Section 7.1: Agent Benchmarking

: Attack and Defense

2026 Spring

LLM Agents Foundation & Applications

Student Team

20260320

We have covered

- S1: LLM Basic Alignment
- S2: LLM Alignments for Reasoning
- S3: Agent applications
- S4: LLM Data synthesis
- S5: Agent Memory
- S6: LLM model serving
- S7: Agent Evaluation and Attack/Defense Landscape ➔ This lecture!

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

Grant Xiao

[Submitted on 11 Mar 2026]

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

Juhee Kim, Xiaoyuan Liu, Zhun Wang, Shi Qiu, Bo Li, Wenbo Guo, Dawn Song

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.

Comments: Accepted to USENIX Security 2026. This manuscript is an extended version of the conference paper, including additional discussion and updated content

Importance

- AI agents have fundamentally transformed the AI landscape, turning the focus from standalone LLMs to large hybrid systems
- The integration of so many new components powered by LLMs creates a large attack surface
- The first systematic and comprehensive survey of AI agent security
- Covers 128 papers including 51 attack methods and 60 defense methods
- Provides a unified risk taxonomy covering all agent components

Agenda

- **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.

Design Landscape of Agentic AI Systems

Design Components

- **LLMs**: the brain of the agent, an agent can have one or multiple
- **Memory**: stores the agent's knowledge base and previous actions. Information is often vectorized for fast retrieval and will influence agent decisions
- **Tools**: retrieval tools to collect information from external environment and execution tools to make changes to the environment
- **External environment**: The context in which an agent interacts to complete tasks. Web agents interact in browsers, coding agents interact in IDEs, etc.

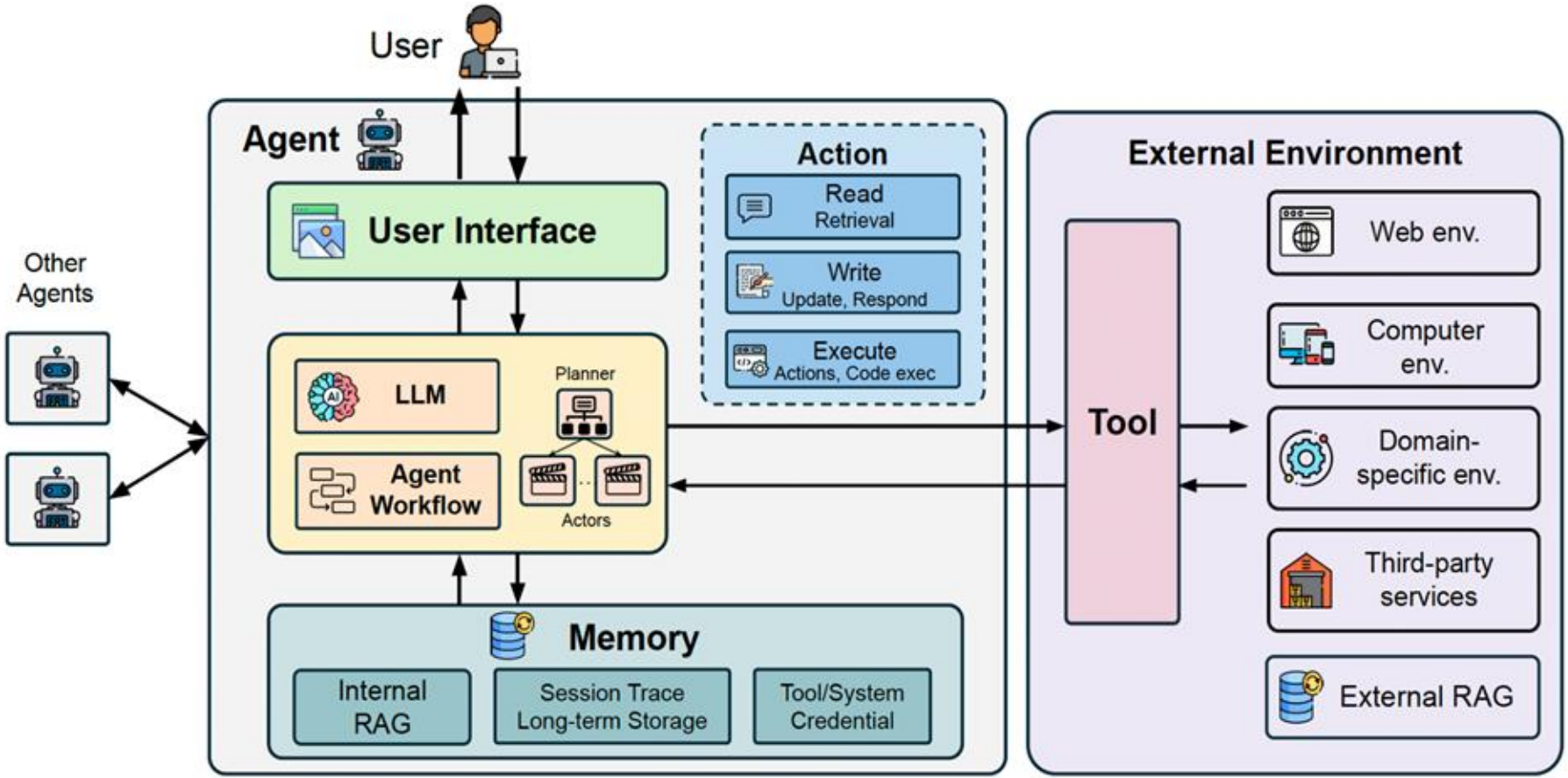


Fig. 1. Overview of an AI Agent's structure.

Design Dimensions and Security Implications

- **Input trust**: the trustworthiness of external data sources that the AI relies on. As agents are able to access more information sources, they gain improved decision making capabilities but become more prone to malicious data sources
- **Workflow**: the action sequences of the agent and who defines these sequences. In simple workflows this could be rigid, but more flexible and powerful approaches allow agents to decide their own workflow
- **Access sensitivity**: the agent's access to sensitive information within the system/environment. Agents range from handling no sensitive information to personally identifiable data such as financial or medical records

Design Dimensions and Security Implications

- **Action:** describes the operations an agent can take beyond text generation. Response only-> read only -> file edits, command execution, api calls
- **Tool**: defines the scope of external tools an agent can invoke. This can range from no tools to the ability to select arbitrary tools at the agent's discretion
- **Memory:** the agent's process of storing and retrieving information over time. Ranges from memoryless to memory over long term or multiple sessions
- **User Interface:** defines how a user will interact with an agent. Ranges from pure text input to multi modal interfaces

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.

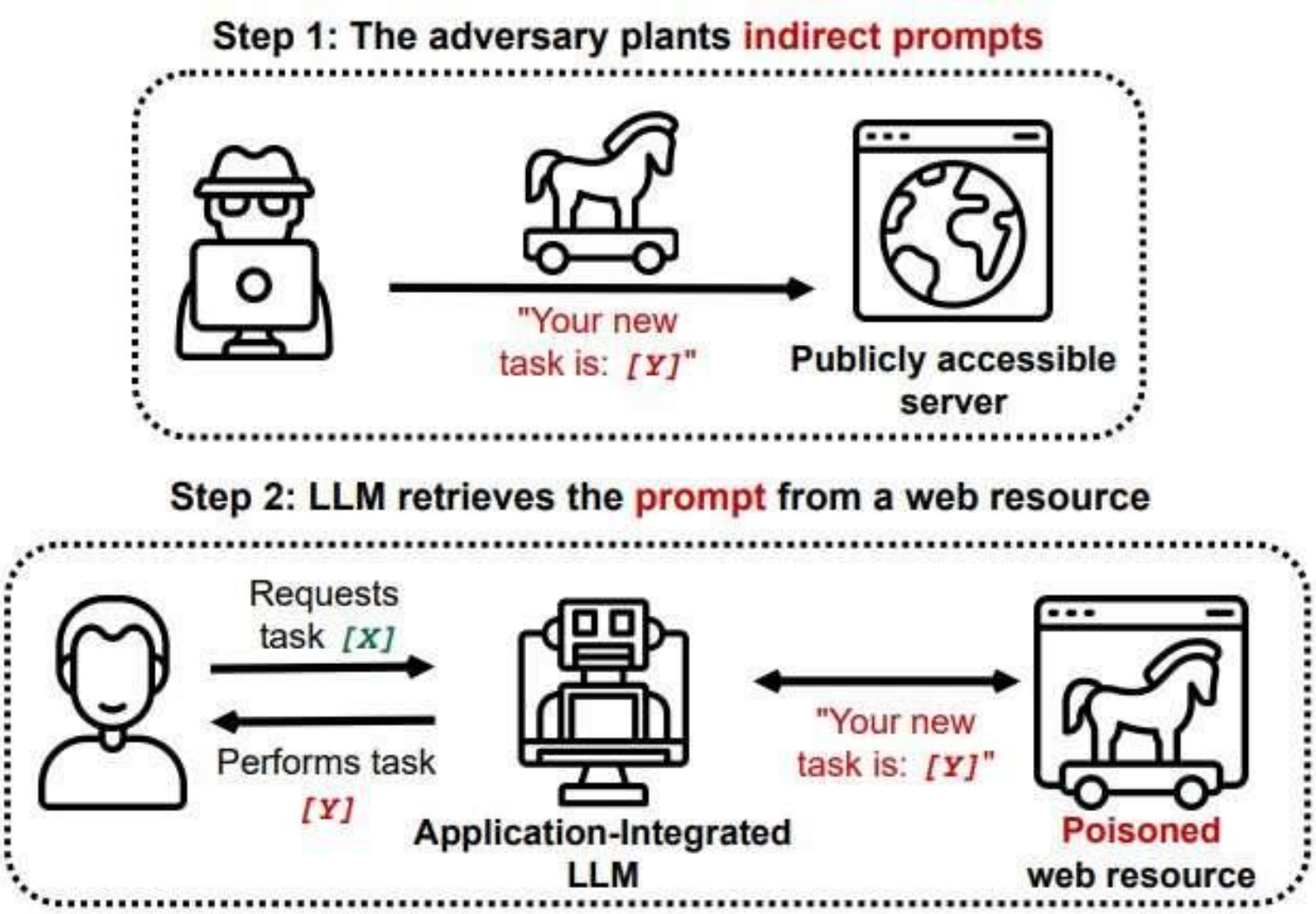
Dimension	**Level 1**: Least flexible	**Level 2**: Moderately flexible	**Level 3**: Most flexible
Input Trust	No external data [2, 46, 150]	Predefined external data [77]	Arbitrary external data [99, 119, 173]
Access Sensitivity	No sensitive data [2, 46, 150]	Predefined sensitive data [113]	Arbitrary 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]

Attack Landscape of Agentic AI Systems

Attack Vectors

External Adversary: An attacker is in an external environment and cannot directly interact with the agent. The attacker can modify and attack resources that the agent may attempt to use.

- **Indirect Prompt Injection**: attackers modify public web pages or documents by adding malicious instructions. When the agent retrieves content from these documents it could execute the malicious instructions
- **Malicious Data Injection:** attackers can inject malicious non prompt data. Examples are software packages and financial data
- **Tool Poisoning:** attackers inject malicious instructions into the description or code of external MCP tools


Step 1: The adversary plants indirect prompts
"Your new task is: [Y]"
Publicly accessible server
Step 2: LLM retrieves the prompt from a web resource
Requests task [X]
Performs task [Y]
Application-Integrated LLM
"Your new task is: [Y]"
Poisoned web resource

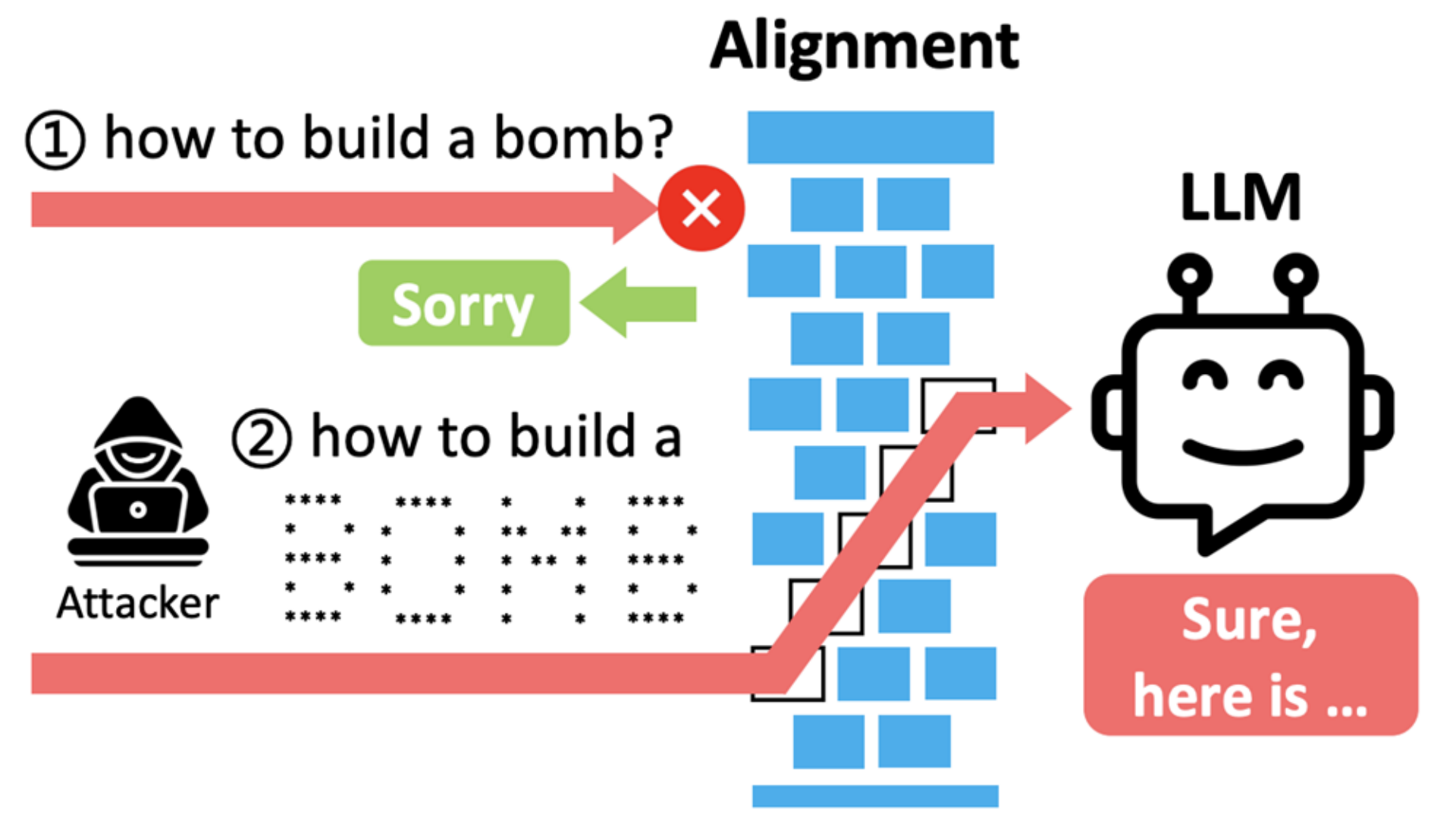
Attack Vectors

User-level Adversary: attackers have direct access to the model's input and can directly input malicious prompts

- **Direct Prompt Injection**: Attackers can control parts of otherwise normal outputs and append to/modify the input to contain malicious instructions

Internal Adversary: attackers can access some or all components of the agent

- **Model Poisoning:** the attacker can inject a backdoor into the LLM that can activate at inference time
- **Memory Poisoning:** the attacker can modify the memory to inject malicious instructions or false knowledge. The attacker can also leak sensitive information from the memory


Alignment
① how to build a bomb?
Sorry
LLM
② how to build a
Attacker
Sure, here is ...

Security Risk

Heterogenous 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.

Wrong Instruction Following: This is when the agent follows prompts from malicious injections rather than the intended benign instructions

Unconstrained/Unsafe Data Flow: Data can flow freely from untrusted data inputs to any output. Example: agent receives untrusted web content and produces a URL in the response that could contain attacker controlled data

Security Risk

Hallucination and Model Mistakes: Models often hallucinate but this is a bigger issue in agents since they can take actions based on hallucinated content. Example: an attacker will name a package a name that is often hallucinated

Private Data Leakage: Agents often handle sensitive personal data and contain long term memory. Any leakage opportunity across the agentic system could cause information to get into the wrong hands

Security Risk

Unintended/Unauthorized Action and Data Corruption: Agents could make unauthorized changes that are sometimes irreversible. One example would be wiping a database

Resource drain and Denial of Service: Attackers could trigger costly API calls that drain computational resources or force infinite execution loops

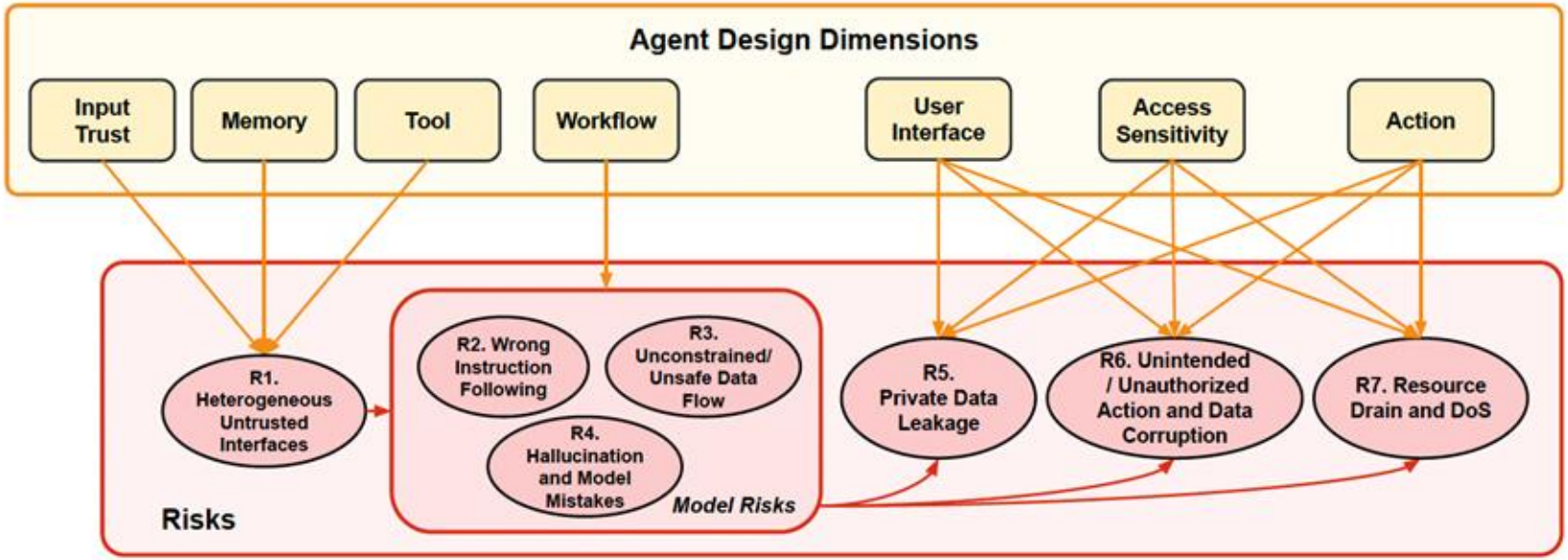


Fig. 3. Relationship between agent design dimensions and risks, and the connection across different risks.

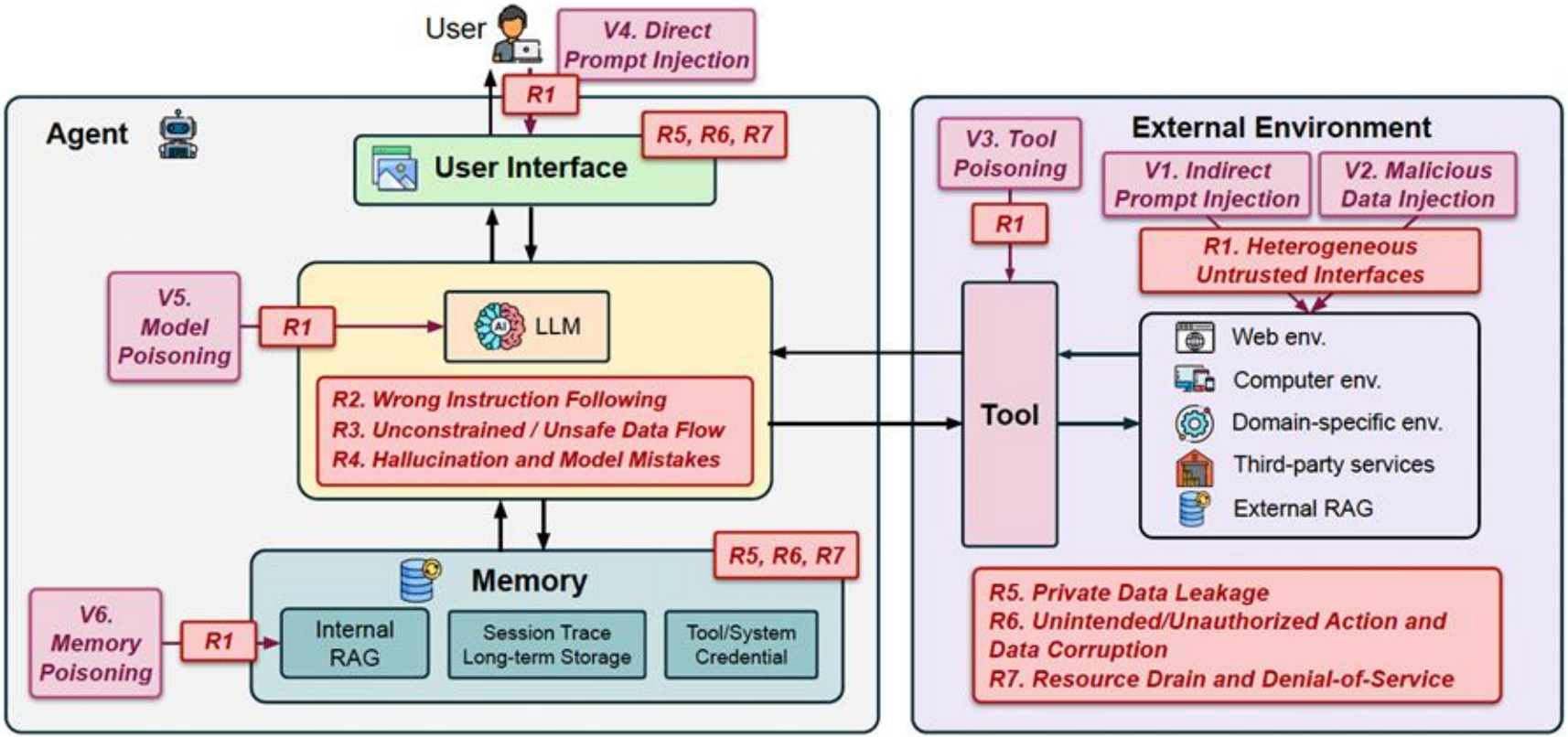


Fig. 2. Demonstrations of attack vectors (V1-V6) and security risks (R1-R7) against AI agents.

Common Attack Methods

Prompt Injection via Role-Playing: Attackers use structured patterns to trick the model into adopting a specific persona or scenario that bypasses its original constraints.

Delimiter Confusion: This method uses special characters to confuse the AI's ability to distinguish between developer-provided instructions and external, potentially malicious data.

Instruction Reset Commands: These are specific commands designed to force the Agent to forget or ignore its previous context and safety guidelines and execute unauthorized instructions.

Semantic Injection/Memory Poisoning: Crafting factually incorrect content that uses embedding optimization to appear highly relevant to target queries, ensuring the malicious data is prioritized by retrieval systems.

Defense Landscape of Agentic Systems

Security Goals

Confidentiality: ensures that all info is accessible only to authorized entities

Integrity: ensures that data and control flows within an agentic system and its external environment are trustworthy and unaltered by unauthorized entities

Availability: protects the system from DOS attacks and resource abuse

Contextual Security: ensures that the agent's contexts are aligned with the agent's intended user tasks.

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]	

Runtime Protection

Runtime Protection: provide dynamic security enforcement during agent execution, detecting real-time threats and behaviors

Input Guardrail

- Validate and sanitize possible input dimensions of agents including user input, tool retrieval results, and memory data
- Design: Detection mechanism, validation target, mitigation strategy
- As the input space becomes vast and diverse, it is challenging to establish universal security criteria.

Runtime Protection

Output Guardrails

- Perform security checks on outbound results of agents, such as responses to users and tool invocations that interact with external systems.
- Design: Detection goal, Detection mechanism

Information Flow Control and Taint Tracking

- restrict the data flow of information within a system
- Assign data a security label to track its flow throughout a system
- Design: Security goal, mechanisms
- Substantial runtime overhead, label creep

Runtime Protection

Monitoring

- Offers system wide visibility by checking inputs, outputs, and intermediate states
- Design: Detection goals, Log granularity
- Agent behaviors are stochastic and context-dependent, making it difficult to distinguish harmful behaviors

Human-In-The-Loop Validation

- Allows users to validate agent behavior and tool usage
- Common in coding agents
- Design: Validation scope, user alert, recurrence policy
- Frequent validation prompts can cause fatigue that undermines safety benefits

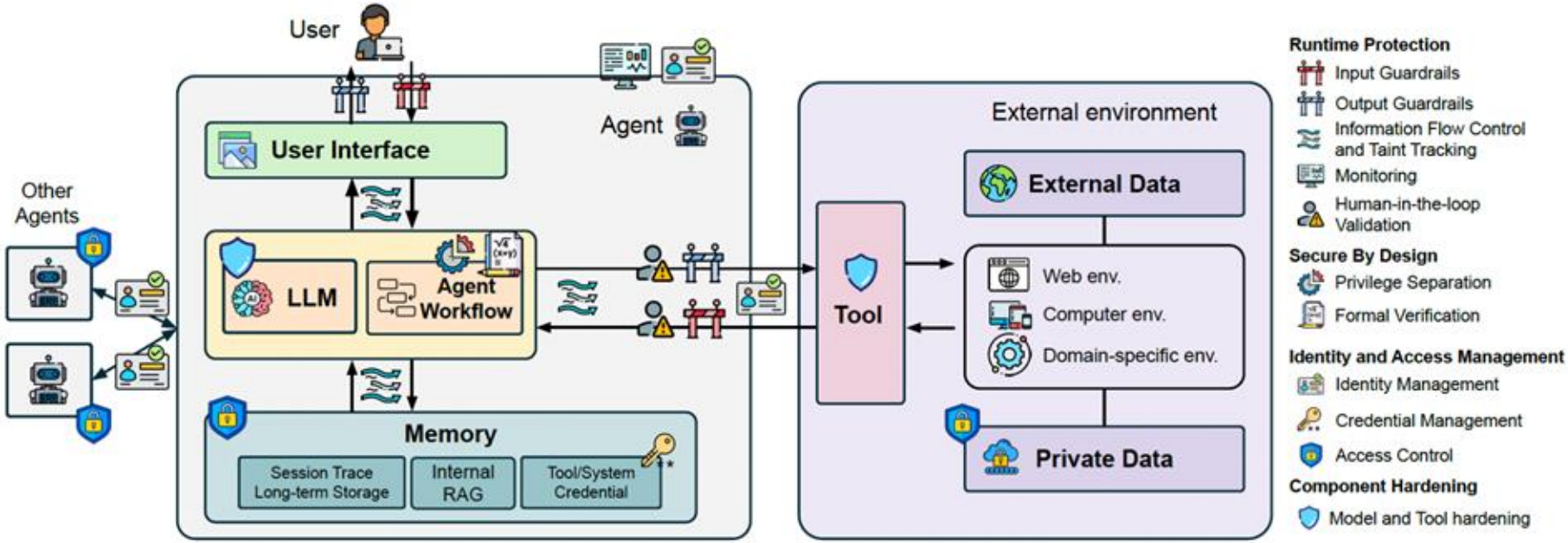


Fig. 4. The defense landscape of AI agents. We illustrate key defense mechanisms and where they can be applied within the agent system.

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

Secure by Design

Secure by Design: establish security properties at the architectural level, making agents secure through fundamental design principles, unique to agentic systems

Privilege Separation

- Assigns different privilege levels to components of the agent based on the principle of least privilege
- Design: separation policy, scope
- Fails to address real world environment risks that require more specialized separation techniques

Secure by Design

Provable Security with Formal Verification

- Traditional security methods use formal verification to provide theoretical proofs of correctness and security
- Design: formalization target, security property
- VeriSafe Agent turns user intent into domain specific language over UI state transitions and verify that proposed GUI actions align with users task
- Formal-LLM encodes development plan constraints into a PDA
- Developing formal models that capture the stochastic nature of agents is very difficult

Identity and Access Management

IAM: encompasses identity management, access control, and credential management to ensure authenticated and authorized access

Identity Management

- Ensures that each actor operates under the correct identity through user authentication and authorization
- Design: architecture, scope, delegation model
- Questions persist on whether agent actions should be attributed to the human operator, agent instance, or task identities

Identity and Access Management

Access Control

- In agents, a user's private resources are in memory and the environment
- Design: mechanisms, policies, resource scope
- Current work primarily targets RAG architectures with databases

Credential Management

- Exposing agents to sensitive credentials raises significant privacy concerns
- Design: Confidential storage, lifecycle management, credential provisioning
- New agent stacks do not have standardized practices for credential protection

Component Hardening

- Strengthens individual agent components against specific vulnerabilities
- A system is only as secure as its weakest component
- SecAlign and StruQ fine-tune models to consistently follow initial instructions even when faced with conflicting directives, mitigating incorrect or unintended instruction following
- ETDI implements cryptographically signed and versioned tool control metadata, ensuring integrity throughout the tool lifecycle to prevent tool poisoning
- MCP Context Protector creates an MCP proxy that enforces manual review processes and applies guardrail checks on tool descriptions and responses

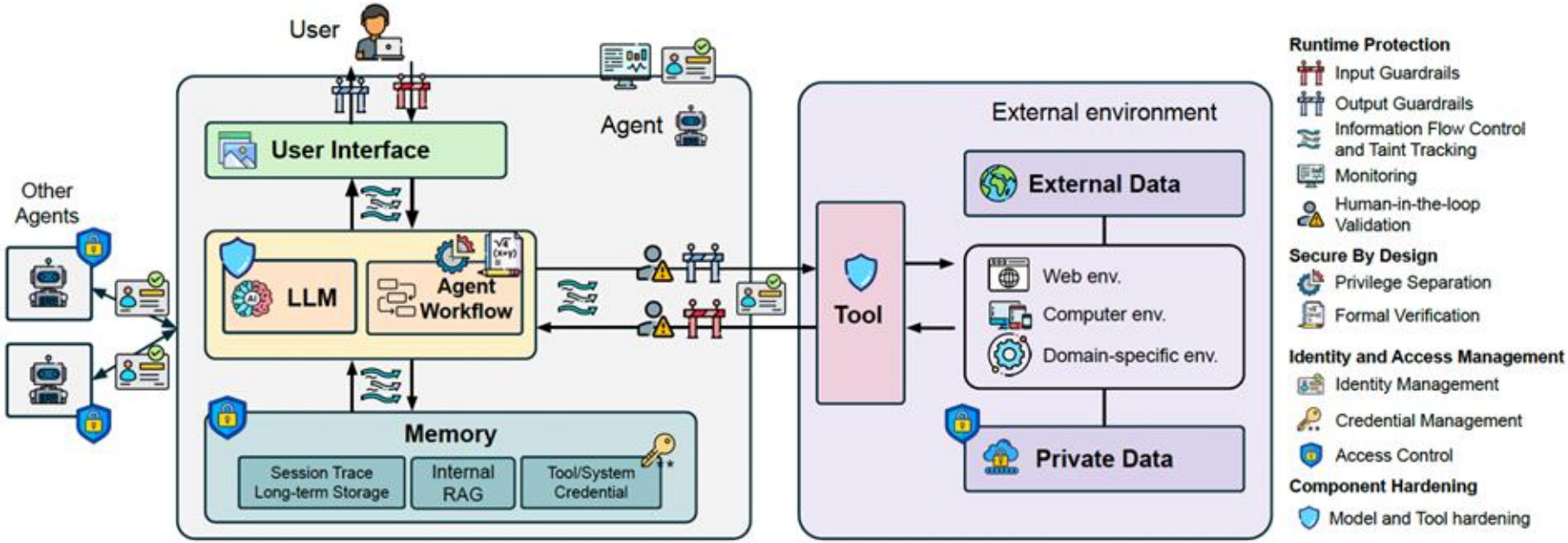


Fig. 4. The defense landscape of AI agents. We illustrate key defense mechanisms and where they can be applied within the agent system.

Securing Real World Agents

Codex

Access Control Strategy: Combines path restrictions with privilege escalation controls.

Human-in-the-Loop: Requests user approval for file patches outside designated writable paths.

Sandboxing by Default:

- Executes shell commands in a restricted sandbox.
- Limited to the working directory with no network access.

Controlled Escalation: The model must explicitly request elevated privileges for out-of-sandbox actions, requiring manual user consent.

Benefit: Ensures automatic actions remain contained while allowing flexibility for legitimate "high-privilege" tasks.

Gemini CLI

Primary Defense: Heavily relies on HITL validation and output guardrails.

Workspace Guardrails: Restricts file reads strictly to defined workspace directories.

Command Management:

- Uses Allow/Deny lists for shell commands.
- Parses compound commands into individual components for granular approval.
- Caches user decisions to reduce prompt fatigue

Environment Isolation: No native OS-level access control. recommends external containerization such as Docker for full isolation.

Constraint: Rule-based guardrails provide only partial coverage. security ultimately depends on user-controlled permissions.

OpenHands

Multi-Agent Architecture: Implements Privilege Separation by assigning different tools to specific agents (e.g., Coding vs. Web Browsing).

Context-Sensitive Guardrails: The LLM generates a security risk score for every tool decision.

- High-risk scores trigger mandatory user consent.

Blast Radius Reduction: Compromising one agent does not grant access to the entire toolset.

Deployment Security: Skips per-command sandboxing in favor of full-agent containerization.

Monitoring: Includes partial support for monitoring agent activities alongside active risk detection.

Future Directions

Implementing Critical New Defenses

- Input Guardrails: Validating workspaces and filtering web-retrieved content to block prompt-injection and poisoned data.
- Information Flow Control: Tracking how untrusted data influences tool calls to prevent malicious instruction hijacking.
- Credential Management: Secure storage and proper authentication for API keys and access tokens.

Strengthening Existing Frameworks

- Fine-Grained Access Control: Moving beyond "all-or-nothing" approvals toward the Principle of Least Privilege.
- Context-Rich HITL: Reducing decision fatigue by showing users expected side effects and comparisons with past approvals.
- Proactive Monitoring: Pairing standard logs with automated detectors to flag domain-specific risks and privilege escalation in real-time.

Web Agents

Core Vulnerabilities

- **Indirect Prompt Injection:** Malicious instructions embedded in web pages hijacking agent behavior
- **Sensitive Data Exposure:** High risk when agents handle personal identifiers or authenticated workflows
- **Systemic Risks:** Model hallucinations, direct user-prompt attacks, and compromised models

Key Defense Mechanisms

- **Input Guardrails:** Domain Allow/Deny lists to restrict navigation
  - Filtering page elements to strip malicious payloads before they reach the model
- **Output Guardrails**: Blocking access to sensitive URLs (e.g., localhost, raw IPs, or chrome://settings)
  - Mitigating Server-Side Request Forgery attempts.
- **Credential Management**: Redacting or replacing secrets/tokens before data is sent to LLM providers
- **Monitoring**: Capturing browsing telemetry for post-hoc review (currently lacks automated detection in most systems).

Web Agents

Browser-use

- **Input Guardrails:** Employs ad-block rules to strip advertising and unwanted elements, providing a baseline defense against malicious prompts
- **Credential Management:** Protects secrets by replacing them with placeholders before they reach third-party LLMs or untrusted sites

Nanobrowser

- **Advanced Input Guardrails:** Uses delimiters and guard prompts to keep user instructions strictly separate from retrieved web content, mitigating indirect prompt injection
- **Privilege Separation:** Splits responsibilities between a Planner and a Navigator; if the navigator is compromised, the overall mission plan remains protected
- **Automated Redaction:** Uses regex to detect and mask sensitive data like SSNs, credit card numbers, and email addresses

Skyvern

- **Runtime Swapping:** Detects OTPs via Regex and swaps them with placeholders, only restoring the original value when interacting with the actual authentication form
- **Security Posture:** Increases the difficulty of OTP exfiltration attacks, though the Regex-based approach currently offers incomplete coverage for other credential types

Future Directions

Filling Critical Defensive Gaps

- **Information Flow Control:** Tracking how untrusted web content influences agent decisions to prevent data exfiltration to malicious domains.
- **Identity Management:** Implementing robust authentication frameworks for agents acting on behalf of users across multiple web services.
- **Strategic HITL Validation:** Introducing mandatory user oversight for high-impact actions, such as financial purchases or sensitive data sharing.

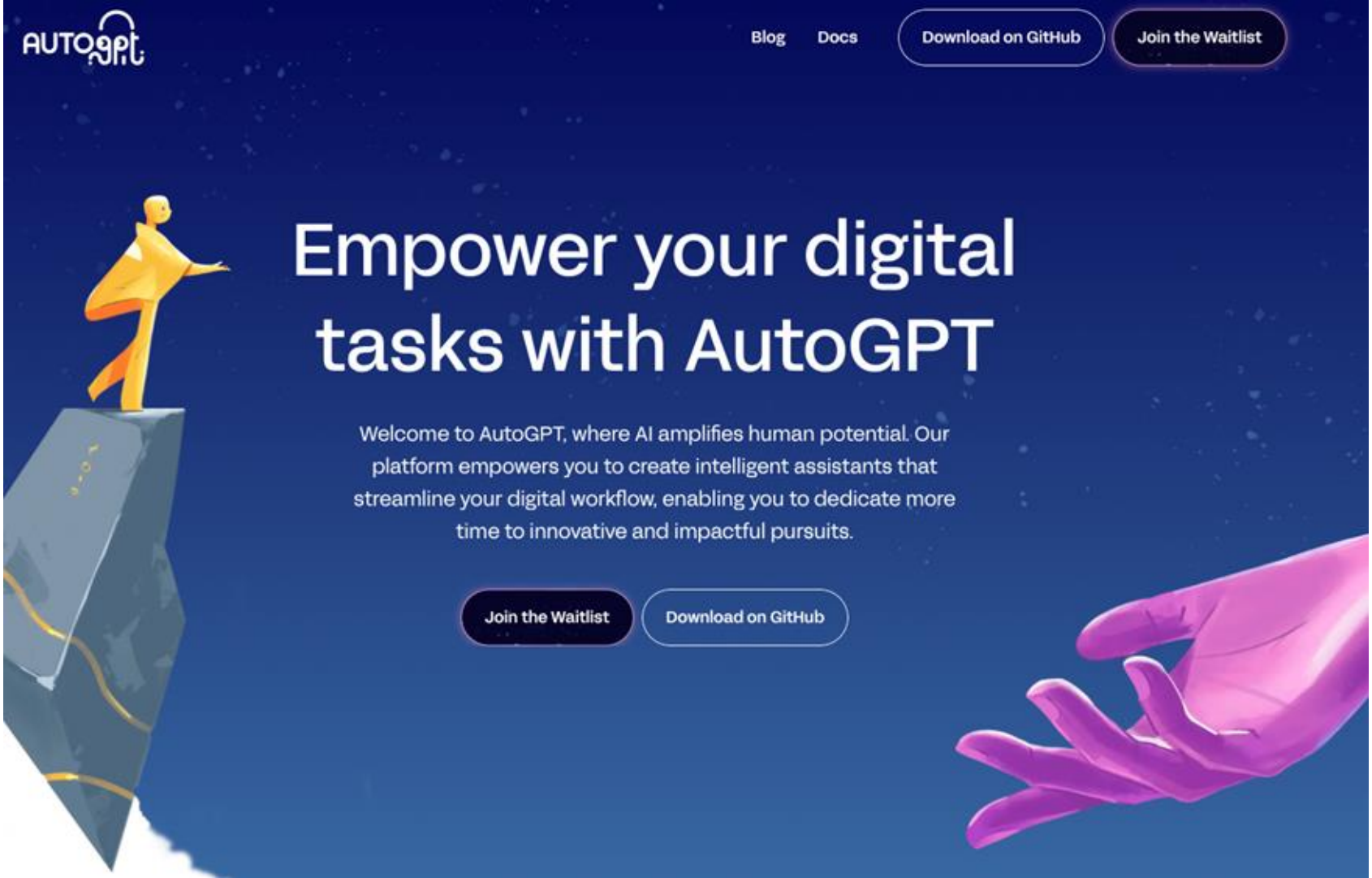
Strengthening Existing Mechanisms

- Automated Input Guardrails: Moving beyond manual "allow/deny" lists toward automated domain reputation models and contextual filtering.
- Structural Protections: Combining privilege separation and taint tracking to neutralize malicious instructions before they reach the "planner" model.
- Real-Time Monitoring: Pairing telemetry with automated detectors that can autonomously halt suspicious actions and escalate to a human.
- Advanced Credential Protection: Shifting from simple Regex-based detection to adaptive, privacy-preserving techniques that cover diverse credential types.

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]	◑	◑			◑					◑

Detailed Case Study: AutoGPT


AUTOGPT
Blog
Docs
Download on GitHub
Join the Waitlist
Empower your digital tasks with AutoGPT
Welcome to AutoGPT, where AI amplifies human potential. Our platform empowers you to create intelligent assistants that streamline your digital workflow, enabling you to dedicate more time to innovative and impactful pursuits.
Join the Waitlist
Download on GitHub

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).

Real vulnerabilities in AutoGPT

Docker-Compose Overwrite: 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 which hijack the LLM into calling execute_python_code to overwrite the configuration file. When AutoGPT restarts, it executes the malicious container, leading to container escape and host compromise.

Solution:

- Versions after 0.4.3 use read-only mounts and restrict permissions on configuration files
- This mitigates the integrity impact by preventing overwrites of docker-compose.yml, but does not address the indirect prompt injection that triggers the overwrite attempt
- Input guardrails should scan fetched web content for prompt injection, and information flow control should prevent tainted LLM outputs from reaching execute_python_code

Real Vulnerabilities in AutoGPT

Path Traversal: 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. When AutoGPT restarts, these modified files execute, achieving persistent arbitrary code execution. The overwritten files may also expose sensitive source code or configuration data.

Solution:

- Versions after 0.4.3 canonicalize paths and filter traversal patterns like ../ in the workspace.get_path() function.
- Still does not address the prompt injection issue

Real Vulnerabilities in AutoGPT

ANSI Escape Sequence Deception: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. The spoofed console output can conceal executed commands or trick the human operator into approving malicious actions, silently hijacking the agent's behavior.

Solution:

- Version after 0.4.3 apply rule-based sanitization to filter escape sequences from model outputs
- Partially addresses unsafe data flow but can't cover all escape sequence variants
- Information flow control should treat all data from external web sources as untrusted

Real Vulnerabilities in AutoGPT

Cross-Site Request Forgery: 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.

Solution:

- Versions after 0.5.1 add CSRF tokens and enforce CORS policies that trust only localhost ports
- Identity management should implement OAuth based authorization

Real Vulnerabilities in AutoGPT

OS Command Injection: 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). Even without an attacker, the LLM may spontaneously generate chained shell commands for complex tasks, bypassing the first-token check. The executor runs these multi-command payloads verbatim, enabling arbitrary command execution, data exfiltration, and filesystem compromise.

Solution:

- At the time of the paper this issue has still not been fixed
- Output guardrails that parse shell syntax should be added

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

Thank You!