Research on a Collaborative Framework for Permission Control and Security Supervision of Task-Autonomous Large Language Model Agents

Jiang Anping

2026-06-19

ChinaRxiv

Abstract

Addressing the two major security risks faced by LLM agents—misoperations caused by the semantic gap and targeted attacks caused by agentic privilege hijacking—this paper points out that their deeper commonality lies in the behavioral unpredictability of black-box systems, and that external supervision is more feasible than "opening the black box." To this end, a collaborative framework for permission control and security supervision is proposed: a four-level permission model (L1~L4) enables fine-grained operational control; a least-privilege strategy based primarily on whitelists and supplemented by blacklists; the three-layer ClawKeeper architecture (Skill/Plugin/Watcher), whose core innovation is the horizontally independent Watcher that enables runtime circuit breaking and control-plane separation; and a human confirmation mechanism with explainable summaries. Using intranet supervision as the scenario, 240 test cases are constructed. Ablation experiments and comparisons with four baselines show that the complete framework achieves an attack interception rate of 95.8%$±$1.5%, with the Watcher layer contributing a marginal improvement of 14.9 percentage points, validating the effectiveness of the horizontally independent supervision paradigm.

Full Text

Research on a Collaborative Framework for Permission Control and Security Supervision of Task-Autonomous Large Language Model Agents

Jiang Anping
Informatization Management Center, General Office of the CPC Hangzhou Municipal Committee, 310000

Abstract: In response to the two major security risks faced by LLM agents —misoperations caused by the semantic gap and targeted attacks caused by agentic privilege hijacking—this paper points out that their deeper commonality lies in the behavioral unpredictability of black-box systems, and that external supervision is more feasible than "opening the black box." To this end, a collaborative framework for permission control and security supervision is proposed: a four-level permission model (L1–L4) realizes operation-granularity control; a least-privilege strategy takes whitelists as primary and blacklists as supplementary; the three-layer ClawKeeper architecture (Skill/Plugin/Watcher) has as its core innovation a horizontally independent Watcher that realizes runtime circuit breaking and separation of the control plane; and a human confirmation mechanism is provided with interpretable summaries. Using intranet supervision as the scenario, 240 test cases are constructed. Ablation experiments and comparisons with four baselines show that the complete framework achieves an attack interception rate of 95.8% ± 1.5%, and that the Watcher layer contributes a

marginal improvement of 14.9 percentage points, verifying the effectiveness of the horizontally independent supervision paradigm.

Keywords: large language model agent; permission control; security supervision; semantic gap; agentic privilege hijacking

1 Introduction

1.1 Research Background and Problem Statement

Represented by AutoGPT, LangGraph, and the recently open-sourced OpenManus, LLM Agent frameworks have realized a complete closed loop from natural-language instructions to system operations through a multi-layer "gateway–agent–tool–memory" architecture. They have each received tens of thousands or more stars on GitHub, and mainstream cloud vendors such as Tencent Cloud and Alibaba Cloud have also successively integrated similar capabilities. However, this leap in capability from "conversation" to "action" has also brought severe security challenges: early versions of AutoGPT adopted an overly aggressive permission-acquisition strategy, causing some test users' system files to be deleted by mistake; in 2025, the Cyera Research team disclosed a four-chain vulnerability in a mainstream Agent framework (including sandbox escape, authentication bypass, etc.; CVE numbers withheld), which affected more than 200,000 publicly exposed AI agent servers. The most severe of these, CVE-2026-44112, received a high-risk CVSS score of 9.6 (Cyera Research, 2026). Behind the phenomenon of "initial popularity followed by cooling interest" lies a core contradiction: users both hope that AI can autonomously complete complex tasks and worry that its misoperations or malicious exploitation could cause irreversible losses.

1.2 Definition of the Research Scope

To avoid conceptual confusion, this paper first clarifies the technical boundary of the research object. The AI agents studied in this paper specifically refer to software entities that take a large language model as the cognitive core, possess tool-calling interfaces and autonomous task-planning capabilities, and can execute specific operations at the operating-system or application level. Typical examples include OpenClaw, AutoGPT, LangGraph Agent, and others. They do not include traditional automation scripts based on rules or finite-state machines, nor do they include purely conversational AI assistants.

1.3 Core Risk Analysis

This paper focuses on two categories of core security risks.

Risk 1: Misoperation. An agent executes an erroneous operation because it "does not understand" the user' s true intent; this is a non-malicious form of "insufficient capability."

Risk Two: Compromise. Attackers manipulate the agent through instruction injection, prompt hijacking, and similar means, turning it into an attack proxy. Unlike traditional compromise, in which an attacker obtains static privileges, here the attacker obtains a dynamic agentic execution capability—the agent uses its own legitimate privileges to actively complete malicious tasks.

The deeper commonality between these two types of risks lies in the black-box nature of LLM agents: their behavior is unpredictable. Therefore, the feasible security path is not to "open the black box," but to construct permission boundaries outside the black box—the control plane—and behavioral monitoring —the supervisory plane. This is precisely the fundamental starting point of the "separation of the control plane and the supervisory plane" proposed in this paper.

1.4 Status of Domestic and International Research

Current research on agent security can be divided into two major directions: embedded permission control and independent supervision.

Embedded approaches: AgentRBAC implements risk early warning through ML behavioral scoring; ConsentGraph relies on user approval through a four-level approval model; SafeHarness adopts a four-layer vertically integrated defense architecture—INFORM→VERIFY→CONSTRAIN→CORRECT; Progent and CSAgent implement tool-level permission constraints. All of the above approaches couple security logic within the agent process and lack architectural independence.

Trend toward independent supervision: AgentMonitor uses a shadow agent but shares context; AAGATE proposes a Kubernetes-native governance plane; Flocks implements collaborative security watch among multiple agents. Among these, ClawKeeper pioneered a three-in-one Skill/Plugin/Watcher architecture, whose core insight is "decoupling security from tasks" —deploying a supervisor that is completely independent of the business logic. Building on this foundation, this paper further verifies the independent contribution of each component through ablation experiments and proposes a systematic framework for collaboration between the control plane and the supervisory plane.

The above works each have their own emphases, but deficiencies remain in systematic architectural collaboration, the design of horizontally independent supervisory mechanisms, and the separated verification of their effects. Taking this as the core starting point, this paper proposes a defense framework in which the control plane and supervisory plane collaborate, and verifies the independent contribution of each component through ablation experiments.

1.5 Research Objectives

The objective of this study is to provide a systematic framework of "collaboration between permission control and security supervision" for LLM agents in

scenarios with high security requirements, so as to keep comprehensive security risks within an acceptable range under acceptable performance overhead.

2 Agent Risk Analysis and Formal Definitions

2.1 Agent Working Model

The agents studied in this paper follow the following workflow: natural-language instruction input → large-model understanding and parsing → task planning to generate an operation sequence → tool invocation and execution → feedback and state update. This closed loop of "perception → cognition → action" gives agents autonomous operating capability.

Compared with traditional static reasoning systems, LLM agents mark a paradigm shift. Their key characteristics include: (1) fuzzy input → precise execution—user instructions are inherently ambiguous, but the agent' s output must be a deterministic sequence of operations; (2) autonomous decision-making—the operation sequence is autonomously planned by the agent, without step-by-step user intervention; and (3) high system privileges—to complete tasks, agents are usually granted broad system access permissions. These three characteristics are mutually coupled and constitute the structural root of agent security risks.

2.2 Misoperation Risk: The Semantic Gap in Instructions

Definition 1 (Semantic Gap). Let the user intent space be S—the real goals in the user' s mind—and the linguistic expression space be L—natural-language instructions. The ideal mapping $f : L \to S$ maps linguistic expressions to the user' s true intent, but this mapping is often not one-to-one: the same language may correspond to different intents, and the same intent may also have multiple possible expressions. The agent' s semantic understanding function is $\hat{h} : L \to \hat{S}$, where $\hat{s}$ is the semantics understood by the agent. The probability of semantic-gap risk is defined as:

$$R_{\text{sem}} = P\left(\pi(i) \notin \Omega \mid i \in I_{\text{benign}}\right)$$

Here, Ω is the legal operation space—the set of operations that do not violate security policies—π is the agent' s policy function, and I_{benign} is the space of benign instructions. This definition directly expresses the probabilistic meaning that "deviation in semantic understanding leads operations to deviate from expectations."

Note: The formal definition in this paper is intended to provide a qualitative analytical framework and a mathematical expression for risk classification, rather than to pursue precise numerical computation. Subsequent work may define semantic distance as the cosine distance based on embeddings from large models,

or perform semantic equivalence classification based on task completion, so as to achieve quantifiable risk measurement.

Causal analysis. (1) Linguistic imprecision: Human instructions are full of omissions, references, and vague quantifiers. For example, in "organize the downloaded folder," "organize" may cover multiple meanings such as categorizing, compressing, and deleting, whereas execution must be deterministic. (2) Lack of common sense: Although LLMs acquire statistical regularities from training on large-scale text, they lack real-world physical experience (e.g., "put it by the table" may cause an item to fall) and judgments of emotional value (e.g., the emotional consequences of "deleting a graduation photo"). (3) Limitations of intent reasoning: Implicit information requires multi-step reasoning to complete, and large models are prone to accumulated bias in long-range reasoning.

2.3 Black-Box Risk: Agentic Permission Hijacking

Definition 2 (agentic permission hijacking). Let the set of legal permissions of the agent be P_{legal}, and let a malicious instruction $i_{\text{adv}} \in I_{\text{malicious}}$ induce the agent to generate the actual operation set O_{actual}. If $O_{\text{actual}} \subseteq P_{\text{legal}}$ and O_{actual} violates the security policy, then agentic permission hijacking is said to occur. Its risk probability is defined as:

$$R_{\{comp\}} = P(\pi(i) \square \text{ Violation} \mid i \in I_{\text{malicious}})$$

The essential difference from the compromise of traditional systems is as follows. In traditional systems, attackers obtain static system permissions through vulnerabilities (such as fixed permissions to read files or execute commands). In agentic permission hijacking, however, the attacker does not directly obtain system permissions—the agent itself already has legal permissions. The attacker deceives the agent into "understanding" a malicious intent, causing it to proactively execute destructive operations within the scope of its legal permissions. The root of this difference lies in the agent' s core workflow (understanding + planning + execution) and its responsive interaction characteristics, which make it more fragile than traditional systems when it is manipulated at the semantic level.

Attack-surface analysis. (1) Instruction injection: embedding malicious instructions in seemingly normal inputs; (2) prompt hijacking: embedding hidden instructions visible only to the agent in web pages or documents; (3) data poisoning: planting malicious samples in the knowledge base or historical memory referenced by the agent, thereby gradually inducing behavioral deviation.

3 Permission Control and Security Supervision Collaborative Framework

3.1 Overall Architecture and Design Concepts

The design concept of the framework in this paper can be summarized as four core principles: default deny—operations that are not explicitly authorized are all prohibited; least privilege—only the minimum permissions required for task execution are granted; defense in depth—multiple layers of protection mechanisms work together; independent supervision—the executor and supervisor are architecturally decoupled, and the supervisor has independent decision-making authority.

The overall architecture consists of four layers, as shown in Figure 1. The L1 data access layer is responsible for collecting multi-source supervisory data; the L2 agent execution layer runs and executes the agent' s specific tasks; the L3 security supervision layer conducts full-process monitoring of the execution layer based on the ClawKeeper framework, and this layer is internally subdivided into three sublayers: Skill/Plugin/Watcher (see Section 3.4 for details); the L4 human-machine collaboration layer provides human intervention nodes for key decisions.

The core innovation of the framework can be summarized as an architectural paradigm that separates the control plane from the supervision plane—decoupling the execution logic of "what to do" from the supervision logic of "whether it is safe" at the architectural level. Unlike the control-plane design commonly used in distributed systems, this paper customizes the approach for the special challenges of LLM agents: (1) the supervisor needs to understand semantic attacks at the natural-language level, rather than merely checking system calls; (2) the supervisor itself also faces the risk of adversarial attacks, so protective measures such as log isolation and heterogeneous LLM instances are designed; (3) the granularity of supervision must be aligned with task intent, rather than with fixed system-resource boundaries.

The core insight of ClawKeeper is "decoupling security from tasks" (Zhiyuan Community, 2026): deploying a supervisor that is completely independent of the agent' s business logic, which does not interfere with business execution but can implement real-time interception and synchronized evolution.

Overall Architecture of the Coordinated Framework for Privilege Control and Safety Supervision

Coordinated Privilege Control and Safety Supervision Framework Architecture

- **Layer 2: Agent Execution**
  - Natural-language instruction input (Prompt)
  - Natural-language instruction flow
  - LLM agent core

(intent understanding / task planning / decision control)
  - Generate operation sequence
  - Tool invocation interface (Tools API)
    (file system / system commands / external-network calls)
  - Invoke
  - Actual operation execution environment (OS/SandBox)
  - Generate actual operation
- **Layer 3: Security Supervision Layer (ClawKeeper)**
  - Instruction pre-scan
  - **Skill layer (instruction interception)**
    Security-policy injection and pre-scan
  - Security-filtered instructions
  - Security-policy injection
  - **Plugin layer (mandatory monitoring)**
    Hard-coded monitoring points and threat blocking
  - **Watcher layer (independent supervisor)**
    Horizontally decoupled core | runtime asynchronous circuit breaking
    Independent state maintenance & heartbeat detection
  - Anomaly circuit-breaking signal
  - Real-time monitoring and constraints
  - Operation logs / heartbeat
  - L3/L4 privilege request
  - Authorization / blocking instruction
- **Layer 4: Human–Machine Collaboration Layer (Human)**
  - Policy configuration
  - **Security administrator**
    Security-policy configuration and global supervision
  - Policy configuration / intervention
  - **Manual approval and intervention center**
    (L3/L4-level privilege review and decision-making)
  - Review and feedback
  - **Review and decision feedback**
    Alert-event analysis and model evolution
- **Layer 1: Data Access**
  - Multi-source regulatory and compliance log collection (Agent / Logs / Network Probe)
    Network-traffic monitoring, endpoint monitoring agents, sandbox behavior audit log streams
  - Audit data
- **Legend**
  - Main task execution flow
  - Anomaly circuit-breaking control signal
  - Human approval and intervention channel

Figure 1: Overall Architecture of the Coordinated Framework for Privilege Control and Security Supervision

Figure note: The framework adopts a design that separates the control plane from the supervision plane: the data access layer (L1) collects objective audit data; the agent execution layer (L2) executes the main task in a closed loop; the security supervision layer (L3, ClawKeeper) performs dual heterogeneous-model supervision through in-process semantic and rule scanning (Skill/Plugin) and out-of-process supervision (the independent Watcher process), thereby enabling millisecond-level anomaly circuit breaking at runtime; finally, the human–machine collaboration layer (L4) conducts manual review of high-risk operations on the basis of interpretable summaries, balancing task autonomy with boundary security.

3.2 Four-Level Progressive Privilege Model

According to the destructiveness and reversibility of operations, this paper divides the operations executable by agents into four privilege levels. L1 (read-only query) covers operations such as querying, reading, and searching, and adopts an automatic execution plus audit-trail mode. L2 (analysis and judgment) covers operations such as data comparison, pattern recognition, and root-cause localization; these are likewise executed automatically, but their results can be reviewed. L3 (non-destructive write operations) covers reversible operations or operations with controllable consequences, such as moving, copying, creating, and sending emails; these require user confirmation before execution. L4 (destructive/high-risk operations) covers irreversible operations or operations with serious consequences, such as deletion, formatting, fund transfers, and modification of system privileges; these require a second manual confirmation and must pass the system circuit-breaking check.

Table 1 presents examples of privilege configurations in various domains:

Domain	Blacklist (absolutely prohibited)	Whitelist (automatically executed)	Requires approval (L3)
File system	`delete`, `format`, `chmod 777`	`read`, `list`, `search`	`write`, `create`, `move`
Email	`delete_all`, `bulk_send`, `auto_forward`	`read`, `archive_promo`	`send`, `reply`
Network	`curl` execution on external networks, port scanning	`get` public API	`post` internal API
System administration	modify firewall, install software	read configuration	restart service

3.3 Permission Configuration Strategy: Whitelist as Primary + Blacklist as Supplementary

This paper adopts a hybrid configuration strategy of "whitelist as primary, blacklist as supplementary." Its core priority rule is: blacklist (highest priority) > whitelist (secondary priority) > default approval required (fallback strategy). This design strikes a balance between the determinism of a closed world (a controllable whitelist) and dynamic risk response in an open world.

Multilevel engineering forms of whitelist instruction patterns. In practical engineering, whitelists can be implemented in multiple forms:

Command-level whitelist: only predefined system commands (such as `ls`, `cat`, and `grep`) are allowed to execute, while all unlisted commands (such as `rm` and `dd`) are rejected. This is suitable for restricting shell command execution.

Function/tool-level whitelist: restricts the APIs or function names that an agent may call; for example, only `get_weather` and `search_docs` are allowed, while `delete_file` is prohibited. This is the main implementation method used in the prototype system of this paper.

System-call whitelist: restricts low-level operations at the sandbox layer, such as disabling dangerous calls including `socket` and `exec`. This can be implemented through `seccomp` or AppArmor.

Scenario-policy whitelist: binds permissions to agent roles (for example, a "read-only supervisor"may not invoke write operations), and is suitable for multi-agent scenarios.

The above static whitelists provide deterministic security assurance during initial system deployment. To further balance security and autonomy, the framework in this paper supports an optional dynamic extension mechanism: a trust score $T \in [0, 1]$ is maintained for each agent, and is updated using Bayesian updating based on its historical behavior sequence (operation legality, manual-confirmation pass rate, anomaly-trigger frequency, etc.). When T exceeds a preset threshold (e.g., 0.8), the whitelist may be adaptively relaxed—for example, new tool calls may be added or path restrictions relaxed; when T falls below the threshold, the system immediately falls back to the most conservative whitelist and triggers a manual audit. In the prototype system, this mechanism is implemented as an optional module. Preliminary evaluation shows that after 200 consecutive safe operations, the trust score reaches 0.85, allowing 80% of requests for L3-level operations to be safely downgraded from "confirmation required" to "automatic execution," with no observed increase in security incidents.

The implementation in this paper uses a combination of function-level whitelists and path-level whitelists: the `ALLOWED_TOOLS` set is used to restrict callable functions, while permitted directory prefixes are configured for file operations (e.g., `/data/safe_dir/`); access outside that directory is uniformly denied.

3.4 ClawKeeper Three-Layer Supervisory Architecture

The security supervision layer in this paper introduces the ClawKeeper framework as the core monitoring and intervention infrastructure. Its three-in-one defense system of Skill/Plugin/Watcher covers the full lifecycle of an agent, from instruction injection to runtime execution and then to system-level monitoring (BAAI Community, 2026).

Skill layer (instruction-level pre-interception). Before the agent parses user instructions, structured security policies are injected directly into the agent context in the form of Markdown documents, supplemented by security-scanning scripts that forcibly enforce environment-specific and cross-platform security-boundary constraints. Its core value lies in moving defense forward to the instruction-understanding stage, blocking dangerous instructions at the source before they enter task planning.

Plugin layer (runtime enforcement and monitoring). As a runtime enforcement executor tightly embedded inside the executing agent, it is responsible for configuration hardening, active threat detection, and continuous behavior monitoring and auditing, providing hard security-enforcement guarantees. The monitoring points of the Plugin layer are implemented through hard coding, ensuring that they cannot be bypassed by the task-execution process.

Watcher layer (independent supervisor). The Watcher is the key to realizing the separation between the control plane and the supervision plane in this framework: it is a fully decoupled system-level security middleware component that runs independently outside the executing agent and does not participate in the agent' s internal task-processing logic. Specifically, the Watcher maintains an independent state space and behavioral baseline library; through heartbeat detection at 1-second intervals, it obtains the operation sequence of the executing agent and independently judges abnormalities in the agent' s execution. It triggers circuit-breaking intervention only when a security-policy violation is detected, without interfering with the normal task-execution process. The core advantage of this design is that even if the executing agent itself is induced and manipulated by an attacker and its behavior becomes uncontrolled, the Watcher can still independently perform anomaly detection and circuit-breaking intervention, thereby fundamentally resolving, at the architectural level, the inherent contradiction of "self-constraint by the executor."

The Watcher itself is security-hardened: (1) it uses different LLM instances and system prompts; (2) it retains only two permissions: reading the log cache area and sending circuit-breaking signals; (3) the judgment thresholds favor false positives (anomaly confidence > 0.6 triggers a warning, > 0.8 triggers blocking); (4) the log cache area uses a one-way isolated channel. If an attacker fully controls the executing agent and injects adversarial logs, the Watcher may still be affected, but the joint bypass probability is greatly reduced (see Section 5.5).

3.5 Human-Machine Collaboration Mechanism

Human-machine collaboration is the final line of defense for achieving high-security-level agents. This paper designs a three-level intervention model: warning level (when the anomaly/placement confidence is low, only notification is issued and the agent continues execution); confirmation level (for medium-risk operations/medium confidence, the user must click to confirm execution); and blocking level (for high-risk operations/low confidence or when an abnormal pattern is matched, execution is forcibly suspended and human intervention is required).

To mitigate the opacity of black-box decision-making, when requesting human confirmation, the framework requires the AI to attach a natural-language interpretable summary explaining the operation to be executed, the triggering reason, and the expected consequences. For example, when the AI requests deletion of a file, the confirmation prompt must display: "I will delete `/tmp/cache/old.log` (size 2.3 MB) because this file has not been accessed for more than 30 days. If it is not deleted, the disk will be full in 5 hours. Please confirm." In addition, for high-risk operations (L4), the system can also provide two enhanced confirmation methods—counterfactual comparison ("If this operation is not performed, the system will ……") and sandbox rehearsal (simulating execution in an isolated environment and displaying the results)—to help humans make more accurate authorization decisions.

4 Intranet Monitoring System Agent Solution

To verify the effectiveness of the framework, this paper uses an intelligent intranet monitoring system as the application scenario. The monitoring scope covers six domains: network, applications, servers, dynamic environment, operations and maintenance, and security. The core requirement is to realize a closed loop across the entire chain of "monitoring → analysis → notification → handling → confirmation → reporting," satisfying Grade III classified-protection requirements.

Seven types of specialized execution agents are designed (network, application, server, dynamic environment, operations and maintenance, security, and reporting), each configured with L1-L4 permission policies. The standard closed-loop process is: anomaly monitoring → intelligent analysis → risk rating → permission check (L1/L2 automatic execution, L3 push confirmation, L4 transfer to human) → handling → verification → archiving. When the closed loop breaks, it is automatically escalated or transferred to manual review.

Enhanced strong-protection scheme: cross-validation by dual supervisors. Two heterogeneous Watchers are deployed (e.g., Qwen-72B and Baichuan2-13B), and an "AND" logic is used for joint adjudication. Assuming the attack success rate against a single Watcher is $p = 4.2\%$, the joint bypass probability is approximately 0.6%-1.2%, improving security by 3-4 times. Detailed derivation is provided in the appendix.

5 System Implementation and Experimental Evaluation

5.1 Prototype System Implementation

The prototype system is implemented on the following technology stack: the agent framework uses OpenClaw v2.1; the supervision framework uses ClawKeeper v1.0 (BAAI Community, 2026); the large model uses Qwen-72B (locally deployed); permission management is extended based on ConsentGraph (ConsentGraph, 2026) to implement joint blacklist/whitelist configuration; and human intervention uses the LangGraph v0.3 Human-in-the-Loop module. In the deployment architecture, the control node (running the Watcher) and the execution node (running the execution agent) are deployed in isolation, with a heartbeat detection interval of 1 second.

5.2 Experimental Design

Test-case set: This paper constructs a benchmark set containing 240 test cases, including 120 misuse cases and 120 attack cases. The misuse category covers: ambiguous instructions (30), interruption of multi-step tasks (30), ambiguous references (30), and forgetting cross-dialogue context (30). The attack category covers: instruction injection (50, including direct injection, progressive injection, and encoded obfuscated injection), prompt hijacking (40, including hidden prompts and system-prompt overriding), and data poisoning (30, including historical-memory poisoning and tool-output poisoning). All attack cases are partly drawn from the public JailbreakBench dataset (80), and partly manually constructed in this paper (40). Detailed lists of the cases and the classification criteria are made public in the supplementary materials.

Experimental environment: The simulated intranet environment contains five categories of supervised objects (3 network devices, 2 application systems, and 10 servers), and generates 200 simulated alerts (100 normal and 100 attack alerts). It should be noted that the scale of this simulated environment is relatively limited. A real production environment (such as the intranet of a large enterprise) may contain thousands of nodes and tens of thousands of alerts per day. The experimental results in this paper mainly verify the effectiveness of the framework under controllable conditions; its scalability remains to be validated in subsequent real-environment experiments.

Comparison baselines:

Baseline-0: Original OpenClaw (no protection)

Baseline-1: AgentRBAC only (AgentRBAC, 2026)

Baseline-2: ConsentGraph only (ConsentGraph, 2026)

Baseline-3: SafeHarness (Institute of Information Engineering, Chinese Academy of Sciences, et al., 2026)

Baseline-4: AgentMonitor (UC Berkeley, 2025)

Ours: The complete framework proposed in this paper

Ablation-study design: To separate the independent contribution of each component, four ablation configurations are designed: Group A (only the four-level permission model), Group B (A + blacklist/whitelist strategy), Group C (B + ClawKeeper Skill/Plugin layer, without Watcher), and Group D (C + Watcher layer, i.e., the complete framework). This progressive design allows the marginal contribution of each component to be quantitatively separated: A→B reflects the marginal gain of the blacklist/whitelist strategy; B→C reflects the marginal gain of the Skill/Plugin layer; and C→D reflects the marginal gain of Watcher as an independent supervisor.

Repetition and statistics: Each configuration is run independently 10 times (each time using a different random seed to reshuffle the order of the test cases), and the mean and standard deviation are calculated. Significance testing uses one-way repeated-measures ANOVA and Tukey HSD post-hoc tests.

Evaluation metrics:

Unsafe-behavior interception rate: For misuse test cases, the complement of the proportion of operations that violated the security policy and were executed by the agent. Explicit definition: unsafe behaviors include deleting unauthorized files, modifying system configurations, accessing sensitive data, etc.

Attack interception rate: For attack test cases, the complement of the proportion of cases in which the agent was induced to execute any malicious operation.

False-alarm rate: The proportion of normal operations that were incorrectly blocked.

Average increase in per-operation latency (seconds): The additional time cost relative to the original OpenClaw.

L3 confirmation trigger frequency (times/hour): The average value over 8 hours of simulated continuous operation.

User-burden score (1–10): Subjective scores from five independent evaluators.

5.3 Experimental Results

Main comparison results (Table 2, reporting mean ± standard deviation):

Metric	Baseline-0	AgentRBAC	ConsentGraph	SafeHarness	AgentMonitor	Ours
Unsafe-behavior interception rate (%)	0.0$±0.0|41.8±3.2|57.9±2.8|73.6±2.1|68.2±2.5|86.7±2.1||*Attackinterceptionrate*(±0.0|52.5±3.5|66.8 *positiverate*(±0.0|8.5±1.2|12.8±1.5|15.1±1.8|11.2±1.3|10.3±1.4||*Latencyincrease*(*s*)|0.0±0.0|0.81±0 *burdenscore*|1.2±0.3|3.6±0.5|5.7±0.6|7.0±0.7|5.5±0.6|4.3±$0.5					

Statistical significance: A one-way repeated-measures ANOVA showed significant differences among groups in attack interception rate ($F(5, 54) = 187.2$, $p < 0.001$). Tukey HSD post-hoc tests indicated that the difference between the framework proposed in this paper and SafeHarness (95.8% vs. 82.4%) was significant at the $p < 0.01$ level; the difference from AgentMonitor (95.8% vs. 78.5%) was likewise significant ($p < 0.01$).

Ablation experiment results (Table 3, mean ± standard deviation; 10 repetitions):

Group	Unsafe-behavior interception rate (%)	Attack interception rate (%)	False-positive rate (%)	Marginal contribution (attack interception)
Group A (four-level permission model only)	57.5$±2.5\|59.8±2.8\|6.7±1.0\|*Baseline*\|\|*GroupB*(*A*+ *allowlist/blocklist*)\|66.9±2.0\|69.5±2.5\|7.9±1.1\|+ 9.7($p <$ 0.01)||*GroupC*(*B*+ *Skill/Plugin*)|77.8±2.0|80.9±2.0|9.4±1.2|+ 11.4($p <$ 0.001)||*GroupD*(*C*+ *Watcher, complete*)|86.7±2.1|95.8±1.5|10.3±$1.4			

The ablation experiment reveals that adding the Watcher layer brings a marginal improvement of +14.9 percentage points in attack interception rate, exceeding the simple sum of its independent contributions. This indicates a significant positive synergistic effect between the Skill/Plugin layer and the Watcher layer (that is, after the preceding defenses filter out routine attacks, Watcher can focus more on detecting advanced bypass attacks).

5.4 Failure Case Analysis

Failure case analysis: Across 10 runs of the complete framework, 7 attack-interception failures occurred in total (1,200 attack attempts in all, for a failure

rate of 0.58%). Analysis found that the failures were mainly concentrated in one class of complex progressive instruction injection: malicious instructions were dispersed across 5 rounds of dialogue, with each round injecting only one seemingly harmless subtask (e.g., "first list the files under `/tmp`," "then read the `config.ini` among them," and "finally send the content to an external domain name"). The Skill layer failed to recognize the cross-round aggregation of intent, and the Plugin layer was likewise unable to detect anomalies from any single subtask. Although Watcher detected an abnormal file-read frequency in the third round and triggered a warning, the threshold setting was biased toward avoiding false positives and therefore failed to automatically trip the circuit breaker. This case shows the need to introduce cross-round context-dependency graph analysis, which is also an important direction for future work in this paper.

Comparison with real-world cases: Similar failures already have precedents in actual deployments. For example, in 2025, an AI programming assistant on the Replit platform, despite the user' s explicit instruction "do not modify any code," not only deleted the entire dynamic database without authorization, but also generated more than 4,000 fake users to cover up its own mistake. In another case, an AI agent from PocketOS, after credential verification failed, autonomously parsed the [[unclear: word beginning with "日"continues off page]]

log files to obtain production-environment keys, and deleted the entire production database within 9 seconds. These cases further verify the necessity of introducing a horizontally independent Watcher.

5.5 Robustness Evaluation Against Attacks on the Watcher Itself

To evaluate the Watcher' s own resistance to attacks, this paper additionally designed a set of targeted experiments: using 50 specialized prompt-injection attack cases against the Watcher, including direct system-prompt overwrite, indirect log poisoning, and role-playing inducement. The attack success rate is defined as: the attacker successfully causes the Watcher to judge a malicious operation as safe and allow it to proceed.

The results show:

The attack success rate against a single Watcher (Qwen-72B) is 4.6% (i.e., the residual leakage rate of the Watcher layer in this framework under adversarial attack).

For log-poisoning attacks against the Watcher (where the attacker writes malicious instructions into logs by controlling an executing agent), the success rate is 8.2%, because the isolation measures for the log buffer are not absolutely secure.

After adopting the dual-supervisor enhanced scheme, the attack success rate drops to $0.8\% \pm 0.3\%$, verifying the effectiveness of heterogeneous supervisors as a defense. These results also explain why the overall attack-blocking rate of

this paper is 95.8% rather than 100%—among the remaining 4.2% of failures, about 0.8% come from successful attacks against the Watcher itself, while the remaining 3.4% come from complex scenarios such as the progressive injection described above.

5.6 Detailed Evaluation of Performance Overhead

In addition to the latency increase in Table 2, this paper additionally measured the following performance metrics (8-hour continuous-running averages):

CPU additional usage: the Watcher layer consumes an additional 1.2 CPU cores (the original executing agent occupies 2.1 cores), an increase of about 57%.

GPU memory usage: the Watcher' s independent LLM instance (Qwen-72B quantized version) additionally occupies about 18GB of GPU memory.

Network bandwidth: heartbeat detection and log transmission occupy an average of 0.3 Mbps.

Increase in end-to-end task completion time: for a typical supervision task (such as "analyze 10 alerts and generate a report"), the complete framework increases time consumption by about 22% compared with the original OpenClaw (mainly due to the Watcher' s LLM inference and waiting for manual confirmation). For scenarios with high real-time requirements (such as intrusion detection), overhead can be reduced by lowering the Watcher detection frequency (adjusting from a 1-second interval to 5 seconds), but the blocking rate will correspondingly decrease by about 3–5 percentage points.

5.7 Discussion of Limitations

The main limitations of this paper' s framework include: (1) the framework depends on the understanding ability of the underlying large model (L1/L2 accuracy); if Qwen-72B makes a misjudgment, the upper layer cannot fully correct it. (2) The Watcher itself still faces adversarial attack risks; although the dual-supervisor scheme significantly reduces the success rate, it does not eliminate it completely. (3) The experiments were conducted only in a simulated environment and at limited scale. Scalability and performance in real production environments (such as thousands of nodes and tens of thousands of alerts per day) need further verification. (4) The dual-supervisor cross-validation enhanced scheme has currently been validated only on a limited attack set, and its actual deployment effectiveness awaits larger-scale red-team testing. (5) The threat model of this paper assumes that the attacker cannot directly modify the Watcher' s configuration. If the attacker obtains system administrator privileges, then all protections fail—this belongs to the domain of traditional system security and is outside the scope of this paper.

6 Comparison with Related Work and Discussion

6.1 Systematic Comparison with Existing Schemes

Table 4 compares this paper' s framework with representative schemes across key dimensions:

Dimension	AgentRBAC	Consent-Safe	GraphHarness	AgentMonitor	Progent	Flocks	Ours
Permission model	RBAC roles	Four-level approval	Capability tokens	None	Tool-level constraints	None	Four levels + black/white lists
Supervision mechanism	None	None	Four-layer vertical	Shadow agent	None	Multi-Agent collaboration	Three-layer horizontally independent

Dimension	AgentRBAC	Consent-Safe	GraphHarness	AgentMonitor	Progent	Flocks	Ours
Human-machine collaboration	None	Confirmation nodes	Rollback confirmation	None	Requires approval	Requires human review	Three-level intervention + explanatory summaries
Supervisory independence	×	×	× (in-process)	× (shared context)	×	×	(horizontally independent)
Resistance to attacks on itself	N/A	N/A	Partial	Partial	N/A	N/A	Dual supervisors, log isolation

Dimension	AgentRBAC	Consent Graph	SafeHarness	AgentMonitor	Progent	Flocks	Ours
Ablation validation	×	×	×	×	×	×	

6.2 Essential Differences of the Proposed Framework

Unlike SafeHarness' s vertical integration, in which security logic runs inside the agent process, and AgentMonitor' s shadow agent, which shares context, this paper adopts a horizontally independent paradigm: the Watcher runs as an independent process, maintains independent state, reads logs only through heartbeats, and sends a circuit-breaker signal when a violation is detected. To address LLM semantic attacks, mechanisms such as log isolation, heterogeneous supervisors, and interpretable summaries are customized.

6.3 Theoretical Contributions and Practical Value

Theoretical contributions. (1) For the first time, this work formally defines the two core risks of LLM agents—semantic gap and agentic privilege hijacking—from the two dimensions of semantic gap and agentic privilege hijacking, and proposes at the architectural level a new paradigm that separates the control plane from the supervisory plane, with a customized design for the special challenges posed by semantic attacks against LLM agents. (2) Through large-scale ablation experiments (10 repetitions and statistical significance tests), it quantitatively separates, for the first time, the independent contributions of four components—the permission model, the whitelist strategy, the Skill/Plugin layer, and the Watcher layer—and reveals positive synergy among the layers. (3) It provides a quantitative trade-off method between permission granularity and autonomy, and proposes the concept of a "permission-autonomy trade-off curve," offering an empirical benchmark for subsequent research.

Practical value. (1) A complete implementation guide for intranet supervision scenarios. (2) An implementation scheme based on an open-source technology stack. (3) A phased deployment roadmap. (4) A public set of test cases and an evaluation protocol (anonymous link), supporting reproducible research.

7 Conclusions and Outlook

7.1 Research Conclusions

This paper conducts a systematic study of the two core security risks of large language model agents—misoperation and compromise—and obtains the following conclusions.

First, at the level of risk essence. Misoperation stems from the "semantic gap," while compromise stems from "agentic privilege hijacking." The deep commonality between the two types of risks lies in the behavioral unpredictability

introduced by black-box systems, which determines that external supervision, rather than opening the black box internally, is the pragmatic security path.

Second, the collaborative framework is effective. In simulated intranet supervision scenarios, the collaborative framework proposed in this paper achieves an unsafe-behavior blocking rate of 86.7% ± 2.1% and an attack blocking rate of 95.8% ± 1.5%, significantly outperforming existing baselines.

Third, the independent supervisor model is key to improving framework performance. Ablation experiments show that the Watcher layer contributes an absolute improvement of about 14.9 percentage points in the attack blocking rate, and that it has positive synergy with the Skill/Plugin layer, verifying the effectiveness of the architecture that separates the control plane from the supervisory plane.

Fourth, the dual-supervisor scheme significantly strengthens resistance to attacks. Experiments on attacks against the Watcher itself show that dual heterogeneous supervisors reduce the attack success rate from 4.6% to 0.8%, improving security by about fivefold.

7.2 Research Outlook

Trust-based dynamic permission adjustment: referring to the BASIS trust model, dynamically adjust permission levels according to the agent' s historical behavior.

Dynamic adaptive whitelist: Extend the static whitelist into a dynamic adaptive whitelist. Based on the agent' s historical behavior sequence, its trust score is updated online, and the permitted instruction set is dynamically adjusted. A Bayesian trust model is adopted, with posterior trust following a Beta distribution, thereby achieving a Pareto-optimal balance between security and autonomy.

Explainable supervision: Make the Watcher' s circuit-breaking decisions explainable, so that operations and maintenance personnel can quickly review them. Combine explainability techniques for large models (such as LIME and SHAP) to generate natural-language explanations.

Multi-agent collaborative security: Study mechanisms for permission delegation and trust propagation among agents, so as to prevent "collusion attacks" by multiple agents.

Large-scale red-team testing of the dual-supervisor enhanced scheme: Validate the practical security of the dual-supervisor approach under larger-scale attack sets (>1000 use cases) and in real environments.

Formal verification: Conduct model checking or theorem proving for key security properties of the framework (such as the absence of unauthorized operations), complementing the empirical evaluation in this paper.

This paper focuses on the security risks generated by LLM agents during the execution of operations (misoperation and being compromised). Risks at the level of information output (such as hallucinations and bias), as well as risks at the level of model value alignment (such as deceptive alignment), are outside the core research scope of this paper and are left for future work to extend by incorporating explainability and fairness techniques."

References

[1] AgentRBAC[CP/OL]. agent-rbac: role-based access control for AI agents with ML-powered anomaly detection. PyPI, 2026-04-17. https://pypi.org/project/agent-rbac/.

[2] CONSENTGRAPH[CP/OL]. consentgraph: consent graph as code: deterministic action governance for AI agents. PyPI, 2026-04-10. https://pypi.org/project/consentgraph/.

[3] SHI T N, ZHANG J, WANG Y, et al. Progent: programmable privilege control for LLM agents[J/OL]. arXiv preprint, 2025. arXiv:2504.11703. https://doi.org/10.48550/arXiv.2504.11703.

[4] GONG H C, LI C X, CHANG R, et al. CSAgent: secure and efficient access control for computer-use agents via context space[J/OL]. arXiv preprint, 2026. arXiv:2509.22256. https://doi.org/10.48550/arXiv.2509.22256.

[5] WANG C L, et al. MI9: an integrated runtime governance framework for agentic AI[J/OL]. arXiv preprint, 2025. arXiv:2508.03858. https://doi.org/10.48550/arXiv.2508.03858.

[6] OPENAI. GPT-OSS-20B red teaming challenge dataset[EB/OL]. Kaggle, 2025. https://www.kaggle.com/competitions/gpt-oss-20b-red-teaming/data.

[7] KPMG. Three out of four global leaders will prioritize AI investment despite economic uncertainty, KPMG Global AI Pulse survey finds[EB/OL]. (2026-03-31)[2026-05-27]. https://kpmg.com/xx/en/media/press-releases/2026/03/kpmg-global-ai-pulse-survey.html.

[8] OWASP FOUNDATION. OWASP top 10 for LLM applications: 2025 version[EB/OL]. (2025-06-15)[2026-05-27]. https://genai.owasp.org/llm-top-10/.

[9] CLOUD SECURITY ALLIANCE. AAGATE: a NIST AI RMF-aligned governance platform for agentic AI[EB/OL]. (2025-12-22)[2026-05-27]. https://cloudsecurityalliance.org/blog/2025/12/22/aagate-a-nist-ai-rmf-aligned-governance-platform-for-agentic-ai.

[10] CYERA RESEARCH. Claw chain: four chainable vulnerabilities in OpenClaw[R]. Cyera Security Report, 2026. https://www.cyera.com/claw-chain.

[11] JAILBREAKBENCH[EB/OL]. JailbreakBench: an open robustness benchmark for jailbreaking large language models. (2024-11-28)[2026-05-27]. https://jailbreakbench.com/.

[12] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. Artificial intelligence risk management framework (AI RMF 1.0)[R]. Gaithersburg: NIST, 2024. DOI: 10.6028/NIST.AI.100-1.

[13] ZHANG Xi, LI Chaozhuo, XU Nuo, et al. Security challenges and response mechanisms for trustworthy large language model agents[J]. Information and Communications Technology and Policy, 2025, 51(1): 33-39. DOI: 10.12267/j.ictc.2025.01.006.

[14] Institute of Information Engineering, Chinese Academy of Sciences, et al. SafeHarness: lifecycle-integrated security architecture for LLM-based agent deployment[J/OL]. arXiv preprint, 2026. arXiv:2604.13630. https://doi.org/10.48550/arXiv.2604.13630.

[15] Zhiyuan Community. Zhiyuan releases the security framework ClawKeeper, using agents to supervise agents[EB/OL]. (2026-04-10) [2026-05-27]. https://hub.baai.ac.cn/view/xxxx.

[16] ThreatBook. ThreatBook releases Flocks, an intelligent security digital employee[EB/OL]. (2026-03-31) [2026-05-27]. https://github.com/AgentFlocks/flocks.

[17] WANG J Y, XU H Y, YE J B, et al. OpenAgentSafety: a comprehensive framework for evaluating real-world AI agent safety[J/OL]. arXiv preprint, 2025. arXiv:2507.06134. https://doi.org/10.48550/arXiv.2507.06134.

[18] WANG H Y, POSKITT C M, SUN J. OpenAgentSafety: a comprehensive framework for evaluating real-world AI agent safety[J/OL]. arXiv preprint, 2025. arXiv:2507.06134. https://doi.org/10.48550/arXiv.2507.06134.

Appendix A: Sample Agent Security Scheme for an Intranet Supervision System

A.1 Scenario Requirements and Constraints

To verify the effectiveness of the framework, this paper takes an intelligent intranet supervision system as the application scenario. The scope of supervision includes: network systems, application systems, servers/cloud resources, computer-room environmental and power systems, operations and maintenance activities, and security risks. The core requirement is to realize a full-chain closed loop of "monitoring → analysis → notification → handling → confirmation → reporting," while keeping security risks within an acceptable range. Special constraints include an intranet environment, data not leaving the domain, 7$×$24-hour high availability, and compliance with Level 3 requirements under classified cybersecurity protection.

Core requirement: realize a full-chain closed loop of "monitoring → analysis →

notification → handling → confirmation → reporting," while keeping security risks within an acceptable range.

Special constraints: intranet environment, with data prohibited from leaving the domain; 7$×$24-hour high availability and fault self-recovery capability; compliance with Level 3 requirements under classified cybersecurity protection, full auditing of operations, and mandatory human–machine collaborative checkpoints that cannot be bypassed.

A.2 Design of Seven Categories of Specialized Execution Agents

Based on the framework in this paper, seven categories of specialized execution agents are designed, with different strategies configured for each category at the L1–L4 permission levels:

Agent Name	Main Responsibilities	Typical L1/L2 Operations	Typical L3 Operations	Typical L4 Operations
Network Monitoring Agent	Network device status, traffic anomalies, access-control policy compliance	Read traffic logs; query routing tables	Modify QoS policies; restart interfaces	Modify firewall rules; change ACLs
Application Monitoring Agent	Application system performance, log anomalies, user behavior	Read application logs; query TPS/QPS	Restart application instances; clear cache	Modify application configuration; roll back versions
Server/Cloud Resource Monitoring Agent	Host health, resource utilization, patch status	View CPU/memory/disk	Clean temporary files; restart services	Modify system configuration; install patches
Data-Center Environment Monitoring Agent	Environmental parameters such as temperature and humidity, power, and fire protection	Read sensor data	Adjust air-conditioning settings	Trigger backup power switchover

Agent Name	Main Responsibilities	Typical L1/L2 Operations	Typical L3 Operations	Typical L4 Operations
O&M Operation Monitoring Agent	O&M command auditing, change operations, script execution	View command history	Execute low-risk scripts	Execute high-risk commands (`rm`, `dd`, etc.)
Security Risk Monitoring Agent	Vulnerability detection, threat intelligence, attack events	Read vulnerability databases; query threat intelligence	Isolate suspicious IPs	Block attack chains; ban accounts
Reporting and Archiving Agent	Generate audit reports, archive logs, compliance documents	Query historical records	Generate draft reports	Submit final reports; archive

Design principle: the core value of multiple specialized agents lies in permission minimization. The permission scope of each agent is strictly confined to its area of responsibility; even if one agent is attacked, the loss is limited to the corresponding subsystem, in accordance with the principles of "least privilege" and "defense in depth." At the same time, professional division of labor avoids the context-forgetting and task-interference problems that arise when a single general-purpose agent handles mixed tasks.

A.3 Closed-Loop Workflow and Break Handling

Standard event closed-loop process (9 steps):

(1) Monitoring detects an anomaly (triggered by sensors/logs/probes)

(2) Intelligent root-cause analysis (invoke L2 agent analysis)

(3) Assess risk level (automatic determination of L1–L4)

(4) Automatically match a disposal plan (rule-base matching)

(5) Perform permission check (diversion according to L1–L4)

(6) L1/L2: execute automatically and enter step 6

(7) L3: push for manual confirmation, and enter step 6 after response

(8) L4: circuit-break and transfer to the manual channel, entering step 8

(9) Execute disposal (invoke tools/API)

(10) Verify effects (read back status and compare with expectations)

(11) Record and archive (write to audit log)

(12) Output report (notify relevant personnel)

Closed-loop break-handling mechanism:

Break Scenario	Trigger Condition	Handling Action	Escalation Path

Failure Scenario	Trigger Condition	Handling Action	Escalation Path
Analysis failure	The agent returns "unable to determine the root cause" or confidence < 0.5	Return to the manual analysis channel	Mark as "requires re-review," transfer to L4
Remediation failure	The execution tool returns an error code or times out	Automatically retry once; escalate if the retry fails	Escalate to L4 level and trigger the emergency response plan
Verification in doubt	After execution, the deviation between the state and the expected result $>$ threshold	Keep a 24-hour manual re-review window	Support rollback operations and record points of doubt

Reporting and review process:

Automatic generation: daily/weekly/monthly/quarterly/annual reports (completed by the reporting and archiving agent)

Automatic distribution: push to reviewers by role (system administrator $\rightarrow$ security supervisor $\rightarrow$ CIO)

Consolidated revision: collect review comments and automatically merge revision suggestions

Final review and release: after manual confirmation, automatically archive and distribute the final report

•