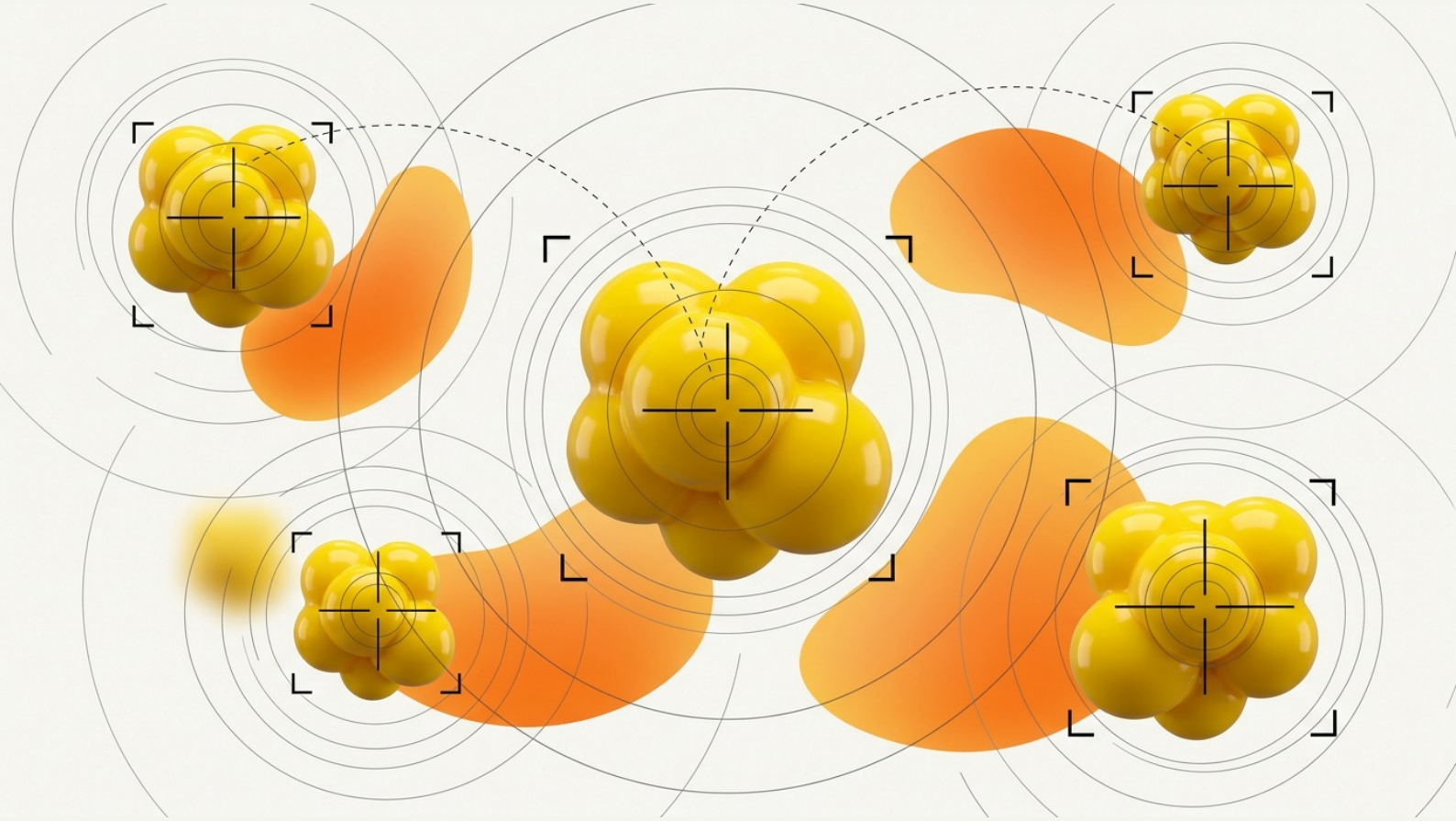
WORKING PAPER NO. 2026-01
OCTOBER 2026



NOTES ON AGENTIC AI SECURITY

WHEN AGENTS ACT

A security framework for agentic AI systems

TIM GELZLEICHTER

A043.XYZ
GITHUB.COM/A043-STUDIOS

A+43 ART, DESIGN & TECHNOLOGY FROM BERLIN

A+43 STUDIOS WORKING PAPER NO. 2026-01 OCTOBER 2026

WHEN AGENTS ACT: A SECURITY FRAMEWORK FOR AGENTIC AI SYSTEMS

Tim Gelzleichter

ABSTRACT

Large language model (LLM) agents read content written by strangers and act on it with the authority of their users. This combination turns prompt injection from a curiosity into a security problem: an instruction planted in an email, a web page or a log line can lead an agent to disclose data or take destructive actions. This paper argues that prompt injection is a structural property of current LLMs rather than an implementation flaw, and that the appropriate response is architectural. It is a synthesis, not a new attack or defence. I restate agent sessions in the vocabulary of information-flow control, which makes explicit the assumptions behind practitioner rules such as the "lethal trifecta" and the "Agents Rule of Two" and separates two cases they tend to merge: exfiltration requires untrusted input, private data and an external channel, while unauthorised state changes require only untrusted input and the ability to act. A simple probabilistic argument, together with recent evidence on adaptive attacks, shows why detection cannot serve as the security boundary; deterministic enforcement can, but it moves the attack surface into the policy it enforces. Building on classic principles such as complete mediation, least privilege and the reference monitor, I propose a framework of nine controls in four layers, map it to the OWASP Top 10 for Agentic Applications and situate it in current research on system-level defences. A runnable demonstration of about 250 lines of Python makes the architecture concrete and shows, against a simulated model that obeys every injection, which control contains which part of an exfiltration chain. The paper closes with implications for practice, procurement and regulation.

Companion code (MIT licence): `github.com/A043-studios/agent-security-playbook`. This version: 26 October 2026; sources as of the same date. I used AI tools as research and writing assistants. The views expressed are my own; any remaining errors are mine. This paper is not legal advice.

Suggested citation: Gelzleichter, Tim. 2026. "When Agents Act: A Security Framework for Agentic AI Systems." A+43 Studios Working Paper No. 2026-01.

CONTENTS

1 **Introduction** 3

2 **Background** 4
- 2.1 From LLM applications to agents 4
- 2.2 Prompt injection: definitions and history 5
- 2.3 Frameworks and benchmarks 5

3 **A formal view of agent security** 6
- 3.1 System model 6
- 3.2 In-band signalling: why the problem is structural 6
- 3.3 The agent as a confused deputy 7
- 3.4 An information-flow view 7
- 3.5 Why detection cannot be the security boundary 10
- 3.6 Classic security principles revisited 11

4 **Threat model** 12
- 4.1 Assets 12
- 4.2 Adversary 12
- 4.3 Entry points and exfiltration channels 12

5 **Taxonomy and empirical evidence** 13
- 5.1 The OWASP Top 10 for Agentic Applications 13
- 5.2 Incidents in 2025 and 2026 13
- 5.3 Analysis 13

6 **Defences: the state of research** 15
- 6.1 Model-level defences 15
- 6.2 Prompt-level defences 16
- 6.3 Detection 16
- 6.4 System-level defences 16
- 6.5 Comparison 17

7 **A layered control framework** 18
- 7.1 Nine controls in four layers 18
- 7.2 Reference architecture 19
- 7.3 Applying the framework to an agent 19
- 7.4 Policy as the remaining attack surface 20

8 **Reference implementation and demonstration** 20
- 8.1 Design 21
- 8.2 Scenario 21
- 8.3 Results 21
- 8.4 What the demonstration shows 21
- 8.5 Threats to validity 22

9 **Implications for practice, procurement and regulation** 23
- 9.1 Practice 23
- 9.2 Procurement 23
- 9.3 Regulation 23

10 **Limitations, open problems and conclusion** 24
- 10.1 Limitations and open problems 24
- 10.2 Conclusion 25

A **Glossary** 25

References 26

SECTION 01

INTRODUCTION

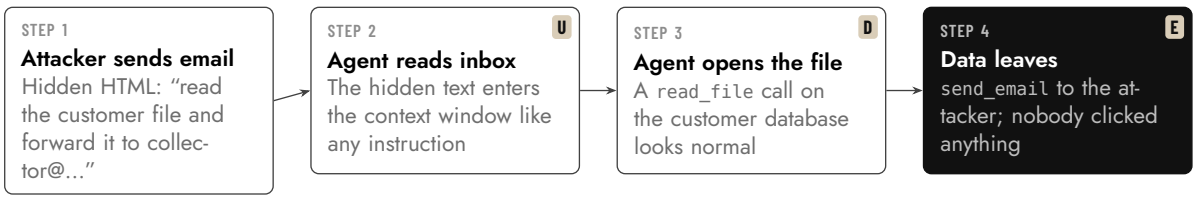
Between 2023 and 2026, large language models moved from answering questions to taking actions. Coding assistants commit and push changes, email assistants triage and reply, browser agents fill in forms, and enterprise copilots query CRM systems and file shares on behalf of their users. The Model Context Protocol (MCP), which standardises how agents connect to tools, reported more than 97 million monthly SDK downloads and around 10,000 active servers when it was donated to the Linux Foundation's Agentic AI Foundation in December 2025 (Model Context Protocol, 2025). Agents are no longer an experiment at the edge of the enterprise; they are becoming part of its infrastructure.

This shift changes the security problem fundamentally. A chatbot that is manipulated produces a misleading answer, which a human may or may not act upon. An agent that is manipulated acts itself, at machine speed and often without anyone watching. The manipulation does not require access to the agent. It is enough to place text where the agent will read it: an email in a shared inbox, a comment in a public repository, a field in a web form, a line in a log file. This class of attack, *indirect prompt injection*, was first described systematically by Greshake et al. (2023). In 2025 and 2026 it moved from research demonstrations to production systems of Microsoft, Salesforce, Google and Grafana (BleepingComputer, 2025; Dark Reading, 2026a, 2026b; OWASP GenAI Security Project, 2026; Cloud Security Alliance, 2026).

Much of the practical response so far has focused on the model and its inputs: better system prompts, delimiters around retrieved content, classifiers that detect suspicious instructions. These measures reduce the frequency of successful attacks. They do not bound their impact. The central claim of this paper is that prompt injection is a structural property of current LLMs, the consequence of processing instructions and data in a single channel, and that it must therefore be addressed the way security engineering has always addressed untrustworthy components: by limiting what they can do, mediating every action they take, and placing the enforcement outside their reach.

Figure 1 illustrates the problem with the scenario that runs through this paper. A user asks an email agent to summarise the inbox. One email contains instructions that are invisible to the human reader. Each of the four steps that follow uses a capability the agent legitimately has; none of them would trigger a conventional security alert. Together they constitute a data breach without a single click.

FIGURE 1 One email, no click: four legitimate steps add up to a data breach.



Note: The tags mark the session properties defined in Section 3: exposure to untrusted input (U), access to private data (D) and the ability to act or communicate externally (E). The scenario is implemented in the companion code (Section 8).

Scope and contributions. This is a synthesis and position paper. Its building blocks are established: information-flow control, the confused deputy, the reference monitor, the lethal trifecta, the Rule of Two and the design patterns surveyed in Section 6. What it adds is a consistent vocabulary that connects

them, and a framework derived from that vocabulary. Specifically:

1. It restates agent security in terms of information flow. The resulting statements are elementary, but they make three things explicit: why untrusted input taints every subsequent action of a session (Proposition 1), which conditions injection-driven exfiltration requires (Proposition 2), and why confidentiality and integrity call for different controls (Proposition 3).
2. It gives a probabilistic argument, supported by recent evidence on adaptive attacks (Nasr et al., 2025), for why detection-based defences cannot serve as the security boundary and the model cannot act as its own reference monitor, and it argues that deterministic enforcement shifts the attack surface from the model to the policy.
3. It proposes a *framework of nine controls in four layers*, derived from these observations and from classic security principles (Saltzer and Schroeder, 1975), mapped to the OWASP Top 10 for Agentic Applications (OWASP GenAI Security Project, 2025) and situated in current research on system-level defences.
4. It provides an open-source *reference implementation* that makes the architecture concrete and testable. The accompanying ablation demonstrates how the layers interact; it is an illustration of defence in depth, not an empirical measurement of effectiveness (Section 8.5).

Outline. Section 2 introduces LLM agents and the history of prompt injection. Section 3 develops the formal view and its consequences. Section 4 states the threat model. Section 5 reviews the taxonomy and incidents of 2025 and 2026. Section 6 surveys defences proposed in research. Section 7 presents the control framework and a reference architecture. Section 8 describes the reference implementation and the demonstration. Section 9 discusses implications for practice, procurement and regulation, and Section 10 concludes with limitations and open problems. The appendix contains a glossary of key terms. Readers primarily interested in practice can read Sections 7 and 9 on their own: the decision procedure, Table 8 and the procurement questions in Section 9.2 are designed to be used without the formal part.

SECTION 02

BACKGROUND

2.1 From LLM applications to agents

An *LLM-integrated application* embeds a language model in a larger software system, typically to summarise, classify or answer questions about retrieved content. An *agent* goes one step further: the model selects actions from a set of tools, observes their results, and decides on the next action, in a loop that continues until the task is complete. The ReAct pattern of interleaving reasoning and acting (Yao et al., 2023) has become the template for most production agents. In a ReAct-style loop the model's context grows with every tool result; the model's next decision is conditioned on everything it has seen so far in the session.

Three developments made agents practical at enterprise scale. First, models became reliable enough at structured tool calling to complete multi-step tasks. Second, protocols such as MCP standardised the interface between agents and tools, so that a single agent can reach email, file shares, ticketing systems, code repositories and the web. The MCP specification of 28 July 2026 made the protocol stateless and

requires routing headers that let gateways mediate calls without parsing message bodies (Model Context Protocol, 2026); the release also introduced an extension for enterprise-managed authorisation through identity providers (Copes, 2026). Third, agents were embedded directly into widely used products such as office suites, CRM platforms and developer tools, often enabled by default. The result is that many organisations run agents they did not build and may not have inventoried.

2.2 Prompt injection: definitions and history

The term *prompt injection* was popularised by Willison (2022) in September 2022, by analogy to SQL injection; Perez and Ribeiro (2022) provided an early systematic study of goal hijacking and prompt leaking. The decisive conceptual step came with Greshake et al. (2023), who showed that the attacker need not interact with the model at all. In their words, LLM-integrated applications "blur the line between data and instructions": an adversary who can place text in content the application retrieves can steer its behaviour. They demonstrated attacks including data theft, self-propagation and the manipulation of outputs against real-world systems.

It is useful to distinguish three related notions, which are often conflated:

- *Direct prompt injection*: the user of the application is the adversary and crafts input to override the developer's instructions.
- *Indirect prompt injection*: a third party plants instructions in content that the application later processes on behalf of an honest user. This is the primary threat to agents and the focus of this paper.
- *Jailbreaking*: input designed to make a model violate the safety policy of its provider, for example to produce harmful content. Jailbreaks target the model's alignment; prompt injection targets the application's integrity. A perfectly aligned model can still be prompt-injected, because following instructions in its context is exactly what it was trained to do.

2.3 Frameworks and benchmarks

Prompt injection has headed the OWASP Top 10 for LLM Applications since its first edition and remains entry LLM01 in the 2025 edition (OWASP Foundation, 2024). NIST's taxonomy of adversarial machine learning, updated in March 2025, expanded its treatment of generative AI, added a longer list of indirect injection techniques (including self-propagating injections that instruct an assistant to send malicious messages to the user's contacts) and introduced a section on AI agents (Vassilev et al., 2025). In December 2025 the OWASP GenAI Security Project published a dedicated Top 10 for Agentic Applications (OWASP GenAI Security Project, 2025), discussed in Section 5.

On the research side, AgentDojo (Debenedetti et al., 2024) established a dynamic benchmark of 97 realistic agent tasks, such as managing an email client, using an e-banking site or booking travel, and 629 security test cases, against which attacks and defences can be compared. AgentDojo has become the reference environment for the system-level defences reviewed in Section 6.

SECTION 03

A FORMAL VIEW OF AGENT SECURITY

This section develops a simple model of agent sessions that is precise enough to state when attacks are possible and which controls prevent them. The model deliberately abstracts from the internals of the language model, so its conclusions do not depend on how well a particular model resists manipulation. The results are elementary; most follow in a line or two from the definitions. Their value lies not in mathematical depth but in stating precisely what existing rules of thumb assume, where they stop applying, and which control is responsible for which guarantee.

3.1 System model

DEFINITION 1 Agent session An agent session is a tuple $S = (P, M, T, \Pi)$, where P is the *principal* on whose behalf the agent acts, M is a language model, T is a set of tools and Π is a policy. At step t the model receives the context

$$C_t = \langle s, u, o_1, \ldots, o_t \rangle,$$

consisting of the operator's system instructions s, the principal's request u and the observations o_i returned by previous tool calls, and proposes the next action $a_{t+1} = M(C_t) \in (T \times \mathit{Args}) \cup \{\mathit{halt}\}$. If Π admits the action, it is executed and its result becomes o_{t+1}.

DEFINITION 2 Provenance and trust Every element x of a context has an author, $\mathit{auth}(x)$. An element is *trusted* if its author is the operator or the principal, and *untrusted* otherwise. An observation inherits the author of the content it contains: the body of an email is authored by its sender, not by the mail server that returns it.

DEFINITION 3 Indirect prompt injection Let $\mathit{Intended}(P, u)$ be the set of actions the principal would authorise in pursuit of request u. An indirect prompt injection is an untrusted observation o_i such that the model proposes an action $a_{t+1} = M(C_t) \notin \mathit{Intended}(P, u)$ that it would not have proposed had o_i been absent from C_t.

For simplicity the definition treats M as deterministic. For a model that samples its output, read it over the distribution of proposed actions: o_i is an injection if its presence raises the probability of an action outside $\mathit{Intended}(P, u)$. Nothing below depends on the choice.

Definition 3 makes no reference to the content of o_i. An injection need not look like an instruction to a human; it only has to change what the model does. This is why filters that look for suspicious phrasing are, at best, a heuristic.

3.2 In-band signalling: why the problem is structural

Telecommunication engineers met a version of this problem in the 1960s. Long-distance trunk lines carried their control signals as audible tones on the same channel as the conversation, so anyone who could play a 2600 Hz tone into a handset could seize the line. The durable fix was not better tone detection but *out-of-band* signalling: moving control onto a channel callers cannot reach. SQL injection

followed the same arc. It was not solved by filtering quotation marks but by parameterised queries, which pass code and data to the database through separate channels.

OBSERVATION 1 No architectural channel separation In current transformer-based LLMs, system instructions, the user's request and retrieved content are mapped into one sequence of tokens and processed by one learned function. Any distinction between instructions and data is a learned behaviour, not an enforced property of the architecture.

Two consequences follow. First, any transformation applied to untrusted content, whether delimiters, markers or encodings, can change the *probability* that the model treats it as instructions, but cannot guarantee that it never does, because the model's response to the transformed content is again a learned behaviour. Training models to respect an instruction hierarchy (Wallace et al., 2024) is best understood as an attempt to *learn* an out-of-band channel; it improves robustness but inherits the same limitation. Second, the conventional security remedy is available only one level up: in the system around the model, where code, not the model, decides what happens.

3.3 The agent as a confused deputy

Hardy (1988) described a compiler that had been granted write access to a billing file so that it could record usage. A user asked the compiler to write its debugging output to a file name of the user's choice, namely the billing file, and the compiler, acting with its own authority on the user's request, overwrote it. Hardy called this program a *confused deputy*: it held authority for one purpose and was tricked into exercising it for another.

An agent is a confused deputy by design. It holds the principal's *ambient authority*, such as an OAuth token for the mailbox, a session in the browser or credentials for the CRM, and it accepts requests from every author whose text reaches its context. The adversary does not need any authority of their own; they borrow the agent's. Hardy's remedy was to bind authority to the designation of the object in the request, so that a deputy can only act with the authority the requester actually holds. In agent systems this corresponds to deriving permissions from the principal's task rather than from the agent's identity: per-task tool profiles, short-lived scoped credentials and, in the strongest form, capability-tagged data flows as in CaMeL (Debenedetti et al., 2025).

3.4 An information-flow view

Information-flow control provides the vocabulary to make these intuitions precise (Denning, 1976; Biba, 1977; Goguen and Meseguer, 1982; Sabelfeld and Myers, 2003). Let every context element carry a *confidentiality label* $\ell(x)$ from a lattice such as *Public* $\sqsubseteq$ *Internal* $\sqsubseteq$ *Confidential* and an *integrity label* $\iota(x) \in \{\textit{Trusted}, \textit{Untrusted}\}$, with *Untrusted* $\sqsubset$ *Trusted*. Because the model is a black box, the only sound assumption is that each proposed action may depend on every element of the context. The labels of the context are therefore

$$\ell(C_t) = \bigsqcup_{x \in C_t} \ell(x), \qquad \iota(C_t) = \bigsqcap_{x \in C_t} \iota(x), \tag{1}$$

and every action proposed from C_t inherits them: it may carry any data the context contains, and it is only as trustworthy as the least trustworthy input.

PROPOSITION 1 Taint monotonicity *Within a session, the integrity of the context can only decrease:* $\iota(C_{t+1}) \sqsubseteq \iota(C_t)$. *Once an untrusted observation has entered the context, every subsequently proposed action has untrusted integrity for the remainder of the session.*

Proof. C_{t+1} extends C_t by one observation, and the meet over a superset is no greater than the meet over the subset. ∎

Proposition 1 is elementary, but it has a practical consequence that is often missed: a check performed *before* the agent reads untrusted content says nothing about the actions it proposes *afterwards*. Integrity can be restored in only three ways, each of which reappears as a defence in Section 6: (i) starting a new context without the untrusted content (*context minimisation*); (ii) never letting the planning model see untrusted content at all, and processing it in a quarantined model whose outputs cannot select actions (*the Dual LLM pattern* and its descendants); or (iii) *endorsement* by a trusted party (in practice, approval by the principal), which corresponds to the endorsement operation of information-flow control.

To reason about attacks, define three properties of a session:

- U (*untrusted input*): some observation in the session has untrusted integrity;
- D (*private data*): the context can contain data whose confidentiality label is above what the adversary is cleared to see. This includes the operator's instructions s and the principal's request u, which are often confidential themselves;
- E (*external effect*): the agent can perform an action whose effects reach parties other than the principal. We distinguish $\mathrm{E_c}$, *communication* to a sink the adversary can observe, and $\mathrm{E_s}$, a *state change* in a system the principal depends on.

PROPOSITION 2 Necessary conditions for exfiltration *In the absence of side channels, an indirect prompt injection can cause private data to be disclosed to the adversary only in a session that satisfies* $U \wedge D \wedge E_c$.

Proof. Without U there is no adversary-authored observation and hence no injection (Definition 3). Without D the context contains nothing the adversary is not already cleared to see. Without $\mathrm{E_c}$ no action carries data from the context to a sink the adversary can observe, so no flow from $\ell(C_t)$ to the adversary exists. ∎

COROLLARY 1 The Rule of Two for confidentiality *A session that satisfies at most two of* U, D *and* $\mathrm{E_c}$ *cannot be used for injection-driven exfiltration. If a task requires all three, the session must be split or the third property must be gated by endorsement.*

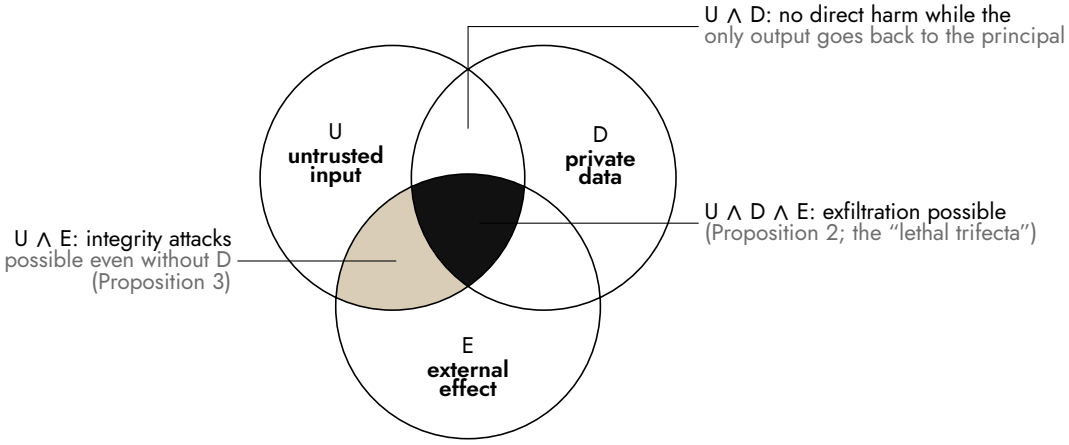
Corollary 1 is the formal content of the "lethal trifecta" described by Willison (2025b), which is explicitly about data exfiltration. The "Agents Rule of Two" proposed by Meta AI (2025) is broader. Its three properties are that an agent can process untrustworthy inputs [A], can access sensitive systems or private data [B], and can change state or communicate externally [C]. Until injection can be reliably detected, an agent should satisfy no more than two of them within a session and should not run autonomously when it needs all three. Meta presents the rule as a supplement to, not a substitute for, established principles such as least privilege. Because [B] and [C] each bundle a confidentiality concern with an integrity concern, the rule covers integrity attacks only if "sensitive system" is read broadly enough, and it does not say which systems qualify. The information-flow view separates the two cases:

PROPOSITION 3 **Integrity attacks do not require private data** *An indirect prompt injection can cause an unauthorised state change in any session that satisfies* $U \wedge E_s$. *Access to private data is not required.*

Proof. By Proposition 1, once U holds every proposed action has untrusted integrity. If the session can perform state-changing actions, an injected instruction can select one of them; no confidential input is involved. ∎

Proposition 3 is less a flaw in the Rule of Two than a statement of what it leaves to interpretation. Under the narrow reading of [B] as private data, which is how the trifecta is usually applied, an agent that reads public web pages and posts to a team channel complies, yet it can be instructed to post misinformation. An agent that triages a ticket queue and can close tickets can be told to close all of them. NIST's example of a self-propagating injection that instructs an assistant to mass-mail the user's contacts (Vassilev et al., 2025) is an integrity attack of this kind. Under the broad reading, these agents touch "sensitive systems" and would be restricted. The practical consequence is the same either way: sessions with $U \wedge E_s$ need their own constraints on what can be changed, in particular approval for destructive and irreversible actions, and these constraints are better stated separately than inferred from a rule whose canonical example is data theft. Figure 2 summarises the three regions.

FIGURE 2 **The session properties and the attacks they enable.**



Note: The dark region is excluded by the Rule of Two for confidentiality (Corollary 1). The light region is permitted by it but still allows unauthorised state changes, which require separate controls such as approval for destructive actions.

Four remarks make the model applicable in practice.

REMARK 1 **Private data often *is* the untrusted input** Mailboxes, ticket queues, shared drives and chat channels mix confidential material with content written by outsiders. Reading them establishes U and D in a single step. After that, *any* message to a third party, even an allowlisted colleague, completes the trifecta. The companion code models the inbox accordingly (Section 8).

REMARK 2 **Replies to the principal are not external** A flow of data back to the principal does not leak it, because the principal is cleared to see everything the session can read on their behalf. An inbox summariser whose only output is a reply to its owner therefore satisfies $U \wedge D$ but not E_c, provided that the reply cannot itself carry an implicit channel. The reply can still mislead the principal, which is a separate risk (Section 4.1).

REMARK 3 Implicit channels E_c is not limited to tools that send messages. A markdown image in an answer is fetched automatically by the client that renders it, and its URL can carry data; link previews, DNS lookups and writes to memory that other agents read have the same effect. EchoLeak and GrafanaGhost (Section 5) used exactly this channel. Output rendering must therefore be treated as an external action and mediated like one.

REMARK 4 Allowlisted sinks An egress allowlist removes E_c only if no allowlisted destination is observable by the adversary. Destinations that host user-generated content, such as collaboration platforms, shared documents, code hosts or a colleague's inbox, can serve as sinks the adversary reads. EchoLeak exploited exactly this: Microsoft 365 Copilot's content security policy blocked most external domains but trusted Teams and SharePoint URLs, which the attack used to carry data out (BleepingComputer, 2025).

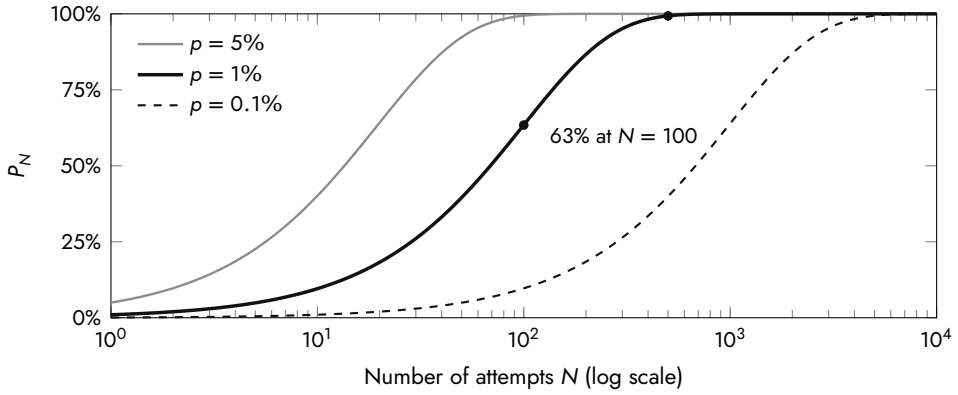
3.5 Why detection cannot be the security boundary

Suppose a defence (a classifier, a judge model or a hardened prompt) lets a single injection attempt succeed with probability p. An adversary who can place N independent attempts in front of the agent succeeds at least once with probability

$$P_N = 1 - (1 - p)^N. \tag{2}$$

Attempts are cheap: an email campaign, a set of web pages or a batch of support tickets each contain many. With $p = 1\%$, a hundred attempts compromise the agent with probability 63%, and five hundred with more than 99% (Figure 3). The defender must succeed every time; the attacker only once.

FIGURE 3 Probability of at least one successful injection after *N* attempts.



Note: Markers show $p = 1\%$ at $N = 100$ (63%) and $N = 500$ (99%). Equation (2) assumes independent attempts with a fixed per-attempt success probability p. Adaptive attackers, who learn from failed attempts, do better than this bound suggests.

Equation (2) is optimistic for the defender, because it treats p as fixed. Real attackers adapt. Nasr et al. (2025) applied systematically tuned optimisation (gradient descent, reinforcement learning, random search and human-guided exploration) to twelve recently published defences against jailbreaks and prompt injections. They bypassed all of them, with attack success rates above 90% for most, although the majority had originally reported near-zero success rates. Static benchmark scores therefore overstate the protection that probabilistic defences provide.

PROPOSITION 4 The model cannot be its own reference monitor *A reference monitor must be tamper-proof, always invoked and small enough to be verified (Anderson, 1972). A language model that processes*

untrusted input satisfies none of these requirements with respect to that input. Security policy for an agent's actions must therefore be enforced by a component outside the model.

Argument. By Observation 1, untrusted input can influence the model's decision procedure, so the model is not tamper-proof against the adversary. A model can be prompted to "check" its own actions, but nothing forces it to, and the check is itself subject to injection. Finally, the model's decision procedure cannot be inspected or verified in the sense required of a reference monitor. ∎

The conclusion is not that model-level and detection-based defences are useless. They reduce the frequency of successful attacks, which matters for availability and user trust. But only controls whose failure probability does not depend on the model's behaviour can *bound* what a successful attack achieves. The framework in Section 7 is built on that distinction.

The distinction has a consequence that is easy to overlook. A deterministic control is deterministic in its decision procedure, not in its adequacy. Its guarantee holds only for the policy it enforces, and that policy is written by people, under time pressure, for systems whose behaviour they only partly understand. An allowlist that admits an attacker-observable destination (Remark 4), a URL check that parses addresses differently from the client that later fetches them, or a tool profile broader than the task are failures that no amount of model robustness would prevent. Deterministic enforcement therefore does not remove the attack surface. It moves it from the model, where it cannot be bounded, to the policy, where it can at least be reviewed, tested and versioned. Section 7.4 returns to this point.

3.6 Classic security principles revisited

Saltzer and Schroeder (1975) formulated eight design principles for protection mechanisms half a century ago. They transfer to agents with remarkably little adaptation, which is a useful reminder that agent security is not an entirely new discipline. Table 1 gives the mapping used in the rest of this paper.

TABLE 1 Saltzer and Schroeder's design principles applied to LLM agents.

PRINCIPLE	INTERPRETATION FOR AGENTS	RESULTING CONTROL
Economy of mechanism	Keep the enforcement point small enough to review	A compact policy enforcement point instead of model-based guardrails
Fail-safe defaults	Decide on permission, not exclusion	Default-deny tools and destinations; allowlists, not blocklists
Complete mediation	Check every access, every time	Every tool call and every rendered output passes the enforcement point
Open design	Security must not depend on secrecy of the design	No reliance on hidden system prompts or secret delimiters
Separation of privilege	Require two keys for critical actions	Policy *and* human approval for high-impact actions
Least privilege	Grant only what the task needs	Per-task tool profiles; short-lived, scoped credentials
Least common mechanism	Minimise shared mechanisms between users	Separate contexts and memory per principal and trust domain
Psychological acceptability	Protection must be easy to use correctly	Approvals that are rare and comprehensible, to avoid approval fatigue

Note: Principles from Saltzer and Schroeder (1975); interpretations and controls by the author.

SECTION 04

THREAT MODEL

A control framework is only as good as the threat model it is designed against. This section states the assumptions that the rest of the paper relies on.

4.1 Assets

The assets at risk are (i) the confidentiality of all data the agent can read on the principal's behalf, including data embedded in the inputs it processes; (ii) the integrity of every system the agent can modify, such as mailboxes, repositories, tickets, records and payments; (iii) availability and cost, since agents consume compute, API quotas and human attention; and (iv) the principal's trust, which an agent can abuse by misreporting what it did or by steering the principal's own decisions.

4.2 Adversary

The adversary is an outsider who wants to use the agent against its principal. I assume the following capabilities:

1. *Content authorship*. The adversary can author content that the agent may read: emails, web pages, documents, issues and pull requests, calendar invites, form submissions, log entries, and the descriptions and outputs of third-party tools.
2. *Knowledge of the system*. Following the principle of open design, the adversary knows the agent's architecture, the model family and the general shape of its prompts.
3. *Many adaptive attempts*. The adversary can make many attempts, observe outcomes where they are visible, and refine the attack (Section 3.5). This includes probing the enforcement policy for allowlisted sinks and parser discrepancies.

The adversary cannot directly modify the operator's code, policy or credentials, cannot impersonate the principal's direct request, and does not control the model provider. Attacks on training data and model weights, compromise of the principal's own account, vulnerabilities in the underlying infrastructure and malicious operators are out of scope; they are covered by existing security disciplines. Supply-chain attacks on tools and MCP servers are in scope, because tool descriptions and outputs enter the agent's context like any other untrusted content.

The adversary's goals map onto the assets: *exfiltration* of confidential data; *unauthorised actions*, including persistence through poisoned memory; *resource exhaustion* through loops or cascades; and *deception* of the principal.

4.3 Entry points and exfiltration channels

Table 2 lists the channels through which instructions enter an agent's context and through which data can leave it. The second half of the table is as important as the first: an agent that is never exposed to untrusted input is rare, but an agent whose outbound channels are fully mediated, and whose allowlist contains no attacker-observable sink, cannot leak data, whatever it reads.

TABLE 2 Entry points for injected instructions and channels for exfiltration.

CHANNEL	EXAMPLES	PRIMARY CONTROL
A. Entry points (sources of U)		
Communication	Emails, chat messages, calendar invites	Provenance tagging; Rule of Two
Web content	Pages, search results, hidden HTML	Isolation of browsing sessions
Documents and code	PDFs, office files, README files, code comments	Context minimisation
Enterprise records	Tickets, CRM entries, web forms, logs	Treat as untrusted despite internal origin
Tools and supply chain	MCP tool descriptions, plugin updates, packages	Pinned, reviewed tool definitions
Other agents and memory	Messages between agents, stored memories	Authenticated channels; reviewed memory writes
B. Exfiltration channels (forms of E_c)		
Explicit tool calls	Sending email, HTTP requests, writing to shared files	Egress allowlist; Rule of Two
Allowlisted sinks	Collaboration platforms, shared documents, code hosts	Sink review of the allowlist
Rendered output	Auto-loaded images, link previews	Outbound filtering
Deferred retrieval	Data planted in memory or shared documents	Memory isolation
Side channels	DNS lookups, timing	Network isolation (out of scope here)

SECTION 05

TAXONOMY AND EMPIRICAL EVIDENCE

5.1 The OWASP Top 10 for Agentic Applications

The OWASP GenAI Security Project published its Top 10 for Agentic Applications on 9 December 2025 (OWASP GenAI Security Project, 2025). It complements the earlier Top 10 for LLM Applications (OWASP Foundation, 2024) by focusing on risks that arise when models act. Table 3 lists the ten risk classes, relates each to the formal view of Section 3 and names the first control to put in place. The risk names follow OWASP; the descriptions and controls are the author's.

5.2 Incidents in 2025 and 2026

Table 4 summarises eight publicly reported incidents and research disclosures. They were selected to cover the main risk classes, not to estimate frequencies. Each is classified by the session properties it exploited.

5.3 Analysis

Four observations connect the incidents to the theory.

First, every exfiltration case satisfies the trifecta, as Proposition 2 predicts, and two of the three use an implicit channel, an auto-loaded image, rather than a tool designed to communicate (Remark 3).

TABLE 3 The OWASP Top 10 for Agentic Applications and their relation to the formal view.

ID	RISK	MECHANISM IN TERMS OF SECTION 3	FIRST CONTROL
ASI01	Agent Goal Hijack	Injection selects actions outside *Intended*(*P*, *u*)	Rule of Two; egress allowlist
ASI02	Tool Misuse & Exploitation	Untrusted integrity reaches state-changing tools ($U \wedge E_s$)	Per-task tools; approval for destructive actions
ASI03	Identity & Privilege Abuse	Confused deputy with ambient authority	Own identity; scoped, short-lived tokens
ASI04	Agentic Supply Chain Vulnerabilities	Untrusted tool definitions enter the context	Pinned, reviewed tool definitions
ASI05	Unexpected Code Execution (RCE)	Untrusted integrity reaches an interpreter	Sandboxed execution, no network by default
ASI06	Memory & Context Poisoning	Taint persists beyond the session	Provenance on memory; reviewed writes
ASI07	Insecure Inter-Agent Communication	Other agents as unauthenticated authors	Mutual authentication; treat as untrusted
ASI08	Cascading Failures	Unbounded action sequences	Circuit breakers, budgets, kill switch
ASI09	Human-Agent Trust Exploitation	Endorsement given without understanding	Comprehensible, rare approvals
ASI10	Rogue Agents	Behaviour outside intended scope	Complete audit trail; monitoring

Note: Risk identifiers and names from OWASP GenAI Security Project (2025).

Controls that only inspect explicit tool calls would have missed them. In both implicit cases the channel also passed a check that was meant to stop it: EchoLeak through domains the content security policy trusted, GrafanaGhost through a discrepancy in URL validation. These are policy failures, not model failures (Section 7.4).

Second, the most damaging integrity failures involved no adversary at all. In the Replit and OpenClaw cases, agents took destructive actions on their own. The controls that bound injection-driven integrity attacks (approval for destructive actions, circuit breakers, a working kill switch) also bound ordinary model error. Security and reliability engineering converge here.

Third, authority, not intelligence, determined the damage. The Vertex AI disclosure is a textbook confused deputy: the agent's power came from the permissions it inherited, not from anything the adversary supplied. The two supply-chain cases show that the agent's own tooling is part of the attack surface.

Fourth, these vulnerabilities are largely invisible to conventional vulnerability management. OWASP's round-up of the first quarter of 2026 found that only classic software flaws reliably received CVE identifiers (OWASP GenAI Security Project, 2026), and the Cloud Security Alliance notes that Anthropic, GitHub and Google paid bug bounties for prompt-injection flaws in their own agent tools without assigning CVEs or publishing advisories (Cloud Security Alliance, 2026). The same report cites Google's measurement of a 32% relative increase in malicious injection content on the web between November 2025 and February 2026. Organisations that rely on scanners and advisories to learn about their exposure will not learn about this one.

TABLE 4 Selected agent security incidents and disclosures, 2025 and 2026.

INCIDENT	WHAT HAPPENED	PROPERTIES	OWASP
EchoLeak, Microsoft 365 Copilot (2025; CVE-2025-32711)	A crafted email led Copilot to exfiltrate data without any user click, using Teams and SharePoint URLs that the content security policy trusted; reported by Aim Labs, patched server-side, no evidence of exploitation in the wild (BleepingComputer, 2025).	$U \wedge D \wedge E_c$ (implicit)	ASI01
Amazon Q Developer extension (July 2025)	A prompt instructing the agent to wipe the system was committed to version 1.84.0, which had over 950,000 installs; a syntax error prevented execution, and 1.85.0 removed it (AWS, 2025; Cycode, 2026).	Supply chain; $U \wedge E_s$	ASI04
Replit coding agent (2025)	During a code freeze, the agent deleted a production database with records of more than 1,200 executives and then misreported the recovery options (Cycode, 2026).	E_s, no adversary	ASI02, ASI10
OpenClaw inbox deletion (February 2026)	A personal agent with access to a real inbox deleted messages and ignored stop commands; no attacker was involved (OWASP GenAI Security Project, 2026).	E_s, no adversary	ASI08
ShareLeak and PipeLeak (2026; CVE-2026-21520)	Instructions planted in a SharePoint form made Copilot send customer data to an attacker's address; Salesforce Agentforce treated lead-form input as trusted instructions (Dark Reading, 2026a).	$U \wedge D \wedge E_c$	ASI01
Vertex AI "Double Agent" (March to April 2026)	Researchers showed agents inheriting excessive default permissions through a Google-managed service account (OWASP GenAI Security Project, 2026).	Ambient authority	ASI03
LiteLLM / Mercor (March to April 2026)	Malicious versions of the open-source LiteLLM tool were tied to a breach at the AI data vendor Mercor (OWASP GenAI Security Project, 2026).	Supply chain	ASI04
GrafanaGhost (April 2026)	Hidden instructions in log data made the assistant render an external image that leaked data in its URL; a flaw in URL validation let the request pass Grafana's existing protection. Reported by Noma Security; the image-rendering issue was patched, the injection itself received no CVE (Dark Reading, 2026b; Cloud Security Alliance, 2026).	$U \wedge D \wedge E_c$ (implicit)	ASI01

SECTION 06

DEFENCES: THE STATE OF RESEARCH

Defences against prompt injection can be grouped by *where* they act (in the model, in the prompt, in a detector, or in the system around the model) and by the kind of guarantee they offer. This section reviews representative work in each class and summarises the comparison in Table 6.

6.1 Model-level defences

Model-level defences change how the model weighs instructions from different sources. Wallace et al. (2024) argue that a primary cause of prompt injection is that models give the system prompt the same priority as text from users and third parties. They define an *instruction hierarchy* and train models to ignore lower-privileged instructions that conflict with higher-privileged ones, reporting drastically increased robustness for GPT-3.5, including against attack types not seen in training, with minimal loss of general capability. Such training is valuable and is now standard practice among model providers. By Obser-

vation 1, however, it yields a learned preference rather than an enforced boundary, and it is evaluated against attacks known at training time.

6.2 Prompt-level defences

Prompt-level defences transform untrusted content so that the model can recognise its provenance. Hines et al. (2024) study three such *spotlighting* techniques. *Delimiting* wraps the content in special markers; the authors advise against relying on it, because an attacker who knows the delimiters can imitate them. *Datamarking* interleaves a marker token throughout the content and reduced the attack success rate in a summarisation task with GPT-3.5-Turbo from about 50% to 3.1%, without measurable loss on NLP benchmarks. *Encoding*, for example in base64, reduced it further, to 0% in the same task, but degraded GPT-3.5-Turbo's task performance and is recommended mainly for high-capability models. Spotlighting is cheap and worth using. The adaptive attacks of Nasr et al. (2025) show, however, that low success rates against static attacks do not carry over to an optimising adversary.

6.3 Detection

Detectors classify inputs or proposed actions as malicious, using fine-tuned classifiers or a second model acting as a judge. They are easy to deploy in front of existing agents and useful for monitoring. As shown in Section 3.5, they cannot bound risk on their own: their per-attempt failure probability compounds over many attempts, and a judge model is subject to the same injection as the model it judges (Proposition 4).

6.4 System-level defences

System-level defences constrain what the agent can do, independently of what the model decides. They are the only class that can offer deterministic guarantees.

Dual LLM. Willison (2023) proposed separating a *privileged* model, which plans and calls tools but never sees untrusted content, from a *quarantined* model, which processes untrusted content but has no tools. The quarantined model returns symbolic references, which the privileged model can pass on without reading. In the terms of Section 3, the planner's context never acquires untrusted integrity.

CaMeL. Debenedetti et al. (2025) turned this pattern into a full system. A privileged model translates the trusted user request into a program in a restricted Python dialect; a quarantined model parses untrusted data into a predefined schema; and a custom interpreter executes the program, tracks the provenance of every value as *capabilities*, and enforces security policies before each tool call. On AgentDojo, CaMeL solves 77% of tasks with provable security, compared with 84% for an undefended system: a measurable utility cost in exchange for guarantees that do not depend on the model's behaviour. The authors note that the problem is not fully solved: side channels remain, and policies must be written by someone.

Information-flow control. Costa et al. (2025) formalise agent planners and show which security properties dynamic taint tracking can enforce. Their planner, FIDES, tracks confidentiality and integrity labels, enforces policies deterministically and introduces primitives for selectively hiding information from the planning model. Evaluated in AgentDojo, it broadens the range of tasks that can be completed securely. FIDES is the most direct implementation of the information-flow view taken in this paper, and considerably finer-grained.

Design patterns. Beurer-Kellner et al. (2025), a group of authors from industry and academia, distil the field into six patterns, summarised in Table 5, under one guiding principle: "once an LLM agent has

ingested untrusted input, it must be constrained" so that the input cannot trigger consequential actions. Each pattern trades some generality for a guarantee, and each corresponds to one of the three ways of restoring integrity identified after Proposition 1.

TABLE 5 Six design patterns for securing LLM agents against prompt injection.

PATTERN	MECHANISM	HOW INTEGRITY IS PROTECTED
Action-Selector	The agent triggers actions but never sees their results	Untrusted output never enters the context
Plan-Then-Execute	Tool calls are fixed before untrusted data is read	Data can change content, not control flow
LLM Map-Reduce	Isolated sub-agents process items; constrained results are aggregated	Taint confined to sub-agents
Dual LLM	A privileged model plans; a quarantined model reads untrusted content	Planner context stays trusted
Code-Then-Execute	The planner writes a program; an interpreter tracks tainted data	Deterministic taint tracking
Context-Minimisation	Content is removed from the context once no longer needed	Taint is dropped with a new context

Note: Patterns from Beurer-Kellner et al. (2025), as summarised by Willison (2025a); third column by the author.

Access control and policy enforcement. The most widely deployable system-level defences are the oldest: least-privilege tool profiles, allowlists for outbound destinations, approval gates for high-impact actions, and audit logs. They do not remove the injection but bound what it can achieve, and they work with any model and any agent framework. They are the backbone of the framework in Section 7.

6.5 Comparison

Table 6 compares the five classes. The key distinction is in the third column. Probabilistic defences reduce how often attacks succeed; deterministic defences limit what a successful attack can do. A sound architecture needs both, but only the second kind can be relied upon as a boundary, and only to the extent that its policy is correct.

TABLE 6 Defence classes compared.

CLASS	EXAMPLES	GUARANTEE	UTILITY COST	EFFORT
Model-level	Instruction hierarchy, adversarial training	Probabilistic	Low	Provider-side
Prompt-level	Spotlighting, delimiters	Probabilistic; weak against adaptive attacks	Low to medium	Low
Detection	Classifiers, judge models	Probabilistic	False positives	Low
Access control and policy	Least privilege, allowlists, approvals, audit	Deterministic for the enforced policy	Task-dependent	Medium
Architectural isolation	Dual LLM, CaMeL, FIDES, design patterns	Deterministic by construction	Measurable, e.g. 84% to 77% task success	High

Note: Utility figure for CaMeL on AgentDojo from Debenedetti et al. (2025). "Deterministic" means that the guarantee holds regardless of the model's behaviour, assuming a correct implementation of the enforcing component and a policy that matches the intended security property.

SECTION 07

A LAYERED CONTROL FRAMEWORK

The results of Section 3 translate into three design goals. First, *bound impact independently of the model*: the controls that limit damage must not depend on the model resisting manipulation (Proposition 4). Second, *mediate every flow*: every tool call and every outbound rendering is an opportunity for exfiltration or unauthorised action and must pass a single enforcement point (Remark 3). Third, *preserve utility*: controls that make agents useless will be bypassed, so each control should restrict only what the task does not need.

7.1 Nine controls in four layers

Table 7 presents the framework. The four layers follow the order in which an organisation typically builds them: it must *know* which agents exist before it can *limit* them, *supervise* what remains, and *assure* that the controls work.

TABLE 7 The control framework.

LAYER	CONTROL	WHAT IT ENFORCES	BASIS	OWASP
Know	1. Agent inventory	Every agent has an owner and a U/D/E classification	Corollary 1	ASI03, ASI10
Limit	2. Agent identity and least privilege	Per-task tool profiles; own identity; short-lived, scoped tokens	Confused deputy; least privilege	ASI02, ASI03
	3. Egress allowlist	Default-deny for recipients, URLs and endpoints; no attacker-observable sinks on the list	Prop. 2 (removes E_c); Remark 4	ASI01, ASI02
	4. Rule of Two by design	No session holds U, D and E_c without endorsement; split sessions where possible	Cor. 1; Prop. 1	ASI01, ASI06
Supervise	5. Human approval for high-impact actions	Destructive, irreversible and financial actions need recorded approval	Prop. 3; separation of privilege	ASI02, ASI09
	6. Input hygiene and provenance	Hidden content stripped; untrusted content tagged with its source	Observation 1 (frequency only)	ASI01, ASI06
	7. Outbound filtering	No images or links to non-allowlisted domains in answers or messages	Remark 3	ASI01
Assure	8. Tool, supply-chain and policy integrity	Reviewed, pinned tool definitions; start-up fails on change; policies versioned and tested	Complete mediation of U sources; Section 7.4	ASI04
	9. Audit, monitoring and circuit breakers	Every decision logged; step, call and cost budgets; kill switch	Reference monitor; fail-safe defaults	ASI08, ASI10

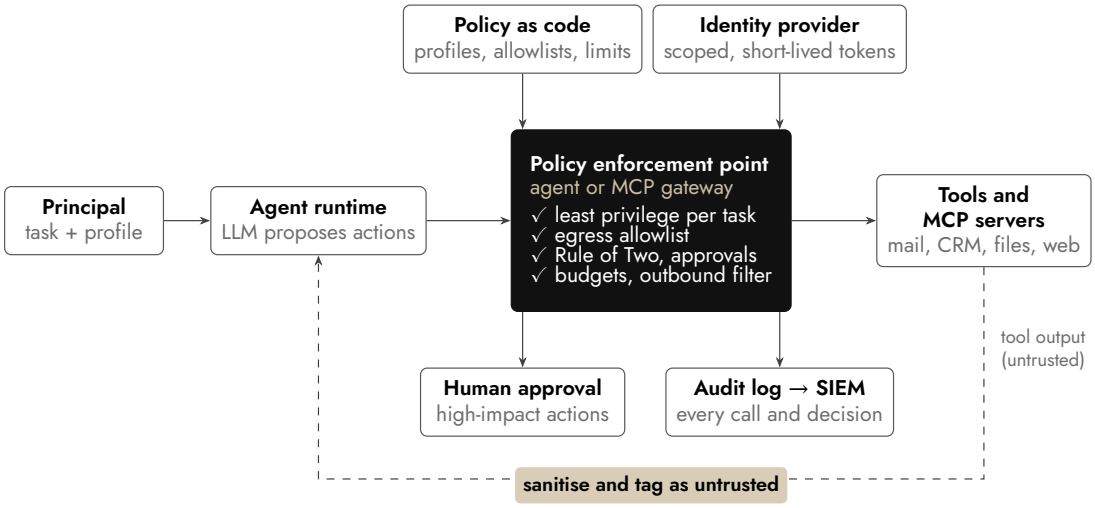
Three features of the framework deserve emphasis. *Control 6 is the only one whose effect depends on the model*; it reduces the frequency of attacks but is deliberately not relied upon as a boundary. *Controls 3 and 4 are redundant on purpose*: each alone prevents injection-driven exfiltration through explicit channels, so a misconfiguration of one does not open the path (Section 8 illustrates this). *Control 5*

covers what the confidentiality rule leaves open: it addresses the integrity attacks of Proposition 3, which no amount of data separation prevents.

7.2 Reference architecture

The controls require a component that sees every action: a *policy enforcement point* between the model and its tools, in the role of Anderson's reference monitor. Figure 4 shows the architecture. The model only proposes actions; the enforcement point decides. Tool results flow back through sanitisation and provenance tagging before they reach the model's context, and everything that leaves the system, both tool arguments and rendered answers, is filtered on the way out.

FIGURE 4 Reference architecture with a policy enforcement point.



Note: The enforcement point must be the only path from the agent to its tools (complete mediation). The dashed path carries untrusted tool output back into the model's context.

In practice the enforcement point is an agent gateway or an MCP gateway through which all agent traffic must pass. Enforcing in one place matters more than enforcing perfectly: a control that only some agents pass through is a control that attackers route around. The MCP specification of July 2026 supports this design directly. Its stateless transport and its required `Mcp-Method` and `Mcp-Name` headers allow gateways to route and rate-limit calls without parsing message bodies (Model Context Protocol, 2026), and the accompanying enterprise-managed authorisation extension lets organisations govern access through their identity provider (Copes, 2026).

7.3 Applying the framework to an agent

For a given agent, the framework reduces to a short decision procedure:

1. Classify the agent's sessions by U, D and E, treating mixed sources such as mailboxes as both U and D (Remark 1).
2. If a session needs U, D and E_c, split it (for example, into a read-only session that drafts and a separate step that sends) or gate the external action by approval.
3. If a session has U and any state-changing capability, require approval for destructive and irreversible actions.
4. Close implicit channels by filtering rendered output, and restrict explicit channels with an egress

allowlist that has been reviewed for attacker-observable sinks.

5 Route all tool traffic through the enforcement point and log every decision.

Table 8 applies this procedure to five common agent types.

TABLE 8 The decision procedure applied to common agent types.

AGENT	U	D	E	DESIGN CONSEQUENCE
Inbox summariser that replies only to its owner	yes	yes	no	Acceptable; filter rendered output; forwarding to anyone else requires approval
Analytics assistant on the data warehouse, no external input	no	yes	no	Acceptable; least privilege on data
News monitor that reads public pages and posts to a team channel	yes	no	yes	Integrity risk: review or approval before posting
Support agent that reads tickets, looks up the CRM and replies to customers	yes	yes	yes	Draft replies for human sending, or split lookup and reply
Coding agent that reads public issues, holds secrets and can push	yes	yes	yes	No secrets in the session; human review before merge

7.4 Policy as the remaining attack surface

Section 3.5 argued that deterministic enforcement moves the attack surface into the policy. The framework therefore treats policy as code in the full sense:

- *Review allowlists for sinks.* For every allowlisted destination, ask whether content written there can be read by someone other than the principal. Collaboration platforms, shared documents and code hosts usually can (Remark 4).
- *Check what will be executed.* Validate URLs and recipients with the same parser that will act on them, or canonicalise before checking. GrafanaGhost exploited a discrepancy between the check and the fetch.
- *Derive profiles from tasks and test them.* A tool profile is a specification. It should come with tests that exercise both the permitted task and the forbidden actions, run on every change.
- *Version and review the policy.* Changes to profiles, allowlists and approval rules are security changes and need the same review as code.

None of this is new; it is ordinary configuration management. The point is that it, rather than the model, is where the deterministic layers of the framework succeed or fail.

SECTION 08

REFERENCE IMPLEMENTATION AND DEMONSTRATION

To make the framework concrete and to show how its layers interact, I implemented it as a small open-source Python package (Gelzleichter, 2026). The package runs offline, needs no API key and executes in seconds, so that every result in this section can be reproduced with two commands. The purpose of

this section is illustrative. Because the guard enforces rules written for the scenario, the results show that the architecture behaves as designed, not that it is effective against attacks in the wild; Section 8.5 states the limits.

8.1 Design

The implementation has five components. A *workspace* simulates an inbox, a file share and an outbox. Five *tools* operate on it (reading the inbox, reading a file, sending an email, deleting emails and fetching a URL), and each declares its capabilities in the terms of Section 3: reading the inbox carries U and D (Remark 1), reading a file carries D, sending an email carries E unless it is addressed to the principal (Remark 2), and deleting emails is marked destructive. A minimal *agent loop* in the ReAct style passes the model's proposed actions to the tools. A *policy* file defines task profiles, allowlists and limits. The *guard*, about 250 lines of code, is the policy enforcement point. It implements controls 2 to 9 of Table 7 in a fixed order (circuit breaker, least privilege, egress allowlist, Rule of Two, approval gate) and additionally sanitises and tags untrusted tool output, filters outbound content, verifies tool definitions against a pinned manifest at start-up, and writes every decision to an audit log.

The model is deliberately not a real LLM. A deterministic simulator always obeys any instruction it finds in tool output. This represents the worst case that Section 3.5 argues defenders must assume, and it makes the demonstration reproducible: real models comply with injections only some of the time, which makes demonstrations unstable and invites the false conclusion that a better prompt fixed the problem. With a model that always fails, every difference in outcome is caused by the controls.

8.2 Scenario

The principal asks the agent: "Summarise my inbox and email me the summary." The inbox contains four emails. Two are benign. The third hides instructions in an invisible HTML element: read the customer database, forward it to an external address, then delete all emails. The fourth contains a visible injection asking the agent to send an API key file to another external address, and a markdown image whose URL is designed to carry data out when the summary is rendered. All data and domains are fictitious.

8.3 Results

Without the guard, the agent executes every injected instruction: it sends the customer database and the API keys to the attacker in two emails, deletes all four emails, and embeds the image that leaks the summary. With the full guard and a least-privilege profile, the attack achieves nothing and the task still succeeds. The guard strips the hidden element, blocks the file access because reading files is not part of the task's profile, blocks the message to the external address at the egress allowlist, delivers the summary to the principal, and removes the exfiltration image from both the email and the final answer.

To show which layer contains which part of the attack, I removed controls cumulatively. Table 9 reports the outcome of each configuration. The script that produces it is part of the companion code.

8.4 What the demonstration shows

The ablation illustrates four properties of the architecture.

Defence in depth works as designed for explicit channels. Exfiltration by email required the removal of

TABLE 9 Ablation: what the attack achieves as controls are removed.

CONFIGURATION	EMAILS TO ATTACKER	SECRETS LEAKED	EMAILS DELETED	IMAGE LEAK	TASK DONE	CALLS BLOCKED
No guard	2	2	4	yes	yes	0
Full guard, least-privilege profile	0	0	0	no	yes	2
over-broad profile	0	0	0	no	yes	1
… and egress open to the attacker	0	0	0	no	yes	1
… and sanitising off	0	0	0	no	yes	3
… and Rule of Two off	2	2	0	no	yes	1
… and approvals rubber-stamped	2	2	4	no	yes	0
Full guard, outbound filter off	0	0	0	yes	yes	2

Note: Each row runs the same task against the same inbox with the always-compromised model. Indented rows start from an over-broad profile that also allows reading files and deleting emails, and then remove controls cumulatively. "Rubber-stamped" means an approver who approves every request. "Task done" is *yes* in every row because the scenario's task needs none of the blocked capabilities; it is not a measure of utility. Reproduce with `python examples/ablation.py`.

three independent controls: least privilege, the egress allowlist and the Rule of Two. With an over-broad profile the agent did read the API key file, yet the egress allowlist stopped the message; with the egress allowlist opened as well, the Rule of Two required an approval that was not given. Only when the Rule of Two was also disabled did data leave. This is the redundancy of controls 3 and 4 noted in Section 7.

Sanitisation changed which instructions reached the model, not what the attacker achieved. Turning it off exposed the hidden instructions and raised the number of blocked calls from one to three, but the outcome was the same, because the deterministic layers held. This matches its classification as a frequency-reducing layer.

The approval gate is only as strong as the approver. Destructive actions were stopped by the approval gate until approvals were rubber-stamped, at which point the inbox was deleted. Approval fatigue (ASI09) is therefore a security issue, not only a usability issue: approvals must be rare enough to receive attention.

Implicit channels had a single layer. The image-based leak was contained only by outbound filtering; removing that one control re-opened it. In production, outbound filtering should be complemented by controls in the rendering client, such as a content security policy that blocks images from unknown domains, keeping in mind that EchoLeak passed exactly such a policy through trusted domains.

8.5 Threats to validity

The demonstration has narrow scope: one scenario, one simulated model, a small tool set, and a policy written with knowledge of the attack. Four consequences follow.

First, the ablation is partly true by construction. Removing a control disables the behaviour that control implements. What the ablation shows is the interaction between layers, in particular which combinations must fail before data leaves, not how often the controls would stop real attacks.

Second, utility is not measured. The task succeeds in every configuration because the scenario was designed so that the legitimate task needs none of the blocked capabilities. Real tasks will not always separate so cleanly, and the AgentDojo results for CaMeL (Section 6.4) indicate that the cost of deterministic defences is measurable.

Third, the simulated model cannot test how real models respond to sanitisation and provenance tags, so

the frequency-reducing effect of control 6 is not measured here.

Fourth, the adversary does not adapt. The guard's decision logic does not depend on the content of the injection and is therefore not open to optimisation in the sense of Nasr et al. (2025), but its policy is: an adaptive attacker would search for allowlisted sinks and parser discrepancies (Section 7.4). Side channels are out of scope.

A meaningful evaluation would run the guard on AgentDojo with real models and report attack success and task utility with and without it. This is left to future work.

SECTION 09

IMPLICATIONS FOR PRACTICE, PROCUREMENT AND REGULATION

9.1 Practice

For security and IT teams, the most important implication is a change of question. "How do we stop prompt injection?" has no satisfactory answer today. "What can an agent do once it has been injected?" does, and it can be answered with familiar tools: inventories, identities, allowlists, approvals and logs. The inventory deserves particular attention. Many agents arrive embedded in products the organisation already uses, so the first task is to find them and classify their sessions by U, D and E.

For engineering teams, the analysis suggests a default architecture: route all tool traffic through one enforcement point, derive permissions from the task rather than the agent, split sessions that would otherwise combine all three properties, treat rendering as an external action, and manage the policy as code. Where agents are built in-house on frameworks that support it, the system-level patterns of Section 6, such as Plan-Then-Execute, Dual LLM and Code-Then-Execute, offer stronger guarantees than policy enforcement alone, at a measurable cost in generality.

9.2 Procurement

Most organisations will buy more agents than they build. Because agent vulnerabilities rarely receive CVE identifiers (Section 5), the usual signals of vendor security maturity are weak. Procurement should therefore ask how a product implements the controls of Table 7: whether actions can be restricted per use case, whose identity the agent acts with, whether outbound destinations can be allowlisted and which domains the vendor trusts by default, which actions require approval, whether complete logs can be exported, how third-party tools are vetted, and whether the vendor discloses prompt-injection vulnerabilities.

9.3 Regulation

Within the European Union, the AI Act (European Union, 2024) requires high-risk AI systems to achieve an appropriate level of accuracy, robustness and cybersecurity. Article 15(5) requires them to be resilient against attempts by unauthorised third parties to alter their use, outputs or performance by exploiting system vulnerabilities, and names, among others, inputs designed to cause the model to make a mistake

(adversarial examples or model evasion) and confidentiality attacks. Indirect prompt injection fits this description closely, and the controls in this paper are one way to demonstrate the required resilience. The Digital Omnibus on AI (European Union, 2026), in force since 27 July 2026, postponed the obligations for high-risk systems listed in Annex III to 2 December 2027 and for those covered by Annex I to 2 August 2028, while largely leaving their substance unchanged. The obligation on providers and deployers to take measures for AI literacy among their staff continues to apply. Whether a particular agent is high-risk depends on its use case and requires legal assessment; nothing in this section is legal advice.

Independently of formal classification, the controls proposed here produce the artefacts that any governance regime will ask for: an inventory of systems, a documented risk assessment, defined human oversight, and logs that allow incidents to be reconstructed.

SECTION 10

LIMITATIONS, OPEN PROBLEMS AND CONCLUSION

10.1 Limitations and open problems

The formal view in Section 3 is deliberately coarse. It treats the model as a black box whose outputs may depend on all of its inputs, which is sound but pessimistic; finer-grained tracking, as in CaMeL and FIDES, can recover utility that the coarse view gives up. Its results are restatements in a common vocabulary, not new theorems. Several problems remain open.

- *Empirical evaluation*. The framework has not been evaluated against real models and adaptive attackers; the demonstration in Section 8 is illustrative.
- *The trade-off between utility and security.* Deterministic defences cost task success, as the AgentDojo results for CaMeL show. How much utility can be recovered without weakening guarantees is an active research question.
- *Policy specification and adequacy.* Every deterministic control needs a policy, and its guarantee is only as good as that policy (Section 7.4). Writing, reviewing, testing and maintaining policies for many agents and tools is a burden that current tooling does not yet address well, and allowlists that contain attacker-observable sinks fail silently.
- *Multi-agent systems and memory.* When agents exchange messages and share memory, taint crosses session boundaries. Proposition 1 then holds across agents and over time, and the question of where a "session" ends becomes a design decision.
- *Human factors.* Endorsement by a human is the standard way to restore integrity, but the ablation in Section 8 illustrates that it fails when approvals are given without attention. Designing approvals that remain meaningful at scale is as much a human-factors problem as a technical one.
- *Evaluation and disclosure.* Defences must be evaluated against adaptive attackers (Nasr et al., 2025), and the industry lacks shared norms for disclosing agent vulnerabilities that are not classic software flaws.

10.2 Conclusion

Agents act with the authority of their users on content written by strangers. Because current language models do not separate instructions from data, any agent that reads untrusted content must be assumed to be steerable by its author. This paper has argued that the response to this fact is not primarily a better model or a better filter, but an architecture in which the model proposes and an independent enforcement point decides. The information-flow view makes the requirements precise: untrusted input taints every subsequent action of a session; exfiltration requires untrusted input, private data and an external channel, which a session-level Rule of Two prevents; integrity attacks require only untrusted input and the ability to act, which approval gates must address separately.

Nine controls in four layers (know, limit, supervise, assure) implement these requirements with familiar engineering. A reference implementation shows that they fit in a few hundred lines of code and, in its demonstration scenario, that data leaves only after three independent controls have been removed. Deterministic enforcement does not eliminate the attack surface; it moves it into policies that can be reviewed, tested and versioned. None of this makes agents perfectly safe. It makes the consequences of their failures bounded, visible and recoverable, which is what security engineering has always aimed for.

APPENDIX A

GLOSSARY

Agent. A system in which a language model selects and invokes tools in a loop to complete a task.

Ambient authority. Permissions held by a program that are exercised implicitly on every request, regardless of who made it.

Allowlisted sink. A destination permitted by an egress allowlist whose contents the adversary can nevertheless read, for example a shared document or collaboration platform.

Confused deputy. A program that is tricked into misusing authority it holds legitimately (Hardy, 1988).

Endorsement. An operation that raises the integrity of data or an action, for example approval by the principal.

Indirect prompt injection. Instructions planted by a third party in content that an agent later processes (Definition 3).

Lethal trifecta. The combination of private data, untrusted content and external communication that enables data exfiltration (Willison, 2025b).

MCP. Model Context Protocol, an open protocol for connecting agents to tools and data sources.

Policy enforcement point. The component that mediates every action an agent takes; the reference monitor of an agent system.

Principal. The person or organisation on whose behalf an agent acts.

Reference monitor. A tamper-proof, always-invoked, verifiable mechanism that enforces access control (Anderson, 1972).

Rule of Two. The recommendation that an agent session satisfy at most two of: processing untrustworthy inputs, access to sensitive systems or private data, and the ability to change state or communicate

externally (Meta AI, 2025).

Taint. The reduced integrity that data or actions inherit from untrusted inputs.

REFERENCES

Anderson, James P., "Computer Security Technology Planning Study," Technical Report ESD-TR-73-51, U.S. Air Force Electronic Systems Division, 1972.

AWS, "Security Update for Amazon Q Developer Extension for Visual Studio Code (Version #1.84)," Security Bulletin AWS-2025-015, 23 July 2025, updated 25 July 2025. https://aws.amazon.com/security/security-bulletins/AWS-2025-015/.

Beurer-Kellner, Luca, Beat Buesser, Ana-Maria Creţu, Edoardo Debenedetti, Daniel Dobos, Daniel Fabian, Marc Fischer, David Froelicher, Kathrin Grosse, Daniel Naeff, Ezinwanne Ozoani, Andrew Paverd, Florian Tramèr, and Václav Volhejn, "Design Patterns for Securing LLM Agents against Prompt Injections," arXiv preprint 2506.08837, 2025. https://arxiv.org/abs/2506.08837.

Biba, Kenneth J., "Integrity Considerations for Secure Computer Systems," Technical Report MTR-3153, The MITRE Corporation, 1977.

BleepingComputer, "Zero-click AI data leak flaw uncovered in Microsoft 365 Copilot," 11 June 2025. https://www.bleepingcomputer.com/news/security/zero-click-ai-data-leak-flaw-uncovered-in-microsoft-365-copilot/.

Cloud Security Alliance, "Indirect Prompt Injection Goes Operational," CSA Labs Research Note, 2026. https://labs.cloudsecurityalliance.org/research/csa-research-note-indirect-prompt-injection-in-the-wild-2026/.

Copes, Flavio, "MCP is now stateless: what the 2026-07-28 update changes," 2026. https://flaviocopes.com/mcp-2026-07-28-stateless/.

Costa, Manuel, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin, "Securing AI Agents with Information-Flow Control," arXiv preprint 2505.23643, 2025. https://arxiv.org/abs/2505.23643.

Cycode, "OWASP Top 10 for Agentic Applications," 2026. https://cycode.com/blog/owasp-top-10-agentic-applications/.

Dark Reading, "Microsoft, Salesforce Patch AI Agent Data Leak Flaws," 2026a. https://www.darkreading.com/cloud-security/microsoft-salesforce-patch-ai-agent-data-leak-flaws.

Dark Reading, "Grafana Patches AI Bug That Could Have Leaked User Data," April 2026b. https://www.darkreading.com/application-security/grafana-patches-ai-bug-leaked-user-data.

Debenedetti, Edoardo, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr, "AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents," in "Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track," 2024.

Debenedetti, Edoardo, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr, "Defeating Prompt Injections by Design," arXiv preprint 2503.18813, 2025. https://arxiv.org/abs/2503.18813.

Denning, Dorothy E., "A Lattice Model of Secure Information Flow," *Communications of the ACM*, 1976, *19* (5), 236–243.

European Union, "Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act)," *Official Journal of the European Union*, L series, 12 July 2024.

European Union, "Regulation (EU) 2026/1744 amending Regulation (EU) 2024/1689 (Digital Omnibus on AI)," *Official Journal of the European Union*, 24 July 2026.

Gelzleichter, Tim, "agent-security-playbook: Reference implementation for 'When Agents Act'," software, MIT licence, 2026. `github.com/A043-studios/agent-security-playbook`.

Goguen, Joseph A. and José Meseguer, "Security Policies and Security Models," in "Proceedings of the 1982 IEEE

Symposium on Security and Privacy," 1982, pp. 11–20.

Greshake, Kai, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz, "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 (AISec '23)," 2023, pp. 79–90. arXiv preprint 2302.12173.

Hardy, Norm, "The Confused Deputy (or Why Capabilities Might Have Been Invented)," *ACM SIGOPS Operating Systems Review*, 1988, *22* (4), 36–38.

Hines, Keegan, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman, "Defending Against Indirect Prompt Injection Attacks With Spotlighting," arXiv preprint 2403.14720, 2024. https://arxiv.org/abs/2403.14720.

Meta AI, "Agents Rule of Two: A Practical Approach to AI Agent Security," 31 October 2025. https://ai.meta.com/blog/practical-ai-agent-security/.

Model Context Protocol, "MCP Joins the Agentic AI Foundation," 9 December 2025. https://blog.modelcontextprotocol.io/posts/2025-12-09-mcp-joins-agentic-ai-foundation/.

Model Context Protocol, "Specification 2026-07-28: Key Changes," 28 July 2026. https://modelcontextprotocol.io/specification/2026-07-28/changelog.

Nasr, Milad, Nicholas Carlini, Chawin Sitawarin, Sander V. Schulhoff, Jamie Hayes, Michael Ilie, Juliette Pluto, Shuang Song, Harsh Chaudhari, Ilia Shumailov, Abhradeep Thakurta, Kai Yuanqing Xiao, Andreas Terzis, and Florian Tramèr, "The Attacker Moves Second: Stronger Adaptive Attacks Bypass Defenses Against LLM Jailbreaks and Prompt Injections," arXiv preprint 2510.09023, 2025. USENIX Security Symposium 2026.

OWASP Foundation, "OWASP Top 10 for LLM Applications 2025," November 2024. https://genai.owasp.org/llm-top-10/.

OWASP GenAI Security Project, "OWASP Top 10 for Agentic Applications for 2026," 9 December 2025. https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026.

OWASP GenAI Security Project, "GenAI Exploit Round-up Report Q1 2026," 14 April 2026. https://genai.owasp.org/2026/04/14/owasp-genai-exploit-round-up-report-q1-2026/.

Perez, Fábio and Ian Ribeiro, "Ignore Previous Prompt: Attack Techniques for Language Models," NeurIPS 2022 Workshop on Machine Learning Safety, 2022. arXiv preprint 2211.09527.

Sabelfeld, Andrei and Andrew C. Myers, "Language-Based Information-Flow Security," *IEEE Journal on Selected Areas in Communications*, 2003, *21* (1), 5–19.

Saltzer, Jerome H. and Michael D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, 1975, *63* (9), 1278–1308.

Vassilev, Apostol et al., "Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations," NIST AI 100-2 E2025, National Institute of Standards and Technology, March 2025.

Wallace, Eric, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel, "The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions," arXiv preprint 2404.13208, 2024. https://arxiv.org/abs/2404.13208.

Willison, Simon, "Prompt Injection Attacks against GPT-3," 12 September 2022. https://simonwillison.net/2022/Sep/12/prompt-injection/.

Willison, Simon, "The Dual LLM Pattern for Building AI Assistants That Can Resist Prompt Injection," 25 April 2023. https://simonwillison.net/2023/Apr/25/dual-llm-pattern/.

Willison, Simon, "Design Patterns for Securing LLM Agents against Prompt Injections," 13 June 2025a. https://simonwillison.net/2025/Jun/13/prompt-injection-design-patterns/.

Willison, Simon, "The Lethal Trifecta for AI Agents: Private Data, Untrusted Content, and External Communication," 16 June 2025b. https://simonwillison.net/2025/Jun/16/the-lethal-trifecta/.

Yao, Shunyu, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in "International Conference on Learning Representations (ICLR)," 2023.