DRAFT · V0.2 · JUNE 9, 2026 · UPDATED JULY 16, 2026

Taming the Confused Deputy: The Agent Proxy Pattern

Bounded, Revocable, Verifiable Authority for Autonomous Agents

Calvin Pak

Independent · Reference implementation: MAP (Molecule Agent Proxy), on Arc · OAP (Orbit Agent Proxy) is a planned instantiation · 2026

Disclosure: the author created Molecule Agent Proxy (MAP) and Orbit Agent Proxy (OAP).

ABSTRACT

Autonomous AI agents increasingly act on real resources (capital, data, APIs) on a principal's behalf. An LLM agent is uniquely susceptible to becoming a *confused deputy*: prompt injection, hallucination, and goal-misgeneralization make it trivial to trick into misusing whatever authority it holds. Prevailing practice, which hands agents raw credentials such as API keys and wallets, grants *ambient authority* with unbounded blast radius. We present the **Agent Proxy Pattern**: rather than give an agent the wallet, give it a *proxy* that holds a scoped, capped, time-bounded, revocable mandate; every action is authorized against the mandate, settled per-action, and recorded for verification. We model the mandate as an *attenuated capability*, state and sketch a **Bounded-Exposure Theorem**, and show the pattern unifies single-sign-on, RBAC, financial mandates, content-licensing, and skill/tool authorization as instances of one framework. A distinctive consequence: because the committed exposure is provably bounded, bounded authority becomes a *necessary precondition for underwriting* agent actions, whether by an insurer or, on a zero-sum market, by a risk-warehousing counterparty. We give a reference implementation on a stablecoin-native chain (MAP), a planned trading instantiation (OAP), and a threat-mitigation analysis.

Keywords: confused deputy, capability security, autonomous agents, delegation, account abstraction, agentic payments, insurability, access control

1 Introduction

In 1988, Norm Hardy described the *confused deputy*: a privileged program induced by a less-privileged caller to misuse its authority [1]. The deputy is merely *confused* about whose intentions it is serving rather than acting from malice. Four decades later, the large-language-model (LLM) agent has emerged as the confused deputy's apotheosis. An LLM agent is a program whose control flow is determined by natural-language input it cannot reliably distinguish from instruction. Prompt injection [12], hallucination, and goal-misgeneralization make it trivial to trick an agent into exercising whatever authority it holds against its principal's interest. The agent is a deputy that can be confused by a sentence.

This would be a manageable problem if agents held little authority. They do not. Agents are being handed wallets, API keys, database credentials, and trading accounts so they can *act* on capital, data, and services on a principal's behalf. The dominant integration pattern is to give the agent the raw credential directly. We call this *ambient authority*: the agent wields the principal's full power, all the time, for any action its reasoning can reach. The blast radius of a confused agent equals the full extent of the credential: the entire wallet, the entire API surface, the entire account.

We argue that *ambient authority is the bug*. The defense is to never give the reasoner ambient authority in the first place, rather than to rely on better prompts or stronger guardrails on an inherently manipulable reasoner. The principle is old (least privilege [2], object capabilities [3]), but its application to autonomous agents acting on real-world resources has not been synthesized into a single, implementable pattern with provable guarantees.

We present that pattern. Rather than give the agent the wallet, give it a **proxy** that holds a *mandate*: a scoped, capped, time-bounded, revocable grant of authority. The agent reasons and proposes actions; the proxy authorizes each action against the mandate, settles per-action, and records the action for independent verification. The agent never holds ambient authority, only the right to *request* actions within an envelope the principal defined and can collapse at will.

Contributions

- (i) The **Agent Proxy Pattern** and a formal model of the mandate as an attenuated capability;
- (ii) security properties, including a **Bounded-Exposure Theorem** that bounds the value an agent can commit by the mandate cap;
- (iii) an **insurability result**: provably bounded exposure is a necessary precondition for underwriting agent actions by a risk-warehousing counterparty (subject to correlated-failure caveats, §6);
- (iv) a **unifying framework** (resource × topology) that subsumes SSO, RBAC, financial mandates, content-licensing, and skill/tool authorization as instances;
- (v) a **reference implementation** (MAP) demonstrating the core mechanism on a stablecoin-native chain, with a planned

trading instantiation (OAP).

We are explicit about what is and is not novel. We do not invent the primitives. Capabilities [2,3], session keys and account abstraction [9], escrow [11], agent identity [10], and agentic-payment rails such as x402 [13] and AP2 [15] already exist. Our contribution is the *pattern*: the synthesis of these primitives into a single authority model, the security properties it yields, the insurability consequence, and the generalization showing that a large class of access-control and delegation problems are instances of one framework.

2 Background & Related Work

2.1 Capabilities and the confused deputy

Dennis and Van Horn introduced capabilities as unforgeable tokens that simultaneously designate a resource and authorize access to it [2]. Object-capability security [3] makes authority follow reference: you can only act on what you hold a reference to, and you can pass attenuated references onward. Hardy's confused deputy [1] is precisely the failure mode that capabilities prevent and that *ambient authority* (identity-based access where a process acts with all its privileges by default) invites. The Agent Proxy is, at its core, a capability discipline applied to an LLM reasoner that cannot be trusted to police its own authority.

2.2 Access control and delegation

Role-Based Access Control [4] binds permissions to roles and roles to principals; Attribute-Based Access Control generalizes this to predicates over attributes. Delegation logics formalize "speaks-for" relationships. OAuth 2.0 [5] introduced *scopes* (coarse, bearer-token restrictions on delegated access), but OAuth scopes are typically not attenuable by the holder, not per-action metered, and not independently auditable. Macaroons [6] are the closest prior art at the credential layer: bearer tokens with *caveats* that any holder can further restrict, giving monotonic attenuation. The Agent Proxy adopts the macaroon insight (attenuation), adds enforced caps, time bounds, and one-transaction revocation, and binds the whole to per-action settlement and an on-chain audit trail.

2.3 Account abstraction and agentic payments

ERC-4337 account abstraction [9] enables smart-contract accounts with programmable validation, including *session keys*: short-lived keys constrained to a policy. ERC-8004 [10] standardizes on-chain agent identity; ERC-8183 [11] specifies escrow primitives for agentic commerce. x402 [13] revives HTTP 402 for agent micropayments. We stress that ERC-8004 is an identity and reputation registry: it establishes *who* an agent is and confers no authority of its own; authorization comes solely from the signed mandate. ERC-8183 and x402 are still emerging drafts; where our construction leans on them it degrades gracefully, since the hard bound rests on the signed mandate and the proxy predicate rather than on any single standard shipping as currently drafted. Google's **A2A** (Agent-to-Agent) [14] standardizes inter-agent *communication* and discovery; **AP2** (Agent Payments Protocol) [15] standardizes payment *authorization* via signed "mandates." We position A2A and AP2 as *complementary, not competing*: A2A is a communication layer and AP2 is a settlement-authorization layer, whereas the Agent Proxy is an *authority* layer that sits above both. AP2's mandate is, in our model, a special case (a capital-scoped mandate for a single payment topology) that our framework generalizes across resources and topologies.

2.4 Content licensing and provenance

A parallel authorization problem is emerging for *data*. **RSL** (Really Simple Licensing) [16] proposes machine-readable, pay-per-inference licensing terms. Microsoft's **Publisher Content Marketplace** [17] and **Story Protocol**'s on-chain Programmable IP License [18] bind content access to enforceable terms. We distinguish these sharply from **C2PA / Content Credentials** [19], which establish *provenance* (what a thing is and where it came from) but *not authorization* to use it. In our framework, content-licensing is the Data resource lens; C2PA is explicitly excluded as it answers a different question.

2.5 Principal-agent theory and Business-as-Code

The economics of delegation under misaligned incentives and asymmetric information is the classical principal-agent problem [7]. The Agent Proxy is, in a sense, a mechanism that makes the principal-agent contract *enforceable in code* rather than merely in law. Finally, we situate the pattern within **Business-as-Code** [20]: "SOPs are source code, AI agents are the runtime." Its COPE framework's "Perspective" dimension is RBAC; the Agent Proxy supplies the missing *authority* layer for that runtime: the bounded, revocable grant under which a BaC agent is permitted to act.

Figure 1: The confused deputy, before and after. Left: the agent holds the full credential; a single confused action can reach the entire resource. Right: the agent holds only a scoped session key and must route every action through the Agent Proxy, which enforces the mandate, collapsing the blast radius to the cap and time-to-live (ttl).

3 Threat Model & Problem Formalization

3.1 Entities

We model four entities. A **Principal** `P` owns the authority and the resource at risk and wishes to delegate *bounded* authority. An **Agent** `A` is an *untrusted reasoner* (typically an LLM with tools) that proposes actions. A **Resource** `R` is the thing of value being acted upon (capital, data, an API, a skill). **Authority** is the right to cause state changes on `R`.

3.2 The agent-authority problem

Classical delegation assumes the deputy faithfully executes the principal's intent and can be trusted to refuse out-of-scope requests. An LLM agent breaks this assumption: its behavior is a function of untrusted input it cannot reliably separate from instruction. The *agent-authority problem* is therefore: *grant `A` enough authority to be useful on `R` while guaranteeing that no behavior of `A`, however induced, can cause `P` a loss exceeding a bound `P` chose in advance.* Note the quantifier: the guarantee must hold over *all* behaviors of `A`, including adversarially induced ones, because we cannot enumerate or predict them.

3.3 Threat model T

We consider an adversary who can achieve any of:

- **Leaked credential**: the agent's session key is exfiltrated;
- **Compromised / hijacked agent**: the agent process or its tools are subverted;
- **Prompt injection** [12]: the *confused-deputy trigger*, in which untrusted content instructs the agent to act against `P`;
- **Runaway loop**: the agent repeats a harmful action without external malice;
- **Malicious counterparty**: the other side of a transaction defrauds the agent;
- **Faulty oracle / data feed**: the agent reasons correctly over wrong inputs.

3.4 Trust assumptions

The trusted computing base (TCB) is small, and we name it explicitly rather than wave at it: the *proxy contract* that evaluates `Auth` (trusted for audited-code correctness; a bug voids the bound), the *settlement chain* (trusted for liveness and correct execution), the *price oracle* that supplies the value used in the cap check (trusted for honest quotes; a corrupt oracle mis-measures value), and the *session-key issuer* [9] (trusted for key custody; a leaked issuer key could mint session keys). Each is auditable and carries the compromise consequence noted; together they enforce the authorization predicate, settle correctly, honor revocation, and produce a tamper-evident audit trail. The *agent* is trusted for *nothing* regarding authority: we assume it may behave arbitrarily within the action interface the proxy exposes. The mandate's authority bounds must hold even if `A` and its session key are fully controlled by the adversary. We do *not* assume the oracle or counterparty is honest; we instead bound the damage they can cause to the mandate cap (§5–6). We do not defend against compromise of the chain or proxy contract itself; those are the explicit trust roots.

4 The Agent Proxy: Formal Model

4.1 The mandate

A **mandate** is the capability the principal issues to the proxy on the agent's behalf:

DEFINITION 1: MANDATE

$M = (id, k_{session}, scope, cap, rate, ttl, revoker)$

```
  id       unique, non-forgeable identifier
  k_session public key the agent signs actions with
  scope    set of permitted action types / targets
  cap      max cumulative value at risk (the bound)
   rate       max actions / value per unit time (velocity
ceiling)
  ttl      expiry timestamp
  revoker   principal key authorized to revoke
```

4.2 The authorization predicate

Every proposed action must satisfy a single predicate the proxy evaluates against the mandate and current state before it is allowed to settle:

DEFINITION 2: AUTHORIZATION PREDICATE

```
Auth(action, M, state) :=
    valid_sig(action, M.k_session)
  ∧ action.type ∈ M.scope
  ∧ state.reserved + action.value ≤ M.cap
  ∧ within_rate(action, M.rate, state)
  ∧ now() < M.ttl
  ∧ ¬ revoked(M.id)
```

An action settles *iff* `Auth` holds. The proxy computes `action.value` itself as `trusted_price × size` from a named signed price oracle rather than trusting any figure the agent self-reports; the oracle is therefore an explicit member of the TCB, and a compromised agent cannot understate what an action puts at risk. The `scope` check is an exact-match test against a literal allowlist of permitted action types and targets (semantic scopes are compiled to such a list before signing), so the enforcement decision is deterministic and contains no LLM. Because multiple actions may be in flight, the accumulator uses atomic *reserve-then-commit* accounting: authorization reserves the action's value against the cap, and the reserve is committed on settlement or released on cancellation, so concurrent actions cannot each pass the cap check before any of them settles. The predicate is total, monotone in the reserved accumulator, and cheap to evaluate on-chain. Critically, it is evaluated by the proxy (part of the trusted base), not by the agent.

4.3 The two-layer mandate

We distinguish two layers that are commonly conflated. The **hard authorization layer** (scope, cap, ttl, revoker) is *enforced and provable*: it bounds what the agent can do regardless of its reasoning. The **soft objective layer** (risk preferences, objectives, an Investment Policy Statement (IPS) in the capital case) is *read by the agent's reasoning to shape behavior, not bound it*. The soft layer tells the agent what it *should* want; the hard layer enforces what it *can* do. A confused agent may ignore the soft layer entirely; it can never exceed the hard layer. Conflating the two is the central error of "guardrail" approaches that try to make the soft layer load-bearing for security.

4.4 Attenuation and sub-delegation

An agent may delegate to a sub-agent by issuing a sub-mandate. Following object-capability discipline [3] and macaroon caveats [6], sub-delegation is *monotonically attenuating*:

DEFINITION 3: ATTENUATION ORDER

```
M' ⊑ M  iff
  M'.scope ⊆ M.scope
 ∧ M'.ttl  ≤ M.ttl
 ∧ revoke(M) ⇒ revoke(M')
 ∧ Σ_j committed(M'_j) ≤ M.cap - state.spent(M)

Siblings share one parent budget: the sum
over all children cannot exceed the parent's
remaining cap.
```

No sequence of sub-delegations can escalate authority above the root mandate, and because every child spends against the same parent accumulator, the summed authority of all siblings stays within the root cap however wide the fan-out; revoking a parent revokes the whole subtree.

4.5 Mandate-as-code

The mandate is *authority as code*: a machine-checked, on-chain artifact whose semantics are the authorization predicate. It is the authorization-layer complement to Business-as-Code's policy-as-code [20]: where BaC encodes *what the business does* as executable SOPs, the mandate encodes *what the agent is permitted to do* as an executable predicate.

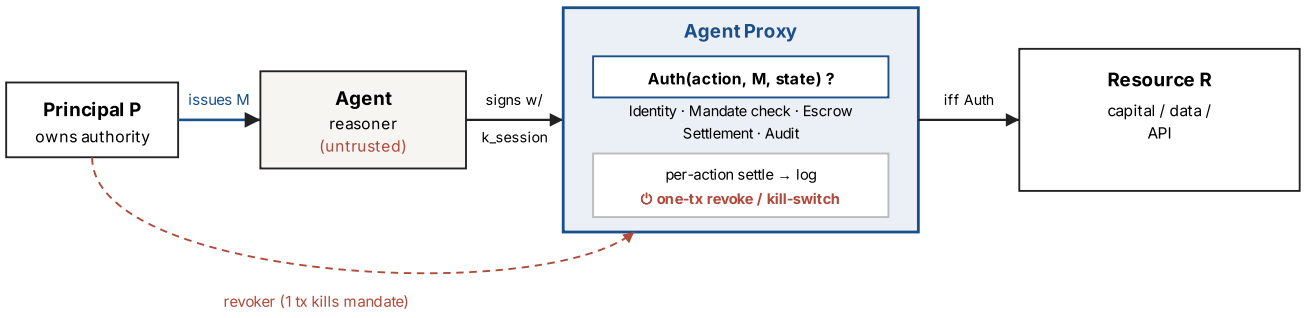

Figure 2: Agent Proxy architecture. The principal issues a mandate `M`; the untrusted agent signs proposed actions with the session key; the proxy evaluates the authorization predicate (identity, mandate check, escrow, settlement, audit) and lets an action reach the resource only if `Auth` holds. The principal's revoker can collapse the mandate in a single transaction.

5 Security Properties & Analysis

Each property below ties to a specific mechanism in §4. We aim for a rigorous tone; full machine-checked proofs are future work, and we give proof *sketches*.

> **Theorem 1 (Bounded Exposure).** Assume the proxy contract, the price oracle, and the settlement chain execute correctly, action values are computed against a trusted signed price reference, and exposure is denominated in the mandate's unit of account. Then under mandate `M` and threat model `T`, the cumulative value agent `A` can commit *through* `M` is at most `M.cap`, and only within the window before `M.ttl` or until revocation, whichever is earlier. The pattern bounds the value an agent may put at risk; it does not bound the realized outcome of an instrument the principal chose to authorize: for a leveraged or short position, funding, slippage, and liquidation are outcome risks the principal accepts, and the cap bounds only what the agent may commit toward it.

Proof sketch. Any state change on `R` caused via the proxy must pass `Auth` (Def. 2). The conjunct `state.reserved + action.value ≤ M.cap`, with `action.value` computed by the proxy from a signed price oracle and enforced against a reserve-then-commit accumulator, ensures the sum of all committed action values never exceeds `M.cap`. The conjuncts `now() < M.ttl` and `¬ revoked(M.id)` ensure no action settles after expiry or revocation. Since `A` holds no authority outside the proxy path (non-custody, below), the total value `A` can commit is ≤ `M.cap`. This holds for *all* behaviors of `A`, including a fully adversary-controlled agent with the leaked session key, because the bound is enforced by the trusted proxy, not by `A`. □

Three boundary conditions lie outside the theorem and are noted in §10: the bound is on committed value in the mandate's unit of account, so a manipulated price reference can cause economic loss the cap does not measure; a principal issuing several concurrent mandates is exposed to the sum of their caps, not a single cap; and for leveraged or short instruments the realized loss is a function of the instrument the principal elected to authorize, so funding and liquidation move outside the cap even though the committed value does not. These are addressed by oracle choice, a portfolio-level cap, and instrument selection, not by Theorem 1.

Least privilege

The `scope` conjunct restricts `A` to a declared set of action types and targets [2]. An agent issued a "trade ETH-perps on venue X" mandate cannot transfer tokens, call arbitrary contracts, or touch another venue; those actions fail `Auth` regardless of what the agent is convinced to attempt.

Revocability

The `revoker` can set `revoked(M.id)` in a single transaction, after which `Auth` is false for all future actions. This bounds the *exposure window*: the time between detecting misbehavior and stopping it is one block, not a credential-rotation scramble across every system the leaked key reached. Revocation stops all *future* authorization; it is subject to block-inclusion latency and can be front-run by an action already in the mempool, and any order resting at an external venue must still be canceled through that venue. It bounds new authority within one block and open exposure as fast as the settlement layer allows.

Non-custody

The proxy never holds `P`'s funds. Value is *pulled at settlement* from the principal's account or escrow only at the moment an authorized action settles. There is no pooled balance for an attacker to drain; compromising the proxy yields no standing custody, only the (still-bounded) ability to authorize actions the predicate already permits.

Verifiability

Every settled action is recorded on-chain with its mandate id, value, and timestamp. Any party (the principal, an auditor, an insurer) can independently reconstruct exactly what the agent did and verify that the cap and ttl were respected, without trusting the agent or the proxy operator's word.

Monotonic attenuation

By Def. 3, any sub-mandate satisfies `M' ⊑ M` and every child draws from the shared parent budget, so the summed authority of all siblings stays within the parent cap; composition of attenuations is itself an attenuation, so no delegation chain can escalate authority. Revoking a parent revokes the subtree. Sub-delegation is therefore safe by construction, which is what makes agent-swarm architectures tractable.

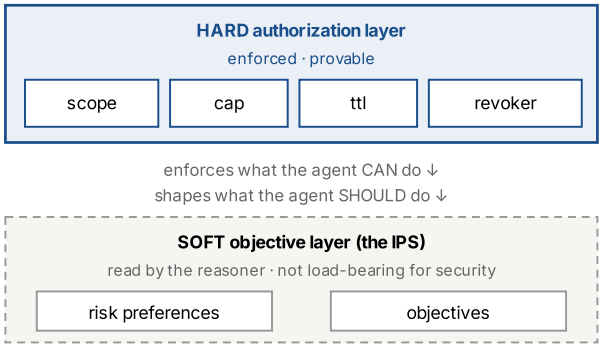

Figure 3: The two-layer mandate. The hard layer (scope, cap, ttl, revoker) is enforced and provable: it bounds the agent. The soft layer (the Investment Policy Statement: risk preferences, objectives) is read by the agent's reasoning to shape behavior, but is never load-bearing for the security guarantee.

6 Insurable Authority

Theorem 1 yields a consequence that, to our knowledge, has not been drawn out for autonomous agents: bounded authorized exposure is the precondition that makes agent risk *underwritable*. The party that underwrites a bounded agent is a risk-warehousing counterparty (an insurer, or on a zero-sum market simply a liquidity provider or counterparty) that prices and warehouses many bounded exposures and manages the resulting book. Underwriting requires a quantifiable, bounded maximum exposure with verifiable claims. Ambient-authority agents fail this on every count: exposure is unbounded (the whole wallet), duration is open-ended, and there is no canonical audit. No counterparty can price a risk with no maximum.

The mandate supplies exactly the missing structure. The `cap` is a *known maximum exposure per mandate*. The `ttl` bounds the *time at risk*. The `rate` bounds *velocity*. The on-chain audit trail enables *claims verification* without trusting the insured. Revocation *caps*

tail risk by bounding the exposure window after an anomaly is detected.

A first-cut model of the book a warehousing counterparty holds: expected loss across a fleet of mandates is

$$E[\text{loss}] \approx \Sigma_i \; p_i \cdot \min(\text{cap}_i, L_i) \cdot (\text{ttl}_i / T_{\text{year}}) \; + \; C(\rho)$$

p_i = agent-failure probability / unit time
L_i = loss given failure ($\le$ cap_i by Thm 1)
ttl_i = time-at-risk for mandate i
$C(\rho)$ = correlated-failure loading

Because $\min(\text{cap}_i, L_i) = L_i \le \text{cap}_i$ is hard-bounded, the worst-case per-mandate exposure is finite and known when the position is warehoused. The independent sum is the easy part; the dominant term the warehouse must actually price is *correlated agent failure* (one model or policy-engine bug firing across every mandate simultaneously), which is precisely what breaks the independence a naive sum assumes. The book must also carry moral-hazard controls: losses to related or colluding counterparties are excluded, and the receipt trail is structured to support counterparty-independence checks, so a principal cannot profit by trading against their own warehoused agent. The free parameters are the empirical failure rates p_i and their correlation structure, which the audit trail lets the warehouse *measure* over time.

This is a genuine *risk-transfer* mechanism for autonomous agents: a principal can buy coverage against a confused agent's bounded exposure, and a risk-warehousing counterparty can write it because the maximum is provable and the history verifiable, subject to correlated-failure pricing. Ambient-authority agents are, by contrast, structurally un-warehousable. We regard this as the pattern's most consequential cross-disciplinary result: bounded authority is the precondition for a *risk market for agent fleets*.

7 A Unifying Framework

The Agent Proxy generalizes beyond capital along two axes: the **resource lens** (what kind of authority is being bounded) and the **topology** (who delegates to whom). Many existing access-control and delegation systems are instances of one cell.

7.1 Resource lens

What varies across lenses is only the *type* of scope and cap; the mandate, the predicate, and the security properties are identical. SSO and RBAC are the identity and role instances long studied [4,5]; Capital is OAP; Data/Content-licensing is RSL/Story/PCM [16,17,18]; Skill/Tool bounds which actions an LLM may invoke. C2PA [19] is provenance, not authorization, and is excluded.

7.2 Topology

The same mandate spans delegation shapes: Agent↔Platform (an agent acting on a venue), Agent↔Agent commerce (one agent hires and pays another, composing with AP2/x402 [13,15]), Agent↔Agent P2P, Human↔Agent (oversight and delegation), and Agent↔Sub-agent (attenuated delegation, Def. 3). The Agent Proxy is *protocol-agnostic and sits above* A2A (discovery/communication) and AP2/x402 (settlement): it composes with them rather than competing.

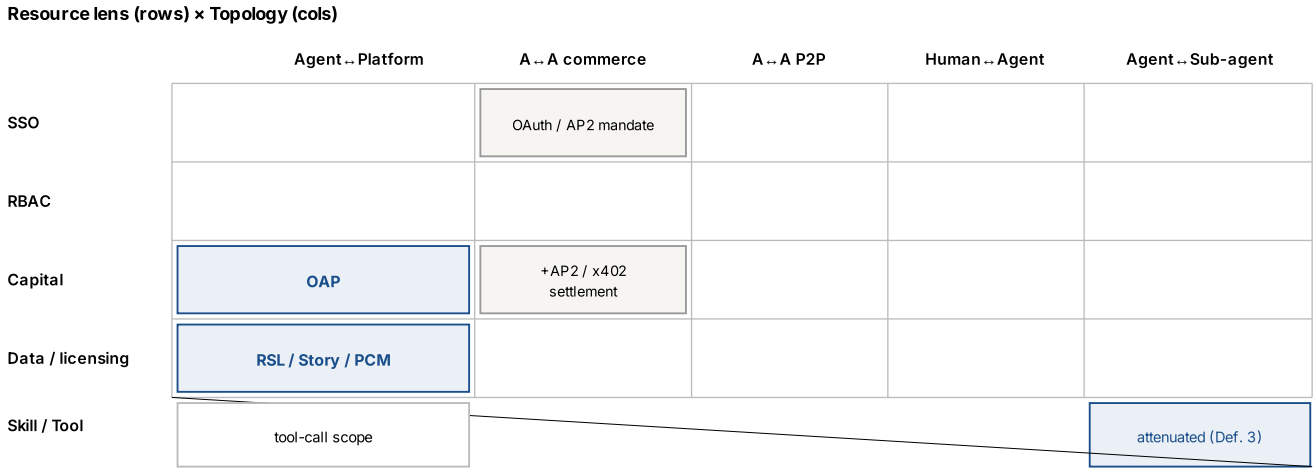

Figure 4: The unifying framework. A 2-axis grid: resource lens (SSO, RBAC, Capital, Data/Content-licensing, Skill) × topology (Agent↔Platform, Agent↔Agent commerce, Agent↔Agent P2P, Human↔Agent, Agent↔Sub-agent). Selected cells: Capital×Agent↔Platform = OAP; Data×Agent↔Platform = RSL/Story/PCM; Agent↔Sub-agent = attenuated delegation (Def. 3); AP2/x402 compose in as the settlement layer.

Table 1: Resource lenses as instances of one mandate. Only the scope/cap type varies; the predicate and security properties are constant. C2PA is excluded (provenance, not authorization).

Lens	What scope bounds	What cap bounds	Prior art / instance
SSO	identity assertions, federated audiences	session lifetime	OAuth/OIDC scopes [5]
RBAC	roles → permitted operations	privilege ceiling	Sandhu et al. [4]; BaC "Perspective" [20]
Capital	instruments, venues, action types	value at risk (USDC)	**OAP**; AP2 mandate [15]
Data / licensing	corpora, usage terms, inference rights	pay-per-inference budget	RSL [16], Story IP [18], PCM [17]
Skill / Tool	which tools/functions an LLM may call	call budget / rate	tool-call allow-lists

7.3 Persistent agents and long-running loops

A recurring pattern for long-running agentic work is the *loop*: an agent that runs unattended over hours or days, iterating perceive–decide–act many thousands of times. This is the confused deputy in its most dangerous form, because confusion and goal-drift *compound across iterations* and no human is in the loop per action. The Agent Proxy is suited to exactly this case, because it bounds a loop in all three dimensions along which it can diverge. The `cap` bounds *cumulative* loss across the entire run, so a loop may iterate without limit yet never exceed it; the `ttl` bounds *time at risk*, forcing re-authorization to continue; and the `rate` ceiling bounds *velocity* (actions or value per unit time), the circuit-breaker against a runaway iteration. Revocation halts a misbehaving loop instantly mid-run (§5); per-action settlement leaves a continuously auditable trace of the job; and the soft objective layer (§4) carries the run's goals and risk preferences across iterations, countering drift. A loop without a proxy is an unbounded liability; under one it is bounded in exposure, duration, and rate, revocable, and audited: the conditions under which a persistent agent can safely run unattended on real resources.

8 Implementation and Intended Deployment

One-line intuition: the Agent Proxy is to an agent what *Stripe* is to a merchant: you never hand over the card; you hand over a bounded ability to charge.

8.1 MAP: Molecule Agent Proxy (reference implementation)

MAP composes ERC-8004 agent identity [10] with an EIP-712 scoped session-key delegation and per-call USDC settlement, deployed on Arc (a USDC-native chain where gas is denominated in USDC). Its deployed instance demonstrates the core mechanism for *metered, identity-gated access to a resource*: a principal issues a scoped delegation; the agent acts under it (in the deployed demo, calling an external LLM/API on the principal's behalf); each call is settled per-action in USDC; and the delegation is revocable. MAP thus realizes the identity → scoped-delegation → per-action-settlement spine of the pattern. The escrow layer (§4) and the capital instance below are the natural extensions of that spine and are *not yet built*. *Built at the Arc hackathon; deployed on Arc testnet (Arc mainnet is not yet live).*

8.2 OAP: Orbit Agent Proxy (intended deployment)

The pattern's intended capital-lens deployment is OAP, in *Orbit*: an APAC-focused, agent-native prediction market for long-tail event markets (regional sports, esports, cultural and crypto-native events) that human market-makers under-serve. Such venues are exactly where bounded agent authority becomes load-bearing: liquidity provision and trading are increasingly agent-driven, spread across thousands of thin markets, and run unattended. As designed, an agent would participate (making markets or trading) under a capped, scoped, revocable mandate, using *single-shot merged-EIP-712 signing*: no pre-approval and no deposit, with the proxy *pulling at settlement* from the principal so funds are never custodied by the proxy or the agent (the non-custody property of §5). The mandate's soft layer would be an Investment Policy Statement encoding the venue's risk and inventory limits; the hard layer the cap/scope/ttl/revoker the principal sets. OAP thus instantiates the full capital lens (scoped delegation, per-action settlement, and one-transaction revocation) on a live, distribution-led venue. *Intended deployment; in development.*

9 Evaluation

We evaluate qualitatively against the threat model and prior approaches; empirical measurements on Arc testnet are pending and marked.

Table 2: Threat → property mapping. Each threat from §3 and the property that neutralizes it.

Threat	Neutralizing property
Leaked credential	Bounded exposure + revocability (committed value ≤ cap; kill in 1 tx)
Hijacked agent	Least privilege + bounded exposure (scope + cap hold for any behavior)
Prompt injection	Hard layer is non-bypassable; soft layer not load-bearing
Runaway loop	Cap (cumulative ceiling) + rate (velocity ceiling) + ttl (auto-expiry)
Malicious counterparty	Non-custody + scope; per-action settlement limits exposure to cap
Faulty oracle / feed	Bounded exposure (cap): does not *prevent* bad inputs, bounds their damage

Table 3: Pattern comparison. Raw keys/wallet vs OAuth scopes vs macaroons vs AP2 vs Agent Proxy.

Property	Raw key	OAuth	Macaroon	AP2	Agent Proxy
Bounded exposure (cap)	–	–	–	●	●
Revocable (fast)	–	~	~	●	●
Attenuable	–	–	●	~	●
Per-action settlement	–	–	–	●	●
Verifiable audit	–	~	~	●	●
Insurable	–	–	–	~	●

● = yes; ~ = partial/protocol-dependent; – = no. AP2 satisfies several properties for the single payment topology; the Agent Proxy generalizes them across resources and topologies and adds the insurability result.

Table 4: Per-action settlement on Arc. Observed network characteristics (testnet, June 2026); a full MAP micro-benchmark is forthcoming.

Metric	Value
Block interval (observed)	≈ 1.4 s
Finality (BFT, Malachite consensus)	sub-second (documented)
Gas denomination	native USDC
Revocation	single transaction (≈ 1 block)

The figures above reflect Arc testnet network characteristics observed on June 10, 2026 (RPC `rpc.testnet.arc.network`, chain 5042002); a full MAP micro-benchmark (per-call USDC cost and end-to-end latency under load) is forthcoming. The qualitative guarantees (Tables 2–3) follow from §5 and do not depend on measurement.

10 Limitations & Open Problems

The pattern bounds *authority*, not *judgment*. It cannot prove an agent makes good decisions, only that bad ones stay within con-

straints and are recorded. Garbage-in yields a valid proof-of-garbage-out: a mandate-compliant action can still be unwise. Several trust roots remain: the *oracle/data feed* is trusted for correctness (we bound its damage, not its honesty); if a TEE is used for the reasoner, *side-channels* are out of scope. The *mandate language* is currently limited in expressiveness: rich conditional scopes need a verified semantics. Aligning the *soft objective layer* is the open AI-alignment problem and we make no claim to solve it. Finally, *on-chain mandate privacy* is a real tension: verifiability and confidentiality pull against each other, and selective-disclosure mechanisms are future work.

11 Conclusion & Future Work

The confused deputy is not a bug to be patched out of LLM agents; it is intrinsic to a reasoner whose control flow is driven by untrusted language. The remedy is to stop giving such reasoners ambient authority. The **Agent Proxy Pattern** gives the agent a mandate, not a wallet: a scoped, capped, time-bounded, revocable capability whose every exercise is authorized, settled, and recorded. From this follow a Bounded-Exposure Theorem, least privilege, fast revocation, non-custody, verifiability, and monotonic attenuation; and, as a cross-disciplinary consequence, *insurability*. The pattern unifies SSO, RBAC, capital, content-licensing, and skill authorization as instances of one model, and composes with A2A and AP2/x402 rather than competing. Future work: insurance markets for agent fleets; content-licensing-as-capability (RSL/Story over Arc micropayments); attenuated agent-swarm delegation at scale; and a formal mandate language with verified semantics and machine-checked proofs of the properties sketched here.

References

1. N. Hardy, "The Confused Deputy: (or why capabilities might have been invented)," *ACM SIGOPS Operating Systems Review*, vol. 22, no. 4, 1988.
2. J. B. Dennis and E. C. Van Horn, "Programming Semantics for Multiprogrammed Computations," *Communications of the ACM*, vol. 9, no. 3, 1966.
3. M. S. Miller, "Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control," Ph.D. dissertation, Johns Hopkins University, 2006.
4. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, 1996.
5. D. Hardt (Ed.), "The OAuth 2.0 Authorization Framework," IETF RFC 6749, 2012.
6. A. Birgisson, J. G. Politz, Ú. Erlingsson, A. Taly, M. Vrable, and M. Lentczner, "Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud," *NDSS*, 2014.
7. M. C. Jensen and W. H. Meckling, "Theory of the Firm: Managerial Behavior, Agency Costs and Ownership Structure," *Journal of Financial Economics*, vol. 3, no. 4, 1976.
8. J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, vol. 63, no. 9, 1975.
9. V. Buterin, Y. Weiss, D. Tirosh, S. Nacson, A. Forshtat, K. Gazso, and T. Hess, "ERC-4337: Account Abstraction Using Alt Mempool," Ethereum Improvement Proposals, 2021.
10. M. De Rossi, D. Crapis, J. Ellis, and E. Reppel, "ERC-8004: Trustless Agents," Ethereum Improvement Proposals, no. 8004, 2025.
11. D. Crapis, B. Lim, T. Weixiong, and C. Zuhwa, "ERC-8183: Agentic Commerce," Ethereum Improvement Proposals, no. 8183, 2026.
12. S. Abdelnabi, K. Greshake, 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," *ACM AISec*, 2023.
13. Coinbase, "x402: An Open Protocol for Internet-Native Payments over HTTP 402," technical specification, 2025.
14. Google, "Agent2Agent (A2A) Protocol Specification," 2025.
15. Google, "AP2: Agent Payments Protocol Specification," 2025.
16. "RSL: Really Simple Licensing, A Machine-Readable Content Licensing Standard," rslstandard.org, 2025.
17. Microsoft, "Publisher Content Marketplace," product documentation, 2025.
18. Story Protocol, "Programmable IP License (PIL): On-Chain IP Licensing," whitepaper, 2024.
19. Coalition for Content Provenance and Authenticity, "C2PA Technical Specification (Content Credentials)," 2024.
20. C. Pak, "Business Engineering: Business-as-Code," *reboot.mba*, 2026.
21. B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in Distributed Systems: Theory and Practice," *ACM Transactions on Computer Systems*, vol. 10, no. 4, 1992.
22. D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete Problems in AI Safety," arXiv:1606.06565, 2016.