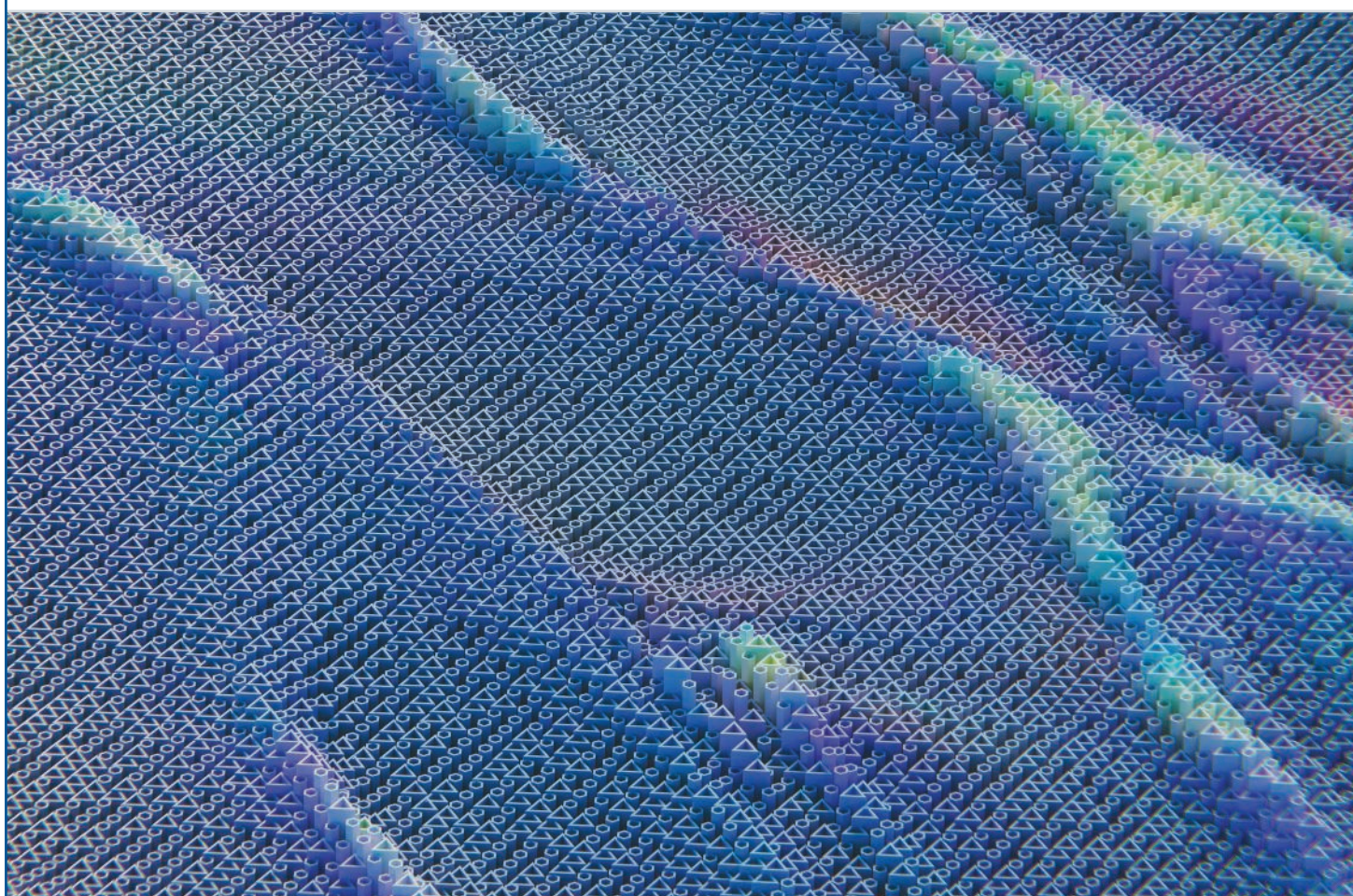


Google's Approach for Secure AI Agents: An Introduction

Santiago Díaz, Christoph Kern, Kara Olive



- 01 Introduction: the promise and risks of AI agents
- 02 Security challenges of how AI agents work
- 03 Key risks associated with AI agents
- 04 Core principles for agent security
- 05 Google's approach: a hybrid defense-in-depth
- 06 Navigating the future of agents securely

This paper is part of our ongoing effort to share Google's best practices for building secure AI systems. Read more about Google's Secure AI Framework at saif.google.

Introduction: The promise and risks of AI agents

We are entering a new era driven by AI agents—AI systems designed to perceive their environment, make decisions, and take autonomous actions to achieve user-defined goals. Unlike standard Large Language Models (LLMs) that primarily generate content, agents act. They leverage AI reasoning to interact with other systems and execute tasks, ranging from simple automation like categorizing incoming service requests to complex, multi-step planning such as researching a topic across multiple sources, summarizing the findings, and drafting an email to a team.

This increasing capability and autonomy promises significant value, potentially reshaping how businesses operate and individuals interact with technology. The rapid development of agent frameworks like Google's Agent Development Kit¹ and open source tools such as LangChain signals a move toward widespread deployment, suggesting "fleets" of agents operating at scale rather than just isolated instances. At the same time, the promise of agents introduces unique and critical security challenges that demand executive attention.



Key risks: Rogue actions and sensitive data disclosure

The very nature of AI agents introduces new risks stemming from several inherent characteristics. The underlying AI models can be unpredictable, as their non-deterministic nature means their behavior isn't always repeatable even with the same input. Complex, emergent behaviors can arise that weren't explicitly programmed. Higher levels of autonomy in decision-making increase the potential scope and impact of errors as well as potential vulnerabilities to malicious actors. Ensuring alignment—that agent actions reasonably match user intent, especially when interpreting ambiguous instructions or processing untrusted inputs—remains a significant hurdle. Finally, there are challenges in managing agent identity and privileges effectively.

These factors create the need for Agent Security, a specialized field focused on mitigating the novel risks these systems present. The primary concerns demanding strategic focus are **rogue actions** (unintended, harmful, or policy-violating actions) and **sensitive data disclosure** (unauthorized revelation of private information). A fundamental tension exists: increased agent autonomy and power, which drive utility, correlate directly with increased risk.

Traditional security paradigms alone are insufficient

Securing AI agents involves a challenging trade-off: enhancing an agent's utility through greater autonomy and capability inherently increases the complexity of ensuring its safety and security. Traditional systems security approaches (such as restrictions on agent actions implemented through classical software) lack the contextual awareness needed for versatile agents and can overly restrict utility. Conversely, purely reasoning-based security (relying solely on the AI model's judgment) is insufficient because current LLMs remain susceptible to manipulations like prompt injection and cannot yet offer sufficiently robust guarantees. Neither approach is sufficient in isolation to manage this delicate balance between utility and risk.

¹ <https://google.github.io/adk-docs/>

Our path forward: A hybrid approach

Building on well-established principles of secure software and systems design, and in alignment with Google’s Secure AI Framework (SAIF),² Google is advocating for and implementing a hybrid approach, combining the strengths of both traditional, deterministic controls and dynamic, reasoning-based defenses. This creates a layered security posture—a “defense-in-depth approach”³—that aims to constrain potential harm while preserving maximum utility. This strategy is built upon three core security principles detailed later in this document.

This paper first explains the typical workflow of an AI agent and its inherent security touchpoints. It then addresses key risks agents pose, introduces core security principles, and details Google’s hybrid defense-in-depth strategy. Throughout, guiding questions are suggested to help frame your thinking. A forthcoming, comprehensive whitepaper will delve deeper into these topics, offering more extensive technical details and mitigations.

² www.saif.google

³ https://google.github.io/building-secure-and-reliable-systems/raw/ch08.html#defense_in_depth

Security challenges of how AI agents work

To understand the unique security risks of agents, it's helpful to start with a mental model that describes a common agent architecture. While details vary, there are several broadly applicable concepts. We will briefly discuss each and identify the risks that apply to each component.

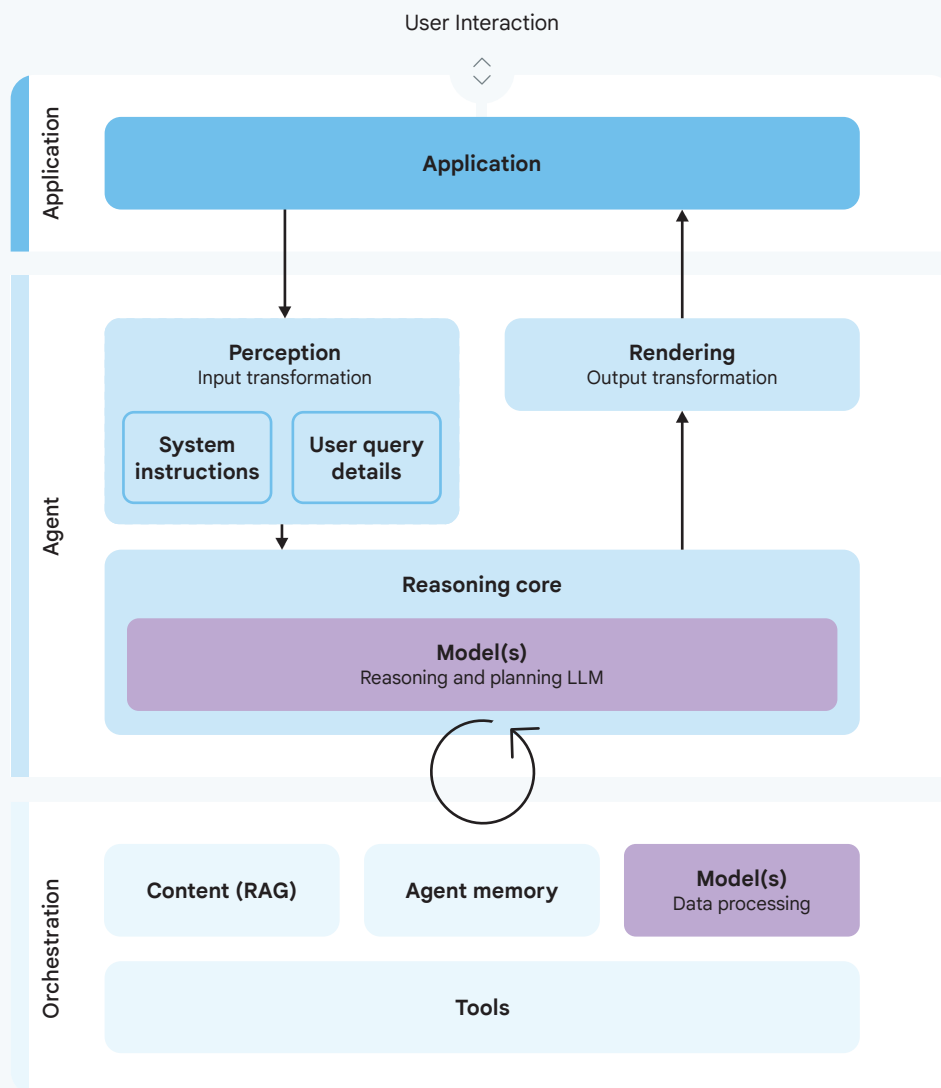


Figure 1: A simplified conceptual agent architecture for visualizing relevant security considerations

Input, perception and personalization: Agents begin by receiving input. This input can be a direct user instruction (typed command, voice query) or contextual data gathered from the environment (sensor readings, application state, recent documents). The input, which can be multimodal (text, image, audio), is processed and perceived by the agent and often transformed into a format the AI model can understand.

 *Security implication:* A critical challenge here is reliably distinguishing trusted user commands from potentially untrusted contextual data and inputs from other sources (for example, content within an email or webpage). Failure to do so opens the door to prompt injection attacks, where malicious instructions hidden in data can hijack the agent. Secure agents must carefully parse and separate these input streams. Personalization features, where agents learn user preferences, also need controls to prevent manipulation or data contamination across users.

Questions to consider

- ☐ What types of inputs does the agent process, and can it clearly distinguish trusted user inputs from potentially untrusted contextual inputs?
- ☐ Does the agent act immediately in response to inputs or does it perform actions asynchronously when the user may not be present to provide oversight?
- ☐ Is the user able to inspect, approve, and revoke permissions for agent actions, memory, and personalization features?
- ☐ If an agent has multiple users, how does it ensure it knows which user is giving instructions, apply the right permissions for that user, and keep each user's memory isolated?

System instructions: The agent's core model operates on a combined input in the form of a structured prompt. This prompt integrates predefined **system instructions** (which define the agent's purpose, capabilities, and boundaries) with the specific user query and various data sources like agent memory or externally retrieved information.

 *Security implication:* A crucial security measure involves clearly delimiting and separating these different elements within the prompt. Maintaining an unambiguous distinction between trusted system instructions and potentially untrusted user data or external content is important for mitigating prompt injection attacks.

Reasoning and planning: The processed input, combined with system instructions defining the agent's purpose and capabilities, is fed into the core AI model. This model reasons about the user's goal and develops a plan—often a sequence of steps involving information retrieval and tool usage—to achieve it. This planning can be iterative, refining the plan based on new information or tool feedback.

 *Security implication:* Because LLM planning is probabilistic, it's inherently unpredictable and prone to errors from misinterpretation. Furthermore, current LLM architectures do not provide rigorous separation between constituent parts of a prompt (in particular, system and user instructions versus external, untrustworthy inputs), making them susceptible to manipulation like prompt injection. The common practice of iterative planning (in a "reasoning loop") exacerbates this risk: each cycle introduces opportunities for flawed logic, divergence from intent, or hijacking by malicious data, potentially compounding issues. Consequently, agents with high autonomy undertaking complex, multi-step iterative planning present a significantly higher risk, demanding robust security controls.

Questions to consider

- ☐ How does the agent handle ambiguous instructions or conflicting goals, and can it request user clarification?
- ☐ What level of autonomy does the agent have in planning and selecting which plan to execute, and are there constraints on plan complexity or length?
- ☐ Does the agent require user confirmation before executing high-risk or irreversible actions?

Orchestration and action execution (tool use): To execute its plan, the agent interacts with external systems or resources via “tools” or “actions.” These could be through APIs for sending emails, querying databases, accessing file systems, controlling smart devices, or even interacting with web browser elements. The agent selects the appropriate tool and provides the necessary parameters based on its plan.

 **Security implication:** This stage is where rogue plans translate into real-world impact. Each tool grants the agent specific powers. Uncontrolled access to powerful actions (such as deleting files, making purchases, transferring data, and even adjusting settings on medical devices) is highly risky if the planning phase is compromised. Secure orchestration requires robust authentication and authorization for tool use, ensuring the agent has appropriately constrained permissions (reduced privilege) for the task at hand. Dynamically incorporating new tools, especially third-party ones, introduces risks related to deceptive tool descriptions or insecure implementations.

Questions to consider

- ☐ Is the set of available agent actions clearly defined, and can users easily inspect actions, understand their implications, and provide consent?
- ☐ How are actions with potentially severe consequences identified and subjected to specific controls or confinement?
- ☐ What safeguards (such as sandboxing policies, user controls, and sensitive deployment exclusions) prevent agent actions from improperly exposing high-privilege information or capabilities in low-privilege contexts?

Agent memory: Many agents maintain some form of memory to retain context across interactions, store learned user preferences, or remember facts from previous tasks.

 **Security implication:** Memory can become a vector for persistent attacks. If malicious data containing a prompt injection is processed and stored in memory (for example, as a “fact” summarized from a malicious document), it could influence the agent’s behavior in future, unrelated interactions. Memory implementations must ensure strict isolation between users and potentially between different contexts for the same user to prevent contamination. Users also need transparency and control over agent memory. Understanding these stages highlights how vulnerabilities can arise throughout the agent’s operational cycle, necessitating security controls at each critical juncture.

Response rendering: This stage takes the agent’s final generated output and formats it for display within the user’s application interface such as a web browser or mobile app.

 *Security implication:* If the application renders agent output without proper sanitization or escaping based on content type, vulnerabilities like Cross-Site Scripting (XSS) or data exfiltration (from maliciously crafted URLs in image tags, for example) can occur. Robust sanitization by the rendering component is crucial.

Questions to consider

- ☐ How is agent memory isolated between different users and contexts to prevent data leakage or cross-contamination?
- ☐ What stops stored malicious inputs (like prompt injections) from causing persistent harm?
- ☐ What sanitization and escaping processes are applied when rendering agent-generated output to prevent execution vulnerabilities (such as XSS)?
- ☐ How is rendered agent output, especially generated URLs or embedded content, validated to prevent sensitive data disclosure?

Key risks associated with AI agents

We think the inherent design of agents, combined with their powerful capabilities, can expose users to two major risks, what we call rogue actions and sensitive data disclosure. The following section examines these two risks and methods attackers use to realize them.

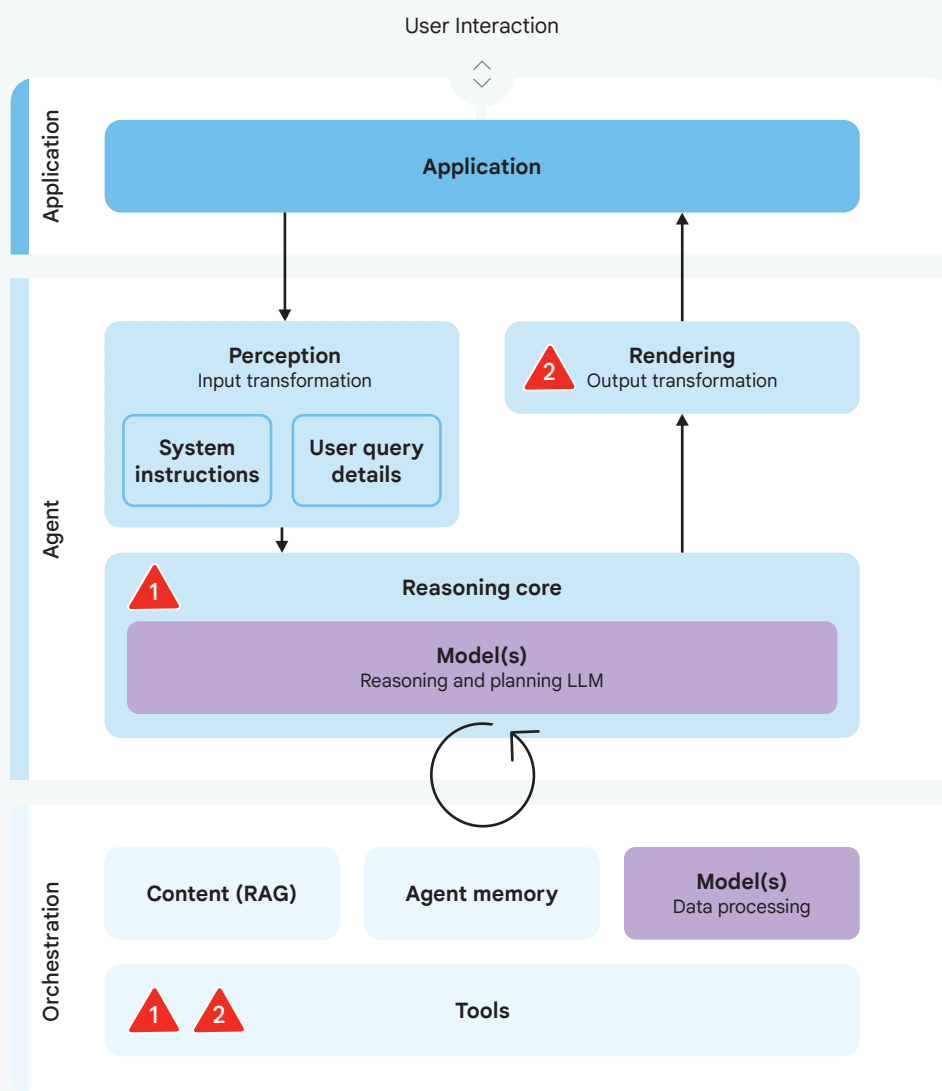


Figure 2: Risks associated with AI agents across the agent architecture: Rogue actions (1) and Sensitive data disclosure (2)

Risk 1: Rogue actions

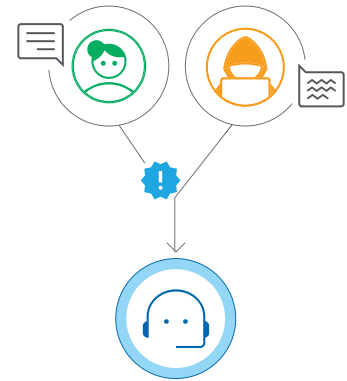
Rogue actions—unintended, harmful, or policy-violating agent behaviors—represent a primary security risk for AI agents.

A key cause is **prompt injection**: malicious instructions hidden within processed data (like files, emails, or websites) can trick the agent's core AI model, hijacking its planning or reasoning phases. The model misinterprets this embedded data as instructions, causing it to execute attacker commands using the user's authority. For example, an agent processing a malicious email might be manipulated into leaking user data instead of performing the requested task.

Rogue actions can also occur without malicious input, stemming instead from fundamental **misalignment or misinterpretation**. The agent might misunderstand ambiguous instructions or context. For instance, an ambiguous request like “email Mike about the project update” could lead the agent to select the wrong contact, inadvertently sharing sensitive information. Such cases involve harmful divergence from user intent due to the agent's interpretation, not external compromise.

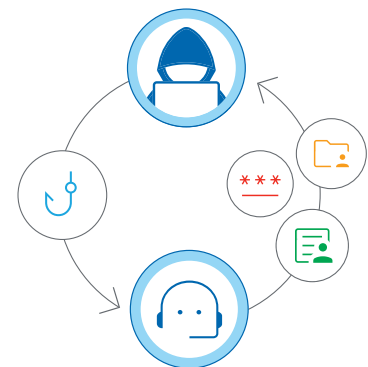
Additionally, unexpected negative outcomes can arise if the agent misinterprets complex interactions with external tools or environments. For example, it might misinterpret the function of buttons or forms on a complex website, leading to accidental purchases or unintended data submissions when trying to execute a planned action.

The potential impact of any rogue action scales directly with the agent's authorized capabilities and tool access. The potential for financial loss, data breaches, system disruption, reputational damage, and even physical safety risks escalates dramatically with the sensitivity and real-world impact of the actions the agent is permitted to take.



Risk 2: Sensitive data disclosure

This critical risk involves an agent improperly revealing private or confidential information. A primary method for achieving sensitive data disclosure is data exfiltration. This involves tricking the agent into making sensitive information visible to an attacker. Attackers often achieve this by **exploiting agent actions and their side effects**, typically driven by prompt injection. Attackers can methodically guide an agent through a sequence of actions. They might trick the agent into retrieving sensitive data and then leaking it through actions, such as embedding data in a URL the agent is prompted to visit, or hiding secrets in code commit messages.



Alternatively, data can be leaked by **manipulating the agent's output generation**. An attacker might trick the agent into including sensitive data directly in its response (like text or Markdown). If this output is rendered insecurely by the application (because it lacks appropriate validation or sanitization for display in a browser, for example), the data can be exposed. This can happen through crafted image URLs hidden in Markdown that leak data when fetched, for instance. This vector can also lead to Cross-Site Scripting (XSS).

The impact of data disclosure is severe, potentially leading to privacy breaches, intellectual property loss, compliance violations, or even account takeover, and the damage is often irreversible.

Mitigating these diverse and potent risks requires a deliberate, multi-faceted security strategy grounded in clear, actionable principles.

Core principles for agent security

To mitigate the risks of agents while benefiting from their immense potential, we propose that agentic product developers should adopt three core principles for agent security. For each principle, we recommend controls or techniques for you to consider.

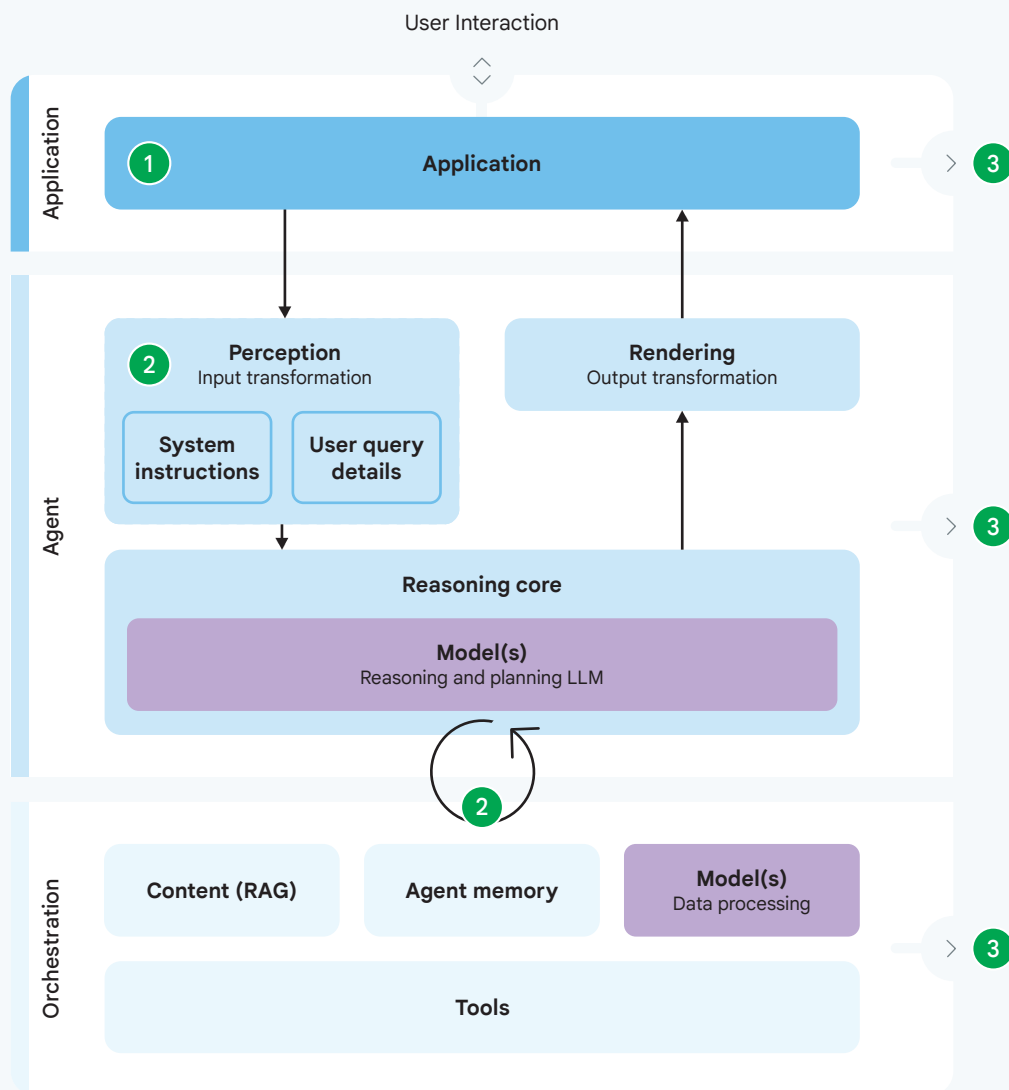
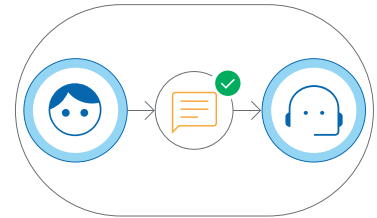


Figure 3: Controls relevant to AI agents: Agent user controls (1), Agent permissions (2), and Agent observability (3)

Principle 1: Agents must have well-defined human controllers

Agents typically act as proxies or assistants for humans, inheriting privileges to access resources and perform actions. Therefore, it is essential for security and accountability that agents operate under clear human oversight. Every agent must have a well-defined set of controlling human user(s). This principle mandates that systems must be able to reliably distinguish instructions originating from an authorized controlling user versus any other input, especially potentially untrusted data processed by the agent. For actions deemed critical or irreversible—such as deleting large amounts of data, authorizing significant financial transactions, or changing security settings—the system should require explicit human confirmation before proceeding, ensuring the user remains in the loop.

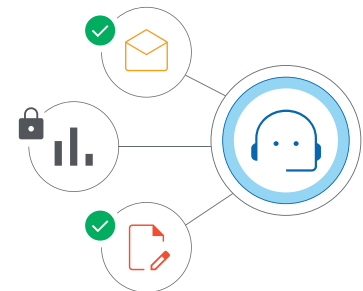


Furthermore, scenarios involving multiple users or agents require careful consideration. Agents acting on behalf of teams or groups need distinct identities and clear authorization models to prevent unauthorized cross-user data access or one user inadvertently triggering actions impacting another. Users should be given the tools to grant more granular permissions when the agent is shared, compared to the coarse-grained permissions that might be appropriate for a single-user agent. Similarly, if agent configurations or custom prompts can be shared, the process must be transparent, ensuring users understand exactly how a shared configuration might alter the agent's behavior and potential actions.

Controls: This principle relies on effective **Agent User Controls**, supported by infrastructure that provides distinct agent identities and secure input channels to differentiate user commands.

Principle 2: Agent powers must have limitations

An agent's powers—the actions it can take and the resources it can access—must be carefully limited in alignment with its intended purpose and its controlling user's risk tolerance. For example, an agent designed for research should not possess the power to modify financial accounts. General-purpose agents need mechanisms to dynamically confine their capabilities at runtime, ensuring only relevant permissions are active for any given query (for example, disallowing file deletion actions when the task is creative writing).



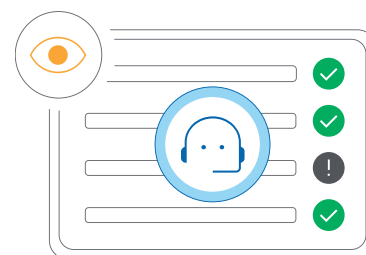
This principle extends traditional least privilege by requiring an agent's permissions to be dynamically aligned with its specific purpose and current user intent, rather than just being statically minimized. This distinction is crucial because model-based agents operate in potentially unbounded ecosystems and are susceptible to reasoning errors, necessitating context-aware limitations that go beyond the typical scope considered for human users or traditional software.

This principle necessitates that maximum permission levels must be definable and enforceable. Critically, agents must be prevented from escalating their own privileges beyond explicitly pre-authorized scopes. Users must also retain the ability to inspect and revoke any authority delegated to an agent. Implementing this requires adapting and strengthening fundamental Authentication, Authorization, and Auditing (AAA) infrastructure for agents, including verifiable agent identities, granular permission systems, and secure management of credentials like scoped OAuth tokens.

Controls: Implementing this principle requires defined **Agent Permissions** controls, enforced by robust Authentication, Authorization, and Auditing (AAA) infrastructure adapted for agents, and utilizing scoped credentials like OAuth tokens to manage access securely.

Principle 3: Agent actions and planning must be observable

Trust, effective debugging, security auditing, incident response, and user understanding all hinge on transparency into the agent’s activities. We cannot ensure an agent is acting faithfully or diagnose problems if its operations are entirely opaque. Therefore, agent actions, and where feasible, their planning processes, must be observable and auditable. This requires implementing robust logging across the agent’s architecture to capture critical information such as inputs received, tools invoked, parameters passed, outputs generated, and ideally, intermediate reasoning steps. This logging must be done securely, protecting sensitive data within the logs themselves.



Effective observability also means that the properties of the actions an agent can take—such as whether an action is read-only versus state-changing, or if it handles sensitive data—must be clearly characterized. This metadata is crucial for automated security mechanisms and human reviewers. Finally, user interfaces should be designed to promote transparency, providing users with insights into the agent’s “thought process,” the data sources it consulted, or the actions it intends to take, especially for complex or high-risk operations. This requires infrastructure investments in secure, centralized logging systems and APIs that expose action characteristics understandably.

Controls: Effective **Agent Observability** controls are crucial, necessitating infrastructure investments in secure, centralized logging systems and standardized APIs that clearly characterize action properties and potential side effects.

These three principles collectively form a strategic framework for mitigating agent risks.

Principle	Summary	Key Control Focus	Infrastructure Needs
1. Human controllers	Ensures accountability, user control, and prevents agents from acting autonomously in critical situations without clear human oversight or attribution.	Agent user controls	Distinct agent identities, user consent mechanisms, secure inputs
2. Limited powers	Enforces appropriate, dynamically limited privileges, ensuring agents have only the capabilities and permissions necessary for their intended purpose and cannot escalate privileges inappropriately.	Agent permissions	Robust AAA for agents, scoped credential management, sandboxing
3. Observable actions	Requires transparency and auditability through robust logging of inputs, reasoning, actions, and outputs, enabling security decisions and user understanding.	Agent observability	Secure/centralized logging, characterized action APIs, transparent UX

Figure 4: A summary of agent security principles, controls, and high-level infrastructure needs

Google’s approach: A hybrid defense-in-depth

Given the inherent limitations of current AI models and the practical impossibility of guaranteeing perfect alignment against all potential threats, Google employs a defense-in-depth strategy centered around a hybrid approach. This approach strategically combines traditional, deterministic security measures with dynamic, reasoning-based defenses. The goal is to create robust boundaries around the agent’s operational environment, significantly mitigating the risk of harmful outcomes, particularly rogue actions stemming from prompt injection, while striving to preserve the agent’s utility.

This defense-in-depth approach relies on enforced boundaries around the AI agent’s operational environment to prevent potential worst-case scenarios, acting as guardrails even if the agent’s internal reasoning process becomes compromised or misaligned by sophisticated attacks or unexpected inputs. This multi-layered approach recognizes that neither purely rule-based systems nor purely AI-based judgment are sufficient on their own.

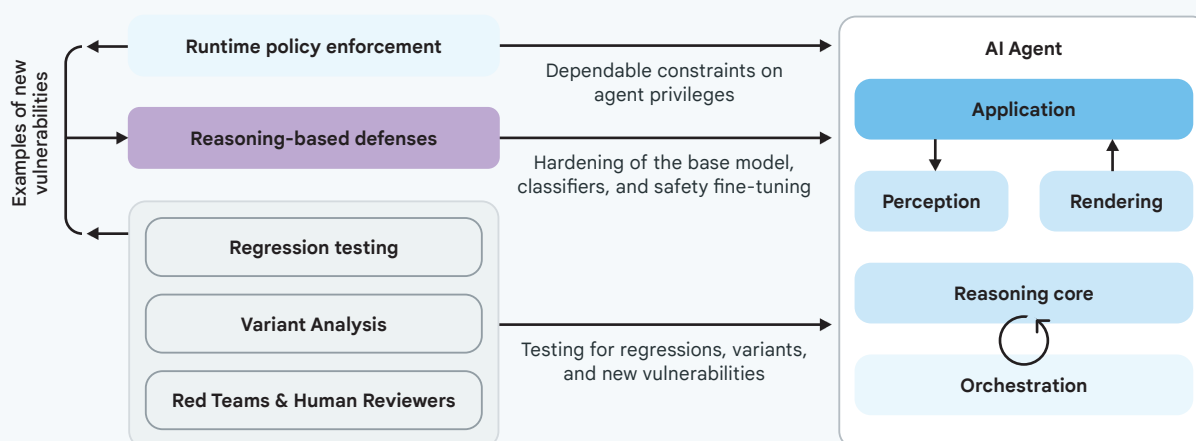


Figure 5: Google’s hybrid, defense-in-depth approach to AI agent security

Layer 1: Traditional, deterministic measures (runtime policy enforcement)

The first security layer utilizes dependable, deterministic security mechanisms, which Google calls *policy engines*, that operate outside the AI model’s reasoning process. These engines monitor and control the agent’s actions *before* they are executed, acting as security chokepoints.

When an agent decides to use a tool or perform an action (such as “send email,” or “purchase item”), the request is intercepted by the policy engine. The engine evaluates this request against predefined rules based on factors like the action’s inherent risk (Is it irreversible? Does it involve money?), the current context, and potentially the chain of previous actions (Did the agent recently process untrusted data?). For example, a policy might enforce a spending limit by automatically blocking any purchase action over \$500 or requiring explicit user confirmation via a prompt for purchases between \$100 and \$500. Another policy might prevent an agent from sending emails externally if it has just processed data from a known suspicious source, unless the user explicitly approves.

Based on this evaluation, the policy engine determines the outcome: it can **allow** the action, **block** it if it violates a critical policy, or **require user confirmation**. This deterministic enforcement provides reliable and predictable hard limits, is testable and auditable, and effectively limits the worst-case impact of agent malfunction, directly supporting the principle of Limited Agent Powers.

However, runtime policy engines also have limitations. Defining comprehensive policies for vast action ecosystems is complex and difficult to scale. Furthermore, policies often lack deep contextual understanding: a rule might block a legitimate action or allow a harmful one in a specific context not anticipated by the policy writers. For instance, sending an email after reading a document is sometimes desired (summarize and send) and sometimes harmful (exfiltrate data); a simple static rule struggles with this nuance.

Layer 2: Reasoning-based defense strategies

To complement the deterministic guardrails and address their limitations in handling context and novel threats, the second layer leverages reasoning-based defenses: techniques that use AI models themselves to evaluate inputs, outputs, or the agent’s internal reasoning for potential risks. This strategy aims to make the agent’s core reasoning and planning processes inherently more resilient to manipulation and better aligned with safe operational parameters.

One common technique is **adversarial training**. During the model’s development, it is exposed to numerous examples of attacks, such as various forms of prompt injection. The training process teaches the model the desired safe response, typically to recognize and ignore the malicious instructions embedded in data or to flag the input for scrutiny. This often involves using structural prompt conventions (like clear delimiters or role tagging) to help the model differentiate trusted commands from untrusted inputs.

Another technique involves using **specialized guard models**. These are typically smaller AI models specifically trained to act as **classifiers**. Functioning like security analysts, classifier models examine inputs or outputs for signs of attack (detecting instruction-like language in user data and checking for malicious patterns, for example) that simpler filters might miss.

Additionally, models can be employed for **analysis and prediction**. By analyzing the agent’s proposed plan of action, a model might predict the probability of that plan leading to an undesirable outcome, potentially flagging high-risk plans for review or triggering stricter policy enforcement.

The strength of reasoning-based defenses lies in their potential for handling dynamic behaviors and context. Models can learn to recognize more nuanced or evolving malicious patterns than static rules typically can. They significantly increase the difficulty and cost for attackers.

However, these strategies are non-deterministic and cannot provide absolute guarantees. Models can still be fooled by novel attacks, and their failure modes can be unpredictable. This makes them inadequate, on their own, for scenarios demanding absolute safety guarantees, especially involving critical or irreversible actions. They must work in concert with deterministic controls.

Validating your agent security: Assurance efforts

Supporting both layers are continuous assurance activities. **Regression testing** ensures fixes remain effective. **Variant analysis** proactively tests variations of known threats to anticipate attacker evolution. Complementing automated testing is crucial **human expertise**: red teams conduct simulated attacks, user feedback provides real-world insights, security reviewers perform audits, and external security researchers (engaged through programs like Google’s VRP⁴) provide diverse perspectives to uncover weaknesses.

4 <https://bughunters.google.com/>

Navigating the future of agents securely

AI agents represent a transformative technological leap, offering immense potential for productivity and innovation. However, their autonomy and power necessitate a proactive and sophisticated approach to security from the outset.

The risks of rogue actions and sensitive data disclosure are significant and stem from the core characteristics of agentic systems, particularly their reliance on complex AI reasoning and interaction with external tools and data. Relying solely on traditional security measures designed for predictable software, or conversely, placing blind faith in the imperfect reasoning of current AI models, is insufficient to meet the challenge.

We believe our hybrid strategy offers a pragmatic and necessary path forward. By layering the deterministic guarantees of runtime policy enforcement with the contextual adaptability of reasoning-based defenses, we can create a more resilient security posture. This approach is grounded in the fundamental principles that agents must operate under **well-defined human control**, their **powers must be carefully limited** according to risk and purpose, and their **actions and planning must be observable** for trust and accountability.

Continuous vigilance, rigorous testing, and sustained commitment to refining these hybrid approaches are paramount. The security of AI agents is not a problem to be solved once, but an ongoing discipline requiring sustained investment and adaptation. By prioritizing security considerations alongside capability development, we can work towards building AI agent systems that are not only powerful and useful, but also trustworthy and aligned with human interests as well, ensuring we harness their transformative potential responsibly.

Acknowledgements

This white paper is the result of the dedicated effort and collaboration of many people.

We extend our sincere gratitude to the following individuals for their valuable contributions, from insightful ideas to constructive feedback on the draft: Damian Bogel, Elie Bursztein, Drew Calcagno, Daniel DiBartolo, Daniel Fabian, Four Flynn, Dirk Göhmann, Royal Hansen, Evan Kotsovinos, David LaBianca, Nicolas Lidzborski, Shan Rao, Laurent Simon, Charley Snyder, and Jacint Szabo.

We would also like to thank Nikita Jain and Michael Temple for their essential support with the diagrams and visual design of the paper.

We appreciate the many other individuals who also supported this project.



Innovate securely with Google's defense infrastructure. Protect your organization across cloud and on-prem environments.