The Attack and Defense Landscape of Agentic AI: A Comprehensive Survey

JUHEE KIM*, UC Berkeley / Seoul National University, USA
XIAOYUAN LIU, UC Berkeley, USA
ZHUN WANG, UC Berkeley, USA
SHI QIU*, UC Berkeley, USA
BO LI, University of Illinois Urbana-Champaign, USA
WENBO GUO, UC Santa Barbara, USA
DAWN SONG, UC Berkeley, USA

AI agents that combine large language models with non-AI system components are rapidly emerging in real-world applications, offering unprecedented automation and flexibility. However, this unprecedented flexibility introduces complex security challenges fundamentally different from those in traditional software systems. This paper presents the first systematic and comprehensive survey of AI agent security, including an analysis of the design space, attack landscape, and defense mechanisms for secure AI agent systems. We further conduct case studies to point out existing gaps in securing agentic AI systems and identify open challenges in this emerging domain. Our work also introduces the first systematic framework for understanding the security risks and defense strategies of AI agents, serving as a foundation for building both secure agentic systems and advancing research in this critical area.

CCS Concepts: • **Security and privacy** → **Systems security**; *Software and application security*; • **Computing methodologies** → **Artificial intelligence**.

Additional Key Words and Phrases: AI agents, large language models, security, prompt injection, systematization of knowledge

1 Introduction

The rapid advancement of agentic AI systems has fundamentally transformed the AI landscape, marking a paradigm shift from isolated language models to integrated hybrid systems that combine Large Language Models (LLMs) with diverse software components. These hybrid systems demonstrate unprecedented capabilities by seamlessly integrating AI reasoning and traditional software, enabling dynamic tool usage and autonomous task execution. Agentic AI systems now power a wide range of applications, from a simple chatbot [48, 111] to software development [26, 30, 47] and web browsing automation [122].

The security implications of agentic AI systems are becoming increasingly critical, as recent incidents demonstrate the severity of their vulnerabilities. Prompt injection attacks have been exploited to access private GitHub repositories [60], while remote code execution vulnerabilities [95, 97] have enabled attackers to gain unauthorized system access. Data exfiltration attacks [125, 128] have compromised sensitive information through malicious document attachments and email forwarding, while servers have exposed user chat and credential data [110, 132]. Attackers are also exploiting web agents to gain access to users' personal banking accounts [17]. These incidents underscore that while the flexibility and automation capabilities make agentic systems powerful,

*Work done during the author's visit to UC Berkeley.

Authors' Contact Information: Juhee Kim, UC Berkeley / Seoul National University, Berkeley, CA, USA, kimjuhi96@snu.ac.kr; Xiaoyuan Liu, UC Berkeley, Berkeley, CA, USA, xiaoyuanliu@berkeley.edu; Zhun Wang, UC Berkeley, Berkeley, CA, USA, zhun.wang@berkeley.edu; Shi Qiu, UC Berkeley, Berkeley, CA, USA, stephenqius@gmail.com; Bo Li, University of Illinois Urbana-Champaign, Champaign, IL, USA, lbo@illinois.edu; Wenbo Guo, UC Santa Barbara, Santa Barbara, CA, USA, henrygwb@ucsb.edu; Dawn Song, UC Berkeley, Berkeley, CA, USA, dawnsong@cs.berkeley.edu.

Extended version of "SoK: Attack and Defense Landscape of Agentic AI Systems".
Conference paper: USENIX Security 2026, pp. 6047-6066.

First published: arXiv:2603.11088v1 [cs.CR] 11 Mar 2026

they also create complex security challenges, which differ fundamentally from those associated with traditional software systems or standalone AI models.

Although existing research has made important contributions to understanding AI agent security, most efforts have focused on specific attack vectors or individual system components, mainly prompt injection attacks [15, 89, 179, 183, 185]. However, these studies lack a comprehensive perspective that considers the defense of agentic systems as a whole. The community needs a systematic framework for understanding how the integration of multiple components introduces novel attack surfaces and demands fundamentally different security approaches.

This paper addresses this gap by providing the first comprehensive systematization of knowledge on the security landscape of agentic AI systems. We approach agent security from a holistic systems perspective, examining how the combination of LLMs and traditional software creates unique security challenges that cannot be mitigated through component-level defenses alone. Our analysis begins with a systematic characterization of agent design dimensions that influence security properties, followed by a comprehensive taxonomy of attack vectors and security risks across the entire agent ecosystem. We then conduct a systematic survey of existing defense mechanisms, categorizing them based on their protection approaches and identifying critical gaps in current security strategies. Finally, we conduct case studies on various real-world agents, including a concrete case of AutoGPT, to further highlight gaps in existing defenses for agentic systems.

Our contributions are threefold:

- **Agent Design Dimensions:** We present a systematic framework that characterizes agentic AI systems through seven key design dimensions: input trust, access sensitivity, workflow, action, memory, tool, and user interface. We then analyze how flexibility along each dimension impacts security risks and map these dimensions to established frameworks, including MITRE ATLAS and OWASP Top 10 for LLM.
- **Comprehensive Attack Landscape and Taxonomy:** We develop a systematic taxonomy of attack vectors organized by threat models (i.e., external, user-level, and internal adversaries) and provide a comprehensive classification of seven security risk categories spanning the CIA triad, along with a system-level analysis of risk interactions and amplification patterns.
- **Defense Landscape Systematization:** We systematically survey existing defense mechanisms and conduct various case studies, identifying specific design dimensions for defense and open challenges.

To the best of our knowledge, this is the first work to systematically analyze the security landscape of agentic AI systems from a comprehensive system perspective. Our systematization provides a foundational framework for understanding security risks and defense strategies in agentic AI systems, guiding future research toward building secure agentic systems. This work serves as a handbook for researchers and developers working with agentic AI systems.

2 Overview

In this section, we define the scope of our work, introduce our methodology, and discuss the key differences from existing agent security surveys.

Scope. We focus on security risks and defenses that are unique to, or significantly amplified in, agentic systems compared to traditional software and standalone LLMs. We first characterize how agents differ from standalone models across seven design dimensions (§3). We then analyze agent-specific security risks both at the component and system levels (§4). For risks that exist in non-agentic LLMs (e.g., jailbreak, hallucination), we emphasize how agent autonomy and environment access magnify their impact (e.g., data exfiltration, unintended system manipulation). We exclude

attacks targeting model internals, such as model inversion [40], as these do not fundamentally worsen in agentic contexts where agents operate purely through inference-time input-output interfaces. Finally, §5 systematizes the agent defense landscape, describing existing approaches and open challenges.

Methodology. First, we define the agent structure and design dimensions based on existing definitions and real-world agent implementations. This forms the foundation of our agent risk taxonomy, which analyzes risks at both the component and system levels. We also refer to the taxonomy of the OWASP Top 10 for LLM Applications [118] and MITRE ATLAS Matrix [98]. We then propose the defense landscape for agentic systems by applying the defense in-depth principle, informed by traditional system security. Under this framework, we conduct a systematic review of academic literature and web documents on agentic AI security from 2023 to October 2025, corresponding to the proliferation of agent systems [178]. We search with keywords across four dimensions: i) general terminology (e.g., *agent security*), ii) OWASP-defined risks (e.g., *prompt injection*, *memory poisoning*), iii) component-centric terms (e.g., *RAG security*, *tool security*), and iv) traditional security adaptations (e.g., *isolation*, *access control*). We prioritize top-tier security venues (e.g., USENIX Security, IEEE S&P, CCS, and NDSS) and ML conferences (e.g., NeurIPS, ICLR, and ICML), alongside high-impact preprints, industry whitepapers, and CVEs. We manually exclude work on standalone models to focus explicitly on agents. The review yields 128 papers, including 51 attack methods and 60 defense methods; the remaining works address both attacks and defenses or focus on case studies without introducing new methods.

Differences From Existing Surveys. Existing surveys on AI and LLM security primarily emphasize model-level threats and overlook the expanded attack surface and downstream consequences introduced by agentic systems [54, 64, 89, 158]. Prior agent-focused studies typically narrow their scope to specific attack vectors [185], agent types [74, 81], or individual components [58], while design-oriented work proposes high-level principles without systematizing attacks or defenses [15, 186]. The closest surveys [33, 181] consider cross-component interactions but largely limit their defense taxonomies to model-based techniques. In contrast, our work provides a unified risk taxonomy covering all agent components and their interactions, together with a defense-in-depth framework that integrates both model-based and system-level defenses, offering actionable guidance for securing agentic systems.

3 Design Landscape of Agentic AI Systems

3.1 Design Components

In this paper, we define AI agents as *hybrid software systems* that combine traditional software components with AI models. An AI agent typically consists of the following components.

LLMs. As the *brain* of the agent, the LLM(s) receive and analyze user tasks, create a step-by-step plan, and take actions following the plan. These processes can be performed by a single or multiple LLMs, each tailored to a specific role.

Memory. Memory stores the agent's internal knowledge base and historical action trajectories. This information is often vectorized for efficient retrieval and can facilitate the agent to make future decisions based on knowledge and prior experience.

Tools. Tools are functions that an agent uses to interact with its external environment. As shown in Figure 1, there are typically retrieval tools and execution tools based on agent's actions. Here, retrieval tools are used to collect information from the external environment (e.g., search or read file), and execution tools are used to make changes to the environment (e.g., write files, send emails,

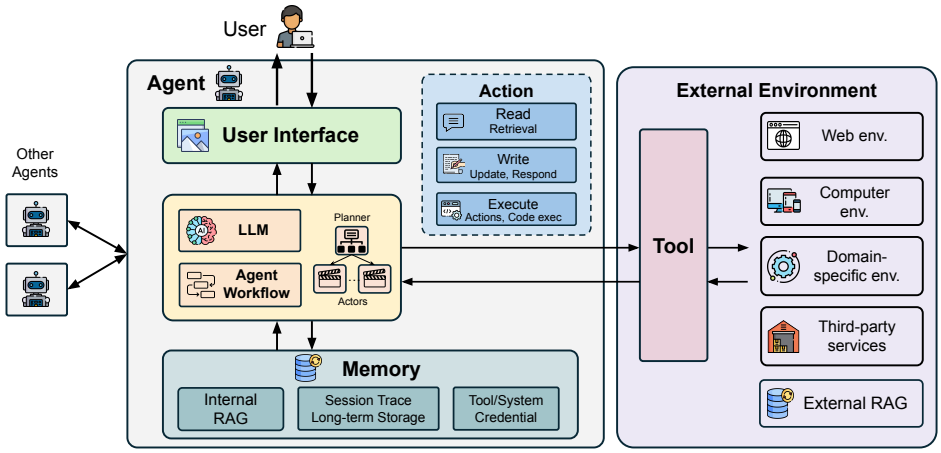

Fig. 1. Overview of an AI Agent's structure.

or run a command). Tools can be implemented through standardized protocols (e.g., MCP [7]). It is worth noting that the tools can be developed by the agent designer or third-party providers [99, 119].

External environment. An external environment is the external context in which an agent interacts to accomplish tasks. Different agents operate in different environments, and the environment largely determines the agent's attack surface and defense priorities. *Web agents* interact with browsers and process arbitrary untrusted content from diverse web sources, making them particularly susceptible to indirect prompt injection and data exfiltration. *Coding agents* operate within IDEs and local file systems, where the primary risks involve unauthorized code execution and workspace manipulation. *Computer use agents* control GUI elements across applications via accessibility APIs or screen-based interfaces, exposing them to UI-based injection and broad system-level access. The agent interacts with its environment (e.g., access, read, and write to a SQL database) through tools.

Overall structure. An AI agent receives a user query and automatically takes a sequence of actions to assist the user. The agent workflow orchestrates this process by coordinating LLM components that typically serve two roles. *Planners* decompose user tasks into step-by-step plans, and *actors* execute individual steps by invoking tools, generating content, or querying memory. In simple agents, a single LLM fulfills both roles. In more complex systems, separate LLMs are dedicated to planning and execution, enabling modular control and finer-grained security boundaries. For example, when asked to update a file, the planner determines the required steps, and the actor sequentially calls a file-read tool, generates new content, and calls a file-write tool to perform the update. This architecture naturally extends to *multi-agent systems* (MAS), where multiple specialized agents collaborate to accomplish complex tasks.

Agentic Systems vs. Traditional Systems. The key uniqueness of agentic systems compared to traditional systems arises in the following aspects. First, agentic systems combine traditional programs with AI model reasoning, whereas traditional systems rely mainly on symbolic logic. This hybrid architecture makes agentic systems more flexible and adaptive. Second, workflows in traditional software are largely pre-programmed, whereas agentic systems can dynamically decide their workflows and actions based on different tasks and *input contexts*. Third, agentic systems use vectorized memory that supports semantic-based retrieval. In contrast, traditional systems use structured memory access through predefined queries.

Table 1. Agent design dimensions and corresponding flexibility for each dimension. The agent design dimensions are conceptually orthogonal, but it practice, their functional implementation can introduce dependencies between them.

Dimension	**Level 1**: Least flexible	**Level 2**: Moderately flexible	**Level 3**: Most flexible
Input Trust	No external data [2, 46, 150]	Predefined external data [77]	Arbitrary external data [99, 119, 173]
Access Sensitivity	No sensitive data [2, 46, 150]	Predefined sensitive data [113]	Arbitrary sensitive data [30, 122]
Workflow	Simple chatbot [2, 46, 150]	Fixed, Developer-defined [68]	Dynamic, LLM-defined [161, 173, 178]
Action	LLM response only [2, 46, 150]	LLM response, Retrieval [77, 113]	LLM response, Retrieval, Execution [30, 122]
Memory	No memory [2, 46, 150]	Transient session memory [71]	Persistent memory across sessions [75, 114]
Tool	No tool [2, 46, 150]	Known tools [133]	Arbitrary tools [119]
User Interface	Text-only [2, 46, 150]	Web-based image preview [44]	Interactable Web elements [43, 112]

3.2 Design Dimensions and Security Implications

Based on the agent structure in Figure 1, we further identify seven agent design dimensions, each of which represents a continuous spectrum of flexibility. In Table 1, we illustrate three representative levels (least, moderately, and most flexible) for simplicity, but agents can operate at any point along this spectrum.

Input Trust. This categorizes the trustworthiness of external data sources on which an agent is relying. It captures the progression from using no external data (highest trust) to relying on arbitrary, potentially untrusted external data sources (lowest trust but highest flexibility). As agents access more diverse and potentially untrusted information sources, they gain improved knowledge and decision-making capabilities, but face increased security risks from compromised or malicious data sources.

Workflow. This refers to the action sequences of the agent and determines who defines these sequences, analogous to the code in traditional software. Flexibility increases as it shifts from no complex workflows (simple chatbots) to developer-defined workflows, and ultimately to LLM-defined dynamic workflows. This progression allows agents to adapt their execution patterns from rigid, predetermined sequences to flexible, context-aware task planning.

Access Sensitivity. This categorizes an agent's level of access to sensitive data within the system or environment. Flexibility increases as agents gain access from no sensitive data to known sensitive data sources, and finally to arbitrary sensitive data. Higher levels enable agents to handle more complex tasks that require sensitive information access while significantly expanding both their operational scope and the potential impact of security breaches. In particular, agents that handle personally identifiable information (PII), such as names, email addresses, financial records, or medical data, face elevated risks of data leakage through unintended tool outputs, logging, or exfiltration attacks.

Action. Action describes what operations an agent can perform beyond text generation. Flexibility increases from response-only, to read-only actions such as retrieval or querying, and eventually to environment-modifying operations including file edits, command execution, or API calls. As actions become more powerful, both task completion and potential security risks grow.

Tool. Tool defines the scope of external tools an agent can invoke. The spectrum begins with no tool usage, expands to a fixed set of curated tools, and culminates in the ability to use or even

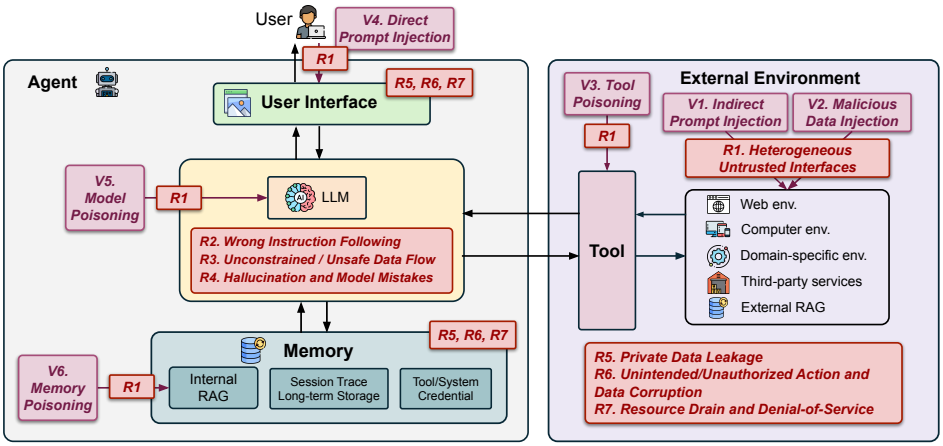

Fig. 2. Demonstrations of attack vectors (V1-V6) and security risks (R1-R7) against AI agents.

select arbitrary tools. Broader tool access enhances functionality but expands the attack surface substantially.

Memory. Memory characterizes how an agent stores and retrieves information over time. Agent ranges from memory-less operation, to transient session-level memory, and finally to persistent memory spanning multiple sessions. Increasing reliance on memory supports personalization and long-term reasoning while introducing risks such as memory poisoning and private data leakage.

User Interface. This dimension defines how users interact with an agent, with flexibility ranging from text-only interactions to graphical interactions through image preview or simple web interfaces, and ultimately to multi-modal interactions such as web, terminal, and integrated development environments (IDEs). Richer interfaces enable more expressive workflows and interactions, but also broaden the range of possible attack vectors.

Security Implication. In general, there is a trade-off between flexibility and security. More specifically, a more flexible agent architecture broadens the attack surface and enables more diverse attack vectors. For example, the attack targets for a simple chatbot are the input data and the model parameters; the sole attack vector runs from user input to model output. In a multi-agent system, however, adversaries can target LLMs, shared memory stores, tools, and the external environment. Attack vectors exist among system components (LLM to tools, memory to LLMs). First, more complex memory designs and user interfaces further enable additional attack opportunities. Second, as the agent's input space becomes more flexible, attackers can more easily inject malicious data or instructions, facilitating poisoning and injection attacks. Third, greater workflow flexibility increases the likelihood of control-flow hijacking, allowing attackers to redirect agent execution and launch various attacks.

4 Attack Landscape of Agentic AI Systems

In this section, we conduct a comprehensive analysis of attacks against agentic systems by identifying key attack vectors (§4.1) and risk taxonomy (§4.2), and by examining how design dimensions correspond to these risks and how different risks interact. Figure 2 provides an overview of the attack landscape.

4.1 Attack Vectors

We discuss attack vectors in AI agents under three threat models, classified based on attackers' access to the agent at the time of attack execution rather than during the attack's development. Assumptions specific to the attack development are considered when we describe attack methods (§4.4).

External Adversary. An attacker is in an external environment and cannot directly interact with an agent, but the attacker can manipulate external resources that the agent may retrieve and process. This threat model represents the most constrained yet highly realistic scenario.

V1. Indirect prompt injection [53, 60, 73, 83, 160, 167]: attackers inject malicious instructions into the external environment that the agent interacts with, such as public web pages or documents. When the agent retrieves content from this environment, it may also retrieve and execute the malicious instructions.

V2. Malicious data injection [120, 147]: The attacker can inject malicious non-prompt data. When such data is consumed during sensitive operations, it can trigger security failures and breaches. Examples include malicious software packages [147] or attacker-supplied values for sensitive financial parameters [120].

V3. Tool poisoning and manipulation [108, 138]: Attackers inject malicious instructions into the names or descriptions of external tools that agents interact with. They can also inject malicious payloads into tool implementations.

User-level Adversary. Here, attackers have access to the agent inputs and can directly feed malicious contents to the agent [88] or inject them into otherwise benign user inputs [41, 42].

V4. Direct prompt injection [41, 42, 88]: Attackers can control parts of otherwise benign inputs and append malicious instructions to user inputs.

Internal Adversary. Attackers can access some or all components inside the agent, which represents the strongest assumption. This attack vector poses the most severe threat, as attackers can control the agent's internals, but it is less practical.

V5. Model poisoning [159, 175]: The attacker injects a backdoor into the LLM, which can be activated during inference to enable malicious behavior.

V6. Memory poisoning [24, 37, 189, 195]: An attacker can directly manipulate the agent's memory to inject malicious instructions or false knowledge. Alternatively, the attacker can leak sensitive user data from the memory.

4.2 Security Risks

We present a comprehensive taxonomy of security risks in agents, categorized by components that can be targeted by different attack vectors. This taxonomy provides a systematic map of the entire security risk landscape for agentic systems.

R1. Heterogeneous untrusted interfaces. Compared to standalone LLMs, agentic systems expose multiple heterogeneous interfaces to users, including external data sources that agents retrieve and process, persistent memory stores that accumulate over time, and third-party tools with various trust levels. These interfaces can be leveraged by attackers as attack entry points, which introduce way larger attack surfaces compared to standalone LLMs.

R2. Wrong instruction following. The agent follows malicious prompts injected by attackers rather than the intended instructions from benign users or system developers. This occurs because inputs from external or attacker-controlled sources are processed by the model inside the agent and can therefore influence its behavior.

R3. Unconstrained/unsafe data flow. LLMs suffer from unconstrained data flow due to their stochastic nature. In agentic systems, this manifests as data flowing freely from any input from untrusted interfaces (R1) to any output (e.g., user responses or subsequent tool calls), unlike traditional systems where data propagation is regulated by programming languages, type systems, and access controls.

This leads to consequential risks: private data leakage (R5), data corruption (R6), and resource drain (R7). For example, when an agent retrieves untrusted web content and produces a URL in its response, that URL may contain attacker-controlled data. Automatically fetching such URLs enables data exfiltration [96, 128], resulting in R5.

R4. Hallucinations and model mistakes. Models often hallucinate and generate incorrect information, and this problem becomes more severe in agent-environment interactions because agents act on hallucinated content, creating real-world consequences beyond misinformation, such as accessing attacker-controlled resources. Attackers exploit this behavior through package hallucination attacks [147], where they register malicious packages with names that LLMs frequently hallucinate. This turns a model limitation into a reliable attack vector for code injection and supply chain compromise.

R5. Private data leakage. Agents handle sensitive data across multiple components such as user conversations, persistent memory, tool credentials, and environment resources, creating opportunities for unauthorized data access. Attackers exploit interactions among these components to exfiltrate private information. Examples include manipulating agents to send data to attacker-controlled servers, triggering automatic URL fetches that leak data through request parameters, or abusing domain-specific vulnerabilities such as SSRF, XSS, path traversal, or SQL injection to reach protected resources. Recent incidents [101, 110, 132] show real confidentiality violations where agent vulnerabilities exposed users' chat histories and personal information.

R6. Unintended/unauthorized action and data corruption. Agents can violate integrity through two closely related risks, unintended and unauthorized actions, and data corruption, both involving unauthorized changes to internal or external state. Unintended actions make irreversible state changes (e.g., unauthorized purchases, arbitrary code execution), while data corruption directly modifies stored resources (e.g., corrupting files or databases). At the user interface, agents may provide false or misleading information. At memory components, agents may inject poisoned knowledge or malicious instructions [24] that corrupt subsequent behavior. At the environment level, agents can be manipulated to execute unauthorized modifications through command injection, SQL injection, or file system manipulation.

R7. Resource drain and denial-of-service. Agents introduce availability risks through their consumption of computational resources, API calls, and interactions with external systems [70]. Attackers exploit this autonomy to trigger costly API calls, force infinite execution loops, or cause excessive memory use, creating denial-of-service conditions that can render the agent and external systems unusable or economically unsustainable.

Other risks in agentic systems. Attackers can also target the agentic AI system itself, rather than the user or surrounding environment, by stealing system prompts, tool descriptions, and configurations that encode proprietary knowledge [137, 176], undermining agent developers and tool providers. Agentic systems may also expose side-channel risks through observable behaviors such as tool execution timing, API call sequences, or network traffic patterns. To the best of our knowledge, we are not aware of prior work studying such side-channel attacks in agentic systems, and thus defer deeper analysis to future work.

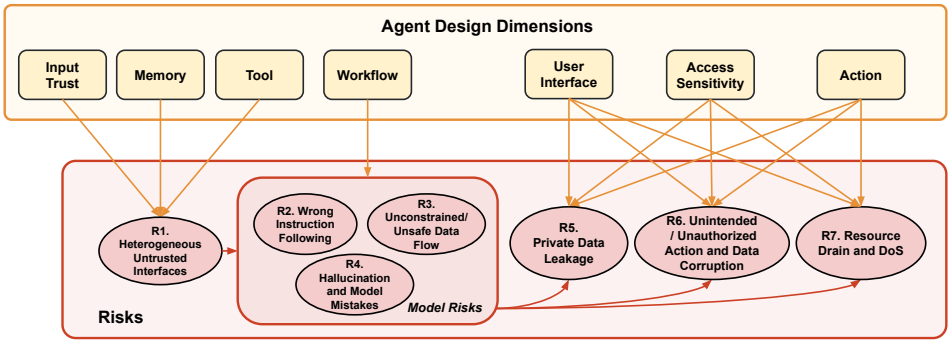

Fig. 3. Relationship between agent design dimensions and risks, and the connection across different risks.

From Attack Vectors To Risks. These risks can be exploited via various attack vectors under different threat models. The wrong instruction following risk (R2) can be triggered by all attack vectors and carried out in all threat models. Private data leakage (R5) can result from indirect prompt injection (V1), data injection (V2), or memory poisoning (V6). Unauthorized actions and data corruption (R6) are commonly executed through indirect prompt injection (V1) but can also be exploited through direct injection (V4), tool poisoning (V3), or model poisoning (V5). Resource drain (R7) can be triggered through any external or user-level vector.

4.3 System-level Analysis of Agent Risks

Figure 3 illustrates how agent design dimensions map to security risks and how these risks interact to amplify system-level threats.

Design Dimensions to Risks. Agent design dimensions map to distinct risk categories, and we observe that greater flexibility amplifies security risks (Figure 3). The Input Trust, Memory, and Tool dimensions directly contribute to increased attack surface (R1). Expanding external data sources enables indirect prompt injection, adding persistent memory creates memory poisoning targets, and incorporating third-party tools introduces supply chain risks.

The Workflow dimension primarily drives model risks (R2, R3, R4). As workflows shift from simple chatbots to LLM-defined dynamic execution, flexible control flows provide more opportunities for hijacking agent reasoning, worsening wrong instruction following and unconstrained data flow. Hallucination risks amplify when agents dynamically select tools and resources, as incorrect model outputs directly trigger real-world actions.

The Access Sensitivity, Action, and User Interface dimensions determine the severity of consequence risks (R5, R6, R7). Granting agents access to more sensitive data amplifies the impact of violations, as compromised agents can leak, corrupt, or drain more valuable resources. Expanding action capabilities from response-only to execution transforms information disclosure into environment corruption. Complex user interfaces introduce new attack channels through automatic URL fetching (confidentiality), executing destructive commands (integrity), and interface manipulation (availability).

Risk Interactions and Amplification. Risks in agentic systems (R1-R7) interact in a cascading manner, where initial failures propagate across components and amplify system-level threats. An expanded attack surface (R1) increases entry points for attacker-controlled data, allowing malicious inputs to reach and exploit model risks (R2, R3, R4). For example, indirect prompt injection through external data can trigger wrong instruction following, which then redirects agent

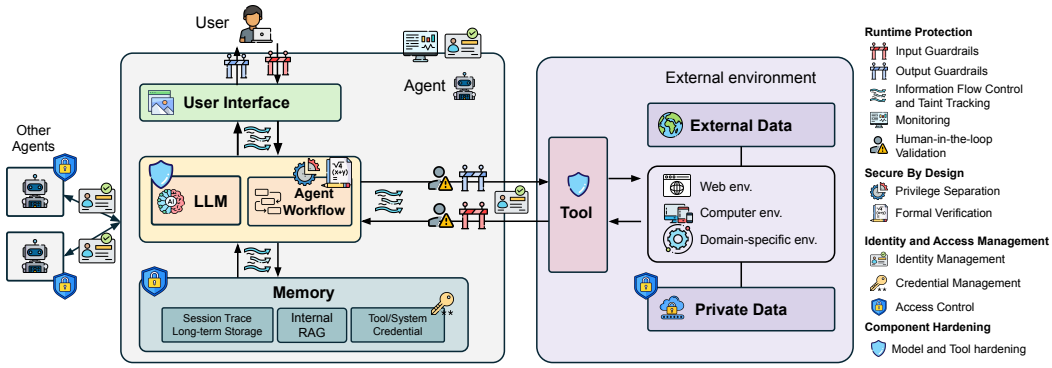

Fig. 4. The defense landscape of AI agents. We illustrate key defense mechanisms and where they can be applied within the agent system.

behavior. Model risks then amplify consequence risks (R5, R6, R7). Wrong instruction following can lead to data exfiltration (confidentiality), data corruption (integrity), or excessive resource consumption (availability). Unconstrained data flow heightens confidentiality and integrity risks by enabling data leakage and malicious code execution. Hallucination increases integrity risks by causing agents to operate on fabricated resources, leading to data leakage and corruption. As a concrete example, EchoLeak [96, 125] demonstrates how a malicious document embedded in an enterprise email exploits heterogeneous untrusted interfaces (R1), triggers unconstrained data flow through the agent's retrieval pipeline (R3), and ultimately exfiltrates sensitive user data to an attacker-controlled server (R5), all without any user interaction.

4.4 Attack Methods

Attack methods refer to the techniques that construct attack paths and generate attack payloads. Due to the attack complexity, most existing attacks heavily rely on human efforts to construct attack paths and payloads [95, 96]. For example, the injection points for just-in-time injection and memory poisoning attacks are almost always set manually. The attack payloads (i.e., malicious instructions) for prompt injection attacks are mainly generated based on pre-specified attack patterns, such as role-playing scenarios [32], delimiter confusion using special characters [164, 165], or instruction reset commands that attempt to ask LLMs to ignore previous context [121, 134].

Recent research has started to explore automated methods for generating attack payloads and injection points. For attack payload, prompt injection attacks design specific fuzzing approaches for AI agents [160, 180], as well as training small attack models for malicious instruction generation [83, 167, 194]. Memory poisoning techniques employ several approaches focused on maximizing retrieval likelihood while maintaining content credibility. Semantic injection represents the most systematically studied approach, crafting factually incorrect content with high semantic similarity to target queries through embedding optimization techniques. This method leverages contrastive learning principles to position malicious content close to legitimate queries in the embedding space, ensuring preferential retrieval by vector similarity search mechanisms [195]. Advanced variants employ gradient-based optimization to craft adversarial passages that achieve optimal embedding similarity while maintaining semantic coherence [87, 184, 189]. In general, the community can benefit from more automated end-to-end attack/red-teaming methods for agents, which can be used as in-house testing tools by agent developers.

5 Defense Landscape of Agentic AI Systems

The comprehensive defense landscape of AI agents has been largely underexplored. We put together the defense landscape by examining security goals (§5.1) and identifying key defense mechanisms in different categories (§5.2,§5.3,§5.4,§5.5). For each category, we analyze their design dimensions and open challenges. Finally, we discuss defense design principles (§5.6). Table 2 summarizes the defense mechanisms for AI agents and the risks that each defense covers, organized by category. Figure 4 illustrates the defense landscape.

5.1 Security Goals

In this section, we discuss the security goals for AI agents. They are based on standard security principles, specifically the *Confidentiality*, *Integrity*, and *Availability* (CIA) triad that forms the foundation of traditional security frameworks. Additionally, we introduce *Contextual Security* as a new security goal for agentic systems.

Confidentiality. Confidentiality ensures that all information is accessible only to authorized entities. It includes protecting system-level secrets (e.g., API keys, credentials), the agent's internal memory, users' private data, and LLM-related data (model parameters and system prompts). Achieving this goal mitigates risks such as unsafe data flow (R3), private data leakage (R5), and unconstrained data flows that expose credentials or sensitive information.

Integrity. Integrity ensures that data and control flows within an agentic system and its external environment remain trustworthy and unaltered by unauthorized entities. It includes preventing tampering with the agent's memory, LLM outputs, tool results, and environmental data, as well as ensuring that agents are not manipulated into performing unintended actions by malicious instructions. Specific security goals associated with integrity can vary across agent types, depending on their components, data, and control flows. Achieving this goal addresses unsafe data flow (R3), hallucinations and model mistakes (R4), wrong instruction following (R2), and unintended/unauthorized actions and data corruption (R6).

Availability. Availability protects the system from denial-of-service attacks and resource abuse (R7), even when LLM inference or tools consume user resources. This includes preventing token draining of LLM, as well as regulating the usage of resources in agent hosts and external system.

Contextual Security. Recent studies define contextual security [139, 152] as a critical security goal for AI agents. Contextual security ensures that the agent's contexts are aligned with the agent's intended user tasks, preventing attacks aimed at manipulating contexts during agent execution. It governs which context elements (e.g., system prompts, user goals, tool descriptions, retrieved snippets) are admissible and how they are prioritized to avoid instruction override, context drift, or unsafe tool selection, complementing confidentiality and integrity that protect data correctness and secrecy.

Contextual security stems from *contextual integrity* [12], a privacy framework that defines appropriate information flows based on social norms and context. Contextual integrity has been applied to various software domains, including general programs [142], mobile applications [163], and IoT systems [65]. In the agentic setting, AirGapAgent [11] applies contextual integrity to ensure that agents only access information relevant to the current task context. However, a recent position paper [141] finds that many works adopting contextual integrity fail to fully follow its principles, simplifying information flows and inadequately addressing the broader context and norms that contextual integrity requires. In AI agents, open-ended inputs and diverse environments further complicate defining such contexts and norms. While contextual integrity primarily addresses privacy concerns, contextual security extends this principle to security by ensuring that the agent's actions remain aligned with user intent and free from adversarial manipulation.

Table 2. Defense mechanisms for AI agents organized by category, as well as the risks each defense covers.

Category	Mechanism	Representative works	Covered risks
Runtime Protection	**Input guardrail**	Prompt Guard2 [25], PromptShield [61], DataSentinel [90], PromptArmor [140], URL allowlist [51]	R1. Heterogeneous Untrusted Interfaces
	Output guardrail	CodeShield [93], Firewalled Agentic Networks [1], Task Shield [63], GuardAgent [172], Agrail [92], AlignmentCheck [25], ControlValve [62], Conseca [152], Progent [139], Maris [29],	R2. Wrong Instruction Following R4. Hallucination and Model Mistakes R5. Private Data Leakage R6. Unintended/Unauthorized Action and Data Corruption
	Information flow control and taint tracking	Permissive [143], MELON [192], RTBAS [188], AgentArmor [156], PFI [67], CaMeL [31], FIDES [28], ACE [78], PrivacyChecker [157]	R3. Unconstrained/Unsafe Data Flow R5. Private Data Leakage R6. Unintended/Unauthorized Action and Data Corruption
	Monitoring	AgentAuditor [91], AgentMonitor [103], SentinelAgent [57], Guardian [191]	R5. Private Data Leakage R6. Unintended/Unauthorized Action and Data Corruption R7. Resource Drain and Denial-of-Service
	Human-in-the-loop	Wu et al. [171]	
Secure By Design	**Privilege separation**	AirGapAgent [11], f-secure [168], CaMeL [31], FIDES [28], PFI [67], IsolateGPT [170], ACE [78], IPIGuard [5], DRIFT [79]	R2. Wrong Instruction Following R3. Unconstrained/Unsafe Data Flow
	Formal verification	Formal-LLM [82], VeriSafeAgent [76], ShieldAgent [23]	
Identity and Access Management	**Identity management**	Authenticated delegation [146], SAGA [148], Agent network protocol [20]	R5. Private Data Leakage R6. Unintended/Unauthorized Action and Data Corruption R7. Resource Drain and Denial-of-Service
	Access control	ControlNet [177], Honeybee [187], Bedrock [4], Authenticated delegation [146], SAGA [148]	
	Credential management	Token vault [10]	R5. Private Data Leakage
Component Hardening	**Model hardening**	SecAlign [22], StruQ [21], Instruction Hierarchy [155], InstructionalAgent [169]	R2. Wrong Instruction Following R4. Hallucination and Model Mistakes
	Tool hardening	Anthropic Connectors [8], ETDI [36], MCP Context Protector [151], MCP Safety Audit [124], MCIP [66]	R1. Heterogeneous Untrusted Interfaces

5.2 Runtime Protection

Runtime protection mechanisms provide dynamic security enforcement during agent execution, detecting real-time threats and behaviors.

5.2.1 Input Guardrail. Input guardrails validate and sanitize possible input dimensions of agents, such as user input, tool retrieval results, and memory data. They provide a *first-line defense* against attack vectors by external and user-level adversaries, preventing malicious instructions and data from reaching their agent internals. Input guardrails apply to both standalone models and agents.

For standalone models, input guardrails focus on detecting malicious inputs such as jailbreak prompts or harmful content [25, 61, 90, 136, 140]. These techniques remain effective in agentic settings to validate various inputs, especially those from external environments. Specifically to agents, input guardrails address risks introduced by dynamic data retrieval and tool execution, verifying the trustworthiness of retrieved data. For instance, agents enforce URL allowlists [51] to constrain data retrieval to trusted sources, mitigating risks from untrusted web content.

Design Dimensions. Input guardrails can be characterized along three design dimensions: *detection mechanism*, *validation target*, and *mitigation strategy*. First, detection mechanisms face a fundamental tradeoff between security strictness and operational flexibility. Rule-based detection offers strict but inflexible protection through predefined patterns [126] or endpoint allowlists [50, 51]. Model-based detection, on the other hand, trains small models [25, 61, 90, 136] or prompts LLMs [138] to detect or filter out malicious prompts. In general, LLMs are more generalizable and effective than small models in defense capabilities, but they also introduce more overhead and latency [158]. Second, validation target specifies what aspect of input data is being inspected. Content-based guardrails [25, 61, 136, 140] examine semantic input for malicious patterns or harmful content. In contrast, source-based guardrails [51] verify data origin trustworthiness, uniquely addressing agents' dynamic retrieval from external sources like web search and databases. Third, mitigation strategy defines how detected threats are handled. Guardrails can simply detect and filter out malicious input [25, 61, 90, 136], sanitize the malicious portion before incorporating the input into the system [140], or normalize input in a structured format with type system and validation [193] to reduce the attack surface. Alternatively, some model-based approaches neutralize threats without explicit detection by perturbing or smoothing inputs to remove adversarial perturbations [129, 190].

Limitations and Open Challenges. As the input space becomes vast and diverse, it is challenging to establish universal security criteria. Model-based detectors are often bypassed by adaptive attacks [6, 105], and rule-based detectors require extensive human effort and are difficult to generalize. Consequently, input guardrails often suffer from false positives and false negatives, which negatively impact the agent's utility and security. False positives are caused partly by the limited detection accuracy, but more importantly, by ambiguous definitions and boundaries between secure and insecure data and instructions. False negatives allow attackers to bypass input guardrails and inject malicious inputs into the target agent.

5.2.2 Output Guardrail. Output guardrails perform security checks on outbound results of agents, such as responses to users and tool invocations that interact with external systems. They can complement input guardrails by preventing attacks that appear benign from input prompts but result in malicious actions [96]. Output guardrails for LLMs typically focus on detecting harmful outputs, using classifiers or programmable rules [93, 100, 126, 136]. Agent-specific output guardrails validate tool usage and action sequences [23, 63, 92, 139, 152, 172], ensuring context-dependent policies and alignment with the user intent. In multi-agent settings, output guardrails can also enforce inter-agent communication policies [1] and permitted control-flow graphs that prevent unauthorized agent transitions [62].

Design Dimensions. Similar to input guardrails, output guardrails can be characterized along two design dimensions: *detection goal* and *detection mechanism*.

First, output guardrails detect harmful content in LLM outputs [126, 136], unsafe code [93, 100], and unsafe actions in tool usage [139, 152]. Further, output guardrails perform alignment checks to ensure agent's behavior remain within user intent [25, 63], data privacy protection [29], domain-specific policy enforcement [23, 92, 172], and runtime-aware contextual security enforcement [139, 152].

Second, mechanisms of output guardrails range from rule-based pattern matching [93, 100] to model-based classifiers [25, 126, 136]. Hybrid approaches adopt a structured policy framework assisted by models to process unstructured data, enabling flexible security enforcement [23, 29, 92, 139, 152, 172].

Limitations and Open Challenges. Output guardrails share similar limitations with input guardrails, including false positives and negatives due to unclear security criteria, and a trade-off between rule-based and model-based solutions. Compared to input guardrails, output guardrails consume more computational resources and time, as they have a dependency on the LLM's output and need to process more data [158]. For guardrail methods, it is critical to balance the trade-off between security and utility by minimizing the latency and reducing guardrail false positives.

5.2.3 Information Flow Control and Taint Tracking. Information Flow Control (IFC) [13, 16, 34, 102] and taint tracking [107] restrict the data flow of information within a system. At a high level, such methods assign each data a security label from a predefined information flow lattice [34] and propagate it along the agent execution, detecting unsafe information flows that violate the lattice constraints.

Design Dimensions. IFC and taint tracking designs vary across *security goals* and *mechanisms*, spanning non-agentic LLM outputs and agent tool executions.

First, previous works provide integrity and confidentiality protection in agents. Integrity protection blocks untrusted inputs from influencing tool call decisions [67, 188, 192]. Confidentiality protection prevents sensitive data from reaching untrusted sinks [28, 31, 78, 157].

Second, IFC and taint tracking mechanisms include symbolic variable-based, multi-execution-based, and model-based approaches. Multi-execution [143, 192] measures the influence of an input on an output by performing the LLM inference multiple times with and without the input. Variable-based approaches replace data with trackable variables to mitigate over-tainting while retaining deterministic information flow guarantee [28, 31, 67, 78]. Model-based approaches request LLMs to inspect information flow given agent traces [80, 156, 157, 188].

Limitations and Open Challenges. Existing IFC and taint methods incur substantial runtime overhead with multi-execution or variable-based reasoning, limiting practicality for latency-sensitive agents. They may also suffer from label creep, where conservative propagation renders agents unusable unless automated and safe declassification rules are devised. Bridging these gaps requires lightweight information flow tracking and principled policies for relaxing security labels when it is safe to do so, without compromising security.

5.2.4 Monitoring. Under the dynamic and unpredictable nature of AI agents, monitoring offers system-wide visibility by checking inputs, outputs, and intermediate states. This holistic view helps surface distributed threats where no single input or action appears malicious in isolation but collectively constitutes a malicious goal [162, 182]. Monitoring can be important to observe interactions across tools and services over long runs, especially for multi-agent systems.

Design Dimensions. Monitoring design varies across *detection goals* and *log granularity*.

First, monitoring systems target different threat categories, including anomaly detection over long agent trajectories [57, 91, 103], and interaction-graph monitoring in multi-agent systems [191].

Second, agent activity log ranges from coarse summaries of actions and tool calls to fine-grained traces that capture intermediate reasoning steps and parameters. Finer granularity improves detection power but increases storage, computation, and privacy exposure [19].

Limitations and Open Challenges. While monitoring techniques provide a holistic view of agent execution, they suffer from fundamental limitations of runtime defenses, such as inaccuracy issues and performance overhead. Agent behaviors are stochastic and context-dependent, making

it hard to distinguish benign actions from harmful ones. Static rules miss novel attacks, whereas adaptive models incur overhead and remain susceptible to evasion. Moreover, long-lived executions would further accumulate logs, for which storage overhead and privacy controls remain largely unexplored.

5.2.5 Human-In-The-Loop Validation. Traditional security employs user consent mechanisms for app installation [38] and sensitive data access [9]. In agentic settings, human-in-the-loop validation allows users to validate agent behavior and tool usage, providing user-customized control over security decisions. While limited literature has been dedicated to discussing human-in-the-loop validation for agents, contemporary coding agents such as GitHub Copilot [47], Gemini Code Assist [26], Cursor [30], and Codex [115] ask for user approval when writing a file or executing command-line commands. Agent defense systems [28, 31, 67, 170] often leverage human-in-the-loop validation when agents attempt actions that violate defense policies.

Design Dimensions. Human-in-the-loop has three design dimensions: *validation scope*, *user alert*, and *recurrence policy*.

First, the *validation scope* defines the scope of actions for which the agent requests user approval. Existing coding agents typically require approval for terminal commands, file accesses outside the current workspace, or destructive actions like file deletion.

Second, the *user alert* provides context to support user approval decisions. Effective alerts should clearly explain the agent's intended action, associated risks, and potential consequences to support informed decision-making.

Third, the *recurrence policy* determines how often alerts are presented. To reduce approval frequency and decision fatigue, the agent can offer an option to remember the user's choice. Similar to mobile permission systems [38], these options can include "allow once" (single-use permission), "allow never" (permanent denial), and "allow always" (persistent authorization). Recent work [171] utilizes machine learning to model user preferences of permission approvals in tool-use agents and reduces the frequency of validation prompts by predicting user decisions.

Limitations and Open Challenges. Frequent validation prompts can overwhelm users and cause decision fatigue that undermines the intended safety benefits. Current alert mechanisms often assume a level of security literacy that many users do not possess, causing them to either blindly approve risky actions or overreact to benign ones. Practical systems require principled criteria for when to defer to human judgment, along with informative yet lightweight explanations that keep users engaged without overburdening them.

5.3 Secure By Design

Secure-by-design defenses establish security properties at the architectural level, making agents intrinsically secure through fundamental design principles. As secure-by-design mechanisms depend on a system's architecture and agents differ fundamentally from traditional systems, these defenses are naturally unique to agentic systems.

5.3.1 Privilege Separation. Following the principle of least privilege, traditional security enforces privilege separation by assigning different privilege levels to different software components, exemplified by earlier automation attempts [18]. In agents, this concept extends to assigning privileges to different components and isolating these components to minimize the overall risk to the system.

Design Dimensions. Privilege separation designs vary by *separation policy* and *scope*.

A *separation policy* can be designed in vertical and horizontal directions. Vertical separation divides components into hierarchical privilege levels, where higher-privilege components (e.g., trusted planners) have more authority than lower-privilege components (e.g., untrusted data

processors). Similar to kernel-user separation in operating systems, planner-processor separation designs [5, 28, 31, 67, 79, 166, 168] isolate tool call planning (high privilege) from tool result processing (low privilege). Memory minimization [11] separates the data minimizer (high privilege) from the untrusted data processing unit (low privilege). Horizontal separation partitions the system into parallel components with equal privileges but isolated access scopes. For example, per-application or per-functionality agents [78, 170] each interact only with their dedicated tools and resources, preventing cross-domain exploitation.

The *scope* of privilege separation indicates the component being separated. An LLM can be separated into planning tool calls and processing results [28, 31, 67, 168], preventing tool results from directly influencing planning decisions. Memory separation [11] creates separate memory spaces, protecting sensitive data from lower-privilege components. Agents can separate external environments to enforce least-privilege [67, 78, 170], leveraging sandboxing, containerization [84–86], and access tokens [52, 145].

Limitations and Open Challenges. Current privilege separation research primarily addresses generic indirect prompt injection in simplified environments [32]. It often fails to address real-world environment risks that present diverse and complex challenges requiring specialized separation strategies, such as web, file systems, and databases. Privilege separation entails utility loss by splitting functionality across isolated components. Dividing an agent into an effective set of least-privilege components remains an open challenge. Designing efficient communication across isolated components to reduce utility loss while maintaining security guarantees presents ongoing challenges for practical deployment.

5.3.2 Provable Security with Formal Verification. Traditional security methods employ formal verification to provide theoretical proofs of correctness and security [56, 69]. Formal verification for agentic AI systems remains an emerging but crucial research frontier that aims to bridge symbolic assurance with non-symbolic behavior modeling.

Design Dimensions. Formal verification approaches vary across *formalization target* and *security property*. First, VeriSafe Agent [76] formalizes user intent into a DSL over UI state transitions, verifying that proposed GUI actions align with the user's task before execution. Formal-LLM [82] encodes developer-defined plan constraints (e.g., required tool orderings) as pushdown automata to restrict plan generation, ensuring validity and executability. Second, various security properties are formally verified in agent systems, including predefined safety constraints [23], alignment with user tasks [76], and the correctness of agent behavior with respect to specified function requirements and expected outputs [82].

Limitations and Open Challenges. Traditional formal methods are designed for programs written in code with structured language, while LLM-based agents operate based on probabilistic models. This fundamental difference creates challenges in developing formal models that capture the stochastic behavior of agents while preserving meaningful security guarantees. Moreover, security properties of agents that interact with various environments, such as web, file systems, or mobile applications, remain underspecified. Developing formal verification frameworks capable of addressing the full spectrum of agent security requirements largely remains an open challenge. To develop formal verification for agentic systems, automation is critical, and frontier AI can help with the process, such as automating the specification generation [174].

5.4 Identity and Access Management

Identity and Access Management (IAM) encompasses identity management, access control, and credential management to ensure authenticated entity and authorized resource access. Traditional systems employ well-established IAM frameworks such as role-based access control (RBAC) [131]

and OAuth-based delegation [39, 59], operating with static user identities and predefined permission boundaries. Agentic systems interact with real-world services on behalf of users, thus requiring agent-specific identities, delegation mechanisms, and dynamic access control policies that adapt to runtime contexts.

5.4.1 Identity Management. Identity management ensures that each actor operates under the correct identity through user authentication and authorization. It is a prerequisite for access control (§5.4.2), as proper identity authentication is required before granting access to the resource. The identity management of agents should support delegation, auditability, and regulatory accountability, and align with existing standards [39, 59, 154].

Design Dimensions. Identity management varies by *architecture*, *scope*, and *delegation model*.

First, identity management can be centralized or decentralized. Centralized approaches [146, 148] rely on a central registry or identity providers [27, 109] running on OpenID Connect [39]. Decentralized approaches, on the other hand, use distributed verification protocols for identity verification or authentication. For instance, agent Network Protocol (ANP) [20] utilizes decentralized identity authentication based on the W3C Decentralized Identifier (DID) standard [154], and Microsoft Verified ID [94] enables peer-to-peer identity verification. Decentralized identifiers provide autonomy and resilience compared to centralized identity management, while requiring more complex implementations and coordination mechanisms.

Second, the scope of identity defines the principal, the entity being authenticated and authorized. Identity can be defined at user-level, agent-level, or task-level, each providing different granularity and accountability. User-level identity ties actions directly to a human user, agent-level identity assigns distinct identities to individual agents [19], and task-level identity creates short-lived identities for specific tasks or sessions to support dynamic, least-privilege operation.

Third, *delegation model* governs how authority flows from users to agents. Direct delegation grants specific user permissions to agents, proxy delegation uses intermediary service or tokens to let agent act on behalf of users, and temporary delegation provides time-limited access that automatically expires to limit risk.

Limitations and Open Challenges. Foundational questions persist about whether agent actions should be attributed to the human operator, the agent instance, or transient task identities, and the answer often varies across domains. Without standard frameworks, platforms implement incompatible credential issuance, delegation, and revocation flows that are hard to audit or federate. Robust identity management will require interoperable taxonomies and lifecycle tooling that preserve accountability while supporting seamless agent collaboration.

5.4.2 Access Control. Traditional systems protect private resources with access control by restricting access to authorized entities only. In AI agents, an agent user's private resources reside in memory (e.g., agent usage histories and personalized knowledge bases) and the environment (e.g., file systems, cloud drives). Recent research proposes a few methods to enforce access controls for these resources by constraining the tools and data sources the agent can access.

Design Dimensions. Access control designs can have different *mechanisms*, *policies*, and *resource scope*.

First, access control *mechanisms* determine how permissions are enforced, such as role-based access control (RBAC) [177, 187], attribute-based access control (ABAC) [4], and capability-based security systems. Specifically, for vector database access control, access can be authorized by model activation patterns [177], outputs can be filtered [4], or the database can be partitioned to expose only accessible entries [187]. When an agent accesses an external web service, existing mechanisms such as API key-based authentication or OAuth 2.0 protocol [59] can be utilized, paired with a

secure delegation protocol [146]. In multi-agent systems, managing access control for sub-agents is also important to prevent confused-deputy attacks [55], where an agent illegally gains a privilege via another agent's capability. For example, a recent study [148] proposed a cryptographic protocol to enforce user-defined policies for multi-agent communications, controlling inter-agent access permissions.

Second, access control *policies* define when access is granted, ranging from static permission rules to dynamic policies that adapt to changing contexts, user tasks, and environmental state.

Third, access control can target different *resource scope* including agent internal memory, external databases, tool APIs, and file systems.

Limitations and Open Challenges. Existing access control work primarily targets retrieval-augmented LLM applications with vector databases [4, 177, 187], but agentic systems require broader coverage of diverse tools, data sources, and inter-agent interactions. Current deployments lack adaptive policy frameworks that can dynamically adjust to evolving tasks and trust contexts, instead relying on ad hoc, non-uniform policies that create mismatches across agents and gaps when coordinating with non-agentic services [127]. Usability challenges further exacerbate the problem, as configuration complexity leads to misconfigurations and excessive privileges even for technical users.

5.4.3 Credential Management. AI agents must manage diverse credential types to interact with external services and access user resources. These include tool API credentials (e.g., API keys and access tokens for third-party services) and environmental credentials (e.g., one-time passwords from email, session tokens). The agent's exposure to these sensitive credentials raises significant privacy and security concerns [106, 149], necessitating robust credential and secret management practices.

Design Dimensions. Credential management approaches span *confidential storage*, *lifecycle management*, and *credential provisioning*.

First, *confidential storage* protects credentials through various storage mechanisms. Encrypted storage protects credentials at rest, preventing unauthorized access even if the storage media is compromised. Temporary storage minimizes exposure by maintaining credentials only for the duration of active sessions, exemplified by OpenAI's temporary chat feature [116] that prevents chat history storage and model training usage. Dedicated credential vaults store encrypted tokens separately from agent code, so that credentials are never directly exposed to the agent [10]. Confidential computing techniques can be leveraged to securely manage credentials within hardware-based trusted execution environments [72, 135].

Second, *lifecycle management* determines how credentials are maintained over time, ranging from static credentials that persist throughout agent sessions to dynamic time-limited tokens that automatically expire, reducing exposure windows.

Third, *credential provisioning* determines how agents obtain credentials, including single-sign-on (SSO) mechanisms [27] and authenticated delegation based on OAuth 2.0 [146] for secure credential transfer from users to agents.

Limitations and Open Challenges. Agent stacks still rely on ad-hoc secret handling, lacking standardized practices for credential protection. Many frameworks guide developers to store internal credentials, such as API keys, as unencrypted environment variables, increasing the risk of leakage. Multi-agent workflows further complicate secret sharing making it difficult to maintain least privilege and traceability, and underscoring the need for coordinated credential orchestration.

5.5 Component Hardening

Component hardening strengthens individual agent components, i.e., models and tools, against their specific vulnerabilities. It follows the principle that a system is only as secure as its weakest component.

Model Hardening. SecAlign [22] and StruQ [21] fine-tune models to consistently follow initial instructions even when faced with conflicting directives, mitigating incorrect or unintended instruction following. Instruction-hierarchy-aware model training [155, 169] ensures that system prompts maintain priority over user input and external data.

Tool Hardening. Extended Tool Definition Interface (ETDI) [36] implements cryptographically signed and versioned tool control metadata, ensuring integrity throughout the tool lifecycle to prevent tool poisoning. MCP Context Protector [151] creates an MCP proxy that enforces manual review processes and applies guardrail checks on tool descriptions and responses. MCP Safety Audit [124] introduces systematic protocols to examine agent tools, identifying potentially exploitable behaviors from malicious logic or misleading descriptions. MCIP [66] enhances MCP with observability and an LLM fine-tuned to detect threats in MCP usage. Anthropic's Connectors Directory [8] maintains a curated repository of trusted tools with reviewed descriptions, functionality, and safety policies.

Limitations and Open Challenges. Current component hardening approaches focus on simplified threat scenarios that do not reflect the complexity of real-world agent systems. Model hardening techniques like instruction hierarchy fine-tuning primarily address simple scenarios involving system prompts, user prompts, and tool results, failing to address sophisticated attacks such as shadowing attacks, where malicious tool results override other tool results. The role of each component and appropriate threat models for comprehensive hardening remain unclear.

5.6 Defense Design Principles

Effective agent security requires multiple complementary defense mechanisms working together rather than relying on any single approach. Three fundamental security principles from traditional security guide secure agent defense design. As discussed in §3, AI agents are hybrid software systems that combine LLMs with traditional software components, inheriting the same security concerns that motivated classical defense principles. Agents further amplify the need for these principles due to their autonomous decision-making, heterogeneous trust boundaries, and complex multi-step execution. We note that additional principles can also be applied to agents (e.g., fail-safe defaults and economy of mechanisms).

Defense-in-Depth. Defense mechanisms complement each other by operating at different stages and targeting different attack vectors. Input guardrails (§5.2.1) provide first-line protection by filtering malicious inputs, while output guardrails (§5.2.2) serve as last-line defense by sanitizing agent outputs. Information flow control (§5.2.3) and monitoring (§5.2.4) provide continuous runtime protection throughout agent execution, while access control (§5.4.2) ensures proper authentication and authorization. Secure-by-design approaches like privilege separation (§5.3.1) establish fundamental architectural protections. Component hardening (§5.5) strengthens individual elements, and human-in-the-loop validation (§5.2.5) provides user oversight for critical decisions. However, layering defenses can also introduce *emergent misalignment* [14], where one mechanism inadvertently weakens another. For example, a sanitizer may strip safety instructions relied upon by a downstream guardrail. Defense-in-depth therefore requires coordinated design across the full agent stack.

Table 3. Defense coverage in real-world agents. Defense techniques include input guardrail (Input), output guardrail (Output), access control (Access), information flow control and taint tracking (IFC), monitoring (Monitor), human-in-the-loop (HITL), privilege separation (Priv), formal verification (Formal), identity management (ID), and credential management (Cred). ● indicates full support and ◑ indicates partial support.

Category	Agent	Input	Output	Access	IFC	Monitor	HITL	Priv	Formal	ID	Cred
Coding	Codex v0.53.0 [115]		◑	●		◑	●				
Coding	Gemini CLI v0.13.0 [49]		◑			◑	●				
Coding	OpenHands v0.59.0 [3]		●			◑	●	●			
Web	Browser Use v0.9.0 [153]	◑	◑			◑					◑
Web	Nanobrowser v0.1.12 [104]	●	◑			◑		●			◑
Web	Skyvern v0.2.20 [144]	◑	◑			◑					◑

Principle of Least Privilege. Agents should operate with the minimum necessary permissions and access rights. This principle is implemented through privilege separation techniques (§5.3.1) that isolate agent components and restrict tool access to only what is required for specific tasks, as well as identity management (§5.4.1) that defines appropriate access scopes for agents.

Complete Mediation. All access to sensitive resources should be verified and authorized. This principle is reflected in comprehensive monitoring systems (§5.2.4), access control mechanisms (§5.4.2), and identity management (§5.4.1) that verify every agent interaction with protected resources.

6 Securing Real-World Agents

We analyze six open-source agents to illustrate how real-world agentic systems combine defenses across the dimensions in §5. We focus on system-level defenses rather than component hardening. Table 3 highlights which defense classes each agent enables. We consider a defense partially supported when the system provides incomplete coverage (e.g., only protects against a subset of threats) or requires non-negligible manual effort (e.g., manual configuration or curation). A defense is fully supported when the protection is automated and provides comprehensive coverage, even if its accuracy is imperfect. Note that this case study represents the agents' current status; these agents are actively evolving and they continue to update and strengthen their defenses.

6.1 Coding agents

General Coding Agent Defenses. Coding agents typically operate within directories that users trust. Nevertheless, threats remain from multiple sources: the LLM may hallucinate and generate incorrect code, the model itself could be compromised by backdoors (V5), or seemingly trusted inputs (e.g., user input, code repositories, documentation) may contain hidden malicious instructions (V1,V4) without the user's awareness.

To defend against such threats, coding agents prioritize constraining agent actions over sanitizing inputs. For instance, they gate AI-generated filesystem operations and shell commands through output guardrails (§5.2.2) and access control (§5.4.2) instead of filtering prompts. They also lean heavily on human-in-the-loop validation (§5.2.5) for sensitive operations because, while these actions can disrupt a user's machine, their safety depends on context. Implementations vary in how they define sensitive actions and enforce access control. Current monitoring (§5.2.4) support is partial in all coding agents. They collect logs of agent actions (e.g., tool calls, file modifications) using services like OpenTelemetry [117] or PostHog [123], enabling post-hoc manual review, but lack detections for suspicious patterns or anomalies.

Codex. Codex [115] combines access control with human-in-the-loop validation to secure AI-suggested file patches and shell commands, while also providing partial output guardrails and monitoring. It implements access control through path restrictions and privilege escalation controls. For file patches, the system requests user approval when the target file is outside writable paths (e.g., not in the working directory). For shell commands, the agent executes them in a sandbox by default, which restricts access to the working directory with no network access [35]. The model can request elevated privileges to run a command outside the sandbox when necessary. Such escalation requires user approval, providing human-in-the-loop control over potentially dangerous operations. This design ensures automatic agent actions run under containment while allowing controlled privilege escalation for legitimate use cases.

Gemini CLI. Gemini CLI [49] relies primarily on human-in-the-loop validation for command execution, complemented by partial output guardrails and monitoring. It applies output guardrails to file access and shell commands, restricting file reads to workspace directories. For shell commands, it maintains allow and deny lists, prompting the user before running anything unlisted. Users can cache decisions, and the agent parses compound commands into individual components so each decision is stored independently, reducing redundant user prompts. As a rule-based guardrail, coverage remains partial. The underlying environment ultimately depends on user-controlled permissions, with no additional OS-level access control. Gemini CLI encourages running the agent inside containers (e.g., Docker) to strengthen access control by isolating the entire agent [45], though setting up and maintaining that sandbox adds nontrivial overhead.

OpenHands. OpenHands [3] takes a multi-layered approach, combining output guardrails with human-in-the-loop validation and privilege separation, along with partial monitoring support. It implements a more active output guardrail by asking the LLM to emit a `security_risk` score with every tool decision to detect high-risk tool usage in a context-sensitive manner. Those high-risk tool calls cannot proceed without user consent, preserving human-in-the-loop control. Like Gemini CLI, it skips per-command sandboxing and recommends containerization to strengthen access control by isolating the entire agent. For privilege separation, OpenHands employs a multi-agent architecture where each agent is granted access to different tools and capabilities (e.g., a coding agent has file system access while a web browsing agent has network access). This separation limits the impact of compromising any individual agent and supports secure delegation schemes across agent boundaries.

Future Directions. Effective security for coding agents requires defense-in-depth with multiple complementary mechanisms working together. Current implementations have significant gaps in both coverage and effectiveness.

First, coding agents lack several critical defenses. They should deploy input guardrails (§5.2.1) that validate workspaces and user prompts to surface poisoned data before agents act on it, and filter web-retrieved content for prompt-injection payloads before it reaches the model. Information flow control (§5.2.3) could track how untrusted data influences tool calls, preventing malicious instructions from compromising agent decisions. Identity and credential management (§5.4.1, §5.4.3) would enable proper authentication and secure storage of API keys and access tokens.

Second, existing partial defenses need significant strengthening. Access control (§5.4.2) should move beyond all-or-nothing approvals to seamless, fine-grained permissions that grant only the minimum necessary access across diverse development tools. Human-in-the-loop validation (§5.2.5) needs richer contextual information, such as expected side effects or comparisons with past approvals, to help users avoid decision fatigue and make informed authorizations. Monitoring (§5.2.4) should pair existing logs with automated detectors that identify domain-specific risks and privilege-escalation attempts so users can intervene before damage occurs.

6.2 Web agents

Web agents autonomously perform web tasks that range from summarizing pages to navigating URLs, clicking buttons, and completing forms. As web agents receive arbitrary content from diverse sources, they inherently process untrusted data even when tasks require sensitive inputs such as personal identifiers, payment details, or authenticated workflows (e.g., accessing cloud files, sending email, or making purchases).

General Web Agent Defenses. Web agents are particularly vulnerable to indirect prompt injection attacks (V1), where malicious instructions embedded in web pages hijack agent behavior. Unlike coding agents that primarily operate in trusted workspaces, web agents continuously process untrusted external inputs from vast and diverse web sources. Moreover, web agents often access the web with user authorization, handling sensitive private data and credentials (e.g., accessing cloud files, reading emails, making purchases), which makes them high-value targets. Additional threats include compromised models (V5), hallucinations, and direct attacks from user prompts (V4).

Given the vast untrusted input surface and access to sensitive data, web agents commonly employ four defense mechanisms: input guardrails (§5.2.1), output guardrails (§5.2.2), credential management (§5.4.3), and monitoring (§5.2.4). Input guardrails filter malicious content from web pages by using techniques such as domain allow and deny lists to control which sites agents can visit, and filtering page elements or blocklisted domains to prevent malicious instructions from reaching the model. Output guardrails constrain agent actions by preventing navigation to sensitive URLs such as local-network hosts, raw IP addresses, and browser configuration pages like `chrome://settings`, limiting server-side request forgery attempts. Credential management protects sensitive user data through various techniques such as redacting or replacing secrets before they reach LLM providers or untrusted domains. For monitoring, web agents emit browsing telemetry for post-hoc review, yet none of the surveyed systems pair these logs with automated detection. However, most defenses provide only partial protection, with manually curated controls and incomplete coverage.

Browser-use. Browser-use [153] provides support for input guardrails, output guardrails, credential management, and monitoring. For input guardrails, it applies ad-block rules to strip advertising and other unwanted elements from incoming pages, which offers limited coverage against malicious prompts. For credential management, it protects secrets by replacing them with placeholders before data reaches third-party LLM providers or untrusted domains, using user-defined mappings that specify which secrets may be revealed to which sites.

Nanobrowser. Nanobrowser [104] implements input guardrails and privilege separation, along with partial support for output guardrails, credential management, and monitoring. It strengthens input guardrails by inserting delimiters and guard prompts that keep user instructions distinct from retrieved page content, mitigating indirect prompt injection attacks. For credential management, it redacts sensitive data, such as Social Security numbers, credit card details, and email addresses, protecting those values from LLM providers and untrusted sites. Such data are detected using regular expression rules. For privilege separation, Nanobrowser splits responsibilities between planner and navigator agents with different permissions, so that compromising the navigator cannot directly corrupt the overall agent plan.

Skyvern. Skyvern [144] provides support for input guardrails, output guardrails, credential management, and monitoring, with a particular focus on credential protection. For credential management, it specializes in protecting one-time passwords (OTPs) for automated authorization tasks. It detects OTPs at runtime with regular expressions, swaps them with placeholders to keep the secrets from LLM providers and untrusted websites, and restores the original values only to

the authentication form. While this raises the bar for OTP exfiltration attacks, the regex-based approach provides incomplete coverage and does not extend to other credential types.

Future Directions. Like coding agents, securing web agents requires defense-in-depth with multiple complementary mechanisms. Today's web agents remain early-stage prototypes with utility features still maturing and defenses that are simple and fragile.

First, web agents lack several critical defenses entirely. Information flow control (§5.2.3) could track how untrusted web content influences agent decisions and prevent data exfiltration to malicious domains. Identity management (§5.4.1) would enable proper authentication when agents act on behalf of users across multiple web services. Human-in-the-loop validation (§5.2.5) could provide user oversight for high-impact web actions such as purchases or data sharing.

Second, existing partial defenses need significant strengthening. Input guardrails (§5.2.1) should move beyond manual domain allow and deny lists to automated domain reputation models and contextual filtering, reducing user burden and addressing risks such as expired domains [130]. Structural protections combining privilege separation (§5.3.1), taint tracking, and model-level input guardrails can neutralize malicious instructions before they reach planners. Monitoring (§5.2.4) should pair existing telemetry with real-time detectors that flag or halt suspicious actions and escalate to human validation when necessary. Credential management (§5.4.3) requires adaptive and privacy-preserving detection techniques to better protect diverse credential types beyond simple regex-based approaches.

7 Detailed Case Study: AutoGPT

AutoGPT [173] is one of the most widely used open-source autonomous agents with over 180k GitHub stars. It exposes a broad set of tools enabling LLM interaction with heterogeneous environments, including the Internet, local files, and execution interfaces (e.g., command line). In this section, we analyze multiple versions of AutoGPT since v0.4.3, track their real-world vulnerability reports, and evaluate implemented defenses.

7.1 Tools and Execution Environments

AutoGPT equips agents with retrieval tools for information gathering, execution tools for system-level operations. These tools enable AutoGPT to interact with diverse external environments.

Retrieval Tools. Agents can search the web with `google_search`, fetch webpage content through `browse_website`, navigate local files using `read_file` and `search_files`, and access historical context via `load_from_memory`.

Execution Tools. AutoGPT grants direct system access through `execute_shell` for arbitrary bash commands, `file_manipulation` for modifying workspace contents, and `execute_python_code` for dynamic code execution.

External Environments. AutoGPT interacts with the web (Internet access via `google_search`), the computer (filesystem read/write access), and domain-specific environments (Python interpreter, shell, OS-level interfaces).

7.2 Real-world Vulnerabilities in AutoGPT

We study five representative CVE vulnerabilities since 2023 and map them to our risk taxonomy.

A. Docker-Compose Overwrite (CVE-2023-37273). The `docker-compose.yml` file of the project lacks write protection, allowing malicious LLM outputs to overwrite container configurations. Attackers embed malicious instructions in external content (e.g., a web page fetched by the agent), which hijack the LLM into calling `execute_python_code` to overwrite the configuration file (R2).

When AutoGPT restarts, it executes the malicious container, leading to container escape and host compromise (R6).

B. Path Traversal (CVE-2023-37274). Unsanitized basename parameters allow path traversal attacks that write files outside the sandbox. Attackers inject instructions into external content that trick the LLM into calling `execute_python_code` with a traversal path such as `../../main.py`, overwriting critical AutoGPT source files (R2, R6). When AutoGPT restarts, these modified files execute, achieving persistent arbitrary code execution. The overwritten files may also expose sensitive source code or configuration data (R5).

C. ANSI Escape Sequence Deception (CVE-2023-37275). ANSI escape sequences are special control codes interpreted by terminals to perform actions such as moving the cursor, clearing the screen, or changing text color. When AutoGPT fetches external web content via `browse_website`, it passes the retrieved content—including any embedded ANSI codes—directly to the console without sanitization. An attacker crafts a malicious web page embedding JSON-encoded ANSI escape sequences. The sequences are not instructions that hijack the LLM; rather, they flow as unsanitized data through the agent pipeline and are rendered by the terminal (R3). The spoofed console output can conceal executed commands or trick the human operator into approving malicious actions, silently hijacking the agent's behavior (R6).

D. Cross-Site Request Forgery (CVE-2024-1879). Missing CSRF protection and permissive CORS settings allow authenticated API requests from malicious webpages. An attacker crafts a webpage that, when visited by an authenticated user, silently triggers agent actions through cross-origin requests. This enables unauthorized command execution and data exfiltration (R5, R6).

E. OS Command Injection (CVE-2024-1881). AutoGPT validates shell commands using an allowlist that checks only the first token. This approach blocks individual dangerous commands but fails to detect operator-chained payloads or multiple commands in a single line. Attackers inject instructions into external content that manipulate the LLM to generate commands with chaining operators (e.g., `ls && rm -rf /`, `cat file; curl attacker.com`) (R2). Even without an attacker, the LLM may spontaneously generate chained shell commands for complex tasks, bypassing the first-token check (R4). The executor runs these multi-command payloads verbatim, enabling arbitrary command execution, data exfiltration, and filesystem compromise (R5, R6).

7.3 Defenses in AutoGPT

AutoGPT has deployed patches across multiple versions to address the known vulnerabilities discussed above. Table 4 summarizes the defense landscape per CVE: the risks each vulnerability exploits, the defense mechanism applied, which risks the patch mitigates, which remain open, and what defenses are still missing. Notably, all patches target downstream consequences (access control and output sanitization) rather than the upstream causes, leaving indirect prompt injection (R2) and unsafe data flow (R3) unaddressed at their source.

A. Docker-Compose Overwrite (CVE-2023-37273). Versions after 0.4.3 use read-only mounts and restrict permissions on configuration files. This mitigates the integrity impact (R6) by preventing overwrites of `docker-compose.yml`, but does not address the indirect prompt injection (R2) that triggers the overwrite attempt. Input guardrails (§5.2.1) should scan fetched web content for prompt injection, and information flow control (§5.2.3) should prevent tainted LLM outputs from reaching `execute_python_code`.

B. Path Traversal (CVE-2023-37274). Versions after 0.4.3 canonicalize paths and filter traversal patterns like `../` in the `agent.workspace.get_path()` function. This blocks basic traversal attacks (R6) when the workspace root is properly configured, but the indirect prompt injection vector (R2) that causes the LLM to generate traversal paths remains unmitigated. Input guardrails

Table 4. Defense analysis of AutoGPT CVEs. For each vulnerability, we list the exploited risks, the defense mechanism deployed in the patch, which risks are mitigated, which remain open, and which defense categories are missing.

CVE	Exploited Risks	Defense Mechanism	Risks Mitigated	Risks Still Open	Missing Defenses
A. Docker-Compose CVE-2023-37273	R2, R6	Access control (read-only mounts)	R6	R2	Input guardrails, Information flow control
B. Path Traversal CVE-2023-37274	R2, R5, R6	Output guardrail (path canonicalization)	R6	R2, R5	Input guardrails, Information flow control
C. ANSI Escape CVE-2023-37275	R3, R6	Output guardrail (regex ANSI filter)	R3 (partial)	R3 (incomplete)	Information flow control
D. CSRF CVE-2024-1879	R5, R6	Access control (CSRF tokens, CORS)	R5, R6 (partial)	R5, R6 (localhost bypass)	Identity management, Monitoring
E. Command Injection CVE-2024-1881	R2, R4, R5, R6	Output guardrail (first-token allowlist)	R6 (partial)	R2, R4, R5	Output guardrails, Human-in-the-loop, Privilege separation, Monitoring, Formal verification

and information flow control (§5.2.3) should block the injection at its source and track taint from web content to filesystem operations.

C. ANSI Escape Sequence Deception (CVE-2023-37275). Versions after 0.4.3 apply rule-based sanitization to filter escape sequences from model outputs. This partially addresses the unsafe data flow (R3), but cannot cover all escape sequence variants and may be bypassed through novel encoding schemes or lesser-known control codes. Information flow control (§5.2.3) should treat all data originating from external web sources as untrusted and enforce sanitization at every output boundary, including the terminal, not just the LLM context.

D. Cross-Site Request Forgery (CVE-2024-1879). Versions after 0.5.1 add CSRF tokens and enforce strict CORS policies that trust only localhost ports. This prevents most cross-origin attacks but leaves open attacks from malicious browser extensions or local applications that can access localhost endpoints. Identity management (§5.4) should implement OAuth-based authorization instead of relying solely on token validation. Monitoring (§5.2.4) should track cross-origin patterns to catch attacks before data theft.

E. OS Command Injection (CVE-2024-1881). Current versions maintain a command allowlist that validates the first token of each input. This blocks individual dangerous commands but does not prevent operator-based chaining or multi-command payloads. The defense remains incomplete: neither the indirect prompt injection (R2) that manipulates the LLM nor the hallucination risk (R4) that produces chained commands spontaneously is addressed. Output guardrails (§5.2.2) should parse shell syntax to catch chained payloads regardless of whether they originate from an attacker or from the LLM's own generation. Human-in-the-loop validation (§5.2.5) should display command risks so users can judge high-risk operations. Privilege separation (§5.3.1) should isolate shell execution with minimal privileges per risk level. Monitoring (§5.2.4) should log command provenance, and formal verification (§5.3.2) should restrict operations to pre-approved command templates.

8 Conclusion

This paper presents an overview of the attack and defense landscape for AI agents, together with an in-depth analysis of AI agent risks, security goals, defense dimensions, case studies, and open challenges. Our survey reveals that while agentic AI security research has made significant progress in mapping the problem space, practical and general-purpose defenses remain largely elusive. Critical directions for the field include realistic evaluation frameworks that bridge research and

production, composable defenses that avoid emergent misalignment, standardized agent identity and access control, and adaptive defenses that balance security with usability. Our SoK can serve as a guide for building secure agents and point out meaningful directions for future research.

References

[1] Sahar Abdelnabi, Amr Gomaa, Eugene Bagdasarian, Per Ola Kristensson, and Reza Shokri. 2025. Firewalls to secure dynamic llm agentic networks. *arXiv preprint arXiv:2502.01822* (2025).

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[3] All Hands AI. 2025. OpenHands. https://openhands.dev/.

[4] Amazon. 2024. Access control for vector stores using metadata filtering with Amazon Bedrock Knowledge Bases. https://aws.amazon.com/blogs/machine-learning/access-control-for-vector-stores-using-metadata-filtering-with-knowledge-bases-for-amazon-bedrock/.

[5] Hengyu An, Jinghuai Zhang, Tianyu Du, Chunyi Zhou, Qingming Li, Tao Lin, and Shouling Ji. 2025. Ipiguard: A novel tool dependency graph-based defense against indirect prompt injection in llm agents. In *EMNLP*.

[6] Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. 2025. Jailbreaking Leading Safety-Aligned LLMs with Simple Adaptive Attacks. In *ICLR*.

[7] Anthropic. 2024. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol.

[8] Anthropic. 2025. Anthropic Connectors Directory FAQ. https://support.anthropic.com/en/articles/11596036-anthropic-connectors-directory-faq.

[9] Apple. 2025. Requesting access to protected resources. https://developer.apple.com/documentation/uikit/requesting-access-to-protected-resources.

[10] auth0. 2025. Calling APIs with Token Vault. https://auth0.com/ai/docs/intro/token-vault.

[11] Eugene Bagdasarian, Ren Yi, Sahra Ghalebikesabi, Peter Kairouz, Marco Gruteser, Sewoong Oh, Borja Balle, and Daniel Ramage. 2024. AirGapAgent: Protecting privacy-conscious conversational agents. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 3868–3882.

[12] Adam Barth, Anupam Datta, John C Mitchell, and Helen Nissenbaum. 2006. Privacy and contextual integrity: Framework and applications. In *2006 IEEE symposium on security and privacy (S&P'06)*. IEEE, 15–pp.

[13] D Elliot Bell and Leonard J LaPadula. 1973. *Secure computer systems: Mathematical foundations*. Technical Report.

[14] Jan Betley, Daniel Chee Hian Tan, Niels Warncke, Anna Sztyber-Betley, Xuchan Bao, Martín Soto, Nathan Labenz, and Owain Evans. 2025. Emergent Misalignment: Narrow finetuning can produce broadly misaligned LLMs. In *Forty-second International Conference on Machine Learning*.

[15] Luca Beurer-Kellner, Beat Buesser Ana-Maria Creţu, Edoardo Debenedetti, Daniel Dobos, Daniel Fabian, Marc Fischer, David Froelicher, Kathrin Grosse, Daniel Naeff, Ezinwanne Ozoani, et al. 2025. Design Patterns for Securing LLM Agents against Prompt Injections. *arXiv preprint arXiv:2506.08837* (2025).

[16] Kenneth J Biba. 1977. *Integrity considerations for secure computer systems*. Technical Report.

[17] Brave. 2025. Agentic Browser Security: Indirect Prompt Injection in Perplexity Comet. https://brave.com/blog/comet-prompt-injection/.

[18] David Brumley and Dawn Song. 2004. Privtrans: Automatically partitioning programs for privilege separation. In *USENIX security symposium*, Vol. 57.

[19] Alan Chan, Carson Ezell, Max Kaufmann, Kevin Wei, Lewis Hammond, Herbie Bradley, Emma Bluemke, Nitarshan Rajkumar, David Krueger, Noam Kolt, et al. 2024. Visibility into AI agents. In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*. 958–973.

[20] Gaowei Chang, Eidan Lin, Chengxuan Yuan, Rizhao Cai, Binbin Chen, Xuan Xie, and Yin Zhang. 2025. Agent network protocol technical white paper. *arXiv preprint arXiv:2508.00007* (2025).

[21] Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. 2025. {StruQ}: Defending against prompt injection with structured queries. In *34th USENIX Security Symposium (USENIX Security 25)*. 2383–2400.

[22] Sizhe Chen, Arman Zharmagambetov, Saeed Mahloujifar, Kamalika Chaudhuri, David Wagner, and Chuan Guo. 2025. SecAlign: Defending Against Prompt Injection with Preference Optimization. In *The ACM Conference on Computer and Communications Security (CCS)*.

[23] Zhaorun Chen, Mintong Kang, and Bo Li. 2025. ShieldAgent: Shielding Agents via Verifiable Safety Policy Reasoning. In *Forty-second International Conference on Machine Learning*.

[24] Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. 2024. Agentpoison: Red-teaming llm agents via poisoning memory or knowledge bases. *Advances in Neural Information Processing Systems* 37 (2024), 130185–130213.

[25] Sahana Chennabasappa, Cyrus Nikolaidis, Daniel Song, David Molnar, Stephanie Ding, Shengye Wan, Spencer Whitman, Lauren Deason, Nicholas Doucette, Abraham Montilla, et al. 2025. Llamafirewall: An open source guardrail system for building secure ai agents. *arXiv preprint arXiv:2505.03574* (2025).
[26] Google Cloud. 2025. Gemini Code Assist. https://cloud.google.com/products/gemini/code-assist.
[27] Composio. 2025. Welcome to Composio. https://docs.composio.dev/docs/welcome.
[28] Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. 2025. Securing AI Agents with Information-Flow Control. *arXiv preprint arXiv:2505.23643* (2025).
[29] Jian Cui, Zichuan Li, Luyi Xing, and Xiaojing Liao. 2025. Safeguard-by-development: A privacy-enhanced development paradigm for multi-agent collaboration systems. *arXiv preprint arXiv:2505.04799* (2025).
[30] Cursor. 2025. Cursor - The AI-first code editor. https://www.cursor.com/.
[31] Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. 2025. Defeating prompt injections by design. *arXiv preprint arXiv:2503.18813* (2025).
[32] Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
[33] Zijian Deng et al. 2025. AI Agents Under Threat: A Survey of Key Security Challenges and Future Pathways. *arXiv preprint arXiv:2406.02630* (2025).
[34] Dorothy E Denning. 1976. A lattice model of secure information flow. *Commun. ACM* 19, 5 (1976), 236–243.
[35] OpenAI Developers. 2025. Codex security guide. https://developers.openai.com/codex/security/.
[36] ETDI Documentation. 2025. Enhanced Tool Definition Interface (ETDI): A Security Fortification for the Model Context Protocol. https://vineethsai.github.io/python-sdk/etdi-concepts/#introduction-the-imperative-for-secure-mcp.
[37] Shen Dong, Shaochen Xu, Pengfei He, Yige Li, Jiliang Tang, Tianming Liu, Hui Liu, and Zhen Xiang. 2025. A practical memory injection attack against llm agents. *arXiv preprint arXiv:2503.03704* (2025).
[38] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. 2012. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the eighth symposium on usable privacy and security*. 1–14.
[39] OpenID Foundation. 2014. OpenID Connect Core 1.0 incorporating errata set 1. https://openid.net/specs/openid-connect-core-1_0.html.
[40] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 1322–1333.
[41] Xiaohan Fu, Shuheng Li, Zihan Wang, Yihao Liu, Rajesh K Gupta, Taylor Berg-Kirkpatrick, and Earlence Fernandes. 2024. Imprompter: Tricking LLM Agents into Improper Tool Use. *arXiv preprint arXiv:2410.14923* (2024).
[42] Xiaohan Fu, Zihan Wang, Shuheng Li, Rajesh K Gupta, Niloofar Mireshghallah, Taylor Berg-Kirkpatrick, and Earlence Fernandes. 2023. Misusing tools in large language models with visual adversarial examples. *arXiv preprint arXiv:2310.03185* (2023).
[43] Gemini. 2025. Gemini Canvas. https://gemini.google/overview/canvas/.
[44] Gemini. 2025. Introducing Nano Banana Pro. https://blog.google/technology/ai/nano-banana-pro/.
[45] gemini cli. 2025. Sandboxing in the Gemini CLI. https://google-gemini.github.io/gemini-cli/docs/cli/sandbox.html.
[46] GeminiTeam. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).
[47] GitHub. 2025. GitHub Copilot. https://github.com/features/copilot.
[48] Google. 2025. Gemini. https://gemini.google.com/.
[49] Google. 2025. Gemini CLI. https://docs.cloud.google.com/gemini/docs/codeassist/gemini-cli.
[50] Google. 2025. Google Safe Browsing. https://safebrowsing.google.com/.
[51] Google. 2025. Mitigating prompt injection attacks with a layered defense strategy. https://security.googleblog.com/2025/06/mitigating-prompt-injection-attacks.html.
[52] Google. 2025. Using OAuth 2.0 to Access Google APIs. https://developers.google.com/identity/protocols/oauth2 (accessed 14, April, 2025).
[53] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*. 79–90.
[54] Kathrin Grosse, Lukas Bieringer, Tarek R Besold, and Alexandre M Alahi. 2024. Towards more practical threat models in artificial intelligence security. In *33rd USENIX Security Symposium (USENIX Security 24)*. 4891–4908.
[55] Norm Hardy. 1988. The Confused Deputy: (or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review* 22, 4 (1988), 36–38.

[56] Chris Hawblitzel, Jon Howell, Jacob R Lorch, Arjun Narayan, Bryan Parno, Danfeng Zhang, and Brian Zill. 2014. Ironclad apps:{End-to-End} security via automated {Full-System} verification. In *11th USENIX symposium on operating systems design and implementation (OSDI 14)*. 165–181.

[57] Xu He, Di Wu, Yan Zhai, and Kun Sun. 2025. SentinelAgent: Graph-based Anomaly Detection in Multi-Agent Systems. *arXiv preprint arXiv:2505.24201* (2025).

[58] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278* (2025).

[59] Internet Engineering Task Force (IETF). 2012. The OAuth 2.0 Authorization Framework. https://www.rfc-editor.org/rfc/rfc6749.html.

[60] Invariantlabs. 2025. GitHub MCP Exploited: Accessing private repositories via MCP. https://invariantlabs.ai/blog/mcp-github-vulnerability.

[61] Dennis Jacob, Hend Alzahrani, Zhanhao Hu, Basel Alomair, and David Wagner. 2024. Promptshield: Deployable detection for prompt injection attacks. In *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy*. 341–352.

[62] Rishi Jha, Harold Triedman, Justin Wagle, and Vitaly Shmatikov. 2025. Breaking and Fixing Defenses Against Control-Flow Hijacking in Multi-Agent Systems. *arXiv preprint arXiv:2510.17276* (2025).

[63] Feiran Jia, Tong Wu, Xin Qin, and Anna Squicciarini. 2025. The Task Shield: Enforcing Task Alignment to Defend Against Indirect Prompt Injection in LLM Agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 29680–29697. doi:10.18653/v1/2025.acl-long.1435

[64] Yuqi Jia, Zedian Shao, Yupei Liu, Jinyuan Jia, Dawn Song, and Neil Zhenqiang Gong. 2025. A Critical Evaluation of Defenses against Prompt Injection Attacks. *arXiv preprint arXiv:2505.18333* (2025).

[65] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Zhuoqing Morley Mao, Atul Prakash, and SJ Unviersity. 2017. ContexloT: Towards providing contextual integrity to appified IoT platforms.. In *ndss*, Vol. 2. San Diego, 2–2.

[66] Huihao Jing, Haoran Li, Wenbin Hu, Qi Hu, Xu Heli, Tianshu Chu, Peizhao Hu, and Yangqiu Song. 2025. Mcip: Protecting mcp safety via model contextual integrity protocol. In *EMNLP*.

[67] Juhee Kim, Woohyuk Choi, and Byoungyoung Lee. 2025. Prompt flow integrity to prevent privilege escalation in llm agents. *arXiv preprint arXiv:2503.15547* (2025).

[68] Agent Development Kit. 2025. Workflow Agent. https://google.github.io/adk-docs/agents/workflow-agents/.

[69] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. 2009. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 207–220.

[70] Abhinav Kumar et al. 2025. Overthink: Slowdown Attacks on Reasoning LLMs. *arXiv preprint arXiv:2502.12345* (2025).

[71] LangChain. 2025. Chat history. https://python.langchain.com/docs/concepts/chat_history/.

[72] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–16.

[73] Donghyun Lee and Mo Tiwari. 2024. Prompt Infection: LLM-to-LLM Prompt Injection within Multi-Agent Systems. *arXiv preprint arXiv:2410.07283* (2024).

[74] Eunkyu Lee, Donghyeon Kim, Wonyoung Kim, and Insu Yun. 2025. Takedown: How It's Done in Modern Coding Agent Exploits. *arXiv preprint arXiv:2509.24240* (2025).

[75] Gibbeum Lee, Volker Hartmann, Jongho Park, Dimitris Papailiopoulos, and Kangwook Lee. 2023. Prompted LLMs as Chatbot Modules for Long Open-domain Conversation. In *Findings of the Association for Computational Linguistics: ACL 2023*. 4536–4554.

[76] Jungjae Lee, Dongjae Lee, Chihun Choi, Youngmin Im, Jaeyoung Wi, Kihong Heo, Sangeun Oh, Sunjae Lee, and Insik Shin. 2025. Safeguarding mobile gui agent via logic-based action verification. *arXiv preprint arXiv:2503.18492* (2025).

[77] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[78] Evan Li, Tushin Mallick, Evan Rose, William Robertson, Alina Oprea, and Cristina Nita-Rotaru. 2026. ACE: A Security Architecture for LLM-Integrated App Systems. In *Network and Distributed System Security (NDSS) Symposium*.

[79] Hao Li, Xiaogeng Liu, Hung-Chun Chiu, Dianqi Li, Ning Zhang, and Chaowei Xiao. 2025. DRIFT: Dynamic Rule-Based Defense with Injection Isolation for Securing LLM Agents. *arXiv preprint arXiv:2506.12104* (2025).

[80] Peiran Li, Xinkai Zou, Zhuohang Wu, Ruifeng Li, Shuo Xing, Hanwen Zheng, Zhikai Hu, Yuping Wang, Haoxi Li, Qin Yuan, et al. 2025. Safeflow: A principled protocol for trustworthy and transactional autonomous agent systems. *arXiv preprint arXiv:2506.07564* (2025).
[81] Yuanchun Li, Yizhen Hao, et al. 2024. Personal LLM Agents: Insights and Survey about the Capability, Efficiency and Security. *arXiv preprint arXiv:2401.05459* (2024).
[82] Zelong Li, Wenyue Hua, Hao Wang, He Zhu, and Yongfeng Zhang. 2024. Formal-llm: Integrating formal language and natural language for controllable llm-based agents. *arXiv preprint arXiv:2402.00798* (2024).
[83] Zeyi Liao, Lingbo Mo, Chejian Xu, Mintong Kang, Jiawei Zhang, Chaowei Xiao, Yuan Tian, Bo Li, and Huan Sun. 2025. EIA: ENVIRONMENTAL INJECTION ATTACK ON GENERALIST WEB AGENTS FOR PRIVACY LEAKAGE. In *The Thirteenth International Conference on Learning Representations*.
[84] Linux. 2024. cgroups(7) — Linux manual page. https://man7.org/linux/man-pages/man7/cgroups.7.html.
[85] Linux. 2024. namespaces(7) — Linux manual page. https://man7.org/linux/man-pages/man7/namespaces.7.html.
[86] Linux. 2024. seccomp(2) — Linux manual page. https://man7.org/linux/man-pages/man2/seccomp.2.html.
[87] Xiaogeng Liu, Zhiyuan Yu, Yizhe Zhang, Ning Zhang, and Chaowei Xiao. 2024. Automatic and universal prompt injection attacks against large language models. *arXiv preprint arXiv:2403.04957* (2024).
[88] Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, et al. 2023. Prompt injection attack against llm-integrated applications. *arXiv preprint arXiv:2306.05499* (2023).
[89] Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. 2024. Formalizing and benchmarking prompt injection attacks and defenses. In *33rd USENIX Security Symposium (USENIX Security 24)*. 1831–1847.
[90] Yupei Liu, Yuqi Jia, Jinyuan Jia, Dawn Song, and Neil Zhenqiang Gong. 2025. DataSentinel: A Game-Theoretic Detection of Prompt Injection Attacks. In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2190–2208.
[91] Hanjun Luo, Shenyu Dai, Chiming Ni, Xinfeng Li, Guibin Zhang, Kun Wang, Tongliang Liu, and Hanan Salam. 2025. Agentauditor: Human-level safety and security evaluation for llm agents. *arXiv preprint arXiv:2506.00641* (2025).
[92] Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. 2025. AGrail: A Lifelong Agent Guardrail with Effective and Adaptive Safety Detection. In *ACL (1)*. 8104–8139. https://aclanthology.org/2025.acl-long.399/
[93] meta llama. 2025. CodeShield. https://github.com/meta-llama/PurpleLlama/tree/main/CodeShield.
[94] Microsoft. 2025. Introduction to Microsoft Entra Verified ID. https://learn.microsoft.com/en-us/entra/verified-id/decentralized-identifier-overview.
[95] MITRE. 2024. CVE-2024-5565. https://www.cve.org/CVERecord?id=CVE-2024-5565.
[96] MITRE. 2025. CVE-2025-32711. https://www.cve.org/CVERecord?id=CVE-2025-32711.
[97] MITRE. 2025. CVE-2025-54795. https://nvd.nist.gov/vuln/detail/CVE-2025-54795.
[98] MITRE Corporation. 2024. MITRE ATLAS – Adversarial Threat Landscape for Artificial-Intelligence Systems. https://atlas.mitre.org/.
[99] modelcontextprotocol. 2025. Model Context Protocol servers. https://github.com/modelcontextprotocol/servers.
[100] Andrew Moffat. 2023. HeimdaLLM. https://heimdallm.readthedocs.io/en/main/.
[101] Mozilla. 2025. Meta: Help Users Stop Accidentally Sharing Private AI Chats. https://www.mozillafoundation.org/en/campaigns/meta-help-users-stop-accidentally-sharing-private-ai-conversations/.
[102] Andrew C Myers. 1999. JFlow: Practical mostly-static information flow control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 228–241.
[103] Silen Naihin, David Atkinson, Marc Green, Merwane Hamadi, Craig Swift, Douglas Schonholtz, Adam Tauman Kalai, and David Bau. 2023. Testing Language Model Agents Safely in the Wild. In *Socially Responsible Language Modelling Research*.
[104] Nanobrowser. 2025. Nanobrowser. https://nanobrowser.ai/.
[105] Milad Nasr, Nicholas Carlini, Chawin Sitawarin, Sander V Schulhoff, Jamie Hayes, Michael Ilie, Juliette Pluto, Shuang Song, Harsh Chaudhari, Ilia Shumailov, et al. 2025. The attacker moves second: Stronger adaptive attacks bypass defenses against LLM jailbreaks and prompt injections. *arXiv preprint arXiv:2510.09023* (2025).
[106] BBC News. 2023. ChatGPT banned in Italy over privacy concerns. https://www.bbc.co.uk/news/technology-65139406.
[107] James Newsome and Dawn Xiaodong Song. 2005. Dynamic taint analysis for automatic detection, analysis, and signaturegeneration of exploits on commodity software.. In *NDSS*, Vol. 5. 3–4.
[108] Trail of Bits. 2025. Jumping the line: How MCP servers can attack you before you ever use them. https://blog.trailofbits.com/2025/04/21/jumping-the-line-how-mcp-servers-can-attack-you-before-you-ever-use-them/.
[109] Okta. 2025. Okta Documentation. https://help.okta.com/en-us/content/index.htm.
[110] OpenAI. 2023. March 20 ChatGPT outage: Here's what happened. https://openai.com/index/march-20-chatgpt-outage/.
[111] OpenAI. 2025. ChatGPT. https://chatgpt.com/.

[112] OpenAI. 2025. ChatGPT Shared Links FAQ. https://help.openai.com/en/articles/7925741-chatgpt-shared-links-faq.
[113] OpenAI. 2025. Connectors in ChatGPT. https://chatgpt.com/features/connectors/.
[114] OpenAI. 2025. Memory and new controls for ChatGPT. https://openai.com/index/memory-and-new-controls-for-chatgpt/.
[115] OpenAI. 2025. OpenAI Codex. https://openai.com/codex/.
[116] OpenAI. 2025. Temporary Chat FAQ. https://help.openai.com/en/articles/8914046-temporary-chat-faq.
[117] OpenTelemetry. 2025. OpenTelemetry. https://opentelemetry.io/.
[118] OWASP Foundation. 2025. OWASP Top 10 for Large Language Model Applications. https://owasp.org/www-project-top-10-for-large-language-model-applications/.
[119] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems* 37 (2024), 126544–126565.
[120] David Patlan, Luis Perez, et al. 2025. Real Vulnerabilities in AI Agents: A Practical Threat Analysis of Web3 Agent Memory Attacks. *arXiv preprint arXiv:2501.12345* (2025).
[121] Fábio Perez and Ian Ribeiro. 2022. Ignore Previous Prompt: Attack Techniques For Language Models. In *NeurIPS ML Safety Workshop*.
[122] Perplexity. 2025. Comet Browser: Browse at the speed of thought. https://www.perplexity.ai/comet.
[123] PostHog. 2025. PostHog. https://posthog.com/.
[124] Brandon Radosevich and John Halloran. 2025. Mcp safety audit: Llms with the model context protocol allow major security exploits. *arXiv preprint arXiv:2504.03767* (2025).
[125] Itay Ravia. 2025. Breaking down 'EchoLeak', the First Zero-Click AI Vulnerability Enabling Data Exfiltration from Microsoft 365 Copilot. https://www.catonetworks.com/blog/breaking-down-echoleak/.
[126] Traian Rebedea, Razvan Dinu, Makesh Narsimhan Sreedhar, Christopher Parisien, and Jonathan Cohen. 2023. Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails. In *Proceedings of the 2023 conference on empirical methods in natural language processing: system demonstrations*. 431–445.
[127] Embrace The Red. 2025. Cross-Agent Privilege Escalation: When Agents Free Each Other. https://embracethered.com/blog/posts/2025/cross-agent-privilege-escalation-agents-that-free-each-other/.
[128] Embrace The Red. 2025. GitHub Copilot Chat: From Prompt Injection to Data Exfiltration. https://embracethered.com/blog/posts/2024/github-copilot-chat-prompt-injection-data-exfiltration/.
[129] Alexander Robey, Eric Wong, Hamed Hassani, and George J. Pappas. 2025. SmoothLLM: Defending Large Language Models Against Jailbreaking Attacks. *Transactions on Machine Learning Research* (2025).
[130] Sebastian Roth, Timothy Barron, Stefano Calzavara, Nick Nikiforakis, and Ben Stock. 2020. Complex security policy? a longitudinal analysis of deployed content security policies. In *Proceedings of the 27th Network and Distributed System Security Symposium (NDSS)*.
[131] Ravi S Sandhu. 1998. Role-based access control. In *Advances in computers*. Vol. 46. Elsevier, 237–286.
[132] Gal Moyal Sasi Levi. 2025. How an AI Agent Vulnerability in LangSmith Could Lead to Stolen API Keys and Hijacked LLM Responses. https://noma.security/blog/how-an-ai-agent-vulnerability-in-langsmith-could-lead-to-stolen-api-keys-and-hijacked-llm-responses/.
[133] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems* 36 (2023), 68539–68551.
[134] Sander Schulhoff, Jeremy Pinto, Anaum Khan, Louis-François Bouchard, Chenglei Si, Svetlina Anati, Valen Tagliabue, Anson Kost, Christopher Carnahan, and Jordan Boyd-Graber. 2023. Ignore This Title and HackAPrompt: Exposing Systemic Vulnerabilities of LLMs Through a Global Prompt Hacking Competition. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 4945–4977. doi:10.18653/v1/2023.emnlp-main.302
[135] AMD Sev-Snp. 2020. Strengthening VM isolation with integrity protection and more. *White Paper, January* 53, 2020 (2020), 1450–1465.
[136] Mrinank Sharma, Meg Tong, Jesse Mu, Jerry Wei, Jorrit Kruthoff, Scott Goodfriend, Euan Ong, Alwin Peng, Raj Agarwal, Cem Anil, et al. 2025. Constitutional classifiers: Defending against universal jailbreaks across thousands of hours of red teaming. *arXiv preprint arXiv:2501.18837* (2025).
[137] Xinyue Shen, Yiting Song, Yixin Li, Yun Zhu, et al. 2024. Prompt Stealing Attacks Against Text-to-Image Generation Models. In *USENIX Security Symposium*.
[138] Jiawen Shi, Zenghui Yuan, Guiyao Tie, Pan Zhou, Neil Zhenqiang Gong, and Lichao Sun. 2025. Prompt Injection Attack to Tool Selection in LLM Agents. *arXiv preprint arXiv:2504.19793* (2025).
[139] Tianneng Shi, Jingxuan He, Zhun Wang, Linyu Wu, Hongwei Li, Wenbo Guo, and Dawn Song. 2025. Progent: Programmable Privilege Control for LLM Agents. *arXiv preprint arXiv:2504.11703* (2025).

[140] Tianneng Shi, Kaijie Zhu, Zhun Wang, Yuqi Jia, Will Cai, Weida Liang, Haonan Wang, Hend Alzahrani, Joshua Lu, Kenji Kawaguchi, et al. 2025. PromptArmor: Simple yet Effective Prompt Injection Defenses. *arXiv preprint arXiv:2507.15219* (2025).

[141] Yan Shvartzshnaider and Vasisht Duddu. 2025. Position: Contextual integrity is inadequately applied to language models. *arXiv preprint arXiv:2501.19173* (2025).

[142] Yan Shvartzshnaider, Zvonimir Pavlinovic, Ananth Balashankar, Thomas Wies, Lakshminarayanan Subramanian, Helen Nissenbaum, and Prateek Mittal. 2019. Vaccine: Using contextual integrity for data leakage detection. In *The World Wide Web Conference*. 1702–1712.

[143] Shoaib Ahmed Siddiqui, Radhika Gaonkar, Boris Köpf, David Krueger, Andrew Paverd, Ahmed Salem, Shruti Tople, Lukas Wutschitz, Menglin Xia, and Santiago Zanella-Béguelin. 2024. Permissive Information-Flow Analysis for Large Language Models. *arXiv preprint arXiv:2410.03055* (2024).

[144] Skyvern. 2025. Skyvern. https://www.skyvern.com/.

[145] Slack. 2025. Installing with OAuth. https://api.slack.com/authentication/oauth-v2 (accessed 14, April, 2025).

[146] Tobin South, Samuele Marro, Thomas Hardjono, Robert Mahari, Cedric Deslandes Whitney, Alan Chan, and Alex Pentland. 2025. Position: AI Agents Need Authenticated Delegation. In *ICML*.

[147] Joseph Spracklen, Ajaykumar Neupane, Sai Krishna Challagundla, and Jagannadh Vaidya. 2025. We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs. In *USENIX Security Symposium*.

[148] Georgios Syros, Anshuman Suri, Jacob Ginesin, Cristina Nita-Rotaru, and Alina Oprea. 2026. SAGA: A Security Architecture for Governing AI Agentic Systems. In *Network and Distributed System Security (NDSS) Symposium*.

[149] New York Times. 2025. South Korea Bans Downloads of DeepSeek, the Chinese A.I. App. https://www.nytimes.com/2025/02/17/business/south-korea-deepseek-china-ai.html.

[150] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[151] trailofbits. 2025. mcp-context-protector. https://github.com/trailofbits/mcp-context-protector.

[152] Lillian Tsai and Eugene Bagdasarian. 2025. Contextual Agent Security: A Policy for Every Purpose. In *Proceedings of the 2025 Workshop on Hot Topics in Operating Systems*. 8–17.

[153] Browser use. 2024. Browser use. https://browser-use.com/.

[154] W3C. 2025. Decentralized Identifiers (DIDs) v1.0. https://www.w3.org/TR/did-1.0/.

[155] Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. 2024. The instruction hierarchy: Training llms to prioritize privileged instructions. *arXiv preprint arXiv:2404.13208* (2024).

[156] Peiran Wang, Yang Liu, Yunfei Lu, Yifeng Cai, Hongbo Chen, Qingyou Yang, Jie Zhang, Jue Hong, and Ye Wu. 2025. AgentArmor: Enforcing Program Analysis on Agent Runtime Trace to Defend Against Prompt Injection. *arXiv preprint arXiv:2508.01249* (2025).

[157] Shouju Wang, Fenglin Yu, Xirui Liu, Xiaoting Qin, Jue Zhang, Qingwei Lin, Dongmei Zhang, and Saravan Rajmohan. 2025. Privacy in action: Towards realistic privacy mitigation and evaluation for llm-powered agents. *arXiv preprint arXiv:2509.17488* (2025).

[158] Xunguang Wang, Zhenlan Ji, Wenxuan Wang, Zongjie Li, Daoyuan Wu, and Shuai Wang. 2026. SoK: Evaluating Jailbreak Guardrails for Large Language Models. In *2026 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 1427–1446. doi:10.1109/SP63933.2026.00076

[159] Yifei Wang, Dizhan Xue, Shengjie Zhang, and Shengsheng Qian. 2024. BadAgent: Inserting and Activating Backdoor Attacks in LLM Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 9811–9827.

[160] Zhun Wang, Vincent Siu, Zhe Ye, Tianneng Shi, Yuzhou Nie, Xuandong Zhao, Chenguang Wang, Wenbo Guo, and Dawn Song. 2025. AgentVigil: Generic Black-Box Red-teaming for Indirect Prompt Injection against LLM Agents. In *Findings of the Association for Computational Linguistics: EMNLP*. Association for Computational Linguistics, 23159–23172.

[161] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

[162] Jiaxin Wen, Vivek Hebbar, Caleb Larson, Aryan Bhatt, Ansh Radhakrishnan, Mrinank Sharma, Henry Sleight, Shi Feng, He He, Ethan Perez, Buck Shlegeris, and Akbir Khan. 2025. Adaptive Deployment of Untrusted LLMs Reduces Distributed Threats. In *The Thirteenth International Conference on Learning Representations*.

[163] Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov. 2015. Android permissions remystified: A field study on contextual integrity. In *24th USENIX Security Symposium (USENIX Security 15)*. 499–514.

[164] Simon Willison. 2022. Prompt injection attacks against GPT-3. https://simonwillison.net/2022/Sep/12/prompt-injection/.

[165] Simon Willison. 2023. Delimiters won't save you from prompt injection. https://simonwillison.net/2023/May/11/delimiters-wont-save-you.

[166] Simon Willison. 2023. The Dual LLM pattern for building AI assistants that can resist prompt injection. https://simonwillison.net/2023/Apr/25/dual-llm-pattern/.

[167] Chen Henry Wu, Jing Yu Koh, Ruslan Salakhutdinov, Daniel Fried, and Aditi Raghunathan. 2024. Adversarial Attacks on Multimodal Agents. *arXiv preprint arXiv:2406.12814* (2024).

[168] Fangzhou Wu, Ethan Cecchetti, and Chaowei Xiao. 2024. System-Level Defense against Indirect Prompt Injection Attacks: An Information Flow Control Perspective. *arXiv preprint arXiv:2409.19091* (2024).

[169] Tong Wu, Shujian Zhang, Kaiqiang Song, Silei Xu, Sanqiang Zhao, Ravi Agrawal, Sathish Reddy Indurthi, Chong Xiang, Prateek Mittal, and Wenxuan Zhou. 2024. Instructional Segment Embedding: Improving LLM Safety with Instruction Hierarchy. In *The Thirteenth International Conference on Learning Representations*.

[170] Yuhao Wu, Franziska Roesner, Tadayoshi Kohno, Ning Zhang, and Umar Iqbal. 2025. IsolateGPT: An Execution Isolation Architecture for LLM-Based Systems. In *Network and Distributed System Security Symposium (NDSS)*.

[171] Yuhao Wu, Ke Yang, Franziska Roesner, Tadayoshi Kohno, Ning Zhang, and Umar Iqbal. 2025. Towards automating data access permissions in ai agents. *arXiv preprint arXiv:2511.17959* (2025).

[172] Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, et al. 2024. Guardagent: Safeguard llm agents by a guard agent via knowledge-enabled reasoning. *arXiv preprint arXiv:2406.09187* (2024).

[173] Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224* (2023).

[174] Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. 2024. Formal mathematical reasoning: A new frontier in ai. *arXiv preprint arXiv:2412.16075* (2024).

[175] Wenkai Yang, Xiaohan Bi, Yankai Lin, Sishuo Chen, Jie Zhou, and Xu Sun. 2024. Watch out for your agents! investigating backdoor threats to llm-based agents. *Advances in Neural Information Processing Systems* 37 (2024), 100938–100964.

[176] Yong Yang, Bo Xu, et al. 2025. PRSA: Prompt Reverse Stealing Attacks against Large Language Models. *arXiv preprint arXiv:2402.19200* (2025).

[177] Hongwei Yao, Haoran Shi, Yidou Chen, Yixin Jiang, Cong Wang, and Zhan Qin. 2025. Controlnet: A firewall for rag-based llm system. *arXiv preprint arXiv:2504.09593* (2025).

[178] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *ICLR*.

[179] Jingwei Yi, Yueqi Xie, Bin Zhu, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. 2025. Benchmarking and defending against indirect prompt injection attacks on large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*. 1809–1820.

[180] Jiahao Yu, Yangguang Shao, Hanwen Miao, and Junzheng Shi. 2025. PROMPTFUZZ: Harnessing Fuzzing Techniques for Robust Testing of Prompt Injection in LLMs. arXiv:2409.14729 [cs.CR] https://arxiv.org/abs/2409.14729

[181] Miao Yu, Fanci Meng, Xinyun Zhou, Shilong Wang, Junyuan Mao, Linsey Pang, Tianlong Chen, Kun Wang, Xinfeng Li, Yongfeng Zhang, et al. 2025. A survey on trustworthy llm agents: Threats and countermeasures. *arXiv preprint arXiv:2503.09648* (2025).

[182] Chen Yueh-Han, Nitish Joshi, Yulin Chen, He He, and Rico Angell. 2025. Monitoring LLM Agents for Sequentially Contextual Harm. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*.

[183] Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. 2024. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 10471–10506. doi:10.18653/v1/2024.findings-acl.624

[184] Chong Zhang, Mingyu Jin, Qinkai Yu, Chengzhi Liu, Haochen Xue, and Xiaobo Jin. 2024. Goal-guided generative prompt injection attack on large language models. In *2024 IEEE International Conference on Data Mining (ICDM)*. IEEE, 941–946.

[185] Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. 2025. Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-based Agents. In *The Thirteenth International Conference on Learning Representations*.

[186] Kaiyuan Zhang, Zian Su, Pin-Yu Chen, Elisa Bertino, Xiangyu Zhang, and Ninghui Li. 2025. LLM Agents Should Employ Security Principles. *arXiv preprint arXiv:2505.24019* (2025).

[187] Hongbin Zhong, Matthew Lentz, Nina Narodytska, Adriana Szekeres, and Kexin Rong. 2025. HoneyBee: Efficient Role-based Access Control for Vector Databases via Dynamic Partitioning. *arXiv preprint arXiv:2505.01538* (2025).

[188] Peter Yong Zhong, Siyuan Chen, Ruiqi Wang, McKenna McCall, Ben L Titzer, Heather Miller, and Phillip B Gibbons. 2025. Rtbas: Defending llm agents against prompt injection and privacy leakage. *arXiv preprint arXiv:2502.08966* (2025).

[189] Zexuan Zhong, Zonglin Huang, Alexander Wettig, and Danqi Chen. 2023. Poisoning Retrieval Corpora by Injecting Adversarial Passages. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 13764–13775.

[190] Andy Zhou, Bo Li, and Haohan Wang. 2024. Robust prompt optimization for defending language models against jailbreaking attacks. *Advances in Neural Information Processing Systems* 37 (2024), 40184–40211.

[191] Jialong Zhou, Lichao Wang, and Xiao Yang. 2025. GUARDIAN: Safeguarding LLM Multi-Agent Collaborations with Temporal Graph Modeling. *arXiv preprint arXiv:2505.19234* (2025).

[192] Kaijie Zhu, Xianjun Yang, Jindong Wang, Wenbo Guo, and William Yang Wang. 2025. MELON: Indirect Prompt Injection Defense via Masked Re-execution and Tool Comparison. In *Forty-second International Conference on Machine Learning*.

[193] Zod. 2025. Zod: Intro. https://zod.dev/.

[194] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043* (2023).

[195] Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. 2025. PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models. *USENIX Security Symposium* (2025). arXiv:2402.07867.

Received March 2026