Crash Recovery After a Stripe Test Payment

A case study of tool interruption and evidence boundaries

Ayobami John Haastrup

23 October 2026

Abstract

A durable agent can recover its execution state after a payment has succeeded without having returned the tool result. We report a bounded integration study of this interval using Cloudflare PiHarness, a separate payment ledger and the Stripe test API. A deterministic model substitute requested one fixed operation in each of two scenarios: a local process kill and a hosted Durable Object reset. Both archived provider readbacks record a succeeded GBP 1.00 test-mode PaymentIntent. Following recovery, each scenario records one tool execution and an interrupted unsafe-tool result. The hosted runner nevertheless failed its original assertion because a marker written immediately before the reset was absent. Subsequent inspection of the same operation recovered matching receipts and unchanged counters; it did not constitute a fresh successful run of the corrected runner. A webhook control recorded three deliveries and one stored payment marker, without testing downstream fulfilment. The case illustrates how execution status, external effects and acceptance assertions can disagree. Its contribution is an inspectable implementation and failure record, not a new delivery guarantee, a model benchmark or a statistical reliability estimate.

1. Question and scope

The motivating application was an automated publishing system. If a process stops after sending a request, restarting the process does not establish whether the publication happened. To examine one tractable boundary, this study substituted a Stripe test payment for a social-platform write and asked: **what survives when the provider has accepted an operation but the agent's tool has not returned?**

The distinction between provider acceptance and tool completion matters here. The tested interruption occurred after a receipt was available to the application. It was not a test of losing Stripe's response before the application could record it. The latter is a separate uncertainty boundary.

Three questions organise the report:

1. Does the recorded unsafe tool run again after the injected interruption?
2. Which observations support recovery when the test runner's original assertion fails?
3. What does a deduplicated webhook record establish about downstream work?

The evidence consists of two provider-backed scenarios, fixture controls, source inspection and exported readbacks from 22 October 2026. Each provider scenario was observed once. The report

preserves the original hosted failure and distinguishes later interpretation from the original acceptance result. It does not claim a previously unknown Pi defect or priority over existing recovery research.

2. Related work and contracts

ToolEmu uses language models to emulate tool environments and evaluate potentially harmful agent actions [1]. Its separation of the environment and evaluator motivates stating what an evaluation can actually observe. Here, the payment boundary used the Stripe test API, while tool selection was deterministic. This difference narrows the kind of evidence collected; it does not make this case study a stronger model evaluation.

The tau-bench evaluation compares final database state with a target state and measures consistency across repeated trials [2]. It also explains that a correct final state can miss a policy violation. Our report likewise treats a terminal execution status as distinct from a justified claim about an external effect. We did not run enough repeated trials to estimate consistency.

LIMBO studies duplicate effects under ambiguous tool outcomes across models, harnesses and simulated service contracts [3]. RetryBench separates permission to retry from realised recovery and includes configured framework controls [4]. These works already cover the broader failure class. The present report adds a small, version-pinned integration record and a documented acceptance-assertion correction. It does not claim to resolve the broader problem or establish a new recovery policy.

Cloudflare documents that interrupted replay-safe tools can run again, whereas an unsafe tool produces an interrupted result for the model [5]. Stripe's API v1 idempotency contract retains the first executed response for a key and checks parameter consistency; retention is not an indefinite application ledger [6]. These are distinct responsibilities. Preventing automatic replay of one tool call does not prevent a later caller from proposing a different business operation.

3. System and evidence

3.1 Pinned implementation

The original execution/source snapshot is commit `ef235c267da51867b042a43e0cc8323781c19991` in the public laboratory repository [8]. The lockfile resolves Agents 0.28.1, Pi Durable 1.3.0, Pi AI 1.3.0 and Wrangler 4.151.1. These versions define the tested baseline. Documentation consulted during manuscript preparation may describe later releases.

The agent uses the real PiHarness with a deterministic faux model that requests the fixed payment tool. This controls the tool sequence without paid model inference. It does not measure how a production language model interprets interruption, chooses a new identifier or seeks further authorisation.

The hosted implementation separates three components. The agent Worker maintains the Pi session and has no Stripe API key. A private gateway and PaymentLedger hold test payment authority and record attempts and receipts. A webhook receiver and WebhookLedger verify incoming events and retain delivery records. The runner observes the gateway through an authenticated control route; Pi does not automatically reconcile Stripe state in this experiment.

3.2 Application admission and provider calls

The hosted gateway accepts one configured operation, amount and currency. Before an outbound payment request, a storage transaction records that the attempt has begun. A prior successful operation returns its receipt. A prior unresolved attempt is refused rather than automatically retried. The outbound request carries an operation-derived Stripe idempotency key and operation metadata.

This policy is conservative. An attempt can remain unresolved even when no payment was created, so suppression of another request is not evidence of eventual completion. The present provider scenarios begin recovery after a successful receipt was available and therefore do not evaluate that liveness cost under a lost provider response.

The field `stripeCreateRequests` is incremented by application code before the HTTP call. It counts attempted dispatches at that boundary, not independently authenticated Stripe requests. Provider object readback supplies a separate observation. The archived hosted request code does not set a `Stripe-Version` header, and the evidence does not retain the account's effective API revision. This is a reproducibility limitation.

3.3 Evidence provenance

The records distinguish the runner's observations, the application ledger and a separate Stripe access path under the same owner's account. The latter is outside the execution runner, but it is not an independent third-party audit. Public readers can inspect the exported JSON; re-querying the payment objects requires account-authorised access.

A later publication snapshot, `74a4521da15f4bec16e207e91323b426ef02b32d`, contains a SHA-256 manifest and an offline checker [8]. For this manuscript, 49 retrieved files were checked against their Git blob identifiers. The repository checker then passed 250 assertions covering 45 manifested files and selected cross-file relationships. No provider requests were made during that audit. Hash agreement establishes byte identity and internal consistency, not that the historical provider responses are authentic.

4. Procedure

4.1 Local crash and restart

The local runner starts Wrangler and a scoped gateway, admits the operation and waits until the external effect has been observed. The unsafe tool remains blocked before returning. The runner kills its own Wrangler process group with SIGKILL and restarts against the same persistence. It then inspects the recovered result, attempts duplicate submission of the same operation and checks the existing receipt and counters.

The local record spans 19:27:27 to 19:27:54 UTC on 22 October. It records one tool execution, one gateway request and one Stripe creation attempt, followed by an interrupted-tool result and rejected duplicate admission [E1].

4.2 Hosted reset and revised interpretation

The hosted runner waits for a successful provider receipt and then calls `abort()` on the Durable Object state. Cloudflare describes this operation as an immediate object reset [7]. It is an injected reset, not an observation of a naturally occurring outage.

The original hosted assertion expected a marker written immediately before the abort to remain available. It did not. The runner recorded failure after observing the deliberate abort response and a different object instance [E2]. The record does not isolate why that marker failed to persist.

A separate recovery-readback procedure subsequently inspected the same operation. Its acceptance conditions were a changed instance, a terminal interrupted-tool result, matching agent and gateway receipts, rejected duplicate admission and unchanged counters [E3]. These conditions were applied after the original failure; they were not preregistered. The corrected full hosted runner was not rerun on a fresh payment.

The readback script contains three additional receipt/counter checks following duplicate admission. The export retains the final state rather than each intermediate sample. Accordingly, this report relies on the exported state and treats the script as evidence of the intended checks, not as three separately inspectable observations.

4.3 Controls

A before-request fixture interrupts the unsafe tool before contacting the gateway. A read-only control compares the same interruption under unsafe and safe replay settings. Neither control creates a Stripe payment [E4, E5].

For the hosted webhook, Stripe's original success event is followed by two signed HTTP replays generated by the test runner. An invalid-signature request is also exercised. The duplicates reproduce the original payload; they do not test Stripe's automatic retry schedule, different events for one payment or arbitrary delivery order [E6].

5. Observations

Table 1 separates provider-backed observations from fixture and webhook controls. Counts are reported per case. Combining these heterogeneous cases into a success rate would give a misleading denominator.

Case	Execution or delivery count	Recorded outcome
Local Stripe test API	Tool 1; gateway 1; create attempt 1	Interrupted tool; same-operation duplicate rejected
Hosted Stripe test API	Tool 1; gateway 1; create attempt 1	Original assertion failed; later same-operation readback passed
Before-request fixture	Tool 1; gateway 0; create attempt 0	Interrupted tool; no payment attempt
Read-only unsafe control	Tool calls 1	Interrupted
Read-only safe control	Tool calls 2	Completed
Hosted webhook control	Deliveries 3; event IDs 1; stored marker 1	Invalid signature rejected; downstream action unmeasured

Table 1. Recorded outcomes at the tested boundaries. Application counters are not provider request logs. Evidence: E1-E6.

5.1 Provider objects and recovered execution

The exported Stripe readback records two PaymentIntents with status `succeeded`, amount and amount received equal to 100 minor units, currency `gbp` and `livemode=false` [E7]. Their operation metadata matches the local and hosted runs. The exported list at 19:45:57 UTC contains those two objects and reports `has_more=false`. This is an observation of that query at that time, not an exhaustive guarantee about all future effects.

The hosted initial record and later readback identify the same PaymentIntent. The post-reset instance differs from the pre-reset instance, while the agent and gateway receipts continue to identify that payment. Terminal status is `done`, but the transcript reports an interrupted tool. Thus terminal status describes the end of the harness operation; it does not by itself communicate whether the external payment succeeded.

No extra tool execution or creation attempt appears in either recorded recovery. Duplicate admission uses the same operation identifier. The study does not test an adversarial or mistaken caller requesting a different identifier for the same intent.

5.2 The failed assertion remains a failed assertion

The hosted failure and subsequent readback are compatible observations. The first says the runner's marker-based acceptance condition was not satisfied. The second says other surviving observations support recovery of the same interrupted operation without another recorded payment attempt. Neither observation should overwrite the other.

An earlier hosted preflight also failed because the runtime rejected the gateway's redirect setting. Its diagnostic records zero Stripe creation attempts. The gateway was corrected before the payment run. These are laboratory corrections, not evidence of a defect in Pi's replay contract [E8].

5.3 A marker is not a fulfilled order

The webhook export records delivery count increasing from one to three, with one event identifier and one payment identifier. The source assigns `businessConsequences = 1` inside the stored record [E6, E9]. There is no measured email, entitlement or fulfilment call behind this field. Its value therefore establishes convergence of that record, not exactly-once execution of an external business action.

6. Interpretation and limits

The results are consistent with the documented unsafe-tool replay contract in the tested scenarios. They also show why a reader needs more than a single pass/fail flag. A failed acceptance assertion can coexist with a succeeded provider object; a completed harness operation can contain an interrupted tool; and a field named for business consequences can represent only a database marker.

For application engineering, the useful pattern is to retain the original operation identity and its observations before deciding whether to retry. This case does not identify which defensive layer is causally necessary. Unsafe-tool handling, application admission, durable attempt recording and provider idempotency were used together, without an ablation that removed each layer. In particular, a single creation attempt does not demonstrate Stripe deduplicating repeated creation requests.

Several limits constrain the conclusions. The two payment scenarios are purposively selected, occur within one owner's test account and do not estimate a failure probability. The local process kill and hosted abort are different interventions. Test mode does not exercise real-money settlement or customer authentication. The model substitute excludes language-model decision quality. The real-provider boundary excludes a lost response before receipt persistence, retention expiry, changed permission and concurrent real-provider recovery. Mocked tests of some such cases are not substitutes for provider observations.

Source provenance is also incomplete. The final snapshot includes the corrected runner, while the failed execution used its earlier assertion. Deployment version identifiers are recorded, but no immutable attestation maps every executed deployed byte to that final source snapshot. The effective Stripe API revision and intermediate repeated readback samples are missing. The owner and the same AI-assisted workflow conducted execution and interpretation; no external reproduction has been obtained.

These limitations leave the work as a descriptive integration case study. A fresh full-run reproduction would strengthen the operational record. A comparative research claim would require an explicit alternative, repeated observations and a failure boundary that the comparison actually exercises. Neither requirement is met by relabelling the present archive.

7. Reproducibility and availability

The public repository [8] supplies the implementation, locked dependencies, failed records and evidence guide. The companion website article [9] provides an accessible account. The evidence commit and publication snapshot above should be used when inspecting this report; a moving default branch is not the reference baseline.

From the publication snapshot, the archive check is:

```
node scripts/verify-evidence.mjs
```

This command performs no payment or network operation. Fresh provider reproduction requires an isolated account configuration, new operation identifiers, test credentials and new deployment. The historical public laboratory endpoints were disabled and hosted payment/signing secrets removed after collection. The archived configuration is intentionally scoped and is not a general public payment demo.

For account-authorised inspection, the local PaymentIntent is `pi_3UTReLGbPfXt7ec50W8SyVvv` and the hosted PaymentIntent is `pi_3UTRr4GbPfXt7ec50z2eLbbo`. These public identifiers are audit pointers, not access credentials or independently verifiable signatures.

8. Conclusion

The two archived recoveries retained one recorded tool execution and matching Stripe test-payment evidence after an interruption before tool completion. The hosted runner's original assertion remained failed, and later readback clarified the same operation without providing a fresh full-run pass. The webhook result established one stored payment marker after repeated deliveries, with downstream fulfilment outside the experiment. Reporting these boundaries together makes the case useful without turning a small integration record into a general reliability claim.

Authorship and assistance

The author is Ayobami John Haastrup, the repository owner known publicly as ayobamih. ChatGPT/Codex assisted with implementation, execution orchestration, source inspection, evidence checking and manuscript drafting. The same assisted workflow conducted the manuscript audit; it was not an independent review. The human author is responsible for the manuscript. No Cloudflare or Stripe endorsement is claimed. This manuscript has not been peer-reviewed.

References

1. Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison and Tatsunori Hashimoto. *Identifying the Risks of LM Agents with an LM-Emulated Sandbox*. arXiv:2309.15817v2, 2024. Paper.
2. Shunyu Yao, Noah Shinn, Pedram Razavi and Karthik Narasimhan. *tau-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains*. arXiv:2406.12045v1, 2024. Paper.
3. Jiapeng Li. *Where Does Exactly-Once Live? Model, Harness, and Tool-Contract Effects on Duplicate Side Effects in LLM Agents*. arXiv:2609.29095v1, 2026. Paper.
4. You Liu, Yue Liu and Quanchao Lu. *RetryBench: Separating Retry Permission from Recovery Outcomes in GPT Tool Use*. arXiv:2610.15884v1, 2026. Paper.
5. Cloudflare. *Pi extensions: Mark tools replay-safe*. Documentation consulted 23 October 2026. Documentation.
6. Stripe. *Idempotent requests*. API v1 contract consulted 23 October 2026. Documentation.
7. Cloudflare. *Durable Object State: abort*. Documentation consulted 23 October 2026. Documentation.
8. ayobamih. *Pi Durable Stripe Crash Lab*. Original execution/source snapshot and subsequent publication snapshot, 2026. Execution/source snapshot; publication snapshot.
9. ayobamih. *The payment succeeded. The agent crashed.* AI Work Accountability, 2026. Companion article.

Appendix A. Evidence index

All E-records are pinned to the original execution/source snapshot, except the explicitly identified publication checker. Each link identifies a concrete file rather than a repository search.

- **E1:** Local Stripe result.
- **E2:** Original hosted assertion failure.
- **E3:** Same-operation recovery readback and readback procedure.
- **E4:** Before-request fixture.
- **E5:** Read-only replay control.
- **E6:** Webhook deliveries and marker.
- **E7:** Separate owner-account Stripe readback.
- **E8:** Preflight diagnostic and execution record.
- **E9:** Webhook handler and gateway.
- **E10:** Offline evidence checker and manifest.