Agent-Payment Enforcement and the Confused Deputy

Oluwasegun Araba, July 2026. Position paper. The claims about enforcement mechanics are implemented and demonstrated in Purse, an open-source enforcement layer published as `@olurabian/purse`. The claims about the open problem are a position, and are labelled as one.

ABSTRACT

AI agents can now hold payment tools and spend real money. The security question this creates is not whether the agent is smart. It is whether a compromised agent can move money outside policy. This paper separates the problem into two failure classes. Forgery, where an agent moves money it was never handed the means to move, is closed by custody, a credential held in a process the agent cannot reach, with execution mediated behind that boundary. Misdirection, where a compromised agent submits a perfectly in-policy request that is not what its principal meant, is not closed by custody, and cannot be fully closed by policy, because policy answers permission and intent lives in the person. This is the confused deputy problem surfacing at the payments layer. The paper describes the levers that bound misdirection today, principal-approved grants, blast-radius limits, and tamper-evident audit, and names the open problem precisely. Verifying that an action matches what the principal actually wanted, at machine speed, without a human on the loop every time. No system, including ours, has solved it.

1. The problem

An AI agent is a loop that picks tools and runs them. Give that loop a payment tool and a wallet, and you have created a spender whose inputs include text from the open internet.

That is the whole threat in one sentence. One prompt injection, one poisoned tool result, one jailbroken instruction, and the loop pays someone. Not because it was hacked in any traditional sense. Because it did exactly what an attacker's text told it to do, with money it was legitimately holding, and nobody clicked approve.

The intelligence layer is trivial here. Wiring a model to a payment API is an afternoon of work, and it gets easier every quarter. The layer that decides what an agent is *allowed* to spend, and proves what it *did* spend, is the part that matters. That layer barely exists in most deployments today. Where it exists, it is usually a system prompt asking the agent nicely.

This paper is about that layer. What enforcement actually requires, what it provably stops, and the one failure class it does not stop, which we argue is the open problem for the field.

2. Enforcement, not advice

Most agent spending controls today are advisory. The policy check runs, produces a verdict, and the agent's own code decides what to do with it. The agent still holds the credential. The agent still executes the payment.

Advisory controls are useful for honest agents. They catch bugs, runaway loops, fat-fingered amounts. But the threat model is not an honest agent. It is a compromised one. And a compromised agent treats a verdict the way an attacker treats a warning banner. It reads it and does the thing anyway. A guardrail an agent can route around is a suggestion.

Enforcement means the denied path does not exist. In Purse's enforcement mode, the payment credential moves into a broker process the agent cannot reach. The agent runs as a spawned child that imports only a client. It can call three operations, request, execute, and status. It cannot call a payment rail, mint grants, approve its own requests, or read policy. The broker validates every field on its own side of the boundary.

Call this custody, not convention. The key lives in a different OS process, so "denied" is not a rule the agent chooses to follow. It is the absence of any other path. Convention says please do not spend this. Custody makes the spend impossible to perform.

Two principles hold the design together. Fail closed, meaning if the policy check cannot run, the spend is denied, never allowed. And single path, meaning the property only holds if the broker is the only way money can move. That second principle is a deployment contract, not a library feature, and section 8 returns to it honestly.

The mechanics that sit behind the boundary are simple and load-bearing.

- **Single-use grants.** A non-denied decision mints one grant bound to an exact payee and amount, single-use, with an expiry. Execution is a separate broker-only step gated on that grant. The agent never converts permission into a blank cheque.
- **Reservation-aware caps.** Open grants reserve budget the moment they are minted. A burst of small spends cannot slip under a daily limit before any of them settle. Caps enforced only at settlement are caps an attacker can race.
- **Out-of-band approval.** Spends over a threshold wait for a principal to approve that exact request. The agent asks. It cannot self-approve.
- **Tamper-evident audit.** Every decision writes a hash-chained record carrying a structured reason. The why is inside the hash, so the log cannot be rewritten after the fact.

These are not hypothetical. A two-process demo settles over the x402 payment rail while a scripted agent tries to misbehave, five scenes, each asserted headlessly in CI. A normal spend settles. An injected off-allowlist payment moves nothing. An over-threshold spend waits for a human. A split-under-cap attack is blocked by reservations. The audit chain verifies.

So enforcement is real, shippable, and it stops a specific class of attack completely. The important question is which class.

3. Forgery versus misdirection

Split the failure space in two.

Forgery is the agent moving money it was never handed the means to move. Calling a rail directly with a key found in its context. Minting its own grant. Executing without a decision. Custody closes forgery. If the key lives outside the agent's process, prompt, memory, and tools, and execution happens behind the broker, there is nothing in the agent's reach to forge with. This class is solved by architecture, and the solution is verifiable. Either the credential is reachable from the agent's runtime or it is not.

Misdirection is different. A compromised agent submits a request that passes every check. Right payee format, on the allowlist, under the cap, plausible intent string. And it is still wrong, because it is not what you meant. The invoice it is paying was injected by an attacker. The vendor is real but the purchase is fabricated. The amount is within budget and entirely wasted.

This is not a new problem. It is the confused deputy, described by Norm Hardy in 1988. A program with legitimate authority is tricked into using that authority on behalf of someone who does not hold it. The deputy is not malicious and does not need to be. It is confused. An LLM agent is the most confusable deputy ever built, because its control flow is influenced by every piece of text it reads, and it reads text from strangers as part of its job.

Custody does nothing against misdirection, because misdirection never touches the key. The request arrives at the broker through the front door, correctly formed, fully in policy. The broker's honest answer is yes.

Name the boundary plainly. Locking the key out of the agent stops forgery. It does not by itself stop misdirection. Any vendor telling you their guardrail product solves both is describing the first and hoping you assume the second.

4. In-policy but wrong

Why can policy not just get better until misdirection closes?

Because policy answers a different question. Policy answers permission. May this class of actor spend this class of amount with this class of payee. It is a predicate over the request. And every fact in the request can be simultaneously true and beside the point.

Intent does not live in the request. It lives in the principal. "Pay $40 to api.stripe.com" is permitted or not on its face, but whether it is *wanted* depends on a fact the broker cannot see, namely what the human was actually trying to get done and whether this spend serves it. A poisoned tool result does not change the request's shape. It changes the reason the request exists. No predicate over the request detects that, no matter how expressive the policy language gets, because the missing information was never in the request to begin with.

Tighter policy shrinks the space of wrong-but-permitted actions. It never empties it. A cap of $5 per action still permits $5 of pure attacker-directed waste, times the velocity limit, times every agent you run. Rules bound misdirection. They do not close it. That distinction is the frontier, and being precise about it is more useful than another feature claiming otherwise.

5. The levers that move it

Three levers act on misdirection today. None eliminates it. Together they change its economics.

Bind consequential actions to an intent a human actually saw. This is what principal-approved grants do. Above a threshold, the spend waits, and a person approves that exact payee and amount, out of band, on a channel the agent does not control. For that spend, misdirection is closed, because the definition of misdirection is a gap between the action and what the principal meant, and here the principal looked at the action itself. The grant that results is single-use and bound to what was approved, so the approval cannot be stretched to cover anything else. The cost is human attention, which is exactly why this lever cannot cover everything.

Where you cannot put a human on every action, bound the blast radius. Auto-granted small spends are where misdirection lives, so make small mean small. Per-action ceilings, reservation-aware velocity caps,

allowlists and category restrictions shrink what a wrong-but-in-policy action can cost, and keep it inside amounts you can absorb or claw back. A confused deputy with a $5 ceiling, a $200 day, and twelve permitted payees is a contained problem. The same deputy with an open wallet is a headline.

Make it provable after the fact. The hash-chained audit records every decision with its structured reason, and the chain breaks if any record is altered, inserted, or removed. This does not prevent a misdirected spend. It guarantees the spend cannot hide. You can reconstruct exactly what was authorized, why the policy said yes, and what settled, and you can prove the record was not edited. Detection with proof turns misdirection from a silent leak into a bounded, investigable event, and it is what makes the other two levers tunable, because you can see where the losses actually happen.

The same three levers structure how we audit third-party setups. The Agent Payment Security Audit scores a deployment on eight dimensions. Single path, custody, mediated execution, intent-binding, no splitting, human approval, provable audit, and continuous verification. The first three close forgery. Intent-binding is the misdirection lever. The rest bound and prove. A setup can score closed on forgery and still leak through misdirection, and the audit says so explicitly rather than letting a green checklist imply safety it does not have.

6. The open problem

Here is the problem, stated as precisely as we can.

Verify that an action matches what the principal actually wanted, at machine speed, without a human on the loop every time.

Human approval solves the per-action version by construction and does not scale. Policy scales perfectly and cannot see intent. The open problem is the gap between them. A mechanism that carries the principal's intent forward into thousands of small autonomous actions with something stronger than a predicate over the request, and cheaper than a person.

Nobody has this. We do not have it. To our knowledge no shipping system has it, and most do not even name it.

Some directions look worth pursuing, offered here as research bets rather than claims. Provenance on the fields of a payment request, so a payee that entered the context from an untrusted tool result is distinguishable from one the principal supplied. Binding auto-granted spends to a task-level intent the principal approved once, then checking each action against that intent rather than against a global policy. Treating text from outside the trust boundary as tainted for the purpose of consequential parameters, the way taint tracking treats user input in web security. Each of these has an obvious failure mode, and each is still more honest than pretending caps are consent.

What we can say from the work so far is the shape of any real answer. It must be enforced, not advised, because a compromised agent ignores advice. It must fail closed. And it must be checkable at the boundary, in the broker's position, from information the broker can actually hold, or it will be another suggestion the deputy is free to be confused about.

That is where this field goes next. The payments layer is simply where it becomes measurable in money, which is why we work there.

This sits inside a larger reframing. Drexler's comprehensive AI services model treats advanced capability as a set of bounded, task-specific services, with agents as one constrained class of product rather than a trusted mind you hand the keys to. Dignum and Dignum make the companion case from the governance side, that agentic autonomy has to be paired with explicit models of accountability and institutional

structure rather than trusted to the agent's own reasoning. Enforcement mode is both arguments made concrete at the money layer. The agent is a service that can request a spend. It is not a mind that holds the wallet. And the confused deputy is what you find the moment you take the bounding seriously enough to put real money behind it.

7. Limitations

An honest list, in the spirit of the harness disclosure this project ships with.

- **Misdirection is bounded, not solved, and this paper does not solve it.** Auto-granted small spends remain exposed up to the caps and allowlist. That exposure is the paper's subject, not a footnote.
- **The enforcement property depends on a deployment contract.** Single path, mediated execution, custody outside the agent, no splitting path, continuous verification. Purse is a hard boundary only when it is the single path money can move. Deployed any other way, it is defense in depth, policy plus logging, not a wall.
- **The boundary is a property you maintain, not a state you reach.** The agent's capability surface widens the moment its runtime gains a tool or dependency, often silently, and any new tool can reopen a money path without touching policy. Continuous capability-surface monitoring is not part of the shipped library. Until it is, re-verification on every change is on the deployment.
- **The evidence base is one implementation.** The enforcement claims rest on a single open-source system and a CI-asserted demo over a mocked x402 rail, with the live path documented. That is a working proof, not a large-scale study, and no formal verification of the broker has been done.
- **Policy expressiveness is deliberately narrow.** Caps, windows, allowlists, categories, thresholds. Narrow is a choice, small surfaces are auditable, but it means some legitimate governance needs live outside what the current policy language can say.

8. Close

The industry is racing to make agents more capable, and capability now includes spending. The permission layer will not build itself, and it will not be built by asking models to behave.

Two claims come out of this work. Forgery is closed by architecture, custody and mediated execution, and any team running a spending agent without that boundary is trusting a system prompt with a wallet. Misdirection, the confused deputy at the payments layer, is open, bounded today by approval, blast radius, and proof, and closed by nothing.

Precision about that line is the contribution. The next contribution is whoever moves it.

Acknowledgments and references

The threat model behind this paper was sharpened in the open, in particular by @runs.dash.

- Hardy, N. "The Confused Deputy (or why capabilities might have been invented)." ACM SIGOPS Operating Systems Review, 22(4), 1988.
- Drexler, K. E. "Reframing Superintelligence: Comprehensive AI Services as General Intelligence." Technical Report #2019-1, Future of Humanity Institute, University of Oxford, 2019. The comprehensive-AI-services frame this paper instantiates at the payments layer.
- Dignum, V., and F. Dignum. "Agentifying Agentic AI." arXiv:2511.17332, 2025. The governance and accountability companion to the bounding argument, agentic autonomy paired with explicit institutional structure.
- Purse, the enforcement layer this paper is grounded in. `npm i @olurabian/purse`, threat model and deployment contract in the project README.
- The Agent Payment Security Audit, the eight-dimension diagnostic. prompts/agent-payment-security-audit.md.
- The x402 governed-agent proof, five scenes, asserted in CI. `npm run demo:x402`, live-path notes under examples/x402/.