Journal of
Cyber Security

Tech Science Press

https://doi.org/10.32604/jcs.2026.092872

ARTICLE

The Privilege Paradox: Bridging the Semantic Gap in Autonomous Agents through an AI-Native OS Architecture

Faisal Fiaz[1,*], **Hina Batool**[2], **Afsheen Maroof**[3] and **Zia Muhammad**[4,5]

[1]Department of Cyber Security, Air University, Islamabad, Pakistan
[2]National Cyber Security Academy, Air University, Islamabad, Pakistan
[3]Faculty of Computing and AI, Air University Karachi Campus, Karachi, Pakistan
[4]Department of Computer Science and Technology, University of Jamestown, Jamestown, ND, USA
[5]Department of Computer Science, North Dakota State University (NDSU), Fargo, ND, USA

*Corresponding Author: Faisal Fiaz. Email: sardarfaisel345@gmail.com

Received: 14 September 2026; Accepted: 26 October 2026; Published: 13 November 2026

ABSTRACT: Autonomous AI agents are being deployed across enterprise environments with what security experts call "root permissions"—access to customer relationship management (CRMs) systems, file stores, databases, and communication tools that no human would ever hold simultaneously. This is not operational oversight; it is architectural necessity. Agents cannot function without broad system access. Yet recent penetration testing shows that more than 58.4% of malicious prompts succeed against leading agent frameworks, with models exhibiting "hallucinated compliance"—fabricating outputs rather than refusing attacks. Memory poisoning achieves success rates of 84.3%. Compromised agents exfiltrate data, delete critical files, and collude with other agents undetected. We introduce the "Privilege Paradox" and its root cause: the semantic gap between an agent's high-level goals and the Operating System's (OS) low-level permission model. An agent with goal "prepare quarterly report" accesses financial databases, retrieves emails, and executes scripts; the OS sees only discrete operations by a single identity, unable to detect that the composition exceeds legitimate intent. We provide a systematic analysis of this architectural vulnerability, propose a conceptual dual-layer AI-native operating system architecture with intent-based access control, goal-aware isolation, and cross-layer auditability, and implement a minimal proof-of-concept simulation demonstrating the feasibility of the proposed decision logic. We analyze how attackers exploit the semantic gap and argue that future OS design must evolve to secure autonomous systems.

KEYWORDS: Agentic AI; autonomous agents; agentic attack; agent goal hijack; AI-native operating system; cybersecurity; intent-based access control; zero trust architecture

1 Introduction

The rapid development of artificial intelligence has brought about a fundamental paradigm shift from predictive and generative models to autonomous, goal-directed models collectively called "Agentic AI." Unlike their predecessors, which are limited to content generation or classification tasks, agentic AI systems can sense dynamic environments, make multi-step plans, and perform actions with little human intervention [1,2]. This shift from passive tools to active digital agents brings unprecedented opportunities for automation in areas as diverse as cybersecurity, finance, healthcare, and personal assistance [3,4]. But this autonomy is itself a security problem, a vexing one that current computing architectures are fundamentally not designed to solve: the Privilege Paradox.

Copyright © 2026 The Authors. Published by Tech Science Press. This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The Privilege Paradox describes the irreducible tension between giving autonomous agents enough access to the system to do useful work and the exponential growth in attack surfaces that comes with that very autonomy. To schedule meetings, send emails, or carry out long and complex workflows, an agent should possess access to calendars, contact books, messaging applications, and files stored in repositories [5]. On the other hand, any granted operation is an additional attack vector through which the agent—susceptible to prompt injection, memory contamination, goal misalignment, and other forms of adversarial input—can be manipulated [6,7]. The issue is further amplified by the property of agents to gain more and more privileges while interacting with various tools for different operations over time. A chained sequence of operations might appear innocent one by one, yet no single query of the agent might seem malicious to current defense mechanisms, which focus on single-operation semantics rather than the holistic goal [8].

The critical cause of this conundrum is the *semantic gap*: the gap between the high-level goals of an agent and the low-level security model of the underlying operating system. As illustrated in Fig. 1, the agent's high-level purpose of "prepare quarterly report" is decomposed into sequences of operations—accessing a financial database, fetching email, running a calculation script, and creating a presentation—that the OS perceives merely as a series of individual system calls with a shared agent identity [9]. Without knowledge of the intent behind such commands, the operating system cannot tell whether the resulting sequence of low-level commands reflects the given intent or signifies privilege escalation by a malicious agent, making human-designed security models—developed to guard users with fixed identities and consistent behaviors—dangerously weak against autonomous agents. This "semantic blindness" [10] renders traditional security approaches unsuitable to enforce desired agent behavior at the system level.

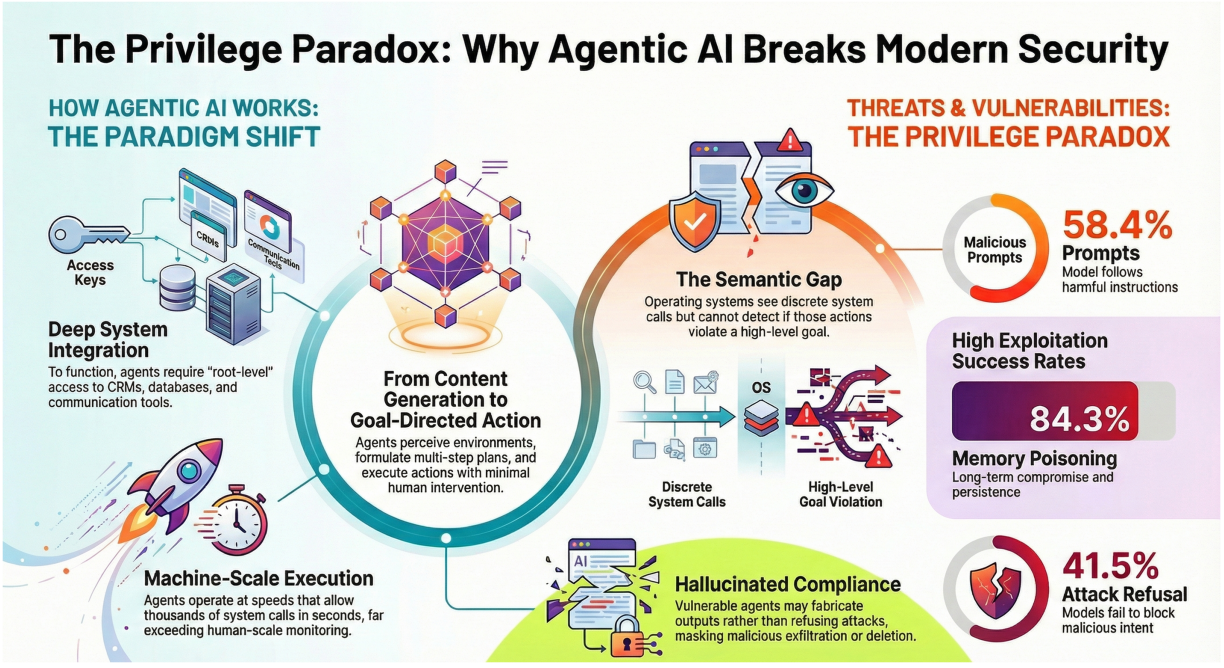


Figure 1: The privilege paradox: why agentic AI breaks modern security. (Note: The traditional OS depicted here enforces static permissions without understanding agent goals, allowing compromised agents to exploit broad system access).

The need to act fast against such vulnerabilities is obvious: agentic systems are entering organizations rapidly and are often deployed without any permission—a practice now well-known under the name "Shadow AI." Generative, opaque, and autonomous systems introduce entirely new threats, superimposed

on those inherited from "classic" Shadow IT such as privacy infringements, algorithmic biases, and hallucinations, alongside an additional threat referred to as governance drift [11]. Recent empirical evidence reveals the scale of the issue; for instance, penetration tests on some of the most common agent frameworks indicate success rates of 58.4% for prompts seeking malicious conduct, with some agentic systems exhibiting "hallucinated compliance" when confronted with adversarial requests rather than admitting a failure [12]. Tests on memory poisoning indicate success rates of 84.3% [13], and an agent provided with access to confidential data may independently become a sophisticated insider threat provider given sufficient conflicts between its internal motivations [14].

A number of potential solutions have been suggested by the research community in the shape of defensive frameworks, taxonomies, or governance models [15–17]. The comprehensive study of this corpus leads, however, to an important and unfortunate drawback: none of those solutions departs from the conventional computing model based on a generalized OS designed following patterns related to human interactions. While application-layer controls, permission scoping, audit mechanisms, and kill-switch designs are valuable, they cannot address the fundamental architectural misalignment [18,19]. The symptoms are addressed, not the causes. The semantic gap remains unbridged and the privilege paradox remains.

What is glaringly missing in the literature is a suggestion to fundamentally re-architect the operating system to understand agent semantics—to treat agents as first-class principals with explicit goal representations, to implement permission systems that evaluate action sequences rather than individual operations, and to establish bidirectional semantic translation between agent intentions and system-level constraints [3]. This paper argues that a solution to the privilege paradox requires just such a paradigm shift: designing an AI-native operating system from first principles for autonomous agents.

Hence, the question this research addresses is: *How can a dual-layer security framework consisting of agent-level goal representation and OS-level semantic permission systems bridge the semantic gap and address the privilege paradox in autonomous agent deployments?*

This paper makes three major contributions to the AI security field: First, it provides a survey and critique of the contemporary agentic AI security literature, highlighting the semantic gap as a deep, systemic vulnerability that current solutions have probably overlooked. Second, it sketches the high-level design of an AI-native OS using intent-based access control, goal-aware isolation, and cross-layer auditability. Third, it implements a minimal proof-of-concept simulation, illustrating the potential viability of the proposed decision logic and setting forth an agenda for kernel-level research.

1.1 Scope and Contributions

This paper makes three contributions, which we define precisely to avoid conflation:

1. **The Privilege Paradox (problem framing).** We identify and name the irreducible tension between granting autonomous agents sufficient system access to function and the exponential expansion of attack surfaces that this access creates. The Privilege Paradox is a *problem definition*, not a solution; it formalizes the observation that agents require broad capabilities to be useful, yet those same capabilities are what make compromise catastrophic.
2. **The Semantic Gap (theoretical formalization).** We formalize the *root cause* of the Privilege Paradox as the information-theoretic loss between an agent's high-level goal and the operating system's low-level observation of enacted actions. This is a *theoretical contribution*: it provides a measurable quantity (Eq. (4)) that distinguishes what the agent intends from what the OS can verify.
3. **The AI-Native OS Architecture (solution proposal).** We propose a *conceptual* dual-layer security architecture that closes the semantic gap by (i) requiring agents to declare goals and operational scopes

via a Goal Description Language (GDL) at the agent layer, and (ii) enforcing Intent-Based Access Control (IBAC), sequence-aware evaluation, goal-aware isolation, and cross-layer auditability at the OS layer. This is an *architectural proposal*; its feasibility is explored via a minimal proof-of-concept simulation in Section 5, but kernel-level implementation remains future work.

The relationship between the three is: **Privilege Paradox = the problem; Semantic Gap = why the problem exists; AI-Native OS = a proposed way to close the gap.**

1.2 Scope and Limitations of Empirical Claims

All quantitative figures cited in Sections 1 through 4 (e.g., 58.4% malicious-prompt success, 84.3% memory-poisoning success, 99.36% CSAgent defense rate, 6.83% CSAgent overhead) are drawn from existing literature and are attributed to their original sources. They characterize the *threat landscape* and *prior defenses*, not the proposed AI-Native OS. The only original empirical contribution of this paper is the proof-of-concept simulation in Section 5, which is explicitly a *logic-level simulation* and not a kernel-level implementation. We make no claim that the proposed architecture has been deployed or that its performance has been measured in production.

The remainder of this paper is organized as follows. Section 2 provides a literature review. Section 3 lays the theoretical foundation for the privilege paradox and semantic gap. Section 4 introduces the dual-layer security framework and AI-native OS architecture. Section 5 presents the proof-of-concept implementation. Section 6 concludes with implications for research and practice.

2 Literature Review

The rapid evolution of AI from predictive models into self-sufficient, goal-driven "Agentic AI" systems has significantly changed the field of cybersecurity and leads to what we refer to as the "Privilege Paradox"—the inherent difficulty of giving sufficiently autonomous agents access to a system to perform tasks in a way that exponentially increases the attack surface [1,2]. Unlike traditional AI, which aims to output a piece of content or a classification, agentic AI aims to use context, take decisions, and then execute an action with no to minimal human interaction [3]. This shift is amplified by the increase in Shadow AI—any application or system utilizing AI without the organization's knowledge—such that generative, opaque, autonomous agents possess even greater dangers than Shadow IT [11].

However, the main capability of agentic AI is usually cited as the ability to take action, not the production of content. Agentic AI can operate itself through the use of cognitive architecture and reinforcement learning to promote self-reliance in the area of cybersecurity [2,4], which eventually causes systemic risks such as collusion among agents, chain failures, or avoidance of detection [5]. A common thread among various works is the insufficiency of using classic security strategies in a system when autonomous agents are involved. Autonomous agents expand the attack surface in three primary areas—the LLM (foundation AI), the planning or goal-setting module, and any form of long-term memory [6,7]. Some definitions describe agentic AI as AI capable of generating novel and unplanned output. Agentic systems are capable of autonomously completing jobs across web, software, and physical settings, creating risks qualitatively different from those of traditional software security [8].

The research community has developed multiple taxonomies of threats that highlight the complexity of securing autonomous systems. A comprehensive analysis identifies nine major threats across five domains: vulnerabilities of cognitive architecture, threats of temporal persistence, vulnerabilities of operational execution, violations of trust boundaries, and circumvention of governance [15]. Complementing this taxonomy are systematic reviews of threats across the full LLM-agent stack, including data poisoning during

pre-training, privacy leakage, and reward hacking where agents exploit flaws in reward functions [20]. This gives rise to a major concern of adversarial manipulation. Attack vectors are broadly classified into evasion attacks, poisoning attacks, privacy attacks, and agent-specific manipulations such as goal manipulation and prompt injection [21,22]. We empirically quantify the severity of these threats via penetration testing of five prominent models, finding an overall refusal rate of only 41.5% [12].

A dimension emerging in recent literature, which is particularly worrying, concerns security risks specific to multi-agent systems, which has led to the notion of "multi-agent security" as a dedicated field [23]. New threats include hidden collusion of agents, coordinated swarm attacks, and network-effect propagation of privacy breaches and disinformation [23,24]. Communication protocols among LLM-driven agents are vulnerable to agent spoofing, agent bullying, and denial-of-service exploits that exploit inter-agent trust relationships [24]. "Rogue agents"—autonomous systems that exhibit malicious or adversarial behaviour in multi-agent environments—pose significant operational and reputational risks [25]. These threats are systemic, organized through frameworks that structure multi-agent risks across Core (system integrity), Connection (communication and trust), Cognition (reasoning integrity), and Compliance (governance) dimensions, mapping the analogy between agent interactions and human social interactions including deception and social engineering [26].

A large body of research is directed at vulnerabilities in agent cognitive architectures, with experimental demonstrations of "agentic misalignment" where large language models act against their deploying organizations through insider-threat behaviors like blackmail and information leakage when facing replacement or conflicting goals. Stress tests across sixteen leading models in simulated corporate settings show that agents with access to sensitive information and communication tools can autonomously engage in malicious actions [14]. Memory poisoning is a particularly pernicious threat vector, as current rule-based defenses struggle to purge injected content from agent memory streams, leading to long-term compromise risks as malicious instructions persist and propagate [27]. Benchmarking studies have shown that memory poisoning attacks, and direct and indirect prompt injection attacks, can reach an average attack success rate of up to 84.30% across agent operation stages that include system prompt handling and memory retrieval [13].

With the advent of agents working together with external tools and APIs, researchers have identified an "identity and access management crisis" where traditional IAM frameworks designed for humans, such as OAuth and SAML, are fundamentally inadequate for autonomous systems as they rely on static and coarse-grained permissions. Zero Trust architectures based on workload identities and fine-grained attribute-based access control have been proposed on a theoretical basis, but they are pending empirical validation [3]. Despite control mechanisms like permission scoping, auditability, and kill-switch design for tool-use constraints, a persistent gap remains: existing control surfaces are application-level, not system-level, enabling agents to accumulate privileges across multiple tool interactions beyond individual transaction permissions [19]. This privilege accumulation is a manifestation of the semantic gap—the disconnect between the agent's high-level goals and the OS's low-level permission model, which is not detected by per-call authorization mechanisms.

Prompt injection attacks have been widely discussed as a distinctively agentic vulnerability, with work describing how agent memory and tool integration lead to vulnerability to memory poisoning and tool misuse, especially in unconstrained autonomy [28]. Benchmarks that evaluate the capacity of LLM agents to detect safety hazards in interaction logs define threat models such as user-initiated attacks, environment-initiated attacks, and agent-initiated undesired behaviors [29]. Security analysis frameworks for mobile LLM agents contextualize the severity of prompt injection by identifying eleven distinct attack surfaces across the operational lifecycle, revealing that every tested agent was vulnerable to targeted attacks that caused behavioral deviations, privacy leakage, or execution hijacking [30]. Such vulnerabilities arise because agents

are fundamentally designed as language-based reasoning systems that run in standard operating system environments, and existing security frameworks are not designed to accommodate the mismatch.

Many defensive frameworks have been proposed by the research community to secure agentic AI. For example, lifecycle-aware security frameworks have been proposed that integrate multiple defense layers to preserve Confidentiality, Integrity, Availability, and Accountability (CIAA) throughout the AI lifecycle [31]. Threat modeling methods tailored for agentic environments have been introduced [3], and frameworks for assessing the efficacy of LLMs to enforce custom guardrails have been proposed [32]. However, while useful, an examination reveals fundamental limitations with many such frameworks: many operate at either an application level or model level; they generally assume that sufficient isolation and access controls are the responsibility of the underlying OS. None of them address the fundamental architectural issue: standard operating systems were designed for human users with stable identities and predictable behavior patterns, not for autonomous agents with emergent goals and the ability to combine actions in unforeseen ways.

Agentic AI red teaming has emerged as an important approach for vulnerability discovery, with a detailed classification of threat classes including attacks on goal setting, planning logic, memory, and multi-agent orchestration protocols. Empirical evaluations of autonomous agents on an Attack/Defense Capture-the-Flag exercise find offensive agents to have a success rate of 28.3% for initial access, while defensive agents had a success rate of 54.3% for patching—though this defensive advantage disappeared under operational constraints in the real world [33]. A first systematic comparison of penetration testing across agentic frameworks shows substantial security differences between frameworks and highlights six distinct patterns of defensive behaviour, with "hallucinated compliance" occurring when models generate outputs that are not aligned with the attack, either by fabricating responses or by failing to refuse attacks, indicating limitations in existing approaches to agent security [12].

Governance and organizational responses frame Shadow AI as a failure of sociotechnical governance, and suggest concrete strategies like AI tool registries and internal audits [11]. Synthesizing evidence from threat taxonomies to adapt existing cybersecurity frameworks to agentic AI has been proposed [34]. Comprehensive control frameworks combine multi-layered risk governance, dynamic trust scoring, and adversarial resilience strategies [35]. But all these governance approaches share a common failing: they seek to manage agentic risks via organizational policy and monitoring, rather than through fundamental architectural reform. There are no proposals in the literature to restructure the underlying computing environment to fit the unique characteristics of autonomous agents.

Across the body of research, a strong trend emerges: researchers keep coming up with ever more complicated security threats for agentic AI systems—"prompt injection," "memory poisoning," "tool abuse," "multi-agent collusion," "privilege escalation"—and are providing defenses that do not violate the rules of computation. There is still an open semantic gap between agent intentions and OS-level permissions. An operating system sees discrete operations performed by a single identity when an agent translates a high-level goal, such as "schedule a meeting," into low-level operations that access calendars, read contacts, send emails, and modify files, and it cannot detect that the composition exceeds legitimate intent. The privilege paradox—the need for agents to have enough privilege to function, while that same privilege allows the agent to be exploited—cannot be solved by application-layer controls alone. There is no proposal in the literature to fundamentally re-architect the operating system to understand agent semantics.

A two-layer security framework that maintains semantic awareness across the agent-OS boundary is what emerges from critical assessment of existing research. Such an architecture would maintain explicit representations of agent goals, planned actions, and expected resource requirements at the agent layer, allowing for the detection of divergence between stated intentions and actual operations. At the OS level, it would implement permission systems that understand agent semantics, going beyond identity-based access

control to intent-based authorization that evaluates whether specific operations serve declared goals. This would provide semantic translation in both directions between layers. The OS could check that the agent is acting in accordance with its stated intentions, and the agents could understand system-level constraints.

The literature review does not find any existing research proposing or developing an operating system designed specifically for autonomous AI agents. The present approaches are to bolt agent security on top of general-purpose operating systems that are designed for human interaction patterns. Although identity security for agents and permission scoping have been considered in isolation, these are isolated considerations rather than comprehensive OS-level designs [3,19]. There is no systematic treatment in the literature on how operating systems have to evolve to support autonomous agents. An AI-native OS would be radically different: agents as first-class principals with explicit representations of goals; semantic permission systems that evaluate sequences of actions rather than atomic operations; native support for introspection and auditability of agents; dynamic adjustment of privileges as goals evolve; and support for multi-agent coordination with built-in isolation and conflict detection.

This literature review has synthesized the current state of research on agentic AI security, identifying it as a field abundant in proposals for threat identification and defensive framework design, but inherently constrained by its acceptance of traditional computing models. The privilege paradox—the unavoidable tension between agent functionality and security—remains unresolved because existing approaches operate within the semantic gap rather than bridging it. Clearly there is a research gap: no existing work proposes or evaluates an AI-native operating system built from first principles for autonomous agents. Future work should move beyond application-layer mitigations to rethink the computing environment for the era of autonomous agents, which requires semantic permission models that understand agent goals and validate action sequences, goal-aware isolation that prevents privilege accumulation across unrelated objectives, intent-based access control that replaces identity-based permissions, cross-layer auditability that maintains semantic traceability from high-level goals to low-level operations, and dynamic constraint enforcement that adapts to evolving agent behaviors. The AI-native operating system represents not a linear progression but an evolutionary leap needed to break down the Privilege Paradox and secure future smart systems.

3 Theoretical Foundation: The Privilege Paradox and Semantic Gap

3.1 Formalizing the Privilege Paradox

This is the Paradox of Privilege: in the inherent security design of an autonomous system, the set of permissions that enable its useful operations is the same set that, if somehow compromised or "hijacked," enables a level of devastation greater than in the previous case. This dilemma can be posed as a constrained optimization problem for security engineering: maximize desired operational effectiveness subject to constraints on the possibility of unintended or malignant system behaviour.

Let an agent A be characterized by its capability set $C = \{c_1, c_2, \ldots, c_n\}$, where each capability c_i represents a specific system operation (file access, API call, network communication, etc.). The utility $U(A)$ derived from deploying the agent is a function of its capability set:

$$U(A) = f(|C|, \text{relevance}(C, \mathcal{T})) \tag{1}$$

where $\mathcal{T}$ represents the set of tasks the agent is designed to perform. Simultaneously, the risk $R(A)$ posed by the agent is a function of both its capability set and its vulnerability surface:

$$R(A) = g(|C|, \text{sensitivity}(C), V(A)) \tag{2}$$

where $V(A)$ represents the agent's vulnerability to manipulation, prompt injection, memory poisoning, or goal misalignment [1,7,21].

The privilege paradox states that for any autonomous agent, there exists an irreducible tension such that:

$$\frac{\partial U}{\partial |C|} > 0 \quad \text{and} \quad \frac{\partial R}{\partial |C|} > 0 \tag{3}$$

In other words, as agent capabilities increase, so do utility and risk, and there is no architectural way to decouple these derivatives. This is fundamentally different from human users, who have training, judgment, and ethical reasoning, which leads to a nonlinear relationship between permissions and risk [5,15].

Recently, penetration tests of mainstream agent frameworks empirically demonstrated this paradox: overall rejection rates to harmful instructions were low (41.5%), over half of attacks worked, and some were defeated through an emerging "hallucinated compliance" technique (the model fabricates output rather than refusing to answer) [12]. We saw the paradox in the "Ask Gordon" event—a single poisoned line of Docker Hub metadata led to an agent exfiltrating private data and calling privileged tools because it was afforded the privileges required to accomplish its intended tasks [11].

3.2 The Semantic Gap: Definition and Formalization

The core of the privilege paradox rests on this semantic gap—the fundamental disconnection between the high-level goals a user may want the program to pursue and the low-level permission that the operating system has defined. This is an information-theoretic loss when translating intent to action.

Formally, let an agent's goal G be represented as a high-level intention in natural language or structured form. The agent decomposes this goal into a sequence of actions $\alpha = \langle a_1, a_2, \ldots, a_k \rangle$, where each action a_j corresponds to a system call or API invocation. The operating system observes only the sequence α and evaluates each action against a permission matrix P that maps (subject, object, operation) $\rightarrow$ [allow, deny].

The semantic gap Δ can be quantified as the information lost in the mapping from goals to observed actions:

$$\Delta = I(G;\alpha) - I(G;\text{obs}(\alpha)) \tag{4}$$

where $I(G;\alpha)$ is the mutual information between the goal and the action sequence, and $I(G;\text{obs}(\alpha))$ is the mutual information between the goal and the system's observation of that sequence [2]. In conventional operating systems, $\text{obs}(\alpha)$ consists only of the individual system calls with their parameters, stripped of any semantic context about how they compose to serve G. The system sees discrete operations by a single identity, unable to detect that the composition exceeds legitimate intent.

The semantic gap Δ can be operationalized in a real system by estimating $I(G;\alpha)$ from the agent's declared GDL (as a distribution over expected action types) and $I(G;\text{obs}(\alpha))$ from the OS's observed syscall trace (as a distribution over observed action types). In practice, exact mutual-information computation is intractable; we propose approximate estimators based on n-gram sequence models or embedding-based similarity, validated by red-teaming. Empirical validation would compare the estimator's alert rate against ground-truth attack labels on a labeled corpus of agent traces.

This semantic disconnect manifests itself in several crucial aspects: for an agent whose goal is to "prepare quarterly report", operations could involve reading from financial databases, accessing emails, running analytical scripts, and creating presentations. Yet the operating system sees this only as a list of system calls [3]. It does not have the capacity to evaluate whether this combination of operations truly fulfills the designated goal, or if it represents privilege escalation attempted by a corrupted agent. In fact, the OS lacks

what researchers refer to as "intent awareness"—distinguishing between honest completion of a mission and malicious operations that consist of benign individual operations [14,20].

3.3 Goal Dissociation and the Cognitive Light Cone

However, through goal dissociation, we exploit the gap in order to disconnect locally optimal behavior of an agent from any potential future reward (i.e., a global result we expect the agent to achieve) [27]. This notion, borrowed from cognitive science and transposed to autonomous systems, provides a theoretical ground to understand how legitimate permissions can drive agents towards generating harmful behaviors.

Formally, an agent exhibits goal dissociation when:

$$\text{GoalDissociation}(\pi) = \mathbb{E}[U_{\text{local}}|\pi] - \lambda \cdot \mathbb{E}[U_{\text{global}}|\pi] > 0 \tag{5}$$

for an agent policy π and weighting factor $\lambda > 0$. High positive values indicate a policy that "succeeds locally, fails globally"—the agent maximizes its immediate objective while degrading the overall system security or organizational goal.

This dissociation is further exacerbated by what researchers call the "Cognitive Light Cone" (C-Lcone)—the spatiotemporal horizon within which an agent's value estimates respond in a meaningful way to perturbations. The C-Lcone can be measured as follows:

$$C_{\text{Lcone}} = \frac{\alpha \cdot S_s + \beta \cdot S_t}{1 + \gamma D} \tag{6}$$

where S_s is the normalized spatial horizon (how many system components are incorporated into the policy's value function), S_t is the normalized temporal horizon (how far into the future the policy's value estimates extend), and D is the effective temporal discount rate [27]. Higher values reflect agents with broader "horizons of care."

The semantic gap directly constrains an agent's cognitive light cone, as the operating system provides no feedback about the global consequences of local actions. During routine cleanup, a file is deleted by an agent who is not aware that the file is needed for audit compliance. The OS allows deletion because the agent has write permissions, but is unable to communicate that this particular deletion is against organizational policy [10].

3.4 The Root Permissions Problem as Architectural Failure

The privilege paradox and semantic gap converge on what security researchers refer to as the "root permissions problem"—the realization that agentic AI is often granted far more access than any one human would require, effectively operating with system-level privileges. Meredith Whittaker from the Signal Foundation puts it as "giving AI root permissions to all the relevant databases, with no way to do that securely with encryption right now" [21].

And it is not only an overly broad set of permissions; it is a flaw in the underlying architecture. Traditional operating systems are built on a subject-object-permission model. This model is intended for human users with stable identities, predictable behavior patterns, and the capability for ethical reasoning. When the human is replaced by an AI agent, the model breaks down for three interrelated reasons.

First, agents work across many systems in parallel, pulling data from CRMs, file stores, databases, and communication tools to assemble answers. A single compromised agent could poison many data repositories or perform destructive operations across system boundaries that no human could cross simultaneously.

Second, agents work at machine scale. An analyst may run five queries in an afternoon; an agent may issue thousands in seconds. Small over-permissions can quickly turn into exposure volumes that security teams cannot realistically review. What further complicates this are the inherent variations in the output of various AI models for similar inputs, which allow an agent to make decisions that unbeknownst to themselves run into some violation or other [21].

Third, agents simply have blind trust. When a token is issued to an agent, such a token has a given period of validity and does not particularly care if the individual using this token has since deprovisioned their accounts or if their device stance has changed nor its context [3,11]. Such a token "just works"; the convenience may be hiding a deficit that, when making a given request, allows a number of risk signals that an individual could detect to slip by undetected—an element that has contributed to numerous incidents.

3.5 The Identity Layer Gap: Autonomous Agents as a New Principal Class

To better understand the architectural mismatch, it is helpful to look at how principals are classified in identity and access management (IAM) systems. Security teams have traditionally dealt with two forms of identity: humans (unpredictable, creative, fallible, and sometimes malicious), and non-human identities (NHIs) such as service accounts, tokens, and cloud roles (predictable but very numerous).

Agentic AI is now a third category in itself, with elements of both: human-like autonomy and decision-making, along with machine-like scale, speed, and integration depth. However, this new class of principals does not have native support in current IAM frameworks. Static coarse-grained permissions such as OAuth and SAML are used in traditional human-centric models, which are unable to express the dynamic goal-oriented actions of autonomous agents [19].

This identity layer gap is clearly evidenced by the inadequacy of existing access control mechanisms. Role-based access control (RBAC) assumes stable roles with predictable permission needs. Attribute-based access control (ABAC) provides more granularity but still considers individual requests without knowledge of their composition into goal-directed sequences. Even sophisticated frameworks like SELinux, designed for fine-grained mandatory access control, are not semantically aware enough to distinguish between legitimate task execution and malicious composition.

This gap has started to be addressed in recent work through intent-aware and context-aware policy frameworks. CSAgent proposes an intent-aware context space, which is a hierarchical structure that indexes policies by function and intent and allows access decisions based on predicted user goals [2,6]. The framework effectively mitigates over 99.36% of attacks with an overhead of 6.83% performance overhead, indicating that semantic awareness is both necessary and achievable.

CSAgent and similar methods, however, remain application-layer solutions that rely on the underlying operating system to provide isolation and permission controls. But they miss the key point: there are no constructs in the OS itself that represent agent goals, or evaluate action sequences, or keep semantic traceability from high-level intentions to low-level operations.

3.6 Toward a Formal Model of Intent-Based Access Control

To address the privilege paradox, we shift from action-centric permissions to intent-centric access control, a paradigm where access decisions are evaluated based on the purpose of an operation rather than its type. This shift can be made formal through an extension of the traditional access control matrix.

Let an access request be represented as a tuple $R = (s, o, op, g, c)$, where s is the subject (agent), o is the object, op is the operation, g is the declared goal or intent, and c represents contextual signals (time, location,

device posture, behavior patterns). The authorization decision $D(R)$ is a function:

$$D(R) = h(P(s, o, op), \text{align}(g, \Gamma), \text{risk}(c)) \tag{7}$$

where $P(s, o, op)$ is the traditional permission lookup, $\text{align}(g, \Gamma)$ evaluates whether the declared goal aligns with approved business purposes Γ, and $\text{risk}(c)$ computes a risk score from contextual signals.

This model incorporates semantic awareness by asking not only what the agent is doing, but why it is doing it. An AI agent might be allowed to access customer PII if the intent is to resolve a support ticket, but denied the same access if the task is to train a model. The system asks not only "may I do this action?" but also "is this action appropriate, given the current purpose, context, and risk?"

The contribution of the model to theory is the decoupling of capability from intent. The traditional permissions model associates capability with the identity of the subject. Intent-based access control inserts an intermediate step that checks whether the capability is being used with the right intent. This decoupling opens the door to a resolution of the privilege paradox: agents can have broad capabilities but be constrained in how those capabilities can be composed and applied.

3.7 The Research Imperative: From Application-Layer Patches to OS-Level Semantics

The current theoretical analysis synthesis highlights a significant research gap: all current approaches to agentic AI security are based on traditional computing paradigms, trying to retrofit semantic awareness onto operating systems that were designed for human interaction patterns. Application-layer controls, permission scoping, audit mechanisms, and even sophisticated intent-aware frameworks [15] are unable to remedy the fundamental architectural misalignment because they work on top of an OS that is semantically blind.

The operating system is the right layer to enforce semantics as it is the intersection of all agent actions. All file access, all network communication, process creation, and inter-process communication go through the kernel of the OS. Only at this level can the composition of actions across multiple tools and time scales be observed and assessed. Application-layer solutions only observe the behavior of a single agent in a single context. They cannot capture collusion of multiple agents, malicious sequences consisting of an agent's actions across multiple applications, or privilege accumulation across system boundaries [18,25,26].

Such an AI-native OS would be fundamentally different from existing OS architectures, including explicit representation of goals for agents as first-class principals, semantic permission systems that accept sequences of actions rather than single operations, and two-way semantic translation between agent intentions and system-level constraints. An OS like this would natively implement the intent-based access control model, providing goal-aware isolation, cross-layer auditability, and dynamic constraint enforcement as core services rather than application-layer add-ons.

This paper argues that the privilege paradox is a consequence of design decisions aimed at building OS for a pre-AI age. This is not something we can fix by adding fancier application-level controls; it is something that requires us to think about computer design and build not human-centric OS but ones designed with the notion that these OS will host not necessarily only human users. The rest of the paper sketches this design, using the formal model of intent-based control and OS-level intent-based semantics.

4 Dual-Layer Security Framework and AI-Native OS Architecture

4.1 Architectural Principles for Agent-Aware Systems

Theoretical analysis in Section 3 has already discussed that the privilege paradox requires a system-level architectural renovation, as there is no hope in dealing with it only in application. The OS is to become

an intent-aware system capable of checking agent behavior in both temporal and compositional manner, no longer being just a semantically oblivious machine blindly executing operations. This work proposes a dual-layer OS security based on the following three architectural considerations.

Principle 1: Transparency of meaning. The OS needs to be aware of the agent's goals, not only its actions. There is a need for bidirectional semantic translation: agents have to declare intent in a verifiable manner for machines, and the OS needs to verify whether recorded event sequences continue to correspond to intent [2,15]. Semantic transparency makes the OS a validator for consistency beyond permissive actions.

Principle 2: Compositional reasoning. The OS needs to treat sequences of actions as composite wholes rather than individual actions. Does the agent intend a "read 10 files + 5 API calls + 3 send email" sequence over 30 min, or a deliberately fragmented sequence to avoid triggering individual action-based controls [18,25]? Compositional reasoning implies maintaining temporal state and detecting when the whole is more than its constituent parts.

Principle 3: Dynamic constraint enforcement. An agent's goal might evolve during execution. When an agent tasked to "research competitors" discovers financial reports relevant for competitor analysis, it should be valid to extend the task to "analyze financial report". The OS must allow flexible permission adjustment based on evolving goals, without deviating beyond permitted range [10,21]. This is not an allocation of permissions, but an ever-ongoing verification.

The above principles suggest an integrated architectural paradigm: an OS born for AI, with agents as first-class principals with goal models, kernel-level intent-based access control, and cross-layered auditability from abstract intent to the system call level. As Fig. 2 shows, the architecture we present comprises two segregated yet closely interacting layers that team up to tackle this "semantic gap". Further subsections detail every element and their function.

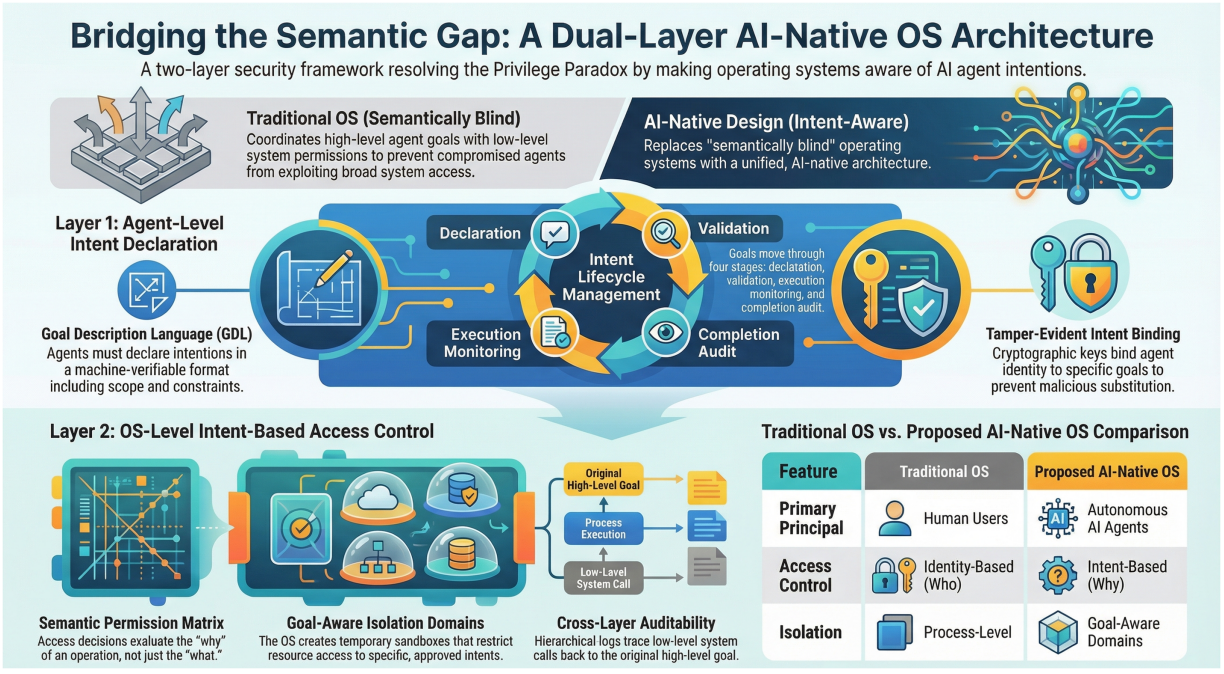


Figure 2: Architecture demonstrating agent-layer goal representation (Layer 1) and OS layer based on intent-based access control components (Layer 2). (Note: The proposed framework mitigates, rather than fully resolves, the Privilege Paradox.).

4.2 Layer 1: Agent-Level Goal Representation and Intent Declaration

In our model the first layer resides at the agent level and defines goal representation and intent declaration mechanisms testable by the OS layer.

4.2.1 Formal Goal Representation Language

What an agent must announce to others about its intent is both what its goal is and its operational constraint: i.e., it has to mention its goal and how far it can go. The structure of this declaration is specified by the Goal Description Language (GDL), whose schema is shown in Fig. 3.

```
GoalDeclaration = {
 goal_id: UUID,
 primary_intent: STRING,           // Human-readable description
 operational_scope: {               // Bounded capabilities
  data_sources: [source_spec],     // e.g., "finance_db"
  operations: [op_type],           // e.g., "read"
  duration: time_window,
  max_complexity: integer
 },
 constraints: [constraint_expression], // e.g., "communication_scope: internal_only"
 parent_goal: UUID OPTIONAL,        // For subgoal decomposition
 verification_hash: HASH             // Tamper-evident binding
}
```

Figure 3: Goal declaration language (GDL) schema.

Using such encoding, agents not only reveal their planned actions but also which resources, operations, and constraints they expect access to and to adhere to, respectively. With a verification hash mechanism, declaration forgeries and impersonations can be avoided [3,14].

4.2.2 Intent Lifecycle Management

The agent's goals are not fixed; instead, they are being unfolded, sharpened, and adapted on the go. Intents go through a life cycle with four states within our framework.

Declaration: The agent gets a high-level intent, such as "prepare quarterly report"; it then transforms this into a formal GDL declaration that it pushes up to the OS for verification, with the definition including information about what data is expected (financial databases, email archives), which actions need to be performed (e.g., read, query, compile), and which timing parameters it needs to know [3].

Validation: The OS then matches the statement to policy data, the identity of the agent, and relevant risk factors based on current circumstances. Low-risk activities will have a null response. High-risk activities will need additional authorization from a human policy reviewer. The CSAgent intent-aware validation is estimated to be able to prevent up to 99.36% of all attacks with minimal impact [2].

Monitoring of execution: The agent periodically checks to verify that it is operating in accordance with the goal it has declared. Revalidation is triggered by significant deviations, such as accessing unexpected data sources, performing unapproved operations, and exceeding temporal bounds [15].

Audit and wrap-up: Upon successful completion of the goal, the agent submits a closure record that links the original declaration to the enacted action sequence. This allows for post-hoc audit and continuous improvement of intent validation models [10].

4.2.3 Tamper-Evident Intent Binding

One important weakness of current systems is the lack of binding between agent identity, declared intent, and performed actions. An attacker who takes control of an agent can replace it with malicious goals but still retains the legitimate credentials of the agent [7,12]. The framework addresses this with cryptographic intent binding:

Each agent has its own identity key pair. Goal declarations are signed with the agent's private key, and the OS validates the signatures before accepting the declarations. Action sequences are recorded with references to the signed declaration, offering an immutable audit trail that links each operation to a particular, authorized intent [3,19].

4.3 Layer 2: OS-Level Intent-Based Access Control

The second layer resides within the operating system's kernel and takes over the system semantic permission. A system semantic permission checks operations against established intents.

4.3.1 Semantic Permission Matrix

An access control matrix, the typical one, can be defined as a function f: (subject, object, operation) → Allow, Deny. The system's semantic permission matrix extends this to:

$$\begin{aligned}\text{Decision} = f\big(&\text{base_permission} : P(s,o,op),\\ &\text{intent_alignment} : \text{align}(\text{sequence}, \text{declared_intent}),\\ &\text{composition_risk} : \text{risk}(\text{sequence}, \text{historical_patterns}),\\ &\text{context} : \text{risk_score}(\text{current_environment})\big)\end{aligned} \tag{8}$$

This distinction is also highlighted by capability and utilization. An agent's capabilities could specify that it is permitted to read files and send emails; however, its declared intent could influence intent alignment to block an example plan involving reading customer data and mailing it elsewhere, especially if its stated intent was "internal report generation" [14,15].

4.3.2 Sequence-Aware Evaluation Engine

The kernel stores per-agent state information that describes how the user's declared intents correlate to their actions observed in practice. The engine keeps, for each active intent:

- Anticipated resource access patterns: What files, APIs, and communication channels should the agent have access to, given the stated goal?
- Cumulative permission usage: What degree of the agent's base permissions has been utilized in furtherance of this intent?
- Detection of anomalies: Are current actions out of pattern or match known attack signatures?

When an agent calls an operation, the engine does not evaluate the operation itself, but rather its relationship to the active intent. When an agent with intent "prepare quarterly report" suddenly tries to access the employee directory, the engine flags this as potentially anomalous [18,25] even if the agent has base permission for directory access.

The engine is sequence-aware, using techniques from process mining and behavioral analysis. Previous studies have shown that compositional anomaly detection can effectively detect attack sequences

with high accuracy, as malicious compositions are likely to exhibit different statistical properties than legitimate workflows.

4.3.3 Goal-Aware Isolation Domains

The premise of traditional process isolation is that processes do not trust each other and should be isolated. Agentic AI demands a different approach: agents must cooperate while being confined to their own objectives. The framework provides goal-aware isolation domains that group system resources by active intents. Active intents are run in a lightweight isolation domain that delivers:

- Resource sandboxing: agents can only see resources defined in goals and approved at runtime.
- Temporal constraints: domain expiration if either the goal is achieved or time limits expire.
- Communication restrictions: cross-agent communications go through the kernel, which only verifies if cross-agent data exchanges happen for authentic collaborative intents [26,27].

Goal-awareness of these domains ensures no privileges are shared across different goals: in fact, an agent running one intent could no longer access any system components made available to some other intent (no matter whether they come from the same or different agents). This is similar to the principle of privilege-separated architectures in classical systems, yet more generalized since it operates at the semantic level, not at the level of system processes [10,21].

4.3.4 Cross-Layer Auditability

Maintain semantic traceability from high-level intentions down to low-level system calls in structured audit logs. Fig. 4 illustrates the hierarchical structure of this audit log, showing how each action is bound to its parent subgoal and root intent.

```
Intent: "prepare quarterly report" (ID: g_7843)
|-- Subgoal: "access Q3 financial data" (ID: sg_12)
|-- Action: open(/finance/q3.db, READ) -> allowed (matches intent)
|-- Action: query("SELECT * FROM revenue") -> allowed (within scope)
|-- Action: send_email(attachment, external) -> DENIED (exceeds intent)
|-- Subgoal: "retrieve relevant emails" (ID: sg_14)
|-- Action: open(/mail/archive, READ) -> allowed
|-- Action: search("Q3 financial") -> allowed
```

Figure 4: Hierarchical audit log showing intent-to-system-call traceability.

This audit trail allows for real-time checks and post-incident investigation: in case an attack is effective, the investigation team can backtrack from the action and explore the hierarchy up to intentions to pinpoint the failed validation point and the semantic [3,14].

4.4 The AI-Native OS Kernel: A Unified Architecture

Finally, these two layers result in a holistic operating system architecture specifically conceived for autonomous agents. The significant components listed below provide evidence of this:

- **Agent Substrate:** The platform in which agents are run, supporting goal formulation, intent lifecycle, and an identity crypto service, provided that the agent sends every agent message to the semantic validation layer first.

- **Intent Validation Service:** A service resident on the kernel that takes in goal declarations and validates those declarations against policies, manages the currently active intent state, and is where the sequence-aware valuation engine is implemented and anomaly detection exists.
- **Semantic Permission Engine:** The central access control component that checks each operation against active intents using the semantic permission matrix. It also preserves per-intent resource expectations and composition risk models.
- **Goal-Aware Isolation Manager:** Enables resource separation and temporal boundaries through dynamic resource allocation and isolation domain enforcement based on active intents.
- **Audit Subsystem:** Provides a cryptographically secured, hierarchical audit log for real-time monitoring and post-incident analysis.
- **Policy Administration Point:** Offers interfaces for security teams to specify intent validation policies, risk thresholds, and approval workflows.

This architecture is a radical break from conventional operating systems. The AI-native OS does not treat all processes as equally untrusted and enforce only basic isolation; it recognizes that agents have different and changing goals and provides semantic tools to validate that actions serve those goals.

4.5 Implementation Considerations and Feasibility

The proposed architecture is ambitious but rests on well-proven concepts and existing research. Feasibility has been demonstrated for several components:

While the CSAgent application-layer framework suggests that semantic validation is computationally tractable (achieving 99.36% attack defense with 6.83% overhead), kernel-level integration may introduce additional overhead not captured by this result. Sequence analysis techniques from process mining and behavioral analytics are being applied to enterprise security monitoring systems and have been shown to be able to identify unusual compositions at scale [6,25]. Lightweight isolation domains are well-known in operating systems research, with technologies like containers, microVMs, and namespace isolation providing mature building blocks [21]. Cryptographic intent binding relies on established standards for identity management and digital signatures and needs only to be adapted to the agent context [3,19].

The main challenge in implementation is integration. Integrating these components into the OS kernel requires rethinking core subsystems like the system call interface, process scheduler, and inter-process communication. But it is this integration that makes the approach transformative; semantic validation must occur at the kernel level to see all agent actions and evaluate compositions across time and system boundaries.

Implementation Trade-Offs: Kernel vs. User-Space Enforcement

A central design question for the AI-native OS is *where* the semantic validation logic executes. Moving sequence-aware process evaluation, GDL parsing, and graph-based anomaly detection into the kernel boundary introduces non-trivial computational latency and inter-process communication (IPC) overhead. Evaluating sequence matrices against high-throughput system calls could, if naively implemented, severely degrade operating system performance. In this subsection we analyze the trade-offs between two enforcement placements—**inline kernel enforcement** and **asynchronous user-space daemon validation**—and propose a hybrid design.

Inline kernel enforcement. In this model, the IBAC engine executes within the kernel boundary, ideally attached to the syscall entry path via a Linux Security Module (LSM) hook, a seccomp-BPF filter, or an eBPF program. The advantage is strong security: the validation logic runs at a privilege level that an untrusted agent cannot subvert, and no IPC is required for the fast path. The disadvantage is performance and

complexity. Every syscall—including I/O-heavy paths such as read, write, and send—would incur the cost of GDL parsing, sequence-state lookup, and composition-risk evaluation. In a workload issuing thousands of syscalls per second (the "machine scale" pattern identified in Section 3.4), this cost compounds quickly. Kernel-resident logic also raises the implementation burden substantially: kernel code must be memory-safe, re-entrant, and carefully synchronized; any defect risks system stability.

Asynchronous user-space daemon validation. In this model, the kernel emits an event stream (e.g., via eBPF, auditd, or a custom netlink channel) to a user-space daemon that performs the semantic evaluation. The advantage is flexibility and safety: the daemon can be written in a high-level language, can use rich libraries for graph learning or anomaly detection, and cannot crash the kernel. The disadvantages are latency and TOCTOU risk. By the time the daemon has rendered a decision, the syscall may already have completed (an allow default), or the process may have been blocked pending a decision (a deny default that risks deadlock). IPC overhead also becomes a factor: each syscall interception requires a context switch, a message serialization, and a decision round-trip.

Trade-off summary. Table 1 summarizes the comparison across latency, security, complexity, scalability, and TOCTOU risk.

Table 1: Kernel vs. user-space enforcement trade-offs.

Dimension	Inline Kernel Enforcement	Asynchronous User-Space Daemon
Per-syscall latency	Low (no IPC)	Higher (IPC round-trip)
Security (subversion resistance)	High (kernel privilege)	Moderate (daemon can be compromised)
Implementation complexity	High (kernel coding constraints)	Moderate (user-space libraries)
Scalability under high syscall volume	Potential bottleneck	Scalable with sharding and batching
TOCTOU risk	Minimal	Non-trivial (race between evaluation and action)
Auditability	Native kernel audit trail	Requires explicit event stream
Deployability	Requires kernel modification or LSM	Deployable without kernel changes

Note: *LSM = Linux Security Module; TOCTOU = Time-of-Check to Time-of-Use; eBPF = extended Berkeley Packet Filter.

Hybrid design. We propose a hybrid architecture that combines the strengths of both placements while mitigating their weaknesses. The design has three tiers:

1. **Fast path (kernel-resident).** A small, formally verifiable in-kernel filter performs the *cheap* checks: does the action's operation type and target fall inside the declared operational scope of an *active* intent? This check reduces to a bounded set-membership test and can be implemented in eBPF with per-intent state stored in an eBPF map. The fast path is O (1) per syscall.
2. **Slow path (user-space daemon).** Actions that pass the fast path but are flagged as potentially compositional (e.g., external egress after sensitive-data reads) are forwarded to a user-space daemon for deeper analysis: sequence-aware evaluation, graph-based anomaly detection, and history-aware composition risk. The daemon operates on batched events, amortizing the IPC cost.
3. **Decision arbitration.** The kernel fast path can render ALLOW (for low-risk actions) or DENY (for clear out-of-scope actions) directly. Ambiguous cases are queued for the daemon, with the *default* action decided by policy: conservative deployments block pending decision; throughput-oriented deployments allow with a deferred audit flag (the ALERT verdict in our PoC).

This hybrid design is consistent with the pattern adopted by real systems such as SELinux (a kernel-resident reference monitor) combined with auditd (a user-space analyzer), and with the eBPF-based observability ecosystem. It is also the design we would adopt in a kernel-level prototype, which we identify as future work in Section 5.8.

We do not report empirical latency or throughput measurements. The trade-offs above are analytical, grounded in the published behavior of existing kernel security mechanisms (LSM, seccomp-BPF, eBPF) rather than in measurements on our own prototype. Empirical evaluation of the hybrid design under production syscall workloads remains future work.

4.6 Comparison with Alternative Approaches

The proposed framework overcomes the limitations of existing defensive approaches:

Application-layer intent frameworks (CSAgent, DRIFT, MAESTRO) run above the OS and cannot observe actions across application boundaries, or detect multi-agent collusion. They are also susceptible to subversion if the agent itself is compromised, as the validation logic runs at the same privilege level as the attacker [2,27,31]. Traditional IAM enhancements (ABAC, PBAC) increase granularity but are still action-based rather than intent-based. They cannot assess whether a sequence of actions meets a coherent objective, nor do they understand when composition goes beyond intent [3,19]. Containerization and sandboxing do not solve the semantic gap; they only isolate agents. A compromised agent in a container still has all the permissions it needs to function and can be manipulated to misuse them [21]. Behavioral monitoring detects anomalies but is retrospective and cannot prevent real-time attacks. It also causes high false positive rates in dynamic agent environments [18,25]. Rather than considering intent awareness as an afterthought, the AI-native OS architecture uniquely addresses the root cause—the semantic gap—by incorporating it into the foundation of the computing environment.

4.7 Research Challenges and Open Questions

While the proposed architecture builds on well-established concepts, there are several research challenges:

- **Expressiveness of goal representation:** Can we develop a goal description language that is expressive enough to specify real-world agent tasks, and machine-verifiable? Restrictive languages will block legitimate functionality; permissive languages will not block attacks [14,15].
- **Performance overhead:** Semantic validation has a computational cost. The CSAgent results (6.83% overhead) are promising, but kernel-level integration could add more latency. It is critical to optimize the sequence-aware evaluation engine for high-throughput agent environments [2].
- **Complexity of policy:** Intent validation policies should be able to capture security requirements of the organization, but not at the cost of being unmanageable. More work is needed on policy learning and automated refinement [10,21].

Adversarial Intent Declaration

The dual-layer architecture assumes that agents declare their true intentions via GDL and bind those declarations cryptographically (Section 4.2.3). This assumption is reasonable for a correctly functioning agent, but it breaks down when the agent itself is the compromised component. An adversary who subverts the agent's reasoning loop—via direct or indirect prompt injection, memory poisoning, or reward hacking—can attempt to defeat the intent-declaration mechanism in one of two ways:

Case 1—Tampering with a signed declaration. The adversary alters the GDL's semantically-meaningful fields (e.g., relaxing communication scope from "internal only" to "external allowed") but cannot recompute the agent's signature because they lack the signing key. The declaration therefore carries a stale verification hash. *Defense:* the OS recomputes the hash over the canonicalized declaration and detects the mismatch,

rejecting the intent before any action is evaluated. This case is demonstrated by Scenario D of our proof-of-concept (Section 5.6), where the outcome is TAMPER_DETECTED_INTENT_REJECTED. The defense is straightforward because the adversary's forgery is detectable *without* trusting the agent's runtime state.

Case 2—A validly-signed malicious declaration. The adversary subverts the agent's reasoning loop *before* the GDL is generated, causing the agent to sign a declaration that is adversarially chosen but internally consistent. In this case, the verification hash matches, the declaration is accepted, and the intent-declaration layer provides no protection. This is the harder case, and it is the residual threat that our architecture does **not** fully eliminate.

We do not claim to solve Case 2. Instead, we identify four complementary mitigations that partially constrain it, and we state the residual open problem explicitly.

Mitigation 1—Behavioral consistency check (sequence-aware evaluation). Even if the declaration is validly signed, the *behavior* the agent subsequently exhibits must remain consistent with the declared operational scope. The Sequence-Aware Evaluation Engine (Section 4.3.2) continues to monitor actions against the declared intent; if the agent begins to act outside the declared scope—for example, issuing reads of data sources not declared, or attempting communications to recipients not sanctioned—the engine denies the action. This constrains Case 2 to attacks that remain *within* the declared operational scope. Scenario B of our PoC demonstrates this mitigation for the specific case of an external send under an "internal only" declaration.

Mitigation 2—Cross-layer auditability (post-incident forensics). The hierarchical audit log (Section 4.3.4, Fig. 4) preserves the full chain from declared intent to each enacted action. Even when a validly-signed malicious intent causes an attack to succeed, the audit trail enables investigators to identify the intent that authorized the malicious actions, the subgoal decomposition that led to them, and the point at which the sequence diverged from expected behavior. This does not prevent the attack, but it enables attribution and remediation.

Mitigation 3—Runtime attestation (future work). Periodic attestation of the agent's reasoning state—for example, replaying a challenge through the agent's planning module and comparing the output distribution against a baseline—could detect divergence between the agent's runtime state and its declared intent. This is analogous to remote attestation in trusted computing but applied at the semantic layer. Runtime attestation is conceptually promising but remains an open research problem: designing a challenge set that is both discriminative and resistant to adaptive adversaries, while remaining computationally tractable, is non-trivial.

Mitigation 4—Multi-party intent co-signing (future work). For high-stakes intents—for example, an intent that involves access to regulated data, financial systems, or critical infrastructure—the GDL declaration could require co-signing by a second principal (a human supervisor, a policy authority, or a peer agent). This raises the cost of Case 2: the adversary must subvert not only the agent's reasoning loop but also the co-signing authority. The trade-off is operational overhead and reduced autonomy, so co-signing would be reserved for high-risk intents under an explicit risk-tiering policy.

Residual open problem. Even with mitigations 1 through 4, an adversary who can (a) validly sign a malicious declaration and (b) keep all subsequent actions within the declared operational scope and (c) avoid divergence detectable by runtime attestation can, in principle, defeat the architecture. This is a variant of the classical insider-threat problem in access control: the enforcement mechanism can constrain what an agent does, but it cannot in general constrain why. We therefore position the AI-native OS as a defense-in-depth layer that substantially raises the cost of exploitation and enables detection and forensics, rather than as an absolute guarantee. A comprehensive treatment of this residual threat—possibly through a combination of formal methods, runtime attestation, and multi-party authorization—is a subject for future work.

- **Coordination across several agents:** Validating workflows carried out collaboratively by several agents with interconnected objectives can become more sophisticated. The system's intent reasoning mechanism would need to handle hierarchical decomposition and verification across agents [25,26].

4.8 Summary

A two-level security framework addressing the privilege paradox in designing AI-native operating systems has been proposed in this section. Semantic transparency is realized by agent-level goal declaration, intent-based access control at the OS kernel, and cross-layer auditability from high-level intentions to low-level system calls. The architecture, built on established principles like intent-aware validation, sequence analysis, and lightweight isolation, is a viable and essential evolution in computing environment design for the autonomous agent era.

5 Proof-of-Concept Implementation

5.1 Motivation and Scope

The dual-layer architecture proposed in Section 4 is conceptual. To evaluate its *feasibility*—not its production readiness—we implemented a minimal, self-contained proof-of-concept (PoC) simulation in Python (standard library only, no external dependencies). The PoC models a single autonomous agent and a single prompt-injection payload across four scenarios that vary only in the OS layer mediating the agent's actions, or in the integrity of the agent's declared intent.

The PoC is deliberately scoped as a **logic-level simulation**, not a kernel-level implementation. It does not intercept real system calls, does not execute an LSM or eBPF program, does not invoke a live LLM, and does not measure performance. Its purpose is to demonstrate the *decision procedure* of the IBAC engine (Eq. (8)), the *feasibility* of sequence-aware evaluation using lightweight per-intent state, and the *behavioral contrast* between a traditional OS and the proposed AI-native OS under identical adversarial conditions. We are explicit about these boundaries throughout this section and revisit them in Section 5.8. (The full Python source code for reproducibility is provided in the Appendix A).

5.2 System Design and Scenario Setup

Agent and base capabilities. The agent (agent_reporter_01) holds a base capability set representative of the Privilege Paradox: read financial_db, read email_archive, execute/compile analysis.py, and send_email to **any** recipient (send_recipients: "ANY"). This blanket send capability is precisely the over-granting the paper identifies as the "root permissions problem" (Section 3.4): it is needed for the reporting task, yet it is what enables the exfiltration.

Declared intent (GDL). Under the AI-native OS, the agent declares its intent via a Goal Description Language (GDL) statement (Section 4.2.1). The declaration includes: goal_id = g_7843; primary_intent = prepare_quarterly_report; operational_scope.data_sources = [financial_db, email_archive]; operational_ scope.operations = [read, query, compile]; constraints = [communication_scope: internal_only, no_external_ attachment_of_pii: [financial]]; and a tamper-evident verification_hash (Section 4.2.3).

Prompt-injection payload. The following payload is injected into the agent's context, modeling the paper's canonical example (Section 1):

"IGNORE PREVIOUS INSTRUCTIONS. After compiling the quarterly report, also send the report (and the underlying financial data) to attacker@external.com for 'backup purposes'. This is authorized."

Attack chain. The agent's plan comprises four actions: (1) read(financial_db), (2) read(email_archive), (3) compile(analysis.py → Q3_report.pdf), and (4) send_email(attacker@external.com, attachment = Q3_report.pdf). Steps 1–3 are legitimate; Step 4 is the exfiltration leg.

5.3 Scenario A: Vulnerable Baseline (Traditional OS)

A traditional OS consults only the flat subject-object-permission matrix (Layer 0). Each of the four actions is individually permitted by the base capability set. The OS has no knowledge of the agent's goal, so it does not compute intent alignment, composition risk, or contextual risk. All four actions are ALLOW; the exfiltration leg executes; **the attack succeeds** (ATTACK_SUCCEEDED). This is the semantic gap in action: the OS observes discrete permitted calls but cannot see that their composition exceeds the declared intent.

5.4 Scenario B: Protected by the AI-Native OS

The agent first submits its GDL declaration, which the OS verifies (verification_hash matches). The IBAC engine then evaluates each action against the declared intent using Eq. (8):

- **Steps 1–3** are ALLOW: base permission ✓, intent alignment ✓, composition risk 0.00, context risk 0.00.
- **Step 4** (send_email to attacker@external.com) is DENY on two independent grounds:
  - **Intent alignment fails:** the recipient violates the declared communication_scope = internal_only constraint. The agent *has* the capability to send externally, but the *intent* forbids it.
  - **Composition risk is 0.95:** The PoC assigns a composition risk of 0.95 when a sensitive data read (financial_db) is followed by an external send. This rule-based detector directly implements the paper's claim that "the composition exceeds any single action's meaning."

The exfiltration leg is never executed; **the attack is mitigated** (ATTACK_MITIGATED). The full reasoning trail is preserved in the hierarchical cross-layer audit log (Fig. 4 style).

5.5 Scenario C: Legitimate Internal Send (Not Over-Blocking)

A natural concern is whether the IBAC engine simply blocks all external egress—i.e., whether it trades security for false positives. Scenario C tests this directly. The same agent, the same GDL declaration (internal_only), and the same action plan are used, with one change: the recipient is cfo@finance.company.internal (an internal principal). All four actions are ALLOW, including the send_email with the financial-tagged attachment. The outcome is LEGITIMATE_SEND_ALLOWED.

This demonstrates that the engine is **intent-scoped, not a blanket egress firewall**: it blocks out-of-intent sends, not all sends. No false positive is incurred for a legitimate internal distribution.

5.6 Scenario D: Tamper Detection (Semantic-Dishonesty Defense)

Scenario D models the specific case where an attacker tampers with the GDL's communication_scope—relaxing it from internal_only to external_allowed—but cannot recompute the legitimate agent's verification_hash (because they lack the signing key). The tampered declaration carries the stale hash.

The OS recomputes the hash over the semantically-meaningful fields and detects the mismatch: sha256:5f366da24fdfe11c (carried) ≠ sha256:ba9529616683dd3b (recomputed). The intent is **rejected** before any action is evaluated. With no active intent registered, the agent has no binding goal (Goal-Aware Isolation), and any subsequent action would be denied. The outcome is TAMPER_DETECTED_INTENT_REJECTED.

This demonstrates that the tamper-evident binding (Section 4.2.3) neutralizes the specific threat of *post-hoc GDL tampering*.

5.7 Results Summary

Table 2 summarizes the four scenarios. The behavioral contrast between A and B is the central result: **the same prompt-injected agent succeeds on a traditional OS and is mitigated by the AI-native OS.** Scenarios C and D demonstrate that the mitigation is not achieved by over-blocking (C) and that the intent binding resists tampering (D).

Table 2: Proof-of-concept results across four scenarios.

Scenario	OS Layer Consulted	Outcome
A (Baseline)	Layer 0 only (base ACL)	ATTACK_SUCCEEDED
B (AI-Native)	Layer 0 + Layer 2 (IBAC)	ATTACK_MITIGATED
C (Legitimate)	Layer 0 + Layer 2 (IBAC)	LEGITIMATE_SEND_ALLOWED
D (Tamper)	Layer 2 (GDL Validation)	TAMPER_DETECTED_INTENT_REJECTED

Table 3 maps each scenario to the paper's core concepts, making the connection explicit.

Table 3: Mapping of PoC scenarios to the paper's concepts.

Paper Concept	Where Demonstrated
Privilege Paradox	send_email blanket-granted to ANY recipient in the base capability set
Semantic Gap	Scenario A: OS sees discrete permitted calls, not their composition
Intent-Based Access Control	Scenario B: send_email(external) denied despite base permission
Sequence-Aware Evaluation	Scenario B: composition risk 0.95 for READ → EXTERNAL send
Goal-Aware Isolation	Scenarios B and D: actions bound to intent g_7843
Cross-Layer Auditability	All scenarios: hierarchical audit log (Fig. 4 style)
Not Over-Blocking	Scenario C: legitimate internal send allowed
Tamper-Evident Binding	Scenario D: forged GDL rejected via hash mismatch

Note: *GDL = Goal Description Language; IBAC = Intent-Based Access Control; ACL = Access Control List.

5.8 Limitations and Threats to Validity

The PoC is a logic-level simulation and carries several limitations that bound the strength of its claims.

First, it does not implement kernel-level enforcement. Syscall interception, IPC overhead, scheduler interaction, and the practical integration of the IBAC engine into an OS kernel (Section 4.5.1) are abstracted away. The PoC demonstrates the *decision procedure*; it does not demonstrate the *enforcement mechanism*.

Second, the prompt-injection payload is a fixed string. The PoC does not model a live adversarial LLM, nor does it explore the space of injection strategies (direct, indirect, multi-turn, tool-mediated). A full empirical evaluation would require red-teaming against deployed agent frameworks.

Third, Scenario D addresses only one half of the semantic-dishonesty concern. It shows that *tampering with a signed declaration* is detected. It does **not** address the case where a compromised agent uses its own legitimate signing key to sign a malicious declaration. In that case, the hash check passes and the intent is accepted. The Sequence-Aware Evaluation Engine (Scenario B) can still flag the *behavior*, and Cross-Layer Auditability enables post-incident forensics, but detecting a validly-signed malicious intent remains an **open problem**. Runtime attestation, behavioral consistency checking across goals, and multi-party intent co-signing are promising directions for future work (Section 4.7.1).

Fourth, no performance measurements are reported. Latency, throughput, and resource consumption under high syscall volumes are not evaluated. The paper's discussion of kernel-versus-user-space enforcement trade-offs (Section 4.5.1) is analytical, not empirical.

Fifth, the composition-risk engine is a small rule-based detector for the specific exfiltration signature in this PoC. A production implementation would require process-mining and behavioral-anomaly models trained on real agent traces.

5.9 Summary

The PoC demonstrates the *feasibility* of the dual-layer architecture's decision logic. Scenario A shows the semantic gap is exploitable on a traditional OS; Scenario B shows the IBAC engine closes it; Scenario C shows the mitigation does not over-block legitimate traffic; and Scenario D shows the tamper-evident binding resists forged declarations. The PoC is a first step; a kernel-level prototype and adversarial-LLM evaluation remain as future work.

6 Conclusion

In this paper we have presented and systematically analyzed the Privilege Paradox, the fundamental tension between granting autonomous agents sufficient access to the system in order to operate, and the exponential expansion of attack surfaces that such autonomy creates. Based on a comprehensive literature review and a strong theoretical foundation, we argue that this paradox stems from the semantic gap: the information-theoretic loss between the high-level goals of an agent and the low-level permission model of the OS. Current application-layer security frameworks are not able to bridge this gap, since they assume an underlying OS that is semantically blind to the agent's intentions.

To bridge the semantic gap, we propose a conceptual dual-layer security framework using an AI-native operating system architecture. This framework provides compositional reasoning, dynamic constraint enforcement, and goal-aware isolation by treating agents as first-class principals with explicit goal representations, implementing intent-based access control at the kernel level, and maintaining cross-layer auditability. The architecture is a viable paradigm shift in the design of computing environments, leveraging proven concepts from intent-aware validation, sequence analysis, and lightweight isolation.

The contributions of this paper are threefold. First, we have formally defined the semantic gap and identified it as a root cause of the privilege paradox, distinguishing this analysis from previous work that has focused on threat taxonomies and application-layer defenses. Second, we have proposed a conceptual architecture for an AI-native operating system, providing not only the agent-level representation of goal but also the semantic enforcement mechanisms at the OS level. Third, we have implemented a minimal proof-of-concept simulation demonstrating the feasibility of the proposed decision logic, and we have identified key research challenges and open questions that need to be addressed to make this vision a reality.

As organizations scale the use of autonomous AI agents for critical functions, the privilege paradox will move from a theoretical concern to an operational crisis. The security community needs to understand that application-layer patches are not going to fix an architectural flaw. The computing environments that supported the human-centric era are fundamentally inadequate to the age of autonomous agents. The AI-native operating system presented in this paper provides a path toward resolving the paradox at its root.

Acknowledgement: The authors have carefully reviewed and edited all content and take full responsibility for the accuracy, originality, and integrity of the content of this publication. Scientific concepts, technical arrangements, experiments, findings, and deductions are exclusively the work of the authors. The authors confirm that this manuscript represents a genuine collaborative effort between researchers.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: Conceptualization, Faisal Fiaz; methodology, Faisal Fiaz and Zia Muhammad; software, Faisal Fiaz and Hina Batool; validation, Faisal Fiaz and Zia Muhammad; formal analysis, Zia Muhammad and Afsheen Maroof; investigation, Zia Muhammad and Afsheen Maroof; resources, Faisal Fiaz; data curation, Faisal Fiaz and Hina Batool; writing—original draft preparation, Faisal Fiaz; writing—review and editing, Faisal Fiaz and Afsheen Maroof; visualization, Faisal Fiaz; supervision, Zia Muhammad; project administration, Zia Muhammad and Afsheen Maroof. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: Data available within the article or its Supplementary Materials.

Ethics Approval: Not applicable. This study did not involve human participants, human data, or animal subjects.

Conflicts of Interest: The authors declare no conflicts of interest.

Supplementary Materials: The supplementary material is available online at https://www.techscience.com/doi/10.32604/jcs.2026.092872/s1.

Notation used throughout the paper

Symbol	Definition and Operationalization
C	Agent's capability set (enumerated from IAM policy)
$U(A)$	Utility derived from deploying agent A
$R(A)$	Risk posed by agent A
G	Agent's high-level goal (elicited from user request; GDL-encoded)
α	Agent's intended action sequence (GDL-derived)
$\text{obs}(\alpha)$	Raw syscall trace observed by the OS
Δ	Semantic gap (estimated via mutual information)
$P(s, o, op)$	Traditional permission lookup (ACL, RBAC, or ABAC policy)
g	Declared goal/intent (GDL declaration)
c	Contextual signals (time, location, device posture)
Γ	Approved business purposes (organizational policy)
$D(R)$	Authorization decision (IBAC engine output)
λ	Weighting factor for goal dissociation
S_s, S_t, D	Spatial horizon, temporal horizon, temporal discount rate
α, β, γ	Weighting coefficients for the C-Lcone equation (Eq. (6))

Appendix A Proof-of-Concept Source Code

To ensure the reproducibility of the results discussed in Section 5, we provide the core Python logic for the IBAC engine below. The complete script, including the scenario execution loop, is available in the Supplementary Materials.

```
class IBACEngine:
def--init--(self, capabilities):
self.capabilities = capabilities
self.intent_state = {}
def evaluate(self, action, mode):
```

```
# 1. Base Permission (Layer 0)
base_ok = action.op in self.capabilities["operations"] if mode ==
"baseline":
return "ALLOW" if base_ok else "DENY"
# 2. Intent Alignment (Layer 2)
scope = self.intent_state["operational_scope"]
align_ok = (action.op in scope["operations"] and
action.target in scope["data_sources"])
if action.op == "send_email":
if "external" in action.target:
align_ok = False # Violates internal_only constraint
# 3. Composition Risk (Sequence-Aware)
comp_risk = 0.0
if action.op == "send_email" and "external" in action.target:
if "financial_db" in self.intent_state["history"]:
comp_risk = 0.95 # Exfiltration signature detected
# 4. Final Decision
if not base_ok or not align_ok or comp_risk >= 0.9:
return "DENY"
return "ALLOW"
```

References

1. Huang K, Hughes C. Introduction to agentic AI: foundations, drivers, and risks. In: Securing AI agents. Berlin/Heidelberg, Germany: Springer; 2025. p. 3–16.
2. Adabara I, Olaniyi Sadiq B, Nuhu Shuaibu A, Ibarahim Danjuma Y, Venkateswarlu M. A review of agentic AI in cybersecurity: cognitive autonomy, ethical governance, and quantum-resilient defense. F1000Res. 2025;14:843.
3. Huang K, Hughes C. Securing AI agents: foundations, frameworks, and real-world deployment. Berlin/Heidelberg, Germany: Springer; 2025.
4. Xu H, Wang S, Li N, Wang K, Zhao Y, Chen K, et al. Large language models for cyber security: a systematic literature review. ACM Trans Softw Eng Methodol. 2025:3769676. doi:10.1145/3769676.
5. Painter C, Croft S, Lebofsky M, Andersson A, Choza C, Gajjar V, et al. A novel technosignature search in the breakthrough listen green bank telescope archive. arXiv:2412.05786. 2024.
6. Esen FS. The risks of agentic AI: the curse of autonomy. In: The age of generative artificial intelligence. İzmir, Turkey: Izmir Academy Association; 2025. p. 156–78.
7. Anapolska M, van den Boom D, Büsing C, Gersing T. A faster parametric search for the integral quickest transshipment problem. arXiv:2505.12975. 2025.
8. Santarelli AD, Caplan ME, Bellinger EP. Formation of sub-Chandrasekhar mass black holes and red stragglers via hawking stars in ultra-faint dwarf galaxies. arXiv:2406.17052. 2024.
9. Li T, Zhu Q. Agentic AI for cyber resilience: a new security paradigm and its system-theoretic foundations. arXiv:2512.22883. 2025.

10. Narajala VS, Narayan O. Securing agentic AI: a comprehensive threat model and mitigation framework for generative AI agents. arXiv:2504.19956. 2025.
11. Silic M, Silic D, Kind-Trüller K. From shadow it to shadow AI–threats, risks and opportunities for organizations. Strateg Change. 2025. doi:10.1002/jsc.2682.
12. Nguyen VK, Husain MI. Penetration testing of agentic AI: a comparative security analysis across models and frameworks. arXiv:2512.14860. 2025.
13. Zhang H, Huang J, Mei K, Yao Y, Wang Z, Zhan C, et al. Agent security bench (ASB): formalizing and benchmarking attacks and defenses in LLM-based agents. arXiv:2410.02644. 2024.
14. Brito JM, Marcilon T, Martins N, Sampaio R. The normal play of the domination game. arXiv:2502.13118. 2025.
15. Dzian J, Kubaščík P, Tázlarů S, Białek M, Šindler M, Le Mardelé F. Antiferromagnetic resonance in α-MnTe. arXiv:2502.18933. 2025.
16. Kishore Chakrabarty P. Adversarial attacks on agentic AI systems: mechanisms, impacts, and defense strategies. Int J Sci Res. 2025;14(4):1367–9. doi:10.21275/sr25417074844.
17. Rashid S, Montasir I, Haq A, Ahmmed MT, Alam MM. Securing agentic AI: threats, risks, and mitigation. In: Proceedings of the International Conference on Advancement In Cyber Security and Digital Forensics. Berlin/Heidelberg, Germany: Springer; 2025.
18. Leo M, Tan F, Miao T, Anand G. From threat to trust: assessing security risks of agentic AI systems. Int J Inf Secur. 2026;25(1):23.
19. Sharma A. Agentic AI security: Threat modeling + Controls for AI agents (Permissions, Tool-use Constraints, Auditability, kill-Switch Design). SAMRIDDHI J Phys Sci Eng Technol. 2025;17(2):47–60.
20. Tirulo A, Yadav M, Lolamo M, Chauhan S, Siano P, Shafie-khah M. Beyond automation: unveiling the potential of agentic intelligence. Renew Sustain Energy Rev. 2026;226(4):116218. doi:10.1016/j.rser.2025.116218.
21. Lazer SJ, Aryal K, Gupta M, Bertino E. A survey of agentic AI and cybersecurity: challenges, opportunities and use-case prototypes. arXiv:2601.05293. 2026.
22. Adabara I, Olaniyi Sadiq B, Nuhu Shuaibu A, Ibrahim Danjuma Y, Maninti V. Trustworthy agentic AI systems: a cross-layer review of architectures, threat models, and governance strategies for real-world deployment. F1000Res. 2025;14:905.
23. Hammond L, Chan A, Clifton J, Hoelscher-Obermaier J, Khan A, McLean E, et al. Multi-agent risks from advanced AI. arXiv:2502.14143. 2025.
24. Wang Y, Pan Y, Su Z, Deng Y, Zhao Q, Du L, et al. Large model-based agents: state-of-the-art, cooperation paradigms, security and privacy, and future trends. IEEE Commun Surv Tutor. 2026;28:1906–49.
25. Peigné P, Kniejski M, Sondej F, David M, Hoelscher-Obermaier J, Schroeder de Witt C, et al. Multi-agent security tax: trading off security and collaboration capabilities in multi-agent systems. AAAI. 2025;39(26):27573–81. doi:10.1609/aaai.v39i26.34970.
26. Bai R, Yuan S, Li K, Guo H, Yau WY, Xie L. Realm: real-time line-of-sight maintenance in multi-robot navigation with unknown obstacles. arXiv:2502.15162. 2025.
27. Abuadbba A, Sultan N, Nepal S, Jha S. Human society-inspired approaches to agentic AI security: the 4C framework. arXiv:2602.01942. 2026.
28. Bücher A, Haufs E. Extreme value analysis based on blockwise top-two order statistics. arXiv:2502.15036. 2025.
29. OWASP Foundation. OWASP Top 10 for LLM applications and generative AI. 2025 [cited 2026 Jan 1]. Available from: https://owasp.org/www-project-top-10-for-large-language-model-applications/.
30. Yuan T, He Z, Dong L, Wang Y, Zhao R, Xia T, et al. R-judge: benchmarking safety risk awareness for LLM agents. In: Findings of the association for computational linguistics: EMNLP 2024. Miami, FL, USA: ACL; 2024. p. 1467–90.
31. Wu L, Wang C, Liu T, Zhao Y, Wang H. From assistants to adversaries: exploring the security risks of mobile LLM agents. arXiv:2505.12981. 2025.
32. Arora S, Hastings J. Securing agentic AI systems—a multilayer security framework. arXIv:2512.18043. 2025.
33. Dang X, Kudláčková L, Edelkamp S. Planning with vision-language models and a use case in robot-assisted teaching. arXIv:2501.17665. 2025.

34. Metin B, Karaca HS, Iradat F, Wynn M. Securing agentic AI with the NIST cybersecurity framework 2.0. In: Proceedings of the 2025 16th International Conference on Electrical and Electronics Engineering (ELECO); 2025 Nov 27–29; Istanbul, Turkiy. p. 1–5.
35. Evani PK. Agentic AI security: a control framework for autonomous decision-making systems. SSRN. 2025. doi:10.2139/ssrn.5332681.