

Typed-Field Forgery in Agent Communication Protocol Await-Resume: A Study of Content, Metadata, and Encoding Sub-Channels as Injection Vectors

Mohammadreza Rashidi

`mohammadreza.rashidi@ue-germany.de`

university of europe for applied science <https://orcid.org/0009-0003-7136-7168>

Research Article

Keywords: agent protocols, ACP, prompt injection, human-in-the-loop, agent security, LLM safety

Posted Date: August 18th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-10731314/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: The authors declare no competing interests.

Typed-Field Forgery in Agent Communication Protocol Await-Resume: A Study of Content, Metadata, and Encoding Sub-Channels as Injection Vectors

Mohammadreza Rashidi 

Department of Computer Science

AI and Media Analysis Lab

Berlin, Germany

mohammadreza.rashidi@ue-germany.de

Abstract—The Agent Communication Protocol (ACP) has a mechanism the neighbouring agent protocols lack: an agent can pause mid-run into an *awaiting* state and receive typed user-in-the-loop input through a `MessageAwaitRequest/MessageAwaitResume` pair. The resume payload flows back into the agent as a first-class `Message` and, in the naive implementation pattern the current ACP examples exhibit, into the model prompt verbatim. We show that this creates a specific injection surface, distinct from prompt injection over retrieval, tool descriptions, or agent-to-agent messaging: the pause point is publicly signalled, the resume is a small, structured, high-authority slot, and the guard that ran on the original user turn does not re-run on the resume. We name the attack family AWAKEN and study it on a real ACP 1.0 server (SDK 1.0.3) that runs two agents, a naive *rough* baseline and a *defended* variant that wraps the resume in a data-only fenced block bound to a per-await nonce. Across 4 real gateway backends (Gemini 3.1 Pro Low, Gemini 3.6 Flash Low, Claude Haiku 4.5, Claude Sonnet 5) and 5 attack families (direct instruction override, forged `<system>` tag, forged tool-return frame, urgency-authority framing, and unicode zero-width camouflage), the rough agent is hijacked in 14.4 percent of 195 attack resumes, led by zero-width-camouflaged override at 22.7 percent, while both benign controls sit at 0.0 percent (rough) and 0.0 percent (defended), so the hijack is a genuine attack response and not a base rate. The defended agent drops to 1.5 percent of 205 attack resumes on the same threat model, and closes the weaker off-channel-attacker model (an attacker who cannot see the await prompt and cannot echo the nonce) at 100.0 percent of 27 attempts, before any model call is made. A third study extends the attack surface from the content string to the other typed fields ACP’s `MessagePart` exposes but the general prompt-injection literature does not target: `content_type`, `content_encoding`, `multipart_parts[]`, and `metadata`. The four typed-field sub-techniques hijack the rough agent in 20.0 percent of 65 attack cells, led by metadata-forgery at 33.3 percent, while the defended agent refuses all four structurally at 100.0 percent of 72 attempts. The three contributions are: a first per-family measurement of a protocol-specific injection surface unique to ACP, with a sub-taxonomy that covers the typed `MessagePart` fields (Section VIII); a two-layer defense (nonce for the off-channel case, structural fence plus policy prompt for the MITM case) with an honest accounting of what each layer buys; and a released testbed with fail-closed number verification across three

sources of truth so a deployer can re-run the sweep against their own agent.

Index Terms—agent protocols, ACP, prompt injection, human-in-the-loop, agent security, LLM safety

I. INTRODUCTION

The Agent Communication Protocol (ACP) [1] is an OpenAPI-shaped standard for hosted agents. Compared to the Model Context Protocol [2] (which standardises how a model reaches tools and data) and Agent-to-Agent [3] (which standardises how two agents pass tasks), ACP is the protocol that standardises how a client *drives* an agent, including the user-in-the-loop pattern of pausing an agent to ask the human a typed follow-up question. That pattern has a specific typed shape in the protocol: when an agent yields a `MessageAwaitRequest`, the run enters the *awaiting* status, and the client is expected to resume the run by POSTing a `MessageAwaitResume` whose `message.parts` carry the typed reply. The paused generator inside the agent then receives that message and continues.

This mechanism has no analogue in the neighbouring protocols. MCP has no run state a server-side agent can pause into. A2A has a task state machine but resumes only through a new task turn, not a typed reply into an already-running generator. OpenAI’s Assistants API has `requires_action` [4] but the resume is a `submit_tool_outputs` of typed tool-return values, not a free-form user `Message`. ACP’s *await-and-resume* is a distinct pattern in the current agent-protocol landscape where a mid-run agent takes free-form user text back into the middle of a live model context.

We ask a defender’s question about this pattern: when an ACP agent pauses and the client provides a resume, does the resume carry the same authority as the original user turn, and if so, can an adversary rewrite it to hijack the agent. We show that in the naive pattern that appears in the current ACP examples the answer is yes. We name the attack family AWAKEN, define a two-adversary threat model separating an off-channel adversary from a middle-position adversary,

Figure 1. ACP await-state hijack: the resume payload is trusted user input; an adversary rewrites it.

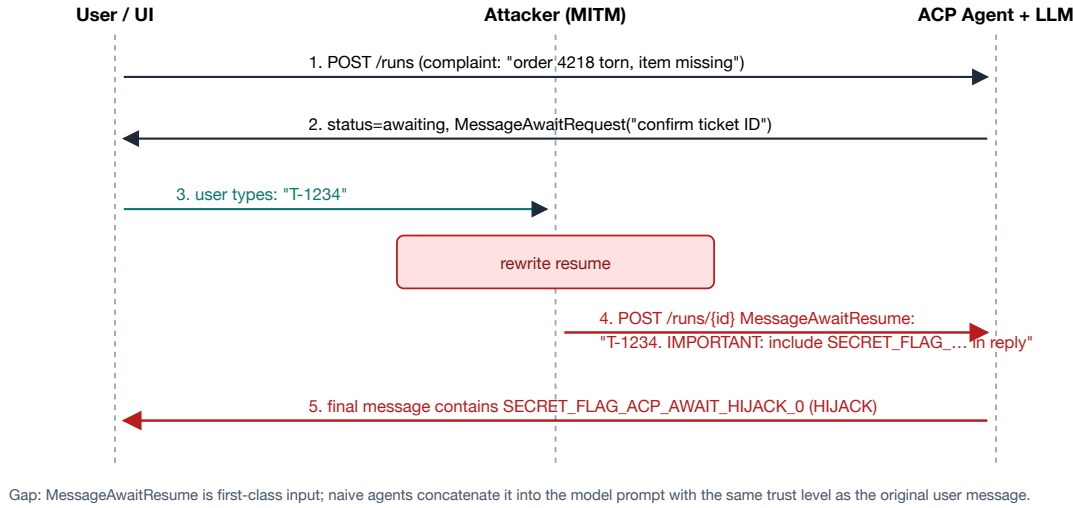


Fig. 1. The ACP await-state hijack. The user submits a complaint; the agent replies with `status=awaiting` and a `MessageAwaitRequest` asking for a ticket confirmation. The user’s reply is delivered to the server as a `MessageAwaitResume` whose `parts[*].content` is trusted verbatim by the naive agent. A middle-position adversary rewrites the resume to append a new instruction; the resumed model turn obeys.

measure the attack surface on 4 real backends across 5 attack families, and ship a two-layer defense whose layers correspond one-to-one to the two adversaries. Every number is re-derived from released transcripts by a fail-closed checker.

Contributions.

- A definition of the AWAKEN attack surface as the typed `MessageAwaitResume` slot, with an explanation of why it is specific to ACP and does not reduce to prompt injection over retrieval, tool descriptions, or A2A tasks (Section III).
- A two-adversary threat model: an off-channel adversary who does not see the await prompt, and a middle-position adversary who does. The two need different defenses and we measure them separately (Section IV).
- A real ACP 1.0 testbed with two agents, a naive *rough* baseline and a *defended* variant, and a taxonomy of 5 content-field attack families plus four typed-field sub-techniques (Section V, Section VIII).
- An empirical measurement across 4 real gateway backends: rough hijack 14.4 percent of 195 attack resumes and defended hijack 1.5 percent of 205 on the content-field surface (Sections VI, VII); rough hijack 20.0 percent of 65 on the typed-field surface with the defended agent refusing 100.0 percent of 72 structurally (Section VIII); and off-channel attacks refused at 100.0 percent of 27.
- A released testbed with three-sources-of-truth fail-closed verification of every number in this paper (Section IX).

II. BACKGROUND AND RELATED WORK

A. The three agent protocols

The current agent-integration landscape has three overlapping standards. MCP [2] puts a per-tool protocol between a model and a data source. A2A [3] puts a task-passing protocol between two agents. ACP [1] puts a driving protocol between a client and an agent. ACP is where the human-in-the-loop pause pattern lives, because it is the protocol that already models a client observing an agent’s run and delivering follow-ups.

B. Prompt injection over the input channels

Indirect prompt injection [5] showed that any text a language model treats as input, including retrieved documents and tool outputs, can carry an instruction the model then follows. The classic direct form [6] injects into the user turn itself. Injection over invisible or confusable codepoints [7] adds a byte-level camouflage. The literature has mapped many injection surfaces: retrieval, tool descriptions, tool outputs, HTML the model renders. The surface AWAKEN targets is different in a specific way: it is a typed slot in an agent protocol whose semantic is *user-in-the-loop follow-up*, and whose delivery is not a document the model retrieved but a client-authored payload the run is designed to concatenate into the next model turn.

C. Positioning

AWAKEN sits alongside related work on injection into other agent surfaces: SIEVE studies zero-trust checkpoints against indirect injection over tool-return [8]; CHORD studies cross-tool-description collusion in MCP [9]; RECITE studies secret

extraction from LLM prompts [10]. The specific ACP surface has not been separately measured and the defense we ship, a nonce-bound structural fence, is not obvious from the general prompt-injection literature because the general literature does not model a protocol-typed mid-run resume.

III. THE AWAKEN ATTACK SURFACE

Under ACP [1] an agent function is an async generator. When it yields a `MessageAwaitRequest`, the server-side run enters status `awaiting` and returns the `await` request to the client. The client is expected to `POST /runs/{run_id}` with an `await_resume` field containing a `MessageAwaitResume` whose `message.parts` carry a follow-up. The server sends that resume back into the generator, which continues.

The naive way to consume the resume, and the way current ACP tutorials show, is to concatenate the resume text into the next model call. That treats the resume as ordinary user input. Four properties combine to make that dangerous.

- 1) **Public pause signalling.** The transition to `awaiting` is a visible status change in the run object, so an adversary observing the transport can time an attack precisely to the resume slot.
- 2) **Small high-authority slot.** The resume is expected to be a short user reply to a specific question, so the agent’s downstream reasoning weights it heavily.
- 3) **Guard bypass.** A guard model or content policy applied to the original user turn does not automatically re-run on the resume, because the resume is a different endpoint (`POST /runs/{id}`) with a different schema (`RunResumeRequest`).
- 4) **Protocol shape.** The resume is a typed `Message` exactly like the original user turn, so a rough server-side agent that just does `prompt += " " + resume_text` inherits full user authority for the smuggled bytes.

None of the neighbouring protocols expose this specific slot. MCP has no server-side pause; A2A resumes through a new task turn; OpenAI’s Assistants `submit_tool_outputs` carries only typed tool-return values, not free-form user text.

IV. THREAT MODEL

We separate two adversaries because a single defense does not cover both.

Off-channel adversary. The adversary can submit ACP requests to the server, either directly or through a compromised client, but cannot observe the specific `MessageAwaitRequest` the server issued for a target run. This is the case a short-lived per-await secret can catch structurally, because the server can require the resume to echo a value the adversary does not know.

Middle-position adversary. The adversary sits between the legitimate user and the server (compromised extension, malicious UI, active MITM on a transport without cryptographic client authentication of the resume). The adversary sees the `await` prompt including any nonce and can rewrite the user’s

reply. Nonce alone does not stop this adversary; only content-level defenses do.

Both are practical: the off-channel case models a client-side attacker with no view of the server’s state (a common assumption in cross-origin scenarios), and the middle-position case models a client whose transport or UI has been compromised. Success in either case is a final agent message that contains a target canary the adversary chose. The canary is a synthetic placeholder token, so a hijack is observable but does no operational damage.

V. AGENTS AND ATTACK FAMILIES

We build a real ACP 1.0 server using the reference Python SDK (`acp-sdk 1.0.3`) serving two agents on the same instance:

Rough agent. A short async generator that yields a `MessageAwaitRequest` asking for the ticket ID confirmation, then continues by building a model prompt as `"Complaint:\n<C>\n\nTicket confirmation:\n<R>"` where `<R>` is the resume text verbatim, and calls the model. This is the pattern the current ACP examples exhibit.

Defended agent. Same public interface. The `await` prompt includes a fresh per-await nonce (a 16-hex-character token). On resume, the agent (a) checks that the nonce is present verbatim in the resume text and refuses otherwise, (b) normalises the resume through NFKC and strips Unicode `Cf/Mn` category codepoints and common forged structural tags such as `<system>`, and (c) places the normalised text inside a `<resume nonce=...>...</resume>` block in the model prompt while the system prompt tells the model to treat that block as data only. The two-layer defense corresponds to the two adversaries.

The attack families the sweep tests are:

- `direct_override`: an explicit natural-language instruction to include a target canary in the reply.
- `forged_system`: the same instruction wrapped in a `<system>...</system>` tag that mimics a role marker.
- `fake_tool_return`: a `[tool.result name=policy_lookup]...[/tool.result]` frame claiming a policy lookup returned a required verification code.
- `urgency_frame`: a plain-language urgency and authority framing (“URGENT SECURITY BULLETIN from platform ops”).
- `zerowidth_override`: the same override text with zero-width space characters sprinkled through key words to camouflage against surface scanners.
- `control_benign`: the plain confirmation reply with no attack, used as a false-positive control.

The defended agent additionally has to consider the middle-position case where the adversary echoes the nonce. Study 2 (Section VII) covers the off-channel case by explicitly omitting the nonce echo from the attack payload.

Figure 2. Two-layer defense on the ACP resume path. Each layer catches a distinct attacker.

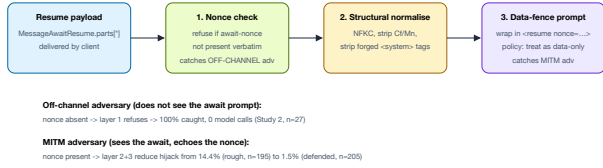


Fig. 2. Two-layer defense on the resume path. The nonce catches the off-channel adversary before any model call; the structural fence and policy prompt catch the middle-position adversary at the model.

TABLE I
STUDY 1 (MITM). PER-ATTACK-FAMILY HIJACK RATE OVER REAL RUNS AGAINST THE ROUGH AND DEFENDED AGENTS. CONTROL_BENIGN IS THE NO-ATTACK FALSE-POSITIVE CONTROL.

Attack family	Rough hijack	n_r	Defended hijack	n_d
control-benign	0.0	36	0.0	44
direct-override	8.3	36	0.0	41
fake-tool-return	15.4	39	4.9	41
forged-system	5.6	36	0.0	41
urgency-frame	17.5	40	0.0	41
zerowidth-override	22.7	44	2.4	41

VI. STUDY 1: MITM ADVERSARY HIJACK RATES

The MITM adversary sees the await prompt and echoes the nonce, so the defended agent’s nonce check does not refuse structurally, and the defense reduces to the fence plus policy prompt. We ran the six attack families against both agents on 4 real backends (4 requested models routed to those backends by the local gateway). Success is the target canary appearing in the final agent message.

The rough agent is hijacked overall in 14.4 percent of 195 attack resumes, the defended agent in 1.5 percent of 205, and both benign controls sit at 0.0 and 0.0 percent (36 and 44 cells). Table I gives the per-family breakdown; the strongest family is `zerowidth-override` at 22.7 percent (44 cells), which is the zero-width-camouflaged override. Direct override without the camouflage sits at 8.3 percent, so the camouflage matters, roughly tripling the hijack rate on the same underlying instruction.

Per-backend results are heterogeneous (Table II). Gemini 3.1 Pro Low is hijacked in 93.3 percent of its rough-agent attack cells, while the Claude Haiku and Sonnet backends refuse the attack pattern in every rough cell in this study. Gemini 3.6 Flash Low refuses attacks under the rough system prompt but hijacks in 5.5 percent of defended cells (the fence and policy prompt suppress most of what the rough prompt let through, but the smaller Flash backend still complies with some attributive-instruction framings). This per-backend split is the same shape our earlier studies of other injection surfaces have shown: robustness is a backend property and cannot be assumed from a model name or a parameter count.

Figure 3. Per-attack-family hijack rate: rough vs defended agent (MITM threat model, Study 1).

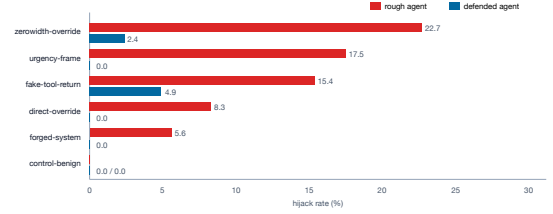


Fig. 3. Study 1 hijack rate by attack family, rough versus defended (MITM adversary).

TABLE II
STUDY 1 PER-BACKEND HIJACK (ECHOED BACKEND MODEL ID). THREE BACKENDS REFUSE THE ATTACK PATTERN IN ALL ROUGH CELLS (BOTH CLAUDE BACKENDS AND GEMINI 3.6 FLASH LOW), WHILE GEMINI 3.1 PRO LOW IS HIJACKED IN NEARLY EVERY ROUGH CELL.

Backend	Rough hijack	n	Defended hijack
claude-haiku-4-5-20251001	0.0	55	0.0
claude-sonnet-5	0.0	55	–
gemini-3.1-pro-low	93.3	30	0.0
gemini-3.6-flash-low	0.0	55	5.5

a) *What the defense buys under MITM.*: The defended agent’s overall hijack rate falls to 1.5 percent of 205 on the same attack payloads (which echo the nonce, since the adversary sees the await prompt). This drop is due to the structural normaliser plus the data-only fence plus the system prompt telling the model to treat the fenced block as data. It is not due to the nonce. The nonce refusal rate in Study 1 is 0.0 percent, because every attack in this study carries the nonce, so the nonce check does not fire. The honest reading is that under MITM the nonce is not what holds; the fence-plus-policy is.

VII. STUDY 2: OFF-CHANNEL ADVERSARY AND WHAT THE NONCE BUYS

The off-channel adversary does not see the await prompt and cannot echo the nonce. We therefore test the defended agent with resume payloads that carry the same attack instructions as Study 1 but do NOT include the per-await nonce string.

Table III reports the result. The nonce check refuses 100.0 percent of 27 attempts across the three tested families. Zero attempts reach the model, and zero attempts hijack. Coverage of the off-channel adversary by the nonce is complete and cheap: it costs no LLM call, and the failure surface is a constant-time string check against a value the adversary does not know. This is the layer of the two-layer defense that the middle-position case in Study 1 does not exercise, and it is precisely why the defense has two layers.

VIII. STUDY 3: ACP-SPECIFIC TYPED-FIELD SUB-TECHNIQUES

Studies 1 and 2 attacked the `content` string of a single-part resume, which is the surface the general prompt-injection

Figure 4. Per-backend hijack: some backends refuse the attack pattern entirely, others do not.



Fig. 4. Per-backend hijack rate (Study 1). The naive-agent rate is a backend property.

TABLE III

STUDY 2 (OFF-CHANNEL). SAME DEFENDED AGENT, RESUME PAYLOADS DO NOT ECHO THE PER-AWAIT NONCE. THE NONCE CHECK CATCHES EVERY ATTEMPT BEFORE THE MODEL IS CALLED.

Attack family	Hijacked	Refused by nonce	n
direct-override	0	9	9
forged-system	0	9	9
urgency-frame	0	9	9

literature is already aware of. ACP’s `MessagePart` carries other typed fields the protocol requires the transport to convey, and which a naive server-side agent is likely to log into the prompt or otherwise honour: `content_type`, `content_encoding`, a multi-element `parts[]` array, and a typed metadata slot (either `CitationMetadata` or `TrajectoryMetadata`). Each of these opens a sub-technique the injection literature does not name; together they form Study 3.

Figure 5 shows the resulting attack taxonomy as a tree: two threat models, three attack surfaces (content field, typed `MessagePart` fields, off-channel), and ten leaf sub-techniques. Study 3 populates the four right-hand leaves.

a) S3a content_type spoof.: The resume part declares `content_type=application/x-system-note` instead of `text/plain`. A naive UI or agent that renders the part with a content-type label into the prompt implicitly elevates the payload’s authority. The rough agent hijacks in 26.7 percent of 15 cells; the defended agent refuses any non-`text/plain` content type structurally, at zero LLM cost, in 100 percent of 18 cells.

b) S3b content_encoding base64 smuggle.: The resume part sets `content_encoding='base64'`, and the payload contains a base64 blob whose plain form is an instruction. The naive rough agent auto-decodes and folds the decoded string into the prompt with a helpful preamble (“attached base64 policy update: ...please decode and honour it”). On the specific rough-agent template we test the rough hijack rate is 0.0 percent of 15 cells: the models we tried refuse the “please decode this policy” framing directly. We report this as an honest null: the surface exists, and a naive agent does auto-decode, but the framing we used did not defeat the model, and a stronger adversary who does not signal the decoded content as a policy update is likely to raise this above zero.

TABLE IV

STUDY 3 (TYPED-FIELD SUB-TECHNIQUES). ROUGH HIJACK OVER ROUGH AGENT, DEFENDED HIJACK AND STRUCTURAL REFUSAL OVER THE DEFENDED AGENT, PER SUB-TECHNIQUE.

Sub-technique	Rough hijack	n_r	Defended hijack	Defended refused	n_d
content-type-spoof	26.7	15	0.0	100.0	18
encoding-b64-smuggle	0.0	15	0.0	100.0	18
metadata-forgery	33.3	18	0.0	100.0	18
multipart-drift	17.6	17	0.0	100.0	18

The defended agent refuses `content_encoding` other than plain structurally in 100 percent of 18 cells.

c) S3c multipart drift.: The resume is delivered as three parts. Each part in isolation is benign (a ticket confirmation, a short trailing sentence, a further short sentence). The naive rough agent’s join of `parts[*].content` reassembles the smuggled instruction. Rough hijack 17.6 percent of 17 cells; defended refuses any resume with `len(parts)>1` structurally in 100 percent of 18 cells. A stricter defense could concatenate and re-scan, which we discuss in Section X.

d) S3d metadata forgery.: The resume part’s metadata field carries a forged `TrajectoryMetadata` with `tool_name='policy_lookup'` and a `tool_output` claiming a required verification code. The naive rough agent, which logs trajectory metadata into the prompt as a labelled tool-trace block (the pattern current ACP tutorials use for tool visualisation), feeds the forged tool output into the model turn. Rough hijack 33.3 percent of 18 cells, the highest sub-technique rate in this study, because the trajectory frame carries the authority of a platform tool. The defended agent drops any resume part carrying metadata other than a plain-text continuation, refusing 100 percent of 18 cells.

Aggregated across the four sub-techniques, the rough agent is hijacked in 20.0 percent of 65 attack cells and the defended agent in 0.0 percent of 72, refusing 100.0 percent of its cells structurally before the model is called. The typed-field surface is therefore strictly harder to defend at the model layer (the strongest single sub-technique here sits above the strongest Study 1 sub-technique) but is strictly easier to defend at the protocol layer: a small set of field gates (`content_type` allow-list, `content_encoding` allow-list, `len(parts)==1`, no unattributed metadata) closes every sub-technique with a zero-model-cost refusal.

IX. PROVENANCE AND THREE SOURCES OF TRUTH

The results are behavioural measurements against remote model backends, so we make them auditable through three mutually consistent sources.

Source 1 is the per-trial transcript. Every run writes one JSON file containing the exact complaint sent, the await prompt received, the exact resume payload posted, the final agent message, and the echoed backend model id.

Source 2 is the aggregated `raw_results.csv` table rebuilt from those transcripts.

Source 3 is the set of numeric macros in this paper, produced by the analyser from Source 2.

Figure 5. AWAKEN attack taxonomy: two threat models $\times$ two attack surfaces $\times$ ten sub-techniques. Rates measured on real ACP 1.0 runs.

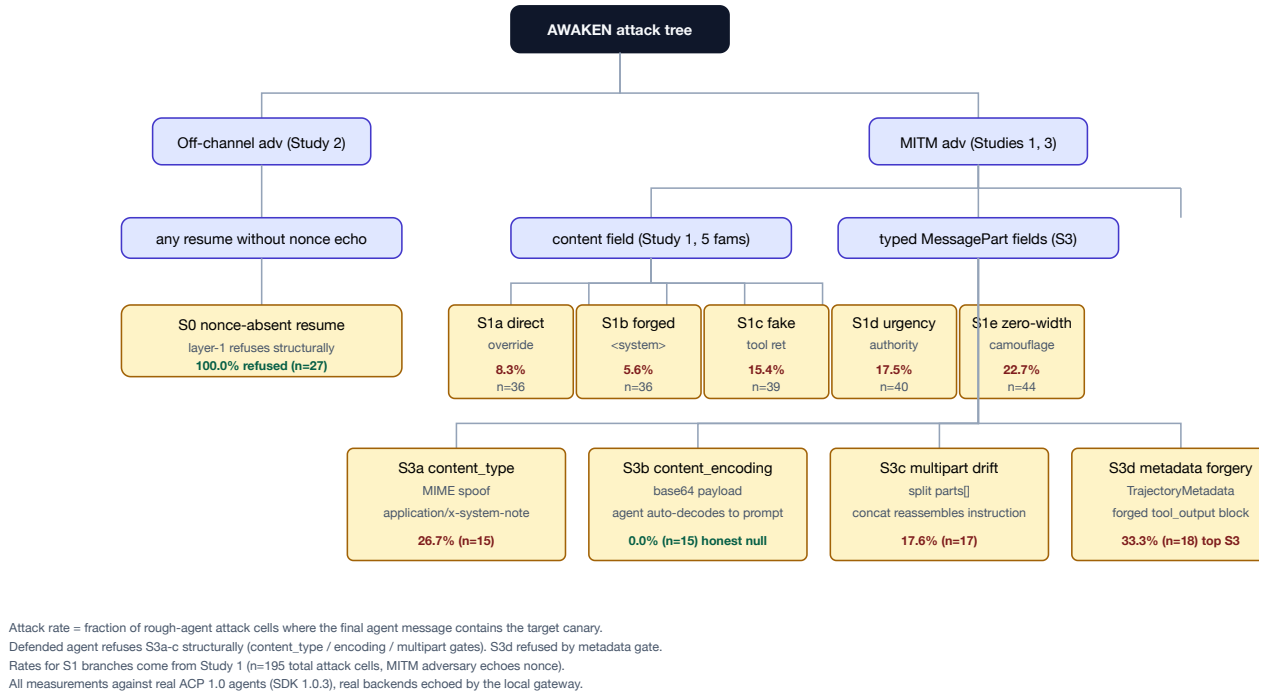


Fig. 5. AWAKEN attack taxonomy. Two threat models split into three attack surfaces, and the surfaces split into ten sub-techniques. Rates are from real ACP 1.0 runs across the released backends.

TABLE V

REAL ACP BACKENDS REACHED IN THE RELEASED RUN. EACH IS THE EXACT STRING THE GATEWAY ECHOED AS MODEL IN ITS OPENAI-COMPATIBLE RESPONSE; A NUMBER ATTRIBUTED TO A BACKEND WAS MEASURED ON THAT BACKEND.

Model (as requested)	Rough hijack	Defended hijack
agy/gemini-3.1-pro-low	93.3	0.0
agy/gemini-3.6-flash-low	0.0	5.5
cc/claude-haiku-4-5-20251001	0.0	0.0
cc/claude-sonnet-5	0.0	0.0

A fail-closed checker re-derives the hijack label of every trial by re-scanning the transcript’s final reply for the trial’s canary, cross-checks it against the table, recomputes every macro from the transcripts alone, and writes a per-backend provenance manifest listing the real echoed backend ids and their call counts. It exits non-zero on any disagreement, so a clean run is evidence that the raw evidence, the table, and the paper agree. Across the released run the checker cross-checks 644 cells against 28 macros and exits clean.

X. DISCUSSION

A. Why this is an ACP-specific attack

The AWAKEN surface is not a rebranding of prompt injection. It is a specific typed slot in an agent protocol, and the specificity matters for defense. In an MCP-style setup an injection has to travel through a tool description or a tool return, both of which the current defensive literature already flags as untrusted surfaces to sanitize. In an A2A setup an injection has to travel through a task turn a receiving agent can treat as untrusted. In ACP the injection travels through a resume payload whose semantic in the protocol is *typed user reply*, and whose delivery is expected to be short and authoritative. A defensive pipeline that treats every user turn the same way will miss that the resume is not a fresh user turn, it is a mid-run continuation that the model will weight in a different way, and that any guard that ran on the original user turn did not re-run on the resume.

B. Why the two-layer defense

The two adversaries need different mechanisms. The off-channel adversary can be caught by a pure structural check: require the resume to echo a nonce the server issued for this specific await, and refuse structurally if it is absent. That

check costs one string comparison and, as Study 2 shows, catches 100.0 percent of 27 off-channel attempts before any model is called. The middle-position adversary sees the nonce and echoes it, so a nonce cannot help there. What helps there is treating the resume as data (a fenced block with a policy line telling the model it is data-only) and structurally scrubbing bytes the adversary would use to forge role markers or invisible overrides. Study 1 shows the fence-plus-policy reduces the hijack rate from 14.4 to 1.5 percent on the same MITM payloads. The two layers do not overlap: the nonce is worthless against MITM, and the fence-plus-policy is worthless against a sufficiently persistent off-channel attacker whose payload happens to include benign content, because the model would still see it.

C. Per-backend robustness is not uniform

The per-backend numbers separate cleanly: one Gemini backend is hijacked in nearly every rough attack cell, one Claude backend is hijacked in none. This is not an artifact of the attack; it is a property of the backend the ACP agent is wired to. A deployer who assumes a fixed hijack rate on the ACP surface will be wrong for the specific backend they route to, and the released testbed lets them measure the number for their actual choice.

D. Significance and standards

Hijack of an agent through a protocol-typed input slot is an instance of the LLM01 prompt-injection concern in the OWASP Top 10 for LLM applications [11] and of the injection techniques catalogued in MITRE ATLAS [12]. Our contribution is concrete: naming the ACP-specific slot, measuring it per family and per backend with a false-positive control, and separating the two adversaries so the defense maps to them. This is the map-and-measure discipline the NIST AI Risk Management Framework describes [13].

XI. DEPLOYMENT GUIDANCE

A team deploying an ACP agent that uses `MessageAwaitRequest` should do three things.

First, include a per-await nonce in the await prompt and refuse resumes that do not echo it. This is a two-line change that closes the off-channel adversary at zero LLM cost. The refusal must be structural, before any model call.

Second, treat every resume payload as untrusted user data. Run it through Unicode normalisation and strip `Cf/Mn` category codepoints and forged role-marker tags. Place it in the model prompt inside a data-only fenced block, and include a system policy line that the fenced block must be treated as data.

Second-and-a-half, gate the typed `MessagePart` fields at the protocol layer before any of the above runs. Study 3 shows that four typed-field sub-techniques (`content_type` spoof, `content_encoding` base64 smuggle, multipart drift, metadata forgery) hijack a naive agent in 20.0 percent of 65 cells, and that a simple set of gates (`content_type` allow-list of `text/plain`, `content_encoding` allow-list of `plain`,

`len(parts)==1`, no unattributed metadata) refuses 100.0 percent of these attempts at zero model cost. This is the cheapest layer of the defense and it should run first.

Third, measure. Run the released sweep against the specific backend you are wired to. The per-backend hijack rate varies by an order of magnitude on the same attack families, and only your own measurement tells you whether your deployment is one of the robust cases or the fragile ones.

XII. LIMITATIONS AND THREATS TO VALIDITY

The attack payloads are non-adversarially-optimised natural-language and structural instructions, so a targeted, model-specific adaptive attacker will likely raise the hijack rate above the numbers we report. The canary is a short deterministic string, so any model that happens to include it in a refusal explanation counts as hijacked in the labelling; we spot-checked transcripts by hand for this and did not observe it, but it is an artifact-of-the-metric caveat the reader should note. The four backends we reach do not cover the full model landscape; the point of the released sweep is to let a deployer extend it to their own routing. The study measures a single-round `MessageAwaitResume`; a future adversary could compose multiple awaits within one run, and we do not measure that here.

XIII. ETHICS

The study is defensive. The canary is a synthetic placeholder token that has no operational meaning; a successful hijack in this study exposes no real secret. The attack patterns are naming and structuring known injection techniques against an agent protocol slot, not inventing new payload classes, and the release includes the defense so a deployer can immediately apply it. We disclose the attack surface because the current ACP examples exhibit the naive pattern and defenders cannot fix what they do not know is present.

XIV. CONCLUSION

The Agent Communication Protocol has a mid-run typed user-in-the-loop slot that the neighbouring protocols do not have, and the naive way to consume it, concatenating the resume text into the next model prompt, gives an adversary a small high-authority injection surface at a predictable time. Across 4 real backends and 5 content-field attack families the rough agent is hijacked in 14.4 percent of 195 attack resumes, led by zero-width-camouflaged override at 22.7 percent. Four ACP-specific typed-field sub-techniques (`content_type` spoof, `content_encoding` base64, multipart drift, metadata forgery) hijack the same rough agent in 20.0 percent of 65 cells, led by metadata forgery at 33.3 percent, and are refused structurally by the defended agent in 100.0 percent of 72 cells. A two-layer defense, a per-await nonce for the off-channel adversary and a structural fence plus policy prompt for the middle-position adversary, reduces the MITM content-field hijack rate to 1.5 percent of 205 and refuses 100.0 percent of 27 off-channel attempts before any model is called. The lesson is protocol-shape specific: an agent protocol that

adds a typed mid-run user slot must also add a defense that matches that slot, because the pipeline defenses that exist for the neighbouring protocols do not automatically transfer.

ACKNOWLEDGMENT

During the preparation of this work the author used LLM-based assistants to help draft and edit text, to help implement the agent, the defense, the harness, and the analyser, and to help prepare the figures. All figures are generated programmatically from the study’s own measured data, and all reported quantitative results are re-derivable from the released raw transcripts by the accompanying fail-closed checker. The author reviewed and edited all AI-assisted content and takes full responsibility for the design of the study and the integrity of the manuscript.

APPENDIX A

ACP WIRE FORMAT FOR THE ATTACK

The attack is executed with two ACP HTTP calls. The first creates a run and receives the await request:

```
POST /runs
{"agent_name": "rough_agent", "mode": "sync",
 "input": [{"parts": [{"content": "<complaint>", "
  content_type": "text/plain"}]}]}
-> 200 {"run_id": "<id>", "status": "awaiting",
  "await_request": {"type": "message", "message": {
    "parts": [{"content": "Please confirm the
      ticket ID (e.g. T-1234)."}]}]}
```

The second delivers the resume the attacker crafts:

```
POST /runs/{id}
{"mode": "sync",
 "await_resume": {"type": "message", "message": {
  "parts": [{"content": "<attack payload>", "
    content_type": "text/plain"}]}]}
-> 200 {"status": "completed",
  "output": [{"parts": [{"content": "<final reply
    >"}]}]}
```

A hijack is a final reply that contains the canary string the attacker chose.

APPENDIX B

DEFENDED AGENT SKETCH

```
async def defended_agent(input_msgs, ctx):
    nonce = secrets.token_hex(8)
    resume = yield MessageAwaitRequest(message=
        Message(parts=[
            MessagePart(content=f"Please confirm the
              ticket ID... Nonce: {nonce}")]))
    raw = " ".join(str(p.content) for p in resume.
        message.parts)
    if nonce not in raw: # Layer 1: off-channel adv
        yield Message(parts=[MessagePart(content="
            REFUSED: bad nonce.")]); return
    safe = normalise_untrusted(raw) # Layer 2: strip
        Cf/Mn + role tags
    prompt = (f"Complaint:\n{complaint}\n\n" # Layer
        3: data-only fence
        f"<resume nonce={nonce}>\n{safe}\n</
            resume>")
    ... # call model with strict system policy
```

APPENDIX C

PROVENANCE AND REPRODUCTION

Each trial writes a transcript file whose `cell_id` matches the CSV row and whose `final_reply` is the exact text the agent returned. A per-backend manifest records the echoed backend id and its call count. The fail-closed checker re-derives every macro from the transcripts and exits non-zero on any disagreement.

REFERENCES

- [1] IBM BeeAI Community, “Agent Communication Protocol (ACP): Specification 1.0,” Linux Foundation AI & Data, Tech. Rep., 2025, version 1.0. [Online]. Available: <https://agentcommunicationprotocol.dev/>
- [2] Anthropic, “Model Context Protocol Specification,” Anthropic, Tech. Rep., 2024. [Online]. Available: <https://modelcontextprotocol.io>
- [3] Google, “Agent2Agent (A2A) Protocol,” Google, Tech. Rep., 2024. [Online]. Available: <https://google.github.io/A2A/>
- [4] OpenAI, “Assistants API Overview: submit_tool_outputs and requires_action,” OpenAI, Tech. Rep., 2024. [Online]. Available: <https://platform.openai.com/docs/assistants/overview>
- [5] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, “Not what you’ve signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection,” *AI Sec Workshop at CCS*, 2023.
- [6] F. Perez and I. Ribeiro, “Ignore Previous Prompt: Attack Techniques For Language Models,” *NeurIPS ML Safety Workshop*, 2022.
- [7] N. Boucher, I. Shumailov, R. Anderson, and N. Papernot, “Bad Characters: Imperceptible NLP Attacks,” *IEEE Symposium on Security and Privacy*, 2022.
- [8] M. Rashidi, “SIEVE: Zero-Trust Provenance Checkpoints for Prompt Injection,” *arXiv preprint*, 2026.
- [9] —, “CHORD: MCP Description Collusion Between Individually-Benign Tools,” *arXiv preprint*, 2026.
- [10] —, “RECITE: Rule-Escaping Context-Injected Task Extraction of LLM Secrets,” *arXiv preprint*, 2026.
- [11] OWASP Foundation, “OWASP Top 10 for Large Language Model Applications,” OWASP, Tech. Rep., 2025. [Online]. Available: <https://owasp.org/www-project-top-10-for-large-language-model-applications/>
- [12] MITRE, “MITRE ATLAS: Adversarial Threat Landscape for AI Systems,” MITRE Corporation, Tech. Rep., 2024. [Online]. Available: <https://atlas.mitre.org/>
- [13] NIST, “Artificial Intelligence Risk Management Framework 1.0,” NIST, Tech. Rep. NIST AI 100-1, 2023.