents

Primary risks: Physical attack surface, model extraction, delayed monitoring during offline operation. **Core controls:** Hardware-attested execution environments where available, encrypted local model and memory storage, network segmentation, policy synchronization upon reconnection. **Monitoring:** Post-reconnection behavioral analysis, model integrity verification, anomalous local resource access.

B.11 Human-in-the-Loop Agents

Primary risks: Approval fatigue, context manipulation, approval bypass. **Core controls:** Approval request detail requirements (full action description, not summarized), approval timeout and escalation, randomized verification challenges, audit of approval-to-action consistency. **Monitoring:** Approval rate anomalies (unusually high approval rates may indicate fatigue), action divergence from approved parameters.

Appendix C: CSA Agent Security Architecture (ASA)

C.1 Purpose

The CSA Agent Security Architecture (ASA) defines the minimum viable layered model required to secure agentic systems in enterprise environments. It serves as a reference architecture for implementing the Identify-Classify-Control-Monitor-Assure operational model described in this paper, and aligns with the CSA AI Controls Matrix (AICM) and Zero Trust principles. The ASA comprises six layers, each addressing a distinct governance function.

C.2 Reference Architecture

Layer 1 — Identity. Agent identity (non-human identity distinct from service accounts), delegation chain tracking from agent action back to human principal, credential lifecycle management including automatic rotation and revocation.

Layer 2 — Policy. Task-level authorization (what is the agent currently permitted to do?), tool-level permissions (which tools can it invoke, with what parameters?), context-aware access controls that adjust based on data sensitivity and autonomy level.

Layer 3 — Guardrails. Input validation and prompt injection detection, output filtering for PII and sensitive data, structured action enforcement (tool invocations validated against schema before execution), policy engine operating out-of-process from the agent runtime.

Layer 4 — Execution. Sandboxed runtime (microVM, hardened container, or gVisor depending on threat model), resource limits (maximum tokens, tool calls, wall-clock time, and cost per task), network egress controls restricting agent communication to approved endpoints.

Layer 5 — Monitoring. Behavioral telemetry capturing prompts, reasoning traces, tool invocations, and outcomes, drift detection comparing current behavior against established baselines, anomaly detection across the agent population for correlated deviations.

Layer 6 — Audit. Full traceability from agent action back through reasoning steps to originating instruction and human principal, immutable logging suitable for regulatory examination and incident forensics, retention and lifecycle management aligned to organizational and regulatory requirements.

Appendix D: Agent Security Failure Scenario

D.1 Example: Zero-Click Prompt Injection Leading to Data Exfiltration

The following scenario illustrates how a prompt injection attack propagates through an enterprise agent's trust boundaries, based on the EchoLeak vulnerability pattern (CVE-2025-32711) [18]:

1. A malicious email is received by an employee. The email body contains hidden instructions (prompt injection payload) invisible to a human reader.
2. An embedded copilot agent summarizes the email as part of its routine operation.
3. The injected instructions enter the agent's context window during summarization.
4. Following the injected instructions, the agent retrieves sensitive internal data from connected enterprise systems (document stores, collaboration platforms, file shares).
5. The agent transmits the retrieved data via a trusted channel (e.g., a link to an attacker-controlled domain embedded in a formatted response), bypassing network-level controls that do not inspect agent-generated traffic.

D.2 Key Observations

No explicit user action was required — this is a zero-click attack initiated by the arrival of an email. The attack propagated across trust boundaries: the email (untrusted content) influenced the agent's behavior within the enterprise environment (trusted context). The agent acted with its legitimate permissions throughout — no privilege escalation was needed because the agent already had access to the data it exfiltrated. Traditional email security controls (spam filters, malware scanning) did not detect the threat because the payload was a natural language instruction, not executable code.

Appendix E: Agent Security Maturity Model

E.1 Purpose

Organizations require a structured way to assess their readiness to govern agentic AI systems. The following maturity model defines five levels of agent security governance, progressing from visibility through control to continuous assurance.

E.2 Maturity Levels

Level	Name	Description	Key Capabilities
0	Unaware	No visibility into agents operating in the environment	None — agents may be deployed without security team knowledge
1	Inventory	Agents identified and cataloged using the classification framework (Appendix A)	Agent registry, basic classification, ownership assignment
2	Basic Controls	Foundational safeguards applied: prompt filtering, credential management, basic access controls	Input/output guardrails, credential rotation, tool allowlisting
3	Type-Specific Governance	Controls mapped to agent types per the playbooks (Appendix B), with type-specific policies	Agent-type-specific policies, risk-calibrated controls, approval workflows
4	Runtime Monitoring	Behavioral monitoring, anomaly detection, and drift analysis operational across agent population	Behavioral telemetry, baseline comparison, automated alerting
5	Continuous Assurance	Automated evaluation, certification, and periodic revalidation integrated into agent lifecycle	Pre-deployment certification, continuous monitoring, triggered re-evaluation

E.3 Progression Path

Agent security maturity progresses from visibility (knowing what agents exist) through control (enforcing appropriate governance) to assurance (verifying governance continuously). Organizations should assess their current level using the Agent Identification Framework (Appendix A) and the Minimum Viable Agent Security Stack (Appendix C) as reference points, then target one-level advancement per planning cycle.

References

[1] Gartner. "By Year-End 2026, 40% of Enterprise Applications Will Integrate Task-Specific AI Agents." March 2025.

[2] Cloud Security Alliance. "State of Non-Human Identity and AI Security Survey." August–September 2025; Cloud Security Alliance. "Securing Autonomous AI Agents Survey." 2025.

[3] Anthropic. "Model Context Protocol Specification." November 2024; JFrog. "CVE-2025-6514: Pre-Authentication RCE in mcp-remote." 2025.

[4] Google. "Agent-to-Agent (A2A) Protocol." April 2025.

[5] Visa. "Trusted Agent Protocol." October 2025; Mastercard. "Agent Pay." April 2025; Google. "Agent Payments Protocol AP2." September 2025; Stripe. "Machine Payments Protocol." March 2026.

[6] Cloud Security Alliance AI Safety Initiative. "Securing the Agentic Control Plane: A Comprehensive Framework for Governing Autonomous AI Systems." March 2026; Cloud Security Alliance. "AI Controls Matrix (AICM) v1.0." July 2025.

[7] SEC. "In the Matter of Knight Capital Americas LLC: Administrative Proceeding." October 16, 2013.

[8] Invariant Labs. "MCP Security Research: Tool Poisoning and Rug-Pull Attacks." 2025; Practical DevSecOps. "MCP Security Analysis 2026." 2026.

[9] Yao, S., et al. "ReAct: Synergizing Reasoning and Acting in Language Models." ICLR 2023.

[10] Wang, L., et al. "Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning." ACL 2023.

[11] OWASP. "Top 10 for Agentic Applications (2026)." December 10, 2025.

[12] Koi Security. "ClawHavoc: Supply Chain Attack on OpenClaw ClawHub." January 2026; SecurityScorecard STRIKE Team. "How Exposed OpenClaw Deployments Turn Agentic AI Into an Attack Surface." February 2026.

[13] Cloud Security Alliance. "MAESTRO: Multi-Agent Environment, Security, Threat, Risk, and Outcome Framework." February 2025.

[14] Cloud Security Alliance. "State of Cloud and AI for Financial Services: 2026 Industry Survey Report." March 2026.

[15] Chen, Z., et al. "MINJA: Memory INJection Attack Against Retrieval-Augmented Generation." NeurIPS 2025 Proceedings.

[16] Shi, Y., et al. "MemoryGraft: Persistent Compromise of LLM Agents via Poisoned Experience Retrieval." arXiv:2512.16962, December 2025.

[17] BlueRock Security / Vulnerable MCP Project. "MCP Server Vulnerability Statistics." 2026; Knostic. "MCP Security: VirusTotal Survey of 18,000 MCP Server Projects." 2026.

[18] Aim Security. "EchoLeak: CVE-2025-32711 — Zero-Click Prompt Injection in Microsoft 365 Copilot." June 2025.

[19] Rehberger, J. "The Month of AI Bugs." Embrace The Red, August 2025.

[20] Swarm Signal. "AI Agent Security 2026: Prompt Injection Prevalence." 2026.

[21] IETF. "draft-klrc-aiagent-auth-00: Agent Identity Management System (AIMS)." March 2, 2026.

[22] OpenID Foundation. "New Whitepaper Tackles AI Agent Identity Challenges." October 2025.

[23] NIST. "Announcing AI Agent Standards Initiative: Interoperable and Secure." February 17, 2026.

[24] Cisco. "State of AI Security 2026 Report." 2026.

[25] Linux Foundation. "Agentic AI Foundation — Founding Members and Projects." December 9, 2025.