Independent Submission
Internet-Draft
Intended status: Experimental
Expires: 4 December 2026

D. Samal
Independent

2 June 2026

Verifiable Agent Protocol (VAP): Intent-Bound Admission Control and Audit for Agent Tool Invocation
draft-samal-vap-00

Abstract

This document specifies the Verifiable Agent Protocol (VAP), a thin, tool-agnostic verification layer for protocols in which an autonomous agent (a client driven by a large language model) invokes tools exposed by a server (for example, the Model Context Protocol, MCP). Existing tool-invocation protocols convey WHAT tool is to be run and, with authorization extensions, WHO is calling, but carry no machine-verifiable statement of WHY a call is being made. VAP adds a declared, structured, optionally signed statement of purpose and a stable per-session scope-and-budget commitment, against which a server performs admission control before executing a call. VAP defines four messages -- Scope Commitment, Intent Envelope, Scope Amendment, and Verdict -- carried inside the host protocol’s existing metadata channel, requiring no change to that protocol and no change to existing servers. VAP targets erroneous and runaway agent behavior, session cost control, and audit; it is defense-in-depth for, not a replacement for, authentication and authorization.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 4 December 2026.

Table of Contents

1. Introduction

Tool-invocation protocols for LLM-driven agents, most prominently the Model Context Protocol [MCP], expose a server's capabilities to a client that decides, at runtime, which tools to call. The decision is made by a language model. The request conveys the tool name and arguments; an authorization extension may convey the caller's identity and granted scopes. The request does not convey, in any machine-verifiable form, the caller's PURPOSE: the goal it is pursuing and the reason this particular call serves that goal.

Because purpose is absent from the wire, a server cannot determine whether a syntactically valid, authorized call is consistent with what the agent is actually trying to do. A confused, mis-prompted, or compromised agent can therefore issue a sequence of individually valid but collectively incoherent calls -- exhausting budget, causing unintended side effects, and producing logs that record WHAT happened but not WHY.

VAP addresses this by making purpose a first-class, declared, and verifiable element of a session, and by giving the server an admission-control procedure that evaluates each call against that declared purpose before execution. VAP is deliberately thin: it reuses the host protocol's transport, identity, and metadata facilities, and it confines itself to tool IDENTITY (the public tool namespace) and universal resource BUDGET. It never encodes a tool's argument semantics; argument-level rules are deployment-local operator policy (Section 9).

VAP is defense-in-depth. Declared purpose is attacker-controllable text; VAP raises the cost of, and the auditability of, incoherent behavior, but MUST NOT be relied upon as an authentication or authorization mechanism.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Agent (Client): The LLM-driven party that initiates tool calls.

Server: The party that exposes tools and executes them.

Verifier: The component that performs VAP admission control. It MAY be co-located with the Server, or deployed as a reverse proxy in front of it (the RECOMMENDED deployment, as it requires no Server change).

Scope Commitment: A per-session declaration of goal, allowed tool namespace, and budget (Section 5).

Intent Envelope: A per-call declaration of the rationale for a single tool call (Section 6).

Scope Amendment: A signed re-baselining of an active Scope Commitment (Section 7).

Verdict: The Verifier’s decision and its justification (Section 8).

Drift: A requested call that escapes the active Scope Commitment’s tool namespace or budget. Drift is defined as escape of the committed envelope, NOT as deviation from a predicted plan.

Meter: A named, additive consumption counter (e.g. "tokens", "usd_opcost") with a ceiling (Section 11).

3. Architecture and Scope

VAP separates two concerns that are frequently conflated:

* Tool IDENTITY -- the name a server advertises for a tool. This is public (it appears in the host protocol’s tool listing) and is the ONLY tool attribute VAP gates on.

* Tool INTERNALS -- the meaning of a tool’s arguments and effects. VAP does NOT encode these. Operators who need argument-level constraints express them in a deployment-local policy hook (Section 9), outside this specification.

This boundary keeps VAP universal: it applies identically to a payments tool and to a read-only search tool, and a Verifier never needs to understand any tool’s domain.

In scope: declared purpose, per-session scope/budget commitment, pre-execution admission control, intent-bound audit.

Out of scope: authentication, authorization, tool/server attestation, transport security, and tool argument semantics. VAP composes with, and relies upon, separate mechanisms for these.

4. Protocol Overview

A VAP session has two tiers:

* Session tier. Once, at session initialization, the Agent sends a Scope Commitment (Section 5). The Verifier MAY perform semantic verification here and, if it accepts, caches the commitment and returns its digest in a Verdict.

* Call tier. For each tool call, the Agent sends an Intent Envelope (Section 6) referencing the session. The Verifier performs deterministic checks against the cached commitment and, gated by a risk score (Section 10), MAY perform semantic verification. It returns a Verdict (Section 8) and, if served, the tool result.

When legitimate re-planning requires widening the committed envelope, the Agent sends a Scope Amendment (Section 7), which the Verifier re-verifies and which re-baselines the commitment. Silent escape of the committed envelope is treated as Drift and denied.

All VAP messages are carried inside the host protocol's metadata channel (Section 12) and are therefore transparent to host-protocol parties that do not implement VAP.

5. Message: Scope Commitment

Sent once per session. Fields:

* vap (string, REQUIRED): protocol version, "0.1".
* type (string, REQUIRED): "scope_commitment".
* session_id (string, REQUIRED): unique session identifier.
* goal (string, REQUIRED): natural-language session purpose; the anchor for semantic verification.
* scope (object, REQUIRED): tools_allow (array of string, REQUIRED) is a set of public tool-name patterns permitted this session, with "*" glob permitted; tools_deny (array of string, OPTIONAL) is a set of deny patterns, evaluated after tools_allow, with deny winning on conflict.
* budget (object, REQUIRED): see Section 11. At least one of max_calls, deadline, or limits MUST be present.
* plan_digest (string, OPTIONAL): hash of the Agent's current plan, opaque to the Verifier; used for verdict caching and re-plan detection. No plan content is disclosed.
* principal (object, REQUIRED): agent_id (RECOMMENDED a DID), and OPTIONAL auth reference and delegated_by.
* signature (string, OPTIONAL): JWS [RFC7515] over the JCS-canonicalized [RFC8785] message. REQUIRED if the Verifier advertises require_signed.

The normative JSON Schema is in [VAP-SCHEMAS].

6. Message: Intent Envelope

Sent per tool call. Fields:

* vap (REQUIRED): "0.1".

* type (REQUIRED): "intent_call".

* session_id (REQUIRED): MUST reference an accepted Scope Commitment.

* intent (object, REQUIRED): rationale (string, REQUIRED) states why this call serves the goal; expected_effect (string, REQUIRED) states the bounded outcome expected; step (integer, OPTIONAL) is an advisory plan position and Drift is NOT defined relative to this value; sensitivity (string, OPTIONAL) is a self-declared risk class, one of "reads", "writes_data", "writes_money", "deletes", "sends_external", "grants_access", which the Verifier MAY override; reasoning_digest (string, OPTIONAL) is a hash of the Agent's full reasoning, retained client-side, enabling later audit without transmitting the reasoning itself.

* call (object, REQUIRED): tool (string) and arguments (object). The Verifier checks tool against the committed namespace. Argument-level constraints, if any, are applied by deployment-local policy (Section 9), not by this protocol.

* signature (OPTIONAL): as in Section 5.

The Agent MUST NOT transmit raw chain-of-thought in the Intent Envelope. A minimal, structured intent claim is REQUIRED; full reasoning, if retained, is referenced by reasoning_digest only.

7. Message: Scope Amendment

Sent to widen an active commitment during legitimate re-planning. Fields: vap, type ("scope_amendment"), session_id, prev_commitment_digest (REQUIRED, forming a tamper-evident chain), reason (REQUIRED), and at least one of add_scope (tools_allow / tools_deny only) or increase_budget (add_calls, extend_deadline, add_limits). An OPTIONAL new_plan_digest and signature MAY be present; the Verifier SHOULD require amendments to be signed because they alter the trust envelope.

The Verifier MUST re-verify an amendment (Section 9, step S2) and MAY accept, downgrade, deny, or require human approval. An amendment is the ONLY legitimate means of widening scope or budget.

8. Message: Verdict

The Verifier's response to any VAP message. Fields: vap, type ("verdict"), session_id, in_response_to, and:

* verdict (string, REQUIRED): "served", "clarify", "downgraded", or "denied".

* result (object, OPTIONAL): the tool output, present when an intent_call is served.

* accepted_commitment_digest (string, OPTIONAL): the commitment the Verifier now enforces (for commitments/amendments).

* verification (object, REQUIRED): checks (array), method ("static", "static+policy", "static+semantic", or "static+policy+semantic"), and OPTIONAL risk_score, semantic_invoked, semantic_trigger, confidence, reason.

* clarification (object): REQUIRED when verdict is "clarify"; maps to a host-protocol user-elicitation request.

* audit_ref (string, OPTIONAL): reference to the audit record.

* signature (string, OPTIONAL): JWS over the verdict. RECOMMENDED for "denied" and "downgraded" so that refusals are non-repudiable and independently auditable by the Agent (mutual accountability).

9. Verification Procedure

On a Scope Commitment (session start):

1. S1. Authentication/authorization (delegated to the host mechanism). On failure, deny the session.

2. S2. Semantic verification: assess whether goal is coherent and the requested scope is proportionate to it. On acceptance, cache the commitment and return its digest.

On an Intent Envelope (per call), in order, failing closed:

1. C1. Bind: session_id MUST map to an accepted commitment.

2. C2. Scope: call.tool MUST match tools_allow and MUST NOT match tools_deny. Only the tool NAME is examined. A scope escape is Drift.

3. C3. Budget: projected consumption MUST NOT exceed any meter, the call count, or the deadline (Section 11).

4. C4. Policy (OPTIONAL, deployment-local): operator-authored rules MAY inspect arguments and impose constraints. This hook is where tool-specific limits (e.g. a per-call monetary cap) live. It is not part of the VAP wire schema and MAY be absent.

5. C5. Semantic (gated): if the risk score (Section 10) crosses the configured threshold, assess whether the call is consistent with the session goal and rationale.

On a Scope Amendment: re-run S2 against the requested widening; on acceptance, re-baseline and chain the commitment digest.

10. Risk-Scored Semantic Admission Control

Semantic verification (C5/S2) is the most expensive check and, if applied naively, can cost more than it saves and can be evaded. A Verifier MUST NOT gate C5 solely on proximity to a budget limit.

A Verifier SHOULD compute, for every call, a cheap deterministic risk score from signals including: the declared or server-assigned sensitivity/irreversibility class; burn-rate anomaly (consumption outpacing the plan); looping (repeated identical calls); novelty (first use of a tool, or arguments far from prior accepted calls); and proximity to a limit. A Verifier MUST additionally invoke C5 on a configurable random sample of otherwise-passing calls, so that an Agent cannot reliably stay below deterministic thresholds while drifting.

A Verifier SHOULD cache semantic verdicts keyed by (plan_digest, tool, argument-shape) to bound repeated cost. Any inference consumed by C5/S2 SHOULD itself be billed to a budget meter (Section 11).

11. Budget Metering

A budget is multi-dimensional and unit-agnostic. It comprises the universal meters max_calls and deadline, and an extensible "limits" map from a meter name to its maximum cumulative value.

A Verifier treats meter names as OPAQUE. A tool self-declares its per-call contribution to one or more meters by returning a cost map in its result metadata (Section 12). The Verifier sums declared contributions and denies any call that would cause any meter to exceed its ceiling. If a tool declares no contribution, the operator MAY supply default per-tool costs.

This design lets a commitment bound an aggregate side effect (for example, total monetary value disbursed across many calls) as an ordinary meter, WITHOUT the Verifier possessing any knowledge of what

the meter represents or what the tool does. Well-known meter names include "tokens" and "usd_opcost" (operational cost); all others are deployment-defined.

12. Transport Binding (MCP)

When the host protocol is MCP [MCP], VAP messages are carried in the "_meta" object that MCP permits on requests and results, under the key "vap". Specifically:

* The Scope Commitment is carried in params._meta.vap on the MCP "initialize" request.

* The Intent Envelope is carried in params._meta.vap on each "tools/call" request.

* A tool’s per-call cost map is carried in the MCP result under result._meta.vap.cost.

* The Verdict is carried in the MCP result _meta, and, when a call is denied, the Verifier returns an MCP tool error result rather than an HTTP-layer error, so that the host session is preserved.

* A "clarify" verdict SHOULD be realized using MCP elicitation.

Because all VAP data rides in _meta, an MCP server that does not implement VAP is unaffected, and a VAP Verifier deployed as a proxy is transparent to it. Other host bindings (e.g. A2A) MAY be defined in companion documents.

13. Capability Negotiation

A Verifier advertises its VAP posture during host-protocol capability negotiation, including whether a Scope Commitment is required, whether messages must be signed, which verification tiers it runs, the conditions under which it invokes semantic verification, and its audit format. An Agent that does not present VAP metadata MAY be rejected, served in a restricted mode, or served with intent treated as advisory, at the operator’s discretion.

14. Audit

For every decision, a Verifier SHOULD emit an append-only audit record binding session_id, the active commitment digest, principal, intent, call, verdict, verification result, a hash of any result, and a timestamp; and a session-level record of the commitment and every amendment. Records SHOULD be signed. Unlike a conventional access log, a VAP audit answers why a call was made, within what committed

envelope, and how it was adjudicated.

15. Security Considerations

Declared purpose is attacker-controllable. An adversarial Agent can author a plausible rationale for a harmful call. VAP therefore MUST NOT be used as an authentication or authorization mechanism; it is defense-in-depth that raises the cost and auditability of incoherent behavior and is most effective against confused or mis-prompted (as opposed to actively malicious) Agents.

Semantic verification employs an LLM and is subject to prompt injection via the goal, rationale, or tool content it examines. Implementations SHOULD isolate the judge’s instruction channel from examined content and SHOULD treat all examined fields as untrusted.

The semantic judge can be evaded by an Agent that keeps each call below deterministic risk thresholds; mandatory random sampling (Section 10) mitigates but does not eliminate this.

A Verifier that calls an external judge endpoint MUST fail closed by default: on judge error or timeout, the call is treated as not consistent (clarify/deny), never served by default.

Signatures (commitments, amendments, verdicts) provide non-repudiation only if key distribution and canonicalization are sound; implementations MUST use JCS [RFC8785] before signing to avoid cross-implementation signature divergence.

Intent and rationale fields may contain sensitive information; raw chain-of-thought MUST NOT be transmitted (Section 6). Audit records may contain sensitive data and MUST be access-controlled.

16. IANA Considerations

This document has no IANA actions. A future revision may request registration of a well-known meter-name registry and a VAP version registry.

17. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, June 2020, <https://www.rfc-editor.org/info/rfc8785>.

18. Informative References

[MCP] Anthropic, "Model Context Protocol Specification", 2025, <https://modelcontextprotocol.io/>.

[VAP-SCHEMAS] Samal, D., "VAP JSON Schemas 0.1", 2026, <https://vap.dev/schemas/0.1/>.

Author’s Address

Deep Samal
Independent
Email: deep.samal@gmail.com