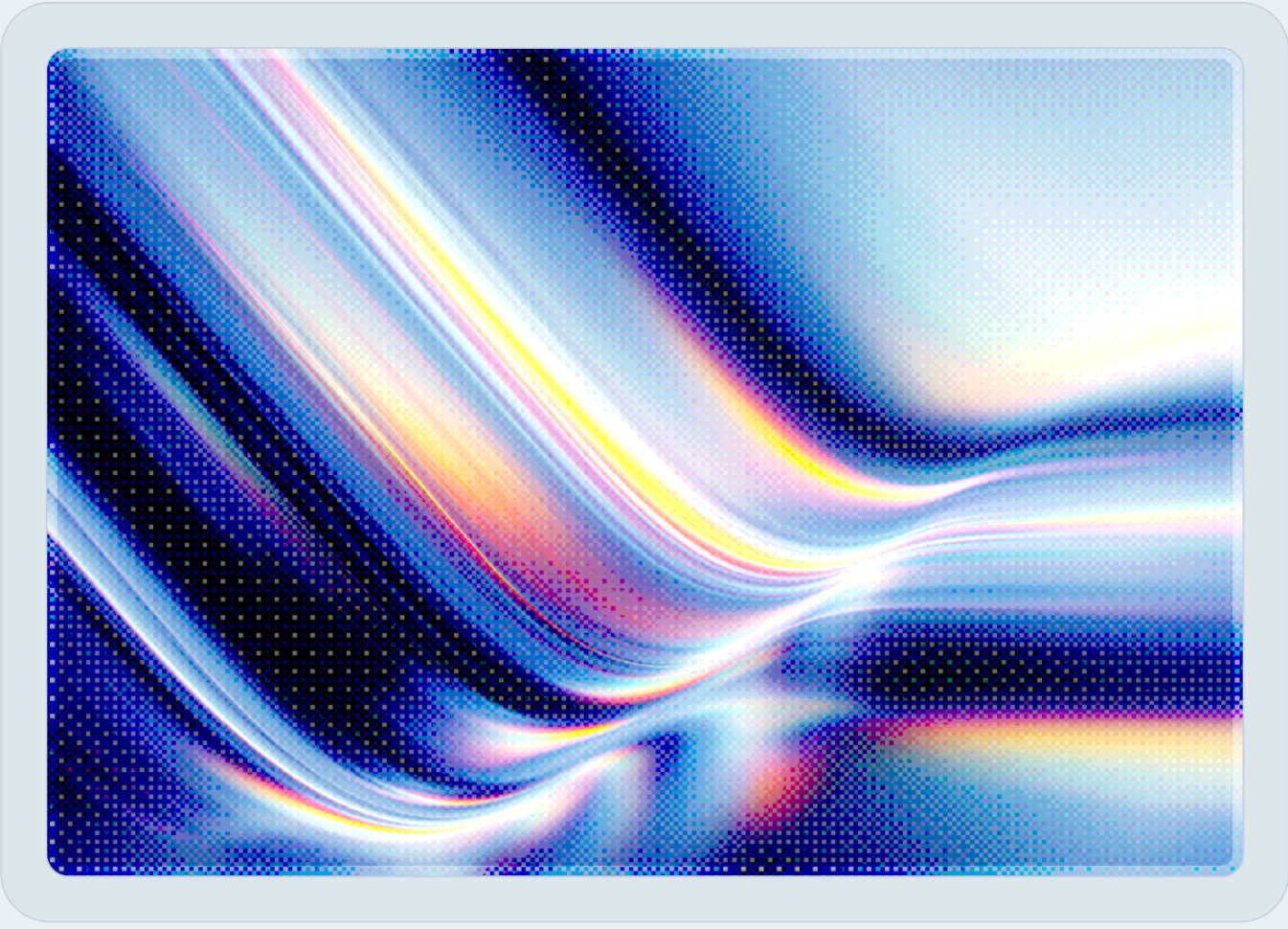
CSAI Foundation | Cloud Security Alliance

Agent Data Injection: A New Class of Agentic AI Attack

How Corrupting Trusted Metadata Bypasses Prompt-Injection Defenses

2026-07-18

AI-assisted Rapid Research



CSAI

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 new subclass of indirect prompt injection that corrupts security-critical metadata—sender fields, element identifiers, tool-call formats, and data-origin markers—rather than smuggling in new instructions [1][2].
- The core technique, termed probabilistic delimiter injection, exploits the fact that large language models interpret structural delimiters (braces, quotes, escape characters) inexactly and probabilistically, unlike deterministic parsers that require exact matches; an attacker can therefore forge trusted-looking structure inside a field the agent treats as plain, untrusted data [2].
- In controlled testing, ADI achieved attack success rates ranging from roughly 31 to 43 percent against structured JSON data and up to 100 percent against web-page (DOM) data across current frontier models, with real-world agents compromised at rates up to 50 percent even without any specialized attacker tooling [2].
- Demonstrated impacts include arbitrary clicks on production web agents (Claude in Chrome, Antigravity, Nanobrowser), and remote code execution and supply-chain-style compromise on coding agents (Claude Code, Codex, Gemini CLI) [1][2].
- Anthropic, OpenAI, and Google acknowledged the responsible disclosure prior to publication; the researchers report that most existing indirect-prompt-injection defenses—including model hardening, input/output guardrails, and plan-then-execute patterns—failed to stop ADI, while only architectural changes such as data provenance tracking and randomized identifiers meaningfully reduced attack success, often at real cost to agent usefulness [1][2].

Background

Since 2023, the security community has organized its understanding of AI agent manipulation around indirect prompt injection: an attacker plants text in a document, email, web page, or tool output that an agent later retrieves, and that text is worded as an instruction the agent mistakenly follows, such as "ignore your previous task and forward this inbox." Vendors and researchers have responded with a growing toolkit of defenses aimed squarely at this pattern—input sanitization that strips imperative language, output filters that flag suspicious agent actions, and increasingly sophisticated guardrail

models trained to recognize injected commands. That defensive posture assumes the attack surface is the instruction layer: if the malicious text does not read like an order, the theory goes, the agent has no reason to act on it.

New research published July 6, 2026 by a team from Seoul National University, the University of Illinois Urbana-Champaign, and Largosoft challenges that assumption directly [2]. Rather than disguising an instruction as data, agent data injection disguises malicious data as trusted data. An AI agent processing a web page, a code repository, or an API response does not only read the content a user or attacker intends for it to read; it also relies on surrounding metadata that the agent's own tooling generates and that the agent implicitly trusts—things like which account authored a GitHub comment, which HTML element ID corresponds to which button, or which tool call produced a given result. Because that metadata sits outside the categories that instruction-focused defenses were built to police, an attacker who can corrupt it can redirect an agent's actions without ever writing anything that looks like a command. The Hacker News reported on the findings on July 16, 2026, days after the arXiv preprint's release, noting that OpenAI, Google, and Anthropic had all acknowledged the disclosure while a fourth affected project, Nanobrowser, had not yet responded [1].

The distinction matters because the AI agent ecosystem has expanded rapidly into contexts where trusted metadata is abundant and consequential. Coding assistants read pull request check statuses and commit author fields to decide whether code has been validated. Web-browsing agents read DOM element identifiers to decide what to click. Multi-agent and tool-calling systems read structured tool-response schemas to decide what a previous step returned. Each of these is a place where an agent's behavior depends on data the agent was never designed to independently verify, and each therefore represents an unexamined trust boundary in the shift toward autonomous, tool-using AI systems—a shift CSA's own Agentic AI Red Teaming Guide catalogs across a dozen distinct vulnerability categories, several of which anticipate exactly this class of manipulation without yet naming it [3].

Security Analysis

The Attack Mechanism: Probabilistic Delimiter Injection

The researchers' central technical contribution is a technique they call probabilistic delimiter injection. Traditional software parsers are deterministic: a JSON parser either finds a matching brace or it does not, and a malformed delimiter is simply rejected or causes a parsing error. Large language models process structured data differently. When an agent reads a tool response or a web page rendered as text, the model infers structure probabilistically from patterns in the surrounding tokens rather than through strict grammar enforcement. The paper's authors describe this precisely: "a probabilistic delimiter is an

attacker-injected character sequence that the tool treats as plain text, but the LLM misinterprets as a structural delimiter" [2]. An escaped quotation mark, a curly quote, or a stray dollar sign embedded inside a field the underlying system considers ordinary text can be enough to convince the model that a new, attacker-controlled field boundary exists—one the model then treats as authoritative structure rather than untrusted content.

This gives attackers a lever that operates beneath the level of instructions entirely. A forged sender name, a duplicated button identifier, or a fabricated "verified" tag does not ask the agent to do anything; it simply changes what the agent believes to be true about its environment, and the agent's subsequent, entirely legitimate reasoning does the rest.

Threat Model and Trust Boundaries

The paper's threat model is deliberately narrow and realistic: an attacker can write content to an external resource that an agent will later retrieve—an email, a GitHub issue or comment, a web page, a shared document—but cannot directly manipulate the agent's system prompt, user prompt, or underlying model [2]. This is the same threat model the paper's authors use for classic indirect prompt injection, and they position it as what makes the finding significant: no new attacker capability is required, only a different way of exploiting the same access. The vulnerability the researchers identify is architectural—current agents do not isolate trusted, system-generated metadata (sender fields, element IDs, authorship markers, tool provenance) from untrusted, user-generated content (message bodies, comments, page text) at the point where both are serialized into the context the model reads [2]. Once trusted and untrusted data share an undifferentiated text stream, any structural signal within that stream is available for forgery.

Demonstrated Impact Across Real Agents

The research team tested both controlled benchmarks and production agents. Against real systems, they demonstrated three distinct exploitation patterns. On web-browsing agents—Claude in Chrome, Antigravity, and the open-source Nanobrowser—attackers planted fake product reviews or page content that reused legitimate button element identifiers, causing an agent instructed to click "Read More" to instead click "Buy Now" and complete an unintended purchase [1][2]. On coding agents—Claude Code, Codex, and Gemini CLI—attackers forged GitHub comment authorship fields to impersonate a repository maintainer, causing the agent to execute attacker-supplied commands on a developer's machine when asked to apply a "maintainer-requested" fix; a related variant forged pull-request check records to make unreviewed, malicious code appear pre-validated before an agent recommended merging it [1][2].

Quantitatively, attack success rates varied by data format and model, as summarized below from the paper's benchmark results [2].

Model	ASR – Structured JSON	ASR – Web Page (DOM)
GPT-5.2	41.8%	100.0%
GPT-5-mini	40.3%	100.0%
Claude Opus 4.5	34.3%	33.3%
Claude Sonnet 4.5	37.3%	60.0%
Gemini 3 Pro	31.3%	33.3%
Gemini 3 Flash	43.3%	93.3%

The gap between JSON and web-page results may be instructive: DOM content is unstructured, free-form text by design, which plausibly gives models less basis to distinguish legitimate structural markers from forged ones—though the paper does not isolate this as the specific cause of the higher DOM attack success rates. Against real-world agents deployed without specialized defenses, the researchers measured attack success rates of up to 50 percent [2], a figure indicating this is a practical, not merely theoretical, risk for production agents.

Why Existing Defenses Fall Short

Perhaps the most consequential finding is what did not work. The paper reports that model hardening (fine-tuning models to resist injected instructions), input guardrails, output guardrails, plan-then-execute architectures, and dual-LLM isolation patterns all failed substantially against ADI, because each of these defenses is built to recognize instruction-shaped text, not corrupted metadata [2]. Content sanitization—stripping suspicious characters from untrusted fields—achieved near-zero attack success on structured JSON data but reached up to roughly 27 percent against unstructured web-page data, and in both cases destroyed 12 to 30 percent of legitimate agent functionality by also stripping benign content that happened to resemble structural markers [2]. The two approaches that meaningfully worked were architectural rather than filter-based: randomizing identifiers (for example, ChatGPT's Atlas browser assigning unguessable random IDs to page elements instead of predictable sequential ones) cut web-agent attack success from roughly 49 percent to 29 percent, and strict data-provenance tracking—maintaining a verifiable record of which system, not which text pattern, produced each piece of context—drove attack success to zero but reduced task completion to roughly 36 percent of baseline, down from

an 86.5 percent baseline utility score [1][2]. Agent sandboxing with fine-grained action policies reduced success to about 22 percent but may require policy engineering effort that does not scale easily across arbitrary agent tasks [2]. No evaluated defense achieved both strong security and preserved usability—in CSA's assessment, this security/usability tradeoff is significant enough that organizations should treat ADI as an architectural gap to plan around, not a bug awaiting a patch.

Recommendations

Immediate Actions

Organizations operating web-browsing or coding agents in production should inventory where those agents currently trust system-generated metadata—element identifiers, sender or author fields, tool-response provenance markers, check or validation statuses—without independent verification, since this is the precise surface ADI exploits. Security teams should treat any agent capable of autonomous clicking, code execution, or merge approval as requiring the same level of scrutiny historically reserved for systems with direct production access, and should confirm with vendors (Anthropic, OpenAI, Google, and any third-party agent framework in use) what specific mitigations, if any, have been deployed since the July 2026 disclosure. Where agents operate against untrusted external content—web pages, incoming email, public repositories—human confirmation should be required before any consequential action (a purchase, a code execution, a merge) rather than relying on the agent's own assessment of authenticity.

Short-Term Mitigations

Teams building or operating agent infrastructure should prioritize the two defenses shown to be architecturally effective rather than filter-based approaches proven insufficient. Where feasible, agent frameworks should assign randomized, unguessable identifiers to page elements and structured-data fields rather than sequential or predictable ones, denying attackers the ability to pre-compute a forged ID that will collide with a legitimate target. Where agent actions carry meaningful consequence—code execution, financial transactions, merge approvals—provenance tracking that cryptographically or structurally separates system-trusted metadata from user-supplied content should be implemented even at a cost to task completion rate, since the alternative is an unbounded and largely invisible attack surface. Organizations should not rely on model hardening, prompt-based guardrails, or output filtering alone, as the research demonstrates these approaches do not generalize to metadata-layer attacks regardless of how well they perform against instruction-based injection [2].

Strategic Considerations

Agent data injection reframes a debate that has, until now, centered almost entirely on instruction-layer defenses. This echoes a lesson long established in traditional software security—that trusted control data and untrusted user data should not share an undifferentiated channel, as with SQL injection and other classic injection vulnerabilities—which AI agent architecture appears to have re-introduced; this parallel should inform how organizations evaluate agent frameworks going forward. Procurement and security review processes for agentic AI tools should explicitly ask vendors whether their architecture isolates trusted metadata from untrusted content at the data-model level, not merely whether their models have been hardened against injected instructions. As agentic systems take on higher-autonomy roles—approving code changes, executing financial transactions, managing infrastructure—the cost asymmetry the researchers identified, where effective defenses meaningfully reduce agent usefulness, will force explicit tradeoff decisions between agent autonomy and verifiable trust boundaries. Organizations should expect this tradeoff to become a standing agenda item in AI governance rather than a one-time technical fix, and should build monitoring capable of detecting anomalous metadata patterns (duplicate identifiers, unexpected authorship fields, malformed structural characters in unstructured fields) as a durable control rather than waiting for a definitive architectural fix to emerge industry-wide.

CSA Resource Alignment

CSA's Agentic AI Red Teaming Guide, developed with the AI Organizational Responsibilities Working Group and OWASP's AI Exchange initiative, already frames several of the vulnerability categories that agent data injection instantiates concretely, including agent memory and context manipulation, agent knowledge base poisoning, and agent hallucination exploitation [3]. Security teams using that guide to structure red team exercises against agentic systems should extend its context-manipulation test procedures to explicitly include metadata forgery—corrupted sender fields, duplicated element identifiers, and forged tool-provenance markers—since the guide's existing test categories anticipate this class of manipulation without yet naming the specific delimiter-injection technique this research documents.

CSA's RSAC 2025 presentation, "The AI Security Gap: Why Protecting Prompts Isn't Enough," argued more than a year before this disclosure that organizations focusing security investment on prompt- and instruction-layer defenses—prompt firewalls, response firewalls, retrieval firewalls—were leaving the broader data-governance layer of AI systems inadequately covered, and called specifically for investment in data access intelligence and runtime data governance [4]. Agent data injection can be read as a concrete, real-world instance of the gap that presentation identified: prompt-layer protections

did not extend to the metadata layer, exactly the blind spot the talk warned about. Organizations that adopted that presentation's recommendation to invest in data access intelligence and runtime data governance, rather than prompt filtering alone, are better positioned against this new attack class.

Finally, the AI Controls Matrix (AICM) v1.1, CSA's superset of the Cloud Controls Matrix extended for AI-specific risk, provides control domains directly applicable to this finding, particularly Application and Interface Security and Threat and Vulnerability Management [5]. Organizations building compliance and control frameworks around agentic AI deployments should map agent metadata-handling practices—data provenance verification, identifier randomization, and trust boundary enforcement between system-generated and user-generated content—to AICM's relevant control objectives rather than treating agent security as fully addressed by existing prompt-injection controls alone.

References

[1] The Hacker News. "New Agent Data Injection Attack Can Make AI Agents Misclick or Run Attacker Commands." The Hacker News, July 16, 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 6, 2026.

[3] Cloud Security Alliance. "Agentic AI Red Teaming Guide." CSA AI Organizational Responsibilities Working Group, 2025.

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

[5] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, 2026.