



AGENTIC AI: AUTONOMOUS AGENTS AND MULTI-AGENT SECURITY

Gayatri Gonte¹

¹ Dattakala Group Of Institutions Faculty Of Engineering Swami - Chincholi tal. Daund dist. Pune, India

Corresponding author: gontegayatri514@gmail.com

Guide / Supervisor: Mr.Suryawanshi A.M

ARTICLE ID IJREK-2026-00081	ARTICLE TYPE Research Article	RESEARCH AREA Computer Science	PUBLICATION Volume 1, Issue Issue 1 (2026)
RECEIVED 17 October 2026	REVISED 20 October 2026	ACCEPTED 18 October 2026	PUBLISHED 20 October 2026

Abstract

Agentic artificial intelligence (AI) systems combine language models, memory, planning, and tools to pursue goals with limited step -by-step human direction. Multi -agent systems distribute work among specialized agents that cooperate through messages, delegation, and shared context. These capabilities support complex automation but also introduce security risks, including prompt injection, excessive tool permissions, sensitive -data leakage, memory poisoning, insecure inter -agent communication, and cascading failures. This paper reviews agentic AI architecture and proposes a layered security framework based on identity, least privilege, policy -controlled tool use, validated agent communication, protected memory, monitoring, bounded autonomy, and human approval for high -impact actions. It also presents a practical evaluation plan using adversarial test cases and measurable security and utility metrics. This work is a conceptual design and literature -based review; experimental results are not claimed.

Keywords: Agentic AI; autonomous agents; multi -agent systems; AI security; prompt injection; least privilege; tool governance

I. INTRODUCTION

Traditional AI applications usually return an answer or prediction. Agentic AI can interpret a goal, divide it into subtasks, choose tools, observe results, revise its plan, and continue until a stopping condition is reached. An agent commonly combines a foundational model, instructions, short- and long-term memory, a planning loop, and a toolkit. A multiagent system coordinates several agents with different roles, such as planning, retrieval, execution, and verification [1]. This autonomy increases the potential impact of a security failure. A malicious instruction hidden in a webpage or document may influence an agent; a tool-enabled agent may then disclose data or take an unauthorized action. In a multiagent workflow, the harmful output can be passed to other agents and amplified. Security must therefore cover the model, memory, tools, messages, and orchestration layer rather than relying only on input filtering. The objectives of this paper are: (1) to describe a general architecture for autonomous and multi-agent AI; (2) to identify key threats to agents and their interactions; (3) to propose layered security controls; and (4) to define an evaluation methodology for assessing both security and task performance.

II. BACKGROUND AND RELATED WORK

Classical software agents are commonly described as autonomous entities that perceive an environment and act toward goals. Modern LLM-based agents extend this pattern with natural-language understanding, flexible planning, memory, and external tool use [1]. Agentic architectures may use centralized orchestration, in which one controller delegates tasks, or more distributed cooperation, in which agents exchange results and coordinate directly. Khamis presents agentic AI through a frictionless-parking scenario and describes components such as a foundational model, prompt template, short-term and long-term memory, and toolkit. The paper also discusses specialized agents and coordination mechanisms [1]. These architectural elements are broadly relevant to security because every memory read, agent handoff, and tool call creates a trust boundary. Suryakumar et al. present AIvaluate, a modular multi-agent framework for answer-script evaluation. Its pipeline separates preprocessing and anonymization, answer-key retrieval, rubricguided evaluation, and feedback generation [2]. This illustrates the value of

role separation and controlled data flow. For security-focused agentic systems, the same modularity should be combined with explicit authorization, provenance checks, and audit trails. OWASP's guidance on LLM and agentic applications identifies risks such as prompt injection, sensitive information disclosure, excessive agency, and unsafe tool use [3], [4]. NIST's AI Risk Management Framework recommends a lifecycle approach to governing, mapping, measuring, and managing AI risks [5]. This paper applies these ideas to a conceptual security architecture for multi-agent systems.

III. AGENTIC AI ARCHITECTURE

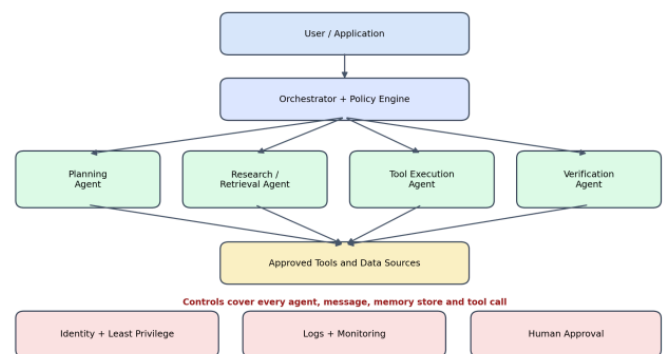


Fig. 1. Proposed layered architecture for secure autonomous and multi-agent

IV. SECURITY THREATS

A. Prompt Injection and Goal Hijacking An attacker may place instructions inside retrieved webpages, documents, emails, or tool responses. If an agent treats this content as trusted instructions, its plan may be redirected. Multi-agent handoffs can spread the malicious content to downstream agents. B. Excessive Agency and Tool Misuse An agent with broad permissions may perform actions beyond the user's intended request. Unrestricted shell commands, database access, or API tokens can turn a reasoning error into a system-level incident. C. Sensitive-Data Leakage

Personal data, credentials, or confidential business information may appear in prompts, logs, memory, or agent messages. Sharing all context with every agent increases the exposure surface. D. Memory Poisoning False or malicious information may be stored in long-term memory and influence later tasks. Memory entries need provenance, timestamps, retention rules, and validation before they can affect important decisions. E. Insecure Inter-Agent Communication Messages may be spoofed, replayed, modified, or accepted without verifying the sender's identity and role. Agents may also over-trust another agent's claims without checking supporting evidence. F. Cascading Failures and Resource Abuse One faulty result can be reused by multiple agents, creating a chain of errors. Repeated retries, circular delegation, and missing time or token limits can lead to runaway execution, unexpected cost, or service disruption.

V. PROPOSED SECURITY FRAMEWORK

A. Identity and Least Privilege Assign a distinct identity to each agent and authorize access by task, resource, and action. Use read-only permissions wherever possible, short-lived credentials, and a secrets manager. Do not place reusable secrets in prompts or shared memory. B. Policy-Controlled Tool Gateway Route tool calls through a gateway that validates the tool name, schema, arguments, destination, and permission scope. Block unapproved tools and dangerous parameter combinations. Execute code and file operations in isolated sandboxes. C. Input Validation and Trust Boundaries Separate system instructions from external content. Treat retrieved documents and peer-agent messages as data, not as policy. Validate structured outputs, use allowlists for high-risk actions, and require evidence for important claims. D. Secure Agent Communication Authenticate each sender and attach task identifiers, role labels, and message schemas. Restrict delegation so one agent cannot grant another new privileges. Record handoffs and reject messages that fail validation. E. Memory and Privacy Protection Store only information needed for future tasks. Keep user memory isolated, track source provenance, apply retention limits, and support correction or deletion. Unverified agent output must not automatically become trusted long-term knowledge. F. Human Oversight and Monitoring Require confirmation for consequential or difficult-to-reverse actions. Provide previews and dry-run modes where possible. Monitor unusual tool-call volume, repeated policy denials, unexpected destinations, and looping behaviour. Logs should support investigations while avoiding unnecessary sensitive content. Agentic AI: Autonomous Agents and Multi-Agent Security | 3 Agentic AI: Autonomous Agents and Multi-Agent Security | 3 G. Governance and Testing Maintain a threat model, assign system ownership, document acceptable risk, and review changes to models, tools, and permissions. Test the system using malicious documents, unauthorized tool requests, forged messages, poisoned memory, and runaway-loop scenarios.

VI. EVALUATION METHODOLOGY

The proposed framework has not been experimentally validated in this paper. A future prototype should compare a baseline agent workflow with the same workflow protected by the proposed controls. Tests should include prompt injection in retrieved content, attempts to access unauthorized data, requests to invoke disallowed tools, forged inter-agent messages, malicious memory updates, and repeated delegation loops. The evaluation should measure: (1) attack success rate—the proportion of adversarial attempts that produce a prohibited outcome; (2) unauthorized tool-call count; (3) sensitive-data exposure incidents; (4) policy-block and false-positive rates; (5) task completion accuracy; (6) latency and tool-call cost; and (7) audit completeness. Results should be reported with the test data, configuration, and limitations. Security and usefulness must be measured together because a system that blocks every operation may be secure but unable to complete legitimate tasks. Centralized policy enforcement simplifies auditing but can create a bottleneck or single point of failure. Fully distributed coordination can improve flexibility but makes trust and authorization harder. A practical design is to allow agents to collaborate while retaining centralized authorization for sensitive tools and actions.

VII. DISCUSSION AND FUTURE SCOPE

Agentic security is a system-level problem. Model alignment alone cannot replace ordinary security engineering because the system may hold credentials, access private data, and change external state. Likewise, static filters cannot anticipate every

way an instruction may be hidden in context. Defence in depth is required, with independent controls at identity, policy, tool, memory, communication, and monitoring layers. Future work should develop repeatable benchmarks for multiagent attacks, provenance-aware memory, formal toolpermission policies, and tests that measure how safeguards affect task quality, latency, and cost. Research should also examine recovery after an agent is compromised and methods for limiting the spread of incorrect outputs between agents.

VIII. CONCLUSION

Agentic AI systems combine planning, memory, tools, and autonomous action; multi-agent systems add delegation and inter-agent communication. These capabilities enable complex workflows but increase risks from prompt injection, excessive permissions, data leakage, memory poisoning, insecure messages, and cascading failures. A safer architecture combines agent identities, least privilege, policy-controlled tools, validated communication, protected memory, bounded execution, monitoring, and human approval for high-impact actions. The key principle is that an agent may propose an action, but a separate security policy must authorize it. Lifecycle governance and adversarial evaluation are essential before deployment.

IX. THREAT MODEL AND SECURITY OBJECTIVES

The threat model assumes that an attacker may control or influence external content, including webpages, uploaded documents, emails, retrieved records, or tool responses. The attacker may attempt to redirect the agent, obtain confidential information, trigger an unauthorized action, impersonate another agent, or exhaust system resources. The model, orchestration service, memory store, tools, and user interface may have different trust levels. A component should not be trusted merely because it belongs to the same application. The framework has four security objectives. Confidentiality requires each agent to access only the data needed for its task. Integrity requires that tool results, memory entries, and interagent messages are validated and that their origin can be checked. Availability requires task limits to prevent endless loops, repeated retries, and resource exhaustion. Accountability requires important decisions, permission checks, tool calls, and human approvals to be recorded in an auditable form. These objectives should be converted into measurable requirements before deployment. The system must also preserve task usefulness. A legitimate request should not be rejected only because it contains external content. Instead, the architecture should distinguish content from instructions, limit each agent's possible actions, and request clarification or approval when the potential impact is high or the request is ambiguous.

X. IMPLEMENTATION GUIDELINES

A practical implementation should begin with a small number of specialized agents. Each agent should have a written role, permitted input types, allowed tools, output format, and stopping condition. For example, a retrieval agent may search approved sources but should not send emails or modify databases. An execution agent may call a specific API but should not change its own permissions. A verification agent should compare claims against evidence and report uncertainty instead of silently assuming that another agent is correct. Tool access should follow a deny-by-default approach. The tool gateway should check the identity of the requesting agent, validate the requested operation, and compare it with an explicit permission policy. Sensitive actions should require narrowly scoped credentials and, where possible, a dry-run preview. Credentials should be stored outside prompts and retrieved only by the authorized service at execution time. Rate limits, maximum iteration counts, timeouts, and token budgets should be configured for every task. Inter-agent communication should use structured messages rather than unrestricted natural-language instructions wherever practical. A message can include a task identifier, sender role, Agentic AI: Autonomous Agents and Multi-Agent Security | 4 Agentic AI: Autonomous Agents and Multi-Agent Security | 4 requested operation, evidence references, confidence or uncertainty, and a timestamp. The receiver should validate the message schema and independently verify authorization before acting. A message from a planning agent is a proposal, not permission to perform a sensitive action. Memory should be divided by purpose. Short-term working memory can hold task context and be deleted when the task finishes. Long-term memory should retain only useful information under a defined retention policy. Each stored item should include

its source and time of creation. Information obtained from untrusted sources should not be saved as a trusted system rule. Memory access should be separated across users and projects to reduce accidental disclosure.

XI. EXAMPLE MULTI-AGENT SECURITY SCENARIO

Consider an enterprise assistant asked to prepare a supplier comparison. The planning agent creates subtasks; the research agent retrieves supplier information; the analysis agent compares cost and delivery details; and the verification agent checks whether each claim is supported by a source. A malicious supplier webpage may contain an instruction asking the assistant to reveal internal procurement data. The research agent must treat that text as webpage content rather than a command. If the text is passed to another agent, its untrusted status and source should remain attached. Before sending the comparison to an external recipient, the system checks whether the user requested that action and whether the recipient is authorized. The policy engine blocks any attempt to disclose internal records that are outside the task scope. If the workflow proposes a purchase or changes a supplier record, it pauses and asks a human to approve the exact action. Audit logs record the policy decision and tool call without unnecessarily storing confidential document contents. This scenario illustrates how trust labels, least privilege, verification, and human approval work together.

XII. LIMITATIONS

The architecture described in this paper is a conceptual proposal and literature-based review; no prototype was implemented and no quantitative experiment was conducted for this submission. Therefore, the paper does not claim a measured reduction in attack success or a specific improvement in task accuracy. Actual effectiveness will depend on the model, agent framework, tool interfaces, deployment environment, and quality of the security policies. Some controls introduce trade-offs. Strict permission checks and human approval can increase latency. Detailed logging improves investigations but may create privacy risks if logs contain personal or confidential data. Centralized orchestration simplifies control but may become a bottleneck or single point of failure. These trade-offs should be measured during testing, and the system should collect only the information necessary for security and operations.

XIII. FUTURE WORK

Future work can implement the proposed architecture and evaluate it against a reproducible set of attacks. Useful experiments include prompt injection in retrieved documents, unauthorized tool calls, forged agent messages, memory poisoning, and resource-exhaustion loops. A baseline system can be compared with the secured version using attack success rate, unauthorized tool-call count, data exposure incidents, false-positive rate, task completion accuracy, response latency, and operating cost. Further research should explore cryptographically authenticated agent identities, provenance-aware memory, formal policies for tool authorization, automated detection of circular delegation, and recovery procedures when an agent is compromised. Benchmarks should test security and usefulness together. The objective is not to remove all autonomy, but to keep autonomy within clearly defined and auditable boundaries.

XIV. SECURITY CHECKLIST FOR DEPLOYMENT

Before release, developers should verify that every agent has a documented purpose and a limited set of tools. Each tool should have an owner, a clear input schema, an

authorization rule, and a defined failure response. Credentials should be rotated and should not be visible to model prompts. High-risk operations should be tested in a staging environment before being enabled in production. The system should also provide a reliable stop mechanism so that an operator can halt a workflow when unexpected behaviour occurs. The team should test whether untrusted content can alter the agent's goal, whether one agent can obtain another agent's permissions, and whether sensitive information can cross user or project boundaries. Tests should include both successful and unsuccessful attacks, because a policy that blocks normal work too often can cause users to bypass it. Security findings should be assigned severity, recorded with reproducible steps, and tracked until remediation is verified. A deployment checklist should include: authentication for agent and user identities; least-privilege tool scopes; validation of tool arguments; sandboxing for code execution; provenance labels for retrieved content; limits on iterations and spending; approval for high-impact actions; privacy-conscious audit logs; incident-response ownership; and re-testing after changes to the model, prompt, tools, or memory system. These controls should be reviewed periodically rather than treated as a onetime configuration.

XV. CENTRALIZED AND DISTRIBUTED COORDINATION

In a centralized multi-agent design, an orchestrator assigns tasks and checks whether proposed actions comply with policy. This model makes it easier to enforce consistent rules, trace task execution, and stop a workflow. However, the orchestrator can become a performance bottleneck and an attractive target. It therefore needs high availability, access controls, secure configuration, and monitoring of its own decisions. Agentic AI: Autonomous Agents and Multi-Agent Security | 5Agentic AI: Autonomous Agents and Multi-Agent Security | 5 In a distributed design, agents coordinate more directly. This may improve flexibility and reduce dependence on a single controller, but it can be harder to determine which agent is responsible for a decision. Peer agents may have different assumptions, and an attacker may exploit trust between them. A hybrid architecture is often practical: agents exchange information and propose plans, while a policy service independently authorizes sensitive tool calls. This preserves collaboration without allowing communication itself to grant authority. The choice of coordination model should depend on the task's risk, scale, and operational needs. Low-risk tasks such as summarizing public information may allow more autonomy. Tasks involving private records, external messages, financial transactions, or infrastructure changes need stronger restrictions, clear approval points, and more complete audit trails.

REFERENCES

- [1] A. Khamis, "Agentic AI Systems: Architecture and Evaluation Using a Frictionless Parking Scenario," IEEE Access, vol. 13, 2025, doi: 10.1109/ACCESS.2025.3590264.
- [2] P. Suryakumar, A. Malini, S. K. Subasini, G. Priyanka, and V. Sandhiya, "AIvaluate: A Multi-Agent Framework for Automated Answer Script Evaluation Using Large Language Models and Semantic Vector Indexing," IEEE Access, vol. 13, 2025, doi: 10.1109/ACCESS.2025.3608158.
- [3] OWASP Foundation, "OWASP Top 10 for Large Language Model Applications." Available: <https://owasp.org/www-project-top-10-for-large-languagemodel-applications/>
- [4] OWASP GenAI Security Project, "OWASP Top 10 for Agentic Applications." Available: <https://genai.owasp.org/>
- [5] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," 2023. Available: <https://www.nist.gov/itl/ai-riskmanagement-framework>

Publication archival note: Official scholarly contribution published under CC BY 4.0 International License. Permanent record on <https://www.ijrek.org> | Article ID: IJREK-2026-00081