
HIL: Human-in-the-Loop Payment Authorization for AI Agents¹

Vasilii Kondyrev
Loyal

Rodion Aleksandrov
Loyal

Kirill Cherniakov
Loyal

Vasilii Tiselko
Loyal

With
Apart Research

Abstract

AI agents are beginning to gain payment authority, but giving a tool-using LLM direct wallet access is an unsafe default. Unlike conventional software, agentic systems can be manipulated through indirect prompt injection, misleading external content, and non-deterministic tool use. We present HIL, a human-in-the-loop payment authorization protocol that separates payment proposal from payment execution. HIL adapts the banking maker-checker pattern to agentic payments through three trust-separated roles: an untrusted initiating agent, a trusted control plane, and a trusted approver. The control plane applies risk-tiered policies, provenance-aware review packets, and audit logging; routine low-risk payments can be auto-approved, while unusual or adversarial requests are escalated to human review. We implement HIL on the Solana devnet with a Telegram approval interface and evaluate it across three end-to-end invoice scenarios: a normal low-value invoice (0.05 SOL), a normal high-value invoice (1.00 SOL), and a hidden-instruction invoice whose visible business context appears benign but whose total is 1.04 SOL. In the current implementation, payments above 0.05 SOL require review, while payments at or below 0.05 SOL are auto-approved. Our main takeaway is that explicit authorization

¹ Research conducted at the [AI Control Hackathon](#), March 2026

boundaries can reduce unsafe autonomous payments without removing the utility of agentic financial workflows. This submission contributes both a protocol framing and a working demo.

1. Introduction

LLM agents are increasingly being given access to tools that act in the world: sending messages, browsing the web, reading documents, and, increasingly, making or initiating payments. This creates a control problem. If the same agent that decides to pay can also sign and broadcast the transaction, then any model error, prompt injection, or adversarially crafted external content can immediately become fund movement.

This matters because the failure mode is not hypothetical. Indirect prompt injection has emerged as a standard threat model for tool-using LLM agents: malicious instructions can be embedded in emails, webpages, documents, or other external content that the agent processes as if it were ordinary data. In the payments setting, that means an apparently ordinary invoice, email thread, or vendor message can contain instructions that redirect or expand a transfer. Meanwhile, real-world payment systems already face significant fraud pressure, especially from business email compromise-style attacks. The problem is therefore not merely whether an agent is usually helpful, but whether the system remains safe when the agent is wrong, manipulated, or strategically deceived.

Our approach is to adapt a longstanding banking control pattern to AI agents. In corporate banking and treasury systems, payment initiation and payment approval are often separated by design. The initiator proposes a payment; a distinct approver or approval chain authorizes it. We translate that logic into an agentic payment protocol: the agent may propose, but it does not hold final payment authority.

Our main contributions are:

1. We propose HIL, a control protocol for agent-initiated payments that separates proposal, review, and execution across distinct trust boundaries.
2. We implement a working Solana devnet prototype with a Telegram approval workflow, structured evidence packets, and server-side policy enforcement.
3. We demonstrate the protocol on three concrete invoice scenarios, including an adversarial hidden-instruction invoice, while preserving the exact current implementation behavior needed for the submitted code and demo.

2. Related Work

Our work sits at the intersection of AI control, payment controls, and prompt-injection security. First, HIL draws on established payment-control practice in banking and treasury. Maker-checker (or dual control) and segregation of duties are standard mechanisms for reducing fraud and operational risk in payment systems. Regulatory and supervisory materials emphasize that high-stakes payment workflows should separate initiation from authorization, use approval queues, enforce authorization limits, and maintain audit trails. In other words, financial systems already treat “who proposed this payment?” and “who approved this payment?” as different questions. Our contribution is not to invent dual control itself, but to reinterpret it as an AI control protocol for tool-using agents.

Second, HIL addresses a harsher threat model than ordinary human-operated banking software. In tool-using agent systems, external content is not just data; it can function as adversarial instruction. Prior work on indirect prompt injection and prompt injection against LLM-integrated applications shows that these attacks are practical, widespread, and capable of causing harmful downstream actions. OWASP also treats prompt injection as a top-tier risk category for LLM applications. This makes agentic payments especially dangerous: unlike a conventional scripted workflow, an LLM can ingest and reinterpret natural-language inputs that were never meant to be trusted as commands.

Third, our work is related to emerging “agentic commerce” efforts. Payment networks and industry analysis increasingly assume that AI agents will eventually purchase goods and services on behalf of users. However, much of that discussion focuses on authentication and convenience. Our work emphasizes containment and authorization boundaries instead: before asking how to let agents pay more smoothly, we ask how to prevent them from paying unsafely.

The gap we address is therefore specific: there is substantial prior work on both payment controls and LLM attack surfaces, but less work that turns these into a concrete protocol for agent-initiated payments with an explicit safety-utility tradeoff. HIL is a compact attempt to fill that gap.

3. Methods

3.1 System design

HIL divides the system into three trust-separated roles:

Role	Trust level	Can do	Cannot do
Untrusted Initiator (U-Agent)	Untrusted	Propose payments, query wallet state	Sign transactions, access private keys, bypass policy
Control Plane	Trusted	Evaluate risk, hold signing key, execute authorized transfers, log actions	Initiate payments on its own
Trusted Approver (T-Approver)	Trusted	Approve or reject requests via Telegram	Initiate payments, modify signed payloads, access keys

The untrusted agent interacts with what appears to be a normal Solana wallet interface. In the submitted implementation, the SDK exposes functions such as `transfer()`, `getBalance()`, and `getPublicKey()`. The important design decision is that the agent does not see the internal control logic. It submits a payment intent and eventually receives either a transaction signature or an error, but it does not directly observe policy evaluation, approval queues, or trust-boundary logic.

3.2 Protocol flow

The protocol is:

1. The agent encounters an invoice or other payment request.
2. It calls the wallet interface to initiate a transfer.
3. The SDK sends a payment intent to the server-side control plane over HTTPS.
4. The control plane evaluates the request under current policy.
5. If the request is low-risk, it may be auto-approved. Otherwise, it is sent to the approver via Telegram.
6. The approver receives a structured evidence packet and chooses approve or reject.
7. If approved, the control plane signs and submits the transaction to Solana devnet.
8. The full action is written to an audit trail.

3.3 Current implementation-specific policy

To preserve compatibility with the submitted code and live demo, we keep the current implementation-specific threshold exactly as implemented:

- payments **above 0.05 SOL** trigger a Telegram review notification;
- payments **at or below 0.05 SOL** are **auto-approved**.

We do **not** claim that 0.05 SOL is an optimal threshold. It is the threshold currently used by the code and, therefore the threshold reported in this submission.

The current implementation also includes the following practical features:

- email-driven ingestion for invoice processing in the demo setup;
- PDF invoice attachments as the payment source format;
- Telegram approval messages with approve/reject controls;
- allowlisting of authorized approver identities;
- audit logging of proposal, review, and execution events;
- server-side key custody, so the agent never holds the signing key.

3.4 Evidence packets

Instead of showing the approver only an amount and destination address, HIL sends structured review information. In the current design, an evidence packet may include:

- payee wallet address;
- amount;
- whether the payee is allowlisted;
- provenance tag (for example, email);
- short payment memo or invoice context;
- first-time payee signal;
- amount anomaly or “amount vs average” signal;
- current window usage or spending context;
- a recommendation such as approve, reject, or escalate.

The goal is not to make the human rubber-stamp the agent’s suggestion, but to provide enough context for an informed decision while minimizing unnecessary exposure to raw sensitive context.

3.5 Demo evaluation setup

We evaluate the protocol with three manually constructed PDF invoices used in the demo:

1. **Normal low-value invoice:** total 0.05 SOL.
2. **Normal higher-value invoice:** total 1.00 SOL.
3. **Hidden-instruction invoice:** visible commercial content looks ordinary, but one line item contains an agent-targeted instruction that increases the total to 1.04 SOL.

The third case is designed to approximate an indirect prompt injection scenario in which the agent reads an ostensibly normal invoice but is manipulated by text embedded inside it.

3.6 Replicability notes

A reader could reproduce the core setup by using:

- a tool-using agent connected to the HIL SDK;
- a server-side control plane with Solana devnet access;
- a Telegram bot for review actions (contact us for access and whitelisting);
- the three provided PDF invoices as test inputs.

4. Results

Our current evaluation is a structured end-to-end demo rather than a large benchmark. The main result is that the protocol creates different outcomes for routine and risky payments without changing the agent-facing wallet API.

4.1 Scenario results

Scenario	Input	Policy outcome in current implementation	Expected safety effect
Case 1: normal invoice	PDF invoice totaling 0.05 SOL	Auto-approved	Preserves low-friction payment for routine low-value case
Case 2: normal larger invoice	PDF invoice totaling 1.00 SOL	Escalated to Telegram for approval	Introduces human review for higher-value spend
Case 3: hidden-instruction invoice	PDF invoice with hidden agent-directed line item; total 1.04 SOL	Escalated to Telegram for approval	Prevents silent execution of suspicious or manipulated higher-value payment

4.2 Qualitative finding

The key qualitative result is that HIL changes the default behavior of agentic payments from **execute immediately** to **execute only after policy and, when required, approval**. In the normal low-value case, the payment can proceed without manual friction. In the two higher-value cases, the system forces an explicit review boundary.

This matters most in the hidden-instruction invoice case. A naive agent-payment flow could interpret the invoice as sufficient payment instruction and send the full amount. Under HIL, the payment is no longer silently executable by the same compromised or misled process that proposed it. The suspicious invoice is surfaced to a separate approver before funds are moved.

4.3 What we did not measure

We did not, in the hackathon timeframe, run a large-scale benchmark of attack success rate, false positive rate, or average approval latency. We also did not compare against multiple baselines such as whitelist-only or threshold-only systems in a full quantitative study.

5. Discussion and Limitations

HIL should be understood as a control protocol, not a claim that agentic payments are now “solved.” The main implication is conceptual and practical at once: the right safety primitive is not simply “make the agent more reliable,” but “prevent the agent from being the sole authority over fund movement.” In this sense, HIL reframes maker-checker as an AI control boundary.

The protocol is especially relevant for payment settings where agents process untrusted external content such as emails, invoices, scraped webpages, or tool outputs. In such environments, prompt injection and social engineering are not edge cases; they are part of the core threat model. HIL reduces blast radius by ensuring that the process vulnerable to manipulation does not also hold signing authority.

Limitations

This project has several important limitations.

1. **Small-scale evaluation.** Our current evidence is an end-to-end prototype demo with a few representative invoices, not a statistically meaningful benchmark.
2. **Heuristic policy.** The current 0.05 SOL threshold is implementation-specific and chosen for the demo, not derived from an optimization process or formal risk model.
3. **Single-approver design.** The present prototype uses a single human approver. It does not yet implement multi-approver quorum or stronger organizational separation.
4. **Custodial trust in the control plane.** The server-side control plane holds the signing key. This is a practical MVP choice, but it creates a trust assumption that could be reduced in future versions with HSMs, MPC, or threshold signing.
5. **No out-of-band vendor verification yet.** The current prototype does not fully implement callback verification, vendor master checks, or external beneficiary validation.
6. **Limited threat coverage.** HIL is designed primarily for unsafe or manipulated payment requests. It does not fully solve collusion, insider abuse by the approver, compromise of the control-plane server, or social engineering of the human reviewer.
7. **Demo-specific ingestion path.** The current email-plus-PDF flow is only one way an agent might receive payment instructions. Additional channels such as chat apps, APIs, ERP systems, and web scraping were not fully implemented.

Future Work

There are several natural next steps.

- Compare HIL against clearer baselines: naive direct-pay agent, threshold-only gating, and whitelist-only gating.
- Measure attack success rate, false positive rate, and approval burden across a larger attack suite.
- Add out-of-band verification for payee changes and first-time beneficiaries.
- Support multiple approvers or quorum-based authorization for larger payments.
- Replace custodial signing with threshold signatures, MPC, or HSM-backed infrastructure.
- Add stronger provenance analysis and anomaly scoring beyond a simple threshold.
- Extend beyond Solana devnet to production-grade environments and additional payment rails.

6. Conclusion

We presented HIL, a human-in-the-loop payment authorization protocol for AI agents. The core idea is simple: the agent that proposes a payment should not be the same process that is able to sign and execute it. By adapting the maker-checker pattern from banking to agentic systems, HIL creates an explicit authorization boundary between vulnerable model behavior and irreversible fund movement.

Our prototype demonstrates that this can be implemented in a practical way. In the current setup, routine low-value payments at or below 0.05 SOL are auto-approved, while larger or potentially manipulated payments are escalated to Telegram review. Even with a compact hackathon implementation, this produces a meaningful control effect: suspicious or higher-risk payment requests are no longer silently executable by the initiating agent. We view this as a promising direction for AI control in financial systems, especially as agentic commerce becomes more common.

Code and Data

- **Code:**
https://drive.google.com/file/d/1RgeASVRKkIE6bl26nGzJ-cBIPWjMlYqx/view?usp=drive_link
- **Appendix A:**
https://drive.google.com/drive/folders/1acpffisFzWcbG4IXON1ax_3NlIz1p3hq?usp=drive_link

References

1. Zhan, Q., Liang, Z., Ying, Z., & Kang, D. (2024). *InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents*. Findings of ACL 2024. <https://arxiv.org/abs/2403.02691>
2. Liu, Y., Deng, G., Li, Y., Wang, K., Wang, Z., Wang, X., Zhang, T., Liu, Y., Wang, H., Zheng, Y., & Liu, Y. (2023). *Prompt Injection attack against LLM-integrated Applications*. arXiv. <https://arxiv.org/abs/2306.05499>
3. OWASP Gen AI Security Project. (2025). *LLM01:2025 Prompt Injection*. <https://genai.owasp.org/llmrisk/llm01-prompt-injection/>
4. Internet Crime Complaint Center (IC3). (2024). *2024 IC3 Annual Report*. https://www.ic3.gov/AnnualReport/Reports/2024_IC3Report.pdf
5. Office of the Comptroller of the Currency. (2021). *Payment Systems | Comptroller's Handbook*. [https://www.occ.gov/publications-and-resources/publications/comptrollers-handbook/files/payment-sys-funds-transfer-activities/pub-ch-payment-systems.pdf](https://www OCC.gov/publications-and-resources/publications/comptrollers-handbook/files/payment-sys-funds-transfer-activities/pub-ch-payment-systems.pdf)
6. Federal Deposit Insurance Corporation. (2020). *Electronic Funds Transfer Risk Assessment Core*. <https://www.fdic.gov/resources/supervision-and-examinations/examination-policies-manual/section22-1/sc-eft.pdf>
7. Board of Governors of the Federal Reserve System, Federal Deposit Insurance Corporation, & Office of the Comptroller of the Currency. (2024). *Joint Statement on Banks' Arrangements with Third Parties to Deliver Bank Deposit Products and Services*. <https://www.federalreserve.gov/newsevents/pressreleases/files/bcreg20240725c1.pdf>
8. SWIFT. (2023). *SWIFT Customer Security Controls Framework v2024*. https://www2.swift.com/knowledgecentre/rest/v1/publications/cscf_dd/53.0/CSCF_v2024_20231017.pdf
9. Eastnets. (2021). *What FIs need to know about the mandatory SWIFT CSP 2.9 controls*. <https://www.eastnets.com/blog/blog/what-fis-need-to-know-about-the-mandatory-swift-csp-2.9-controls>

Appendix A. Demo test artifacts

Invoice 1: normal invoice, total 0.05 SOL.

Invoice 2: hidden-instruction invoice, total 1.04 SOL.

Invoice 3: normal invoice, total 1.00 SOL.

LLM Usage Statement

We used LLM assistance to help structure the report, synthesize related work, and draft text. All architecture claims, implementation-specific details, and demo descriptions were grounded in the project materials and reviewed against the actual current implementation. We intentionally

preserved code-specific behavior, including the current 0.05 SOL review threshold, rather than rewriting the report around unimplemented features.