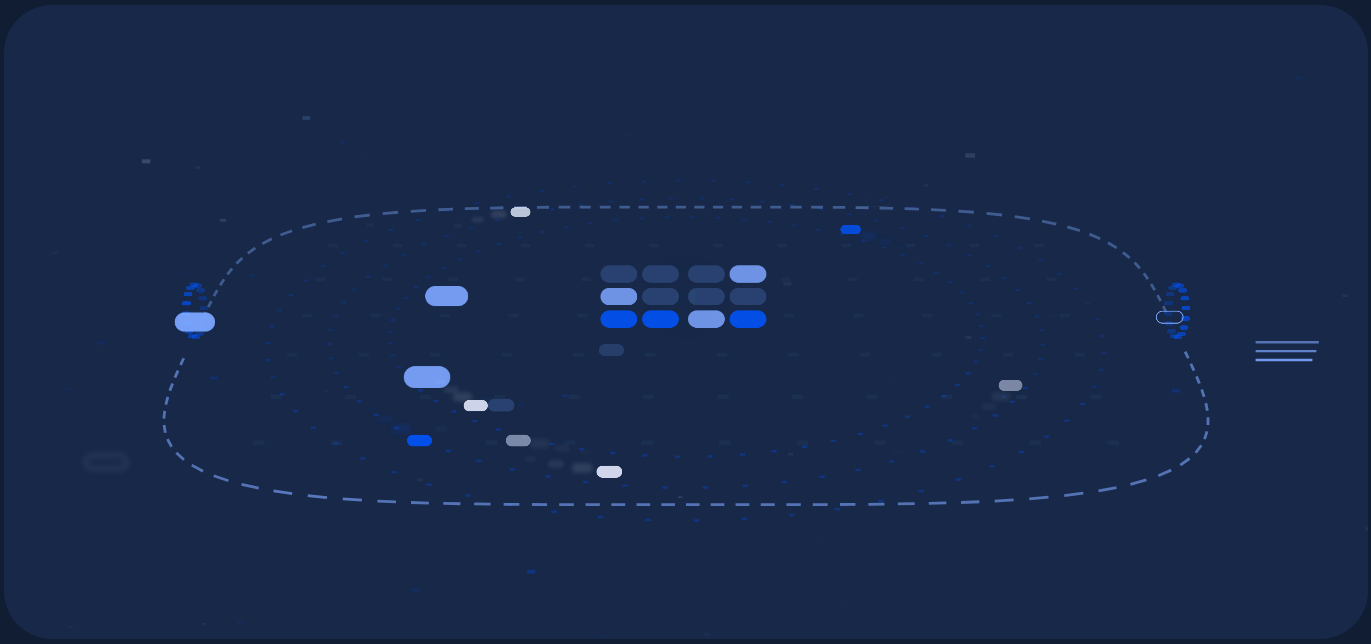
Rillor

EXPLAINER

Prompt injection: how it works and how to test an agent

Why text an agent reads can override its instructions, and what prompt injection testing has to measure on superintelligence (SI) agents and the tools they call.



Rillor

Contents

01 Summary

02 Background

03 Method

04 Findings

05 Implications

06 Limits of this analysis

07 Sources

Summary

Prompt injection lets text an agent reads change what it does, and none of the guidance reviewed here offers a defense that removes it [1, 3, 8].

- LLMs read instructions and data in one channel, so an instruction planted in a web page, email, file or tool result can steer an agent that reads it. NIST, the UK NCSC and the CWE weakness catalog describe the same flaw [3, 7, 8].
- Direct prompt injection arrives in the user's own message. Indirect prompt injection is planted by a third party in content the agent reads later, and the user is often the one harmed [1, 3, 6].
- Disclosed cases include CVE-2025-32711, a zero-click email injection in an enterprise assistant, and a May 2025 demonstration in which a malicious public issue led an agent to leak private repository data [2, 11, 13].
- Defenses lower measured rates and leave attacks that work. In AgentDojo, a benchmark of 97 agent tasks, a tool filter cut one model's attack success from 57.69% to 6.84%, and a detector cut it to 7.95% while benign task success fell from 69.0% to 41.49% [22].
- Attacks adapted to a defense undo low scores measured with fixed attacks. Attacks tuned on some tasks raised success on held-out tasks from 11% to 81% in tests by NIST's Center for AI Standards and Innovation, and an October 2025 study bypassed 12 recent defenses with attack success above 90% against most of them [5, 26].
- Repetition changes the estimate. In tests by NIST's Center for AI Standards and Innovation, 25 attempts per attack raised average success from 57% to 80%, and CWE-1427, the weakness entry for this flaw, advises repeating each case and retesting whenever the model changes [5, 7].
- The OWASP prompt injection cheat sheet asks that each test case state its intended violation or task, the context it needs, the channel its input goes in and the outcome that shows a violation, run on dummy data with sandboxed tools, and report results for each security objective, such as a leaked test marker or an unauthorized tool call, with numerators, denominators and versions [9].
- A passed test shows that a defined set of attacks failed on one model version. It does not show resistance to a persistent adversary, and no source reviewed sets an acceptable attack success rate [9, 16, 18].

Background

What prompt injection is

OWASP's LLM01:2025 entry describes a prompt injection vulnerability as user prompts that alter an LLM's behavior or output in unintended ways [1]. NIST AI 600-1 describes indirect prompt injection as adversaries exploiting LLM-integrated applications remotely, by placing prompts in data likely to be retrieved [4]. MITRE ATLAS technique AML.T0051 describes prompts crafted to make a model ignore parts of its original instructions and follow the adversary's, with effects that can persist through a session [6].

The documented history is short. In September 2022, Riley Goodside showed that a prompt could order a model to ignore its previous directions, and Simon Willison proposed the name prompt injection, by analogy with SQL injection [15]. In November 2022, the PromptInject framework named two attack types, goal hijacking and prompt leaking [19]. In February 2023, Greshake and colleagues described indirect prompt injection and demonstrated it against a deployed search chat assistant and code-completion engines [20].

Why agents raise the stakes

NIST AI 100-2 E2025 describes an agent as software that prompts a model repeatedly, processes its outputs, for example by calling a function it selected, and returns the results in its next prompt. Attacks on agents can let an attacker execute arbitrary code or exfiltrate data from the environment the agent runs in [3]. NIST's Center for AI Standards and Innovation (CAISI) defines agent hijacking as indirect prompt injection that inserts malicious instructions into data an agent may ingest, causing unintended, harmful actions [5]. The OWASP Top 10 for Agentic Applications, published on 9 December 2025, lists Agent Goal Hijack first, as ASI01 [2].

This report is for teams who build, test or deploy agents.

Method

What was examined. Definitions and taxonomies, the explanations of why prompt injection works, its channels and disclosed cases, measured attack and defense rates, adaptive and repeated attacks, design patterns that limit damage, and what guidance says a test should do and report.

Sources. Standards, catalogs and specifications from OWASP, NIST, MITRE and the Model Context Protocol [1, 2, 3, 4, 5, 6, 7, 9, 10], a UK NCSC blog post [8], vulnerability records and disclosures [11, 12, 13, 14], practitioner and developer posts [15, 16, 17, 18] and research papers with their benchmarks [19-35].

How. Every source was opened on 7 November 2026. Papers were read from their arXiv abstracts, and from the full text where a figure comes from a table [22, 25, 26, 34]. Each figure is a measurement under its study's own setup, given with its date. Model developers appear only as publishers in Sources. Numbers computed for this report are labelled.

Terms. Attack success rate (ASR) is the share of attack cases in which the attacker's goal is met. Benign utility is the share of user tasks an agent solves with no attack present. Utility under attack is the share of attack cases in which the agent completes the user's task without the attacker's side effects [22].

Findings

1. Definitions and taxonomies agree on two kinds, and one adds a third

Source	Direct	Indirect	Also covers
OWASP LLM01:2025 [1]	The user's own prompt alters the model's behavior	The model accepts input from external sources such as websites or files	Either kind can be intentional or unintentional
NIST AI 100-2 E2025 [3]	The main user's instructions are appended to higher-trust instructions such as the system prompt	The attacker modifies documents or web pages that the model ingests at runtime	Jailbreaks, filed as direct prompting attacks
MITRE ATLAS 2026.10 [6]	AML.T0051.000	AML.T0051.001, through a separate data channel	AML.T0051.002, triggered by a user action or an event
OWASP ASI01 [2]	Prompt-based manipulation	Hidden payloads, deceptive tool outputs, forged agent messages, poisoned data	Goal hijack across multi-step behavior

Table 1. How each source divides prompt injection, in paraphrase.

ATLAS release 2026.10 rates AML.T0051 and its direct and indirect sub-techniques as realized, and the triggered sub-technique, added on 4 November 2025, as demonstrated [6]. CWE-1427 covers the underlying weakness, prompts built from externally provided data, and counts any external source that may hold untrusted data as potentially malicious [7]. Two of OWASP's nine example scenarios involve job applications: an instruction a company puts in a job description "to identify AI-generated applications", which an applicant's LLM triggers without the applicant knowing, and a résumé split into parts that combine into an injection [1].

On jailbreaking, OWASP notes that the terms are often used interchangeably and treats jailbreaking as a form of prompt injection that makes the model disregard its safety protocols entirely [1]. For agents, OWASP's agentic list sets LLM01:2025, which concerns a single changed response, apart from ASI01, where manipulated inputs redirect an agent's goals, planning and multi-step behavior [2].

2. The mechanism: instructions and data share one channel

NIST's taxonomy states the cause in one line: because generative models "combine the data and instruction channels", attackers can use the data channel to affect system operations [3]. The UK NCSC wrote in December 2025 that LLMs enforce no security boundary between instructions and data inside a prompt, and

make no distinction between the two at all [8]. CWE-1427 describes a model that fails to tell user input from the developer's directives [7], and CAISI traces agent hijacking to the lack of a clear separation between trusted instructions and untrusted external data [5].

Greshake and colleagues found that LLM-integrated applications blur the line between data and instructions, so processing retrieved prompts can act like arbitrary code execution [20]. BIPIA's authors named two causes: models cannot tell informational context from actionable instructions, and they lack awareness that they should not execute instructions found in external content [24]. The instruction hierarchy paper found that models often give system prompts the same priority as text from untrusted users and third parties [32].

"Prompt Injection as Role Confusion", first posted in February 2026, finds the cause in the model's internal representations [35]. Models perceive the source of text from how it sounds, so a command hidden in a web page can hijack an agent because it reads like user text, despite its tool label [35]. Its zero-shot attack, CoT Forgery, reached 60% success against frontier models whose baseline was near zero, and the measured degree of role confusion predicted attack success before any token was generated [35].

3. Channels and disclosed cases

Channel	Example in the sources	Source
User message	Direct injection into a support chatbot	[1]
Web page	A summarized page that inserts an image linking to an attacker's address, leaking data [1], and planted page content that leads an agent to expose private data [2]	[1, 2]
Email	The crafted email in CVE-2025-32711	[2, 11]
Calendar	An invite that injects a recurring "quiet mode" instruction	[2]
Document or file	A modified document in a retrieval store	[1]
Code and repository content	Code comments, commit messages, issues	[9, 13]
Tool result	Data returned by the tools an agent calls	[22]
Tool metadata	Tool annotations from servers that are not trusted	[10]

Channel	Example in the sources	Source
Memory and session	Effects that persist through a session, memory poisoning	[6, 29]
Image	Instructions placed in an image	[1]

Table 2. Channels for injected instructions.

CVE-2025-32711. Published by the National Vulnerability Database on 11 June 2025, it describes command injection in an enterprise assistant that let an unauthorized attacker disclose information over a network, scored 9.3 by the reporting organization and 7.5 by NVD under CVSS 3.1 [11]. OWASP's agentic list describes the same case, known as EchoLeak, as a zero-click indirect injection in which a crafted email led the assistant to exfiltrate confidential emails, files and chat logs [2].

CVE-2025-54135. Published on 5 August 2025, it concerns a code editor before version 1.3.9 [12]. When the editor's MCP configuration file did not exist, an attacker could chain an indirect prompt injection to write to the settings file and run code on the victim's machine without approval [12]. The reporting organization scored it 8.5 and NVD 9.8 [12].

Earlier records. CWE-1427 lists CVE-2023-32786, prompt injection in an LLM integration framework leading to server-side request forgery, CVE-2024-5184, an email-analysis product whose service logic a direct prompt could take over, and CVE-2024-5565, a library that generates SQL through an LLM with retrieval, whose visualization prompt an injection could alter to run arbitrary Python code [7].

The public issue case. In May 2025, Invariant Labs showed that an issue containing a prompt injection, opened on a public repository, could lead an agent working through an MCP server at its owner's request to pull private repository data into its context and leak it in a public pull request [13]. The researchers noted that many users choose an "Always Allow" confirmation policy, and located the flaw in the agent system's architecture, outside the server's code [13].

Applications in general. HouYi, a black-box attack technique from June 2023, found 31 of 36 LLM-integrated applications susceptible, and ten vendors confirmed the findings [21].

On the open web. A scan of Common Crawl published on 23 April 2026 used monthly snapshots of 2 to 3 billion pages each [14]. A naive search turned up mostly benign text, such as research and teaching material about prompt injection, and the injections confirmed after classification and review were mostly pranks, guidance, search engine manipulation or attempts to deter agents [14]. A small number aimed at data theft, and the authors judged that, as of April 2026, attackers had not put that research into production at scale [14]. The malicious category rose by 32% in relative terms between November 2025 and February 2026 [14], and the post gives no overall count. On that date, the authors expected attempted indirect injections to grow in scale and sophistication [14].

4. Measured attack rates without added defenses

Study	Date	Scale	Result	Source
AgentDojo	June 2024 (figures from version 3, November 2024)	97 user tasks, 27 injection tasks, 629 security cases	The model with the highest benign utility: benign utility 78.22%, utility under attack 51.19%, targeted ASR 33.86%	[22]
InjecAgent	March 2024	1,054 cases, 17 user tools, 62 attacker tools, 30 agents	One ReAct-prompted agent attacked successfully 24% of the time, nearly double with a reinforcing prompt	[23]
Agent Security Bench	October 2024	10 scenarios, 10 agents, over 400 tools	Average attack success rates up to 84.30%	[29]
WASP	April 2025 (figures from version 3, May 2025)	84 tasks from 21 attacker goals	Diversion rates of 17% to 86%, end-to-end success up to about 17%	[25]
Public competition, then the ART benchmark built from it	July 2025	Competition: 22 agents, 44 scenarios, 1.8 million attacks. ART: 4,700 curated attacks on 19 models	Over 60,000 policy violations in the competition. On ART, nearly all agents violated policy on most behaviors within 10 to 100 queries	[28]

Table 3. Attack rates as each study reports them.

AgentDojo's four suites cover a workspace, team messaging, travel and banking, with 6, 5, 7 and 9 injection tasks [22]. Its authors found that the models tested fail many tasks even with no attack, that existing attacks break some security properties and leave others intact, and that more capable models tend to be easier to attack [22]. Injections placed toward the end of a tool's response were more effective than other placements in their tests [22]. For the model in Table 3, utility fell by 27.03 points under attack (computed for this report from [22]).

In the July 2025 competition, more than 60,000 of the 1.8 million attacks, about 3.3%, elicited violations such as unauthorized data access (share computed for this report from [28]). Its authors found high transfer of attacks across models and tasks, and limited correlation between robustness and model size, capability or inference-time compute [28].

5. Defenses and their measured limits

NIST groups mitigations into training, such as teaching models to follow hierarchical trust in prompts, detection, and input processing, such as filtering instructions and spotlighting [3]. Because these do not protect fully, NIST suggests designing systems on the assumption that prompt injection is possible whenever a model sees untrusted input, for example with several models holding different permissions [3].

Defense, a second model in AgentDojo	Targeted ASR
None	57.69%
Delimiting (spotlighting)	41.65%
Repeating the user prompt	27.82%
Prompt injection detector	7.95%
Tool filter	6.84%

Table 4. AgentDojo defense results for a second model, not the one in Table 3, against the paper's strongest attack [22].

The detector's low rate came with too many false positives: benign utility fell from 69.0% to 41.49% [22]. The tool filter fails when the tools a task needs are also enough to carry out the attack, which held for 17% of the test cases, and every defense lost 15 to 20% of utility under attack [22].

Defense papers report larger effects under their own tests. Spotlighting cut attack success from above 50% to below 2% for one model family [31]. Training for an instruction hierarchy greatly increased robustness, including against attack types not seen in training [32]. BIPIA's white-box defense brought attack success near zero in its authors' tests [24]. CaMeL, a design that separates control flow from untrusted data, solved 77% of AgentDojo tasks with provable security, against 84% for the undefended system [33], and the OWASP cheat sheet notes that it does not prevent misleading summaries or phishing text [9].

A model developer reported on 24 November 2025 that, against its adaptive Best-of-N attacker with 100 attempts per environment, one model with improved safeguards was hijacked in 1.4% of attempts, against 8.0% for the model alone and 7.6% with the earlier safeguards, and the three other models in the same chart went from between 12.0% and 13.8% for the model alone to between 1.9% and 2.4% with improved safeguards [18]. The developer added that even a 1% rate still represents meaningful risk [18].

Practitioner guidance treats filters as partial. The OWASP cheat sheet says input screening is not a security boundary and that a guardrail model is itself susceptible to prompt injection [9]. The UK NCSC warns that a deny-list can be evaded by rephrasing in endless ways [8]. Willison noted that in web application security, a filter that catches 95% of attacks fails [16].

6. Adaptive, repeated and human attacks

CAISI's tests, published on 17 January 2025 and updated on 19 December 2025, used AgentDojo with one model and added injection tasks for remote code execution, exfiltration of a user's cloud files and automated phishing [5]. With the UK AI Security Institute, CAISI built attacks on a random subset of workspace tasks and tested them on held-out tasks. Success rose from 11% for the leading baseline attack to 81% for the leading new attack, and the new attacks also worked in the other three suites [5]. Across five injection tasks the average success rate was 57%, with some tasks hijacked far more often than others [5]. With 25 attempts per attack, it rose to 80%, a gain of 23 points (computed for this report from [5]). Among CAISI's lessons: "Evaluations need to be adaptive" [5].

"The Attacker Moves Second", posted on 10 October 2025, argues that evaluating defenses against static attack strings, or against weak optimization not designed with the defense in mind, is flawed [26]. With gradient descent, reinforcement learning, random search and human-guided exploration, its authors bypassed 12 recent defenses with attack success above 90% against most of them, although the majority had first reported near-zero rates [26]. On AgentDojo, spotlighting and prompt sandwiching had static attack success as low as 1%, and the search attack exceeded 95% against both [26]. A training defense with 2% static success fell to a 96% search attack [26].

The same paper ran a human red-teaming competition with over 500 participants, $20,000 in prizes and 40 challenges, and the competitors defeated every challenge [26]. On a selected subset, the search attack succeeded 69% of the time and the human attackers, taken together, 100% [26]. Its authors conclude that a defense evaluation should try as hard as possible to break the defense, and that human attackers should be given details of the defenses in the system [26]. A February 2025 study bypassed all eight indirect injection defenses it evaluated, each above 50% success [27].

7. Design patterns that limit what an injection can do

The UK NCSC calls an LLM system inherently confusable [8]. If the model drives tools or APIs, the NCSC equates the impact of an injection with giving an attacker direct access to those tools, and recommends deterministic safeguards outside the model that constrain the system's actions [8]. Citing an exchange between Simon Willison and Baibhav Bista, it adds that when a model processes information from a party, its privileges should drop to that party's [8].

Willison's lethal trifecta names the combination that lets an injection steal data: access to private data, exposure to untrusted content and a way to communicate externally [16]. The Agents Rule of Two, published by a model developer on 31 October 2025, says that within a session an agent should satisfy no more than

two of three properties: it processes untrustworthy inputs, it can reach sensitive systems or private data, and it can change state or communicate externally [17]. An agent that needs all three needs at least supervision, through human approval or another reliable check [17].

A June 2025 paper sets out a principle: once an agent has ingested untrusted input, it must be constrained so that the input cannot trigger consequential actions, meaning actions with negative side effects on the system or its environment [34].

Pattern	How it limits an injection
Action-Selector	The model picks from a fixed list of actions
Plan-Then-Execute	Tool outputs cannot change which actions the agent takes
LLM Map-Reduce	Isolated model calls each process one piece of third-party data
Dual LLM	A privileged model has tools and never processes untrusted data
Code-Then-Execute	The model writes code that calls tools and other models

Table 5. Five of the six design patterns, from [34]. The sixth is Context-Minimization [34].

The authors note that these patterns deliberately limit an agent's ability to perform arbitrary tasks [34]. CaMeL extracts the control and data flow from the trusted query, so untrusted data cannot change the program's flow, and enforces security policies when tools are called [33].

The Model Context Protocol specification, version 2025-11-25, says a person should always be able to deny tool invocations, that clients must treat tool annotations as untrusted unless they come from trusted servers, and that clients should validate tool results before passing them to the model and log tool usage [10]. OWASP's LLM01:2025 mitigations include least privilege, human approval for high-risk actions and segregating external content [1]. Its ASI01 mitigations add treating all natural-language inputs as untrusted [2].

8. What the guidance says to test

Source	What it asks for
OWASP LLM01:2025 [1]	Adversarial testing that treats the model as an untrusted user, to check trust boundaries and access controls

Source	What it asks for
OWASP ASI01 [2]	Periodic red-team tests that simulate goal override, with a check that rollback works
NIST AI 600-1 [4]	Red-teaming to assess resilience against attacks such as prompt injection (action MS-2.7-007)
CWE-1427 [7]	The same test case run several times, and testing again for each new model or weight update
CAISI [5]	Shared frameworks, adaptive attacks, task-level results and multiple attempts
UK NCSC [8]	Logs detailed enough to spot suspicious activity, possibly the full model input and output

Table 6. Testing guidance, in paraphrase.

The NCSC adds that detecting failed tool or API calls can reveal the earlier stages of an attack [8]. For test material, NIST names AgentDojo as an evaluation framework for agent hijacking, and garak and PyRIT as open-source tools for finding weaknesses in models [3]. AgentDojo's authors describe it as an extensible environment for designing and evaluating new agent tasks, defenses and adaptive attacks [22]. The garak paper notes that models produce unpredictable output and are updated constantly, and that a weakness in one setting may not matter in another [30].

9. Designing prompt injection test cases

The OWASP cheat sheet offers 16 attack inputs and nine benign requests as a smoke test, and calls them illustrative, never a security benchmark [9]. Its rules for a test case fill most of Table 7.

Field	What it holds	Source
Objective	The intended security violation, or the legitimate task for a benign case	[9]
Context	What the case needs in place to run	[9]
Channel	Where the input goes, for indirect cases the external content channel under test	[9]
Observable	The outcome that shows a violation, one for each security objective	[9]
Data and tools	Dummy data, sandboxed tool substitutes and test markers, never real secrets	[9]
Language	Each language and script the application accepts, with benign requests in each	[9]

Field	What it holds	Source
Attempts	Repeated runs, recorded as repeats and never as independent cases	[5, 7, 9]

Table 7. The fields of one test case.

Placement decides what a case tests: an indirect injection submitted as a user message tests a different boundary [9]. A realistic threat model limits the attacker to what a third party can do. In WASP, the attacker is a user of a website the agent visits: on a cloned code-hosting site it can open issues and post comments, and it cannot change the layout, create pop-ups or edit other users' comments [25]. AgentDojo pairs each suite's user tasks with injection tasks that state the attacker's goal [22]. The cheat sheet's attack types include encoding and obfuscation, multilingual input, HTML and Markdown injection, multi-turn attacks, system prompt extraction, multimodal input, retrieval poisoning and attacks on an agent's tools, and Best-of-N, which generates variations until one gets through [9].

10. Scoring and reporting

Measure	What it counts	Source
Benign utility	User tasks solved with no attack	[22]
Utility under attack	Attack cases where the user's task is still completed without side effects	[22]
Targeted ASR	Attack cases where the attacker's goal is met	[22]
ASR-intermediate	Cases where the agent was diverted from the user's objective	[25]
ASR-end-to-end	Cases where the attacker's goal was achieved	[25]
False-positive rate	Benign requests blocked or refused, per language	[9]

Table 8. Measures for a prompt injection test.

The cheat sheet's grading rules follow from what a transcript hides. A refusal in the final response does not undo an action already taken, and a clean text response does not show that no data left through another channel, so each objective gets its own observable: disclosure of a test marker, unauthorized tool use or data changes, and external disclosure [9]. Each case ends as violation observed, no violation observed,

inconclusive or not applicable. Missing telemetry, errors and unsupported test contexts never count as blocked attacks, and the grader is validated against known outcomes first [9]. A system that refuses every benign request shows a 100% false-positive rate [9].

Small samples carry wide intervals. Zero false positives in seven independent trials gives a 95% Wilson interval of about 0% to 35.4% [9]. The cheat sheet asks reports to keep per-case outcomes, the numerator and denominator of each rate, the corpus source, model and defense versions, settings and the number of repeated runs, and to report each security objective on its own, with no combined score [9]. CAISI adds that task-level results can inform a risk assessment beyond the aggregate [5]. Passing such a test does not show resistance to a persistent adversary [9].

Implications

For teams building agents. Assume that an agent exposed to untrusted content can be steered, and limit what it can do after reading that content [3, 8, 34]. Check each agent against the Rule of Two and add supervision where it needs all three properties [17]. Validate tool results and log tool use [10].

For owners and buyers of agents. Ask which channels a test covered, which model version it ran on, how many attempts it made per case and whether the attacks were adapted to the defenses [5, 7, 9, 26]. A low rate on a static set says little about an adaptive attacker [26, 27].

For testers. Place each indirect case in the channel it targets, and instrument the tools so that actions and outbound data are observed apart from the final text [9]. Score utility alongside attack success, so that a defense that blocks everything shows its cost [9, 22]. Repeat cases, adapt attacks to the defense, add human attempts and retest on each model change [5, 7, 26].

Rillor's testing. Rillor's agentic security work tests whether instructions sent in a user message (direct) or planted in pages, emails, files and tool results the agent reads (indirect) change what it does. Each test names the model family and the pinned version it ran on, and every run is logged. Rillor tests only systems the client owns or is authorized to test, within a written scope.

Limits of this analysis

- This is an explainer of published definitions, measurements and guidance. It does not rank models, agents or defenses.
- Every figure is a measurement under one study's setup, on the models and defenses of its date [5, 22, 26].
- No source reviewed measures the share of deployed agents that are vulnerable, and the April 2026 scan gives no overall count [14].
- No source reviewed sets an acceptable attack success rate or a standard number of attempts per case [5, 7, 9].
- The browser agent figures are a model developer's measurements of its own models [18].
- CAISI publishes its per-task rates only in a chart, so this report gives none.
- OWASP ids are cited as references, never as certifications. LLM01 carries its 2025 edition [1].

Sources

1. LLM01:2025 Prompt Injection, OWASP Top 10 for LLM Applications 2025. OWASP GenAI Security Project, 2025 edition, viewed 7 November 2026. https://genai.owasp.org/llmrisk/llm01-prompt-injection/

2. OWASP Top 10 for Agentic Applications for 2026. OWASP GenAI Security Project, 9 December 2025, viewed 7 November 2026. https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/

3. Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations, NIST AI 100-2 E2025. National Institute of Standards and Technology, March 2025, viewed 7 November 2026. https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-2e2025.pdf

4. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile, NIST AI 600-1. National Institute of Standards and Technology, July 2024, viewed 7 November 2026. https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

5. Technical Blog: Strengthening AI Agent Hijacking Evaluations. National Institute of Standards and Technology, Center for AI Standards and Innovation, 17 January 2025, updated 19 December 2025, viewed 7 November 2026. https://www.nist.gov/news-events/news/2025/01/technical-blog-strengthening-ai-agent-hijacking-evaluations

6. ATLAS data, release 2026.10 (technique AML.T0051, LLM Prompt Injection). MITRE, 28 October 2026, viewed 7 November 2026. https://github.com/mitre-atlas/atlas-data/blob/main/dist/v6/ATLAS-2026.10.yaml

7. CWE-1427: Improper Neutralization of Input Used for LLM Prompting. MITRE, Common Weakness Enumeration, first published 19 November 2024, last changed 28 October 2026, viewed 7 November 2026. https://cwe.mitre.org/data/definitions/1427.html

8. Prompt injection is not SQL injection (it may be worse). Dave Chismon, NCSC (United Kingdom), 8 December 2025, viewed 7 November 2026. https://www.ncsc.gov.uk/blog-post/prompt-injection-is-not-sql-injection

9. LLM Prompt Injection Prevention Cheat Sheet. OWASP Cheat Sheet Series, undated, viewed 7 November 2026. https://cheatsheetseries.owasp.org/cheatsheets/LLM_Prompt_Injection_Prevention_Cheat_Sheet.html

10. Tools, Model Context Protocol specification, version 2025-11-25. Model Context Protocol, 25 November 2025, viewed 7 November 2026. https://modelcontextprotocol.io/specification/2025-11-25/server/tools

11. CVE-2025-32711. NIST National Vulnerability Database, published 11 June 2025, viewed 7 November 2026. https://nvd.nist.gov/vuln/detail/CVE-2025-32711

12. CVE-2025-54135. NIST National Vulnerability Database, published 5 August 2025, viewed 7 November 2026. https://nvd.nist.gov/vuln/detail/CVE-2025-54135

13. GitHub MCP Exploited: Accessing private repositories via MCP. Marco Milanta and Luca Beurer-Kellner, Invariant Labs, 26 May 2025, viewed 7 November 2026. https://invariantlabs.ai/blog/mcp-github-vulnerability

14. AI threats in the wild: The current state of prompt injections on the web. Thomas Brunner, Yu-Han Liu and Moni Pande,

Google, 23 April 2026, viewed 7 November 2026. https://blog.google/security/prompt-injections-web/

15. Prompt injection attacks against GPT-3. Simon Willison, Simon Willison's Weblog, 12 September 2022, viewed 7 November 2026. https://simonwillison.net/2022/Sep/12/prompt-injection/

16. The lethal trifecta for AI agents: private data, untrusted content, and external communication. Simon Willison, Simon Willison's Weblog, 16 June 2025, viewed 7 November 2026. https://simonwillison.net/2025/Jun/16/the-lethal-trifecta/

17. Agents Rule of Two: A Practical Approach to AI Agent Security. Meta AI, 31 October 2025, viewed 7 November 2026. https://ai.meta.com/blog/practical-ai-agent-security/

18. Mitigating the risk of prompt injections in browser use. Anthropic, 24 November 2025, viewed 7 November 2026. https://www.anthropic.com/research/prompt-injection-defenses

19. Ignore Previous Prompt: Attack Techniques For Language Models. Fábio Perez and Ian Ribeiro, arXiv:2211.09527, 17 November 2022, viewed 7 November 2026. https://arxiv.org/abs/2211.09527

20. Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. Kai Greshake, Sahar Abdelnabi and others, arXiv:2302.12173, version 2, 5 May 2023, viewed 7 November 2026. https://arxiv.org/abs/2302.12173

21. Prompt Injection attack against LLM-integrated Applications. Yi Liu, Gelei Deng, Yuekang Li and others, arXiv:2306.05499, version 3, 29 December 2025 (first version 8 June 2023), viewed 7 November 2026. https://arxiv.org/abs/2306.05499

22. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. Edoardo Debenedetti, Jie Zhang and others, arXiv:2406.13352, version 3, 24 November 2024, viewed 7 November 2026. https://arxiv.org/abs/2406.13352

23. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. Qiusi Zhan, Zhixiang Liang, Zifan Ying and Daniel Kang, arXiv:2403.02691, version 3, 4 August 2024, viewed 7 November 2026. https://arxiv.org/abs/2403.02691

24. Benchmarking and Defending Against Indirect Prompt Injection Attacks on Large Language Models. Jingwei Yi, Yueqi Xie and others, arXiv:2312.14197, version 4, 27 January 2025, viewed 7 November 2026. https://arxiv.org/abs/2312.14197

25. WASP: Benchmarking Web Agent Security Against Prompt Injection Attacks. Ivan Evtimov, Arman Zharmagambetov and others, arXiv:2504.18575, version 3, 16 May 2025, viewed 7 November 2026. https://arxiv.org/abs/2504.18575

26. The Attacker Moves Second: Stronger Adaptive Attacks Bypass Defenses Against LLM Jailbreaks and Prompt Injections. Milad Nasr, Nicholas Carlini and others, arXiv:2510.09023, 10 October 2025, viewed 7 November 2026. https://arxiv.org/abs/2510.09023

27. Adaptive Attacks Break Defenses Against Indirect Prompt Injection Attacks on LLM Agents. Qiusi Zhan, Richard Fang, Henil Shalin Panchal and Daniel Kang, arXiv:2503.00061, version 2, 4 March 2025, viewed 7 November 2026. https://arxiv.org/abs/2503.00061

28. Security Challenges in AI Agent Deployment: Insights from a Large Scale Public Competition. Andy Zou, Maxwell Lin, Eliot Jones and others, arXiv:2507.20526, 28 July 2025, viewed 7 November 2026. https://arxiv.org/abs/2507.20526

29. Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-based Agents. Hanrong Zhang, Jingyuan Huang, Kai Mei and others, arXiv:2410.02644, version 4, 30 May 2025, viewed 7 November 2026. https://arxiv.org/abs/2410.02644

30. garak: A Framework for Security Probing Large Language Models. Leon Derczynski, Erick Galinkin and others, arXiv:2406.11036, 16 June 2024, viewed 7 November 2026. https://arxiv.org/abs/2406.11036

31. Defending Against Indirect Prompt Injection Attacks With Spotlighting. Keegan Hines, Gary Lopez and others, arXiv:2403.14720, 20 March 2024, viewed 7 November 2026. https://arxiv.org/abs/2403.14720

32. The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions. Eric Wallace, Kai Xiao and others, arXiv:2404.13208, 19 April 2024, viewed 7 November 2026. https://arxiv.org/abs/2404.13208

33. Defeating Prompt Injections by Design. Edoardo Debenedetti, Ilia Shumailov and others, arXiv:2503.18813, version 2, 24 June 2025, viewed 7 November 2026. https://arxiv.org/abs/2503.18813

34. Design Patterns for Securing LLM Agents against Prompt Injections. Luca Beurer-Kellner, Beat Buesser and others, arXiv:2506.08837, version 3, 27 June 2025, viewed 7 November 2026. https://arxiv.org/abs/2506.08837

35. Prompt Injection as Role Confusion. Charles Ye, Jasmine Cui and Dylan Hadfield-Menell, arXiv:2603.12277, version 6 (first version 22 February 2026), viewed 7 November 2026. https://arxiv.org/abs/2603.12277

About this report

Published by Rillor on 8 November 2026. Superintelligence (SI) tools assist research and drafting, and every figure cites its source. Rillor builds and secures SI systems, supplies the data they learn from and delivers the infrastructure they run on. We apply SI to financial and compute markets.

Cite as: Rillor. Prompt injection: how it works and how to test an agent. 8 November 2026. https://rillor.com/insights/prompt-injection-testing

Notices

Superintelligence (SI) outputs can be wrong. Important decisions should include human review.

Work with Rillor

Send us the agents, the tools and data they reach, where they run and what you want tested. Contact us at rillor.com/contact?type=security or contact@rillor.com. More on agentic security: rillor.com/si/agentic-security