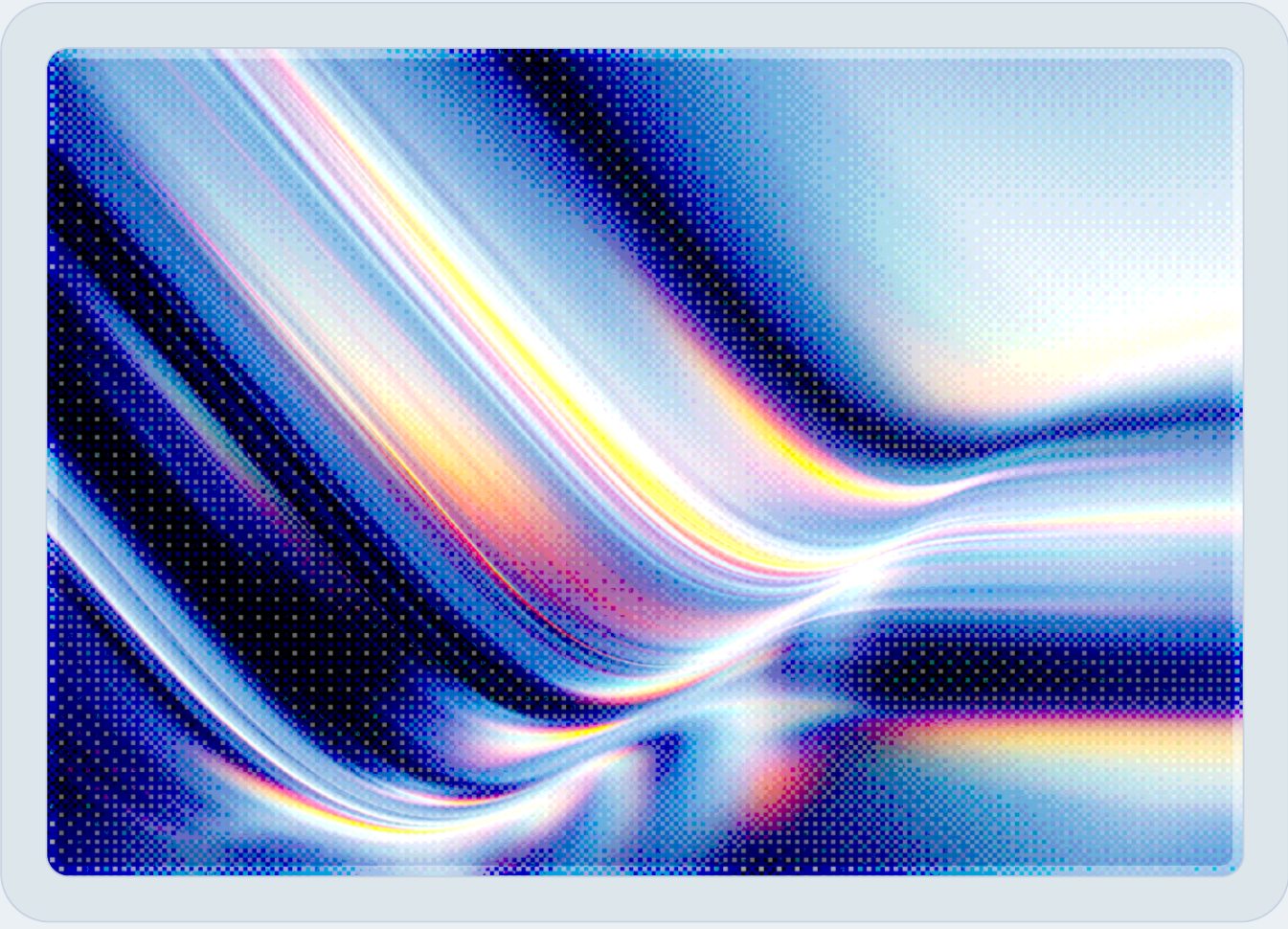
CSAI Foundation | Cloud Security Alliance

Agent Data Injection: A New Attack Class Beyond Prompt Injection

How Corrupted Metadata Bypasses Existing Agent Defenses

2026-07-17

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- Researchers from Seoul National University, the University of Illinois Urbana-Champaign, and Largosoft have identified Agent Data Injection (ADI), a distinct category of indirect prompt injection that corrupts the factual data an AI agent implicitly trusts — such as UI element identifiers, comment authorship metadata, and tool call records — rather than embedding direct instructions [1][2].
- ADI works by exploiting how large language models parse structural markers probabilistically instead of through strict syntax rules; a technique the researchers call probabilistic delimiter injection uses escaped quotes, curly quotes, and similar characters to convince a model that fabricated data boundaries are legitimate ones [2].
- The research team demonstrated working attacks against six frontier models — OpenAI's GPT-5.2 and GPT-5-mini, Anthropic's Claude Opus 4.5 and Sonnet 4.5, and Google's Gemini 3 Pro and Flash — and against real-world deployed agents including Claude in Chrome, Google's Antigravity, Nanobrowser, Claude Code, OpenAI's Codex, and Google's Gemini CLI [1][2].
- Attack success rates against structured data ranged from roughly 31 to 43 percent, and against unstructured webpage content from one-third up to 100 percent; ADI retained up to 50 percent effectiveness against defenses purpose-built to block prompt injection, while those same defenses nearly eliminated classic instruction-based attacks [1][2].
- The underlying weakness is architectural: current agent frameworks do not isolate trusted data from untrusted data at the level of metadata and context records, meaning that even agents hardened against direct prompt injection remain exposed until they adopt provenance tracking or strict data-flow isolation [2].

Background

In CSA's assessment, indirect prompt injection has been the dominant framing for AI agent security risk since agents began consuming untrusted content — web pages, emails, pull requests, tool outputs — as part of their operating context. The defensive posture that has emerged over the past two years assumes the attacker's goal is to smuggle an instruction into that content: a hidden line telling the agent

to exfiltrate data, transfer funds, or execute a command. Guardrail models, input sanitization, and instruction-hierarchy training have all been built around detecting and neutralizing that specific pattern, and CSA's own guidance on securing LLM-backed systems has emphasized authorization boundaries and validation layers as the primary controls against this class of attack [5].

Agent Data Injection, disclosed on July 6, 2026 by a team from Seoul National University, the University of Illinois Urbana-Champaign, and the security firm Largosoft, targets a different layer of the agent's operation entirely [1][2]. Rather than asking the agent to do something it should not, ADI corrupts the facts the agent uses to decide what it is already doing. An agent navigating a web page relies on the identifiers embedded in that page's HTML to know which button is "Buy Now" and which is "Add to Cart." A coding agent reviewing a pull request relies on metadata about who authored a comment and what a prior tool call actually returned. None of this metadata carries an explicit instruction, so instruction-hierarchy defenses have no signal to act on. The Hacker News, reporting on the disclosure, summarized the distinction succinctly: the attack works "one layer down" from where prompt injection defenses are looking [1].

In CSA's assessment, this represents a meaningful shift in how the industry needs to think about agent trust boundaries. It suggests that the defensive investment of the last two years — substantial as it has been — addressed only one dimension of a two-dimensional problem. An agent can be fully resistant to being told what to do by untrusted content and still be manipulated into doing the wrong thing because it was lied to about where it is, what it clicked, or who it is talking to.

Security Analysis

The Core Mechanism: Probabilistic Delimiter Injection

The researchers' central technical contribution is identifying why agents are so susceptible to metadata corruption in the first place. Traditional software parses structured data — JSON fields, HTML attributes, API responses — using deterministic rules: a quote character either closes a string or it does not, based on strict grammar. A large language model, by contrast, reads these same structures as text and infers their boundaries probabilistically, weighing what a delimiter "looks like" against the surrounding context rather than enforcing an exact grammar [2]. The paper terms this vulnerability probabilistic delimiter injection: an attacker embeds characters that resemble legitimate structural markers — escaped quotes, visually similar Unicode quote variants, or symbols like dollar signs used in templating syntax — into a field

the agent expects to be plain data. The model, reading probabilistically, concludes that a new field or a new record boundary exists where none actually does, and the fabricated content is absorbed as if it were a legitimate part of the agent's trusted context.

This matters because it defeats defenses built on the assumption that the trusted/untrusted boundary is enforced correctly by the surrounding application code. If the vulnerability lived only in the model's willingness to follow embedded instructions, then stripping or flagging imperative language in untrusted fields would close the gap. Because the vulnerability lives in how the model perceives structure itself, an attacker does not need to write anything resembling a command — a forged sender name, a duplicated button identifier, or a fabricated verification record accomplishes the same goal without ever containing text a guardrail model would flag as suspicious.

Demonstrated Attack Vectors

The research team validated three concrete attack patterns against production agent systems, organized around where an agent draws metadata from and how that metadata determines its next action.

The first targets web browsing agents through what the paper labels element ID injection. Many web agents identify clickable elements by numeric or sequential identifiers embedded in the page's accessibility tree or DOM structure. By planting a fabricated product review or comment on a page that reuses a real, currently valid button identifier — one associated with a "Buy Now" or "Confirm Payment" element — an attacker can cause the agent to click the wrong target while believing it is completing the user's original request. This was demonstrated against Claude in Chrome, Google's Antigravity, and the open-source Nanobrowser project [1][2].

The second and third vectors target coding agents through what the paper calls origin injection and tool call and response injection. In origin injection, an attacker forges a GitHub comment's apparent authorship metadata so that a coding agent believes an instruction or command originates from a trusted repository maintainer rather than an anonymous or low-privilege account, inducing the agent to execute commands it would otherwise treat with more scrutiny. In tool call and response injection, an attacker fabricates the record of a prior tool execution — for example, a forged test-passing or security-scan result — so that a pull request appears to have already cleared verification steps it never actually underwent, allowing malicious code to slip through automated or semi-automated review. Both variants were confirmed against Claude Code, OpenAI's Codex, and Google's Gemini CLI [1][2].

Quantifying the Threat

The research team measured attack success rate (ASR) across both isolated model tests and full agent-in-the-loop scenarios. On structured JSON-format data, baseline ASR ranged from 31.3 percent (Gemini 3 Pro) to 43.3 percent (Gemini 3 Flash), with GPT-5.2 at 41.8 percent, GPT-5-mini at 40.3 percent, Claude Opus 4.5 at 34.3 percent, and Claude Sonnet 4.5 at 37.3 percent [2]. Against unstructured web DOM data, success rates ranged far more widely, from roughly one-third up to a full 100 percent depending on the specific page structure and target model [1][2]. The consistency of the vulnerability across every tested vendor and model family indicates that ADI is not an artifact of any single company's training approach but a structural consequence of how current agents consume context. These findings originate from a single research team's study; the researchers have released their benchmark suite publicly for independent replication, but the results have not yet been reproduced by a separate group, and the figures below should be read as the best currently available evidence rather than as an established industry-wide consensus.

One of the most operationally significant findings concerns effectiveness against existing defenses. Against a battery of representative mitigations, the paper reports the following agent-level attack success rates, shown against a no-defense baseline of 49.1 percent.

Defense mechanism	Agent-level ASR	Notes
No defense (baseline)	49.1%	–
Input guardrails	50.0%	No measurable improvement
Output guardrails	45.4%	Marginal reduction
Plan-then-execute	40.7%	Modest reduction
Agent sandboxing	22.2%	Meaningful reduction
Dual-LLM architecture	25.0%	Meaningful reduction
Data flow tracking (normal)	23.1%	Meaningful reduction
Data flow tracking (strict)	0%	Full prevention
Randomized identifiers	28.7%	Meaningful reduction

Percentages are from [2]; the qualitative characterizations in the Notes column are CSA's own.

Input guardrails, among the most commonly deployed defenses against conventional prompt injection, produced no improvement against ADI and in one configuration performed marginally worse than having no defense at all. This finding supports the interpretation that ADI is not simply a harder version of an already-solved problem, but a gap the current generation of commercial and open-source defenses was not designed to close. Only architectural controls — strict data flow tracking between trusted and untrusted sources, and randomized rather than sequential resource identifiers — produced substantial reductions, and the fully effective strict data-flow-tracking configuration came at a significant utility cost: overall agent task completion fell to 36.5 percent of baseline performance, compared with an 86.5 percent baseline utility measured without any defense in place [2]. One deployed system, OpenAI's ChatGPT Atlas browser, already benefits from partial protection by design: it identifies page elements using unguessable, randomly generated identifiers rather than simple sequential counters, the same randomized-identifier approach that reduced attack success from 49.1 percent to 28.7 percent in the researchers' own benchmark across the six tested models [1][2]. The researchers did not separately measure an attack success rate for the deployed Atlas product itself; the relevant point is that Atlas's existing engineering choice happens to match the defense category their benchmark validated as effective.

Disclosure and Vendor Response

The research team reported findings to Anthropic, OpenAI, Google, and the Nanobrowser project prior to publication. Anthropic, OpenAI, and Google acknowledged the validity of the findings; as of publication, no vendor had reported a shipped or scheduled fix, and Nanobrowser had not responded to the disclosure [1][2]. The researchers released their benchmark suite and attack code to support vendor testing and further community research [1].

Recommendations

Immediate Actions

Organizations operating agentic systems in production — particularly web-browsing agents and coding assistants with commit, merge, or deployment authority — should inventory where those agents derive metadata they treat as authoritative: element identifiers, comment or commit author fields, and tool call or test result records. Any agent that trusts sequential or predictable identifiers for UI elements should be flagged as a near-term priority for remediation, since the research indicates this pattern is directly and reliably exploitable. Security teams should also review whether their current prompt injection

defenses — particularly input and output guardrail models — were validated against metadata-corruption attacks specifically, since the evidence here indicates that validation against instruction-based injection alone provides a false sense of coverage.

Short-Term Mitigations

Where agent frameworks support it, organizations should move away from sequential or human-predictable resource and element identifiers toward randomized, unguessable ones, following the pattern that already reduced ADI effectiveness in at least one production browser agent. For coding agents with merge or execution authority, tool call and response records — test results, security scan outcomes, review approvals — should be cryptographically bound to the actual execution event rather than represented as freely rewritable text in the agent's context, so that a forged "tests passed" record cannot be inserted downstream of the real tool invocation. Agent sandboxing and dual-LLM architectures, which separate a privileged planning model from a lower-privilege model that handles untrusted content, produced meaningful (though not complete) reductions in the study and represent a reasonable interim architecture for agents that cannot yet support full data provenance tracking.

Strategic Considerations

The research's central conclusion — that current agents do not isolate trusted data from untrusted data at the level of metadata and context records — points toward a broader architectural requirement rather than a patchable defect. Full data flow tracking eliminated the attack entirely in testing but reduced task-completing utility to 36.5 percent of the 86.5 percent no-defense baseline, more than halving task success, which will be an unacceptable utility cost for many production deployments, though organizations with a lower risk tolerance for agent errors may find the tradeoff justified [2]. This gap between the only fully effective defense and what is currently practical to deploy should inform how organizations sequence their agent security investments: rather than treating prompt injection defenses as sufficient coverage for agent risk, security and engineering leadership should treat data provenance — the ability to distinguish where every piece of an agent's context actually originated, and to prevent an agent from treating unverified data as if it were verified — as the next major control category to mature, on the same timeline organizations are already using to build out agent authorization and sandboxing controls.

CSA Resource Alignment

CSA's Agentic AI Red Teaming Guide [3] is the most directly applicable prior CSA publication. Its taxonomy of Agentic AI vulnerability categories already includes agent context manipulation and knowledge base poisoning as distinct threat classes alongside instruction-based prompt injection, and its guidance on testing agent trust boundaries maps closely onto the metadata-corruption vectors this research note describes; organizations using that guide's testing methodology should extend their element ID, comment-authorship, and tool-response test cases specifically to cover the ADI attack patterns documented here rather than assuming existing prompt-injection test coverage is sufficient.

CSA's presentation "The AI Security Gap: Why Protecting Prompts Isn't Enough" [4], delivered at the CSA Summit 2025 at RSAC, previewed precisely this argument at the architectural level — that prompt-level defenses address only part of the agent security surface and that data governance and runtime controls are required to close the remainder. ADI is a concrete, empirically validated instance of the gap that presentation warned practitioners to anticipate, and its inclusion here should reinforce for readers that the concern was not speculative.

CSA's guidance on "Securing LLM Backed Systems: Essential Authorization Practices" [5] provides the authorization-layer controls — external, out-of-band authorization checkpoints that do not rely on the agent's own interpretation of context — that offer a practical mitigation path for the coding-agent attack vectors described in this note, particularly the forged tool-call and verification-record attacks against pull request review.

Finally, CSA's MAESTRO Agentic AI Threat Modeling Framework [6] offers the layer-by-layer reference architecture organizations should use to locate where ADI-style risk enters their own systems: the framework's Data Operations and Agent Frameworks layers correspond directly to the metadata and context-handling surfaces this research targets, and threat models built using MAESTRO should be updated to treat agent-consumed metadata as an explicit trust boundary distinct from the instruction channel already covered by conventional prompt injection threat modeling.

References

[1] The Hacker News. "New Agent Data Injection Attack Can Make AI Agents Misclick or Run Attacker Commands." The Hacker News, July 2026.

[2] Choi, Woohyuk, Juhee Kim, Taehyun Kang, Jihyeon Jeong, Luyi Xing, and Byoungyoung Lee. "Agent Data Injection Attacks are Realistic Threats to AI Agents." arXiv:2607.05120, July 2026.

[3] Cloud Security Alliance. "Agentic AI Red Teaming Guide." Cloud Security Alliance, 2025.

[4] Cloud Security Alliance. "The AI Security Gap: Why Protecting Prompts Isn't Enough." CSA Summit 2025 at RSAC, 2025.

[5] Cloud Security Alliance. "Securing LLM Backed Systems: Essential Authorization Practices." Cloud Security Alliance, 2024.

[6] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." Cloud Security Alliance, February 2025.

