RESPONSIBLE TESTING REPORT

MultiHopper Agentic Flow Reproducible Bugs and Documentation Fixes

A reproducible review of the published create, prepare, sign/broadcast, confirm-broadcast, monitoring, idempotency, and webhook flows.

Bounty	Break It Before Users Do
Prepared	20 June 2026
Contact	X: @rdv_kaya_57, TG: @rdvkaya57
Evidence	Included in Appendix A of this report. Private source/evidence package available on request.

Scope note: no bounty-approved API key or funded test wallet was included in the supplied material. This draft reports public documentation-contract and executable reference-code findings only.

Executive summary

I approached this as an integrator rather than as a prose-only documentation review. I started from the public LLM documentation index, implemented the published TypeScript transfer sequence, and then turned each failure into a small regression test. I also built a stricter state-machine version of the client so I could compare the documented behavior with a fail-closed implementation.

The most important problems are concentrated at the boundaries between steps: the documentation index sends agents to the wrong host, the reference client treats rejected HTTP requests as success, and the advertised expiry/resume loop does not actually retry. The monitoring example also expects a response envelope that conflicts with the OpenAPI schema. Separately, the webhook example is unsafe in a normal Express setup because it neither preserves the signed raw body nor validates signature length before `timingSafeEqual`.

These are agent-relevant failures, not UI preferences. An autonomous client following the published examples can fail before integration begins, continue after an unsuccessful keeper-funding confirmation, stop instead of resuming after blockhash expiry, or lose reliable status monitoring after transactions have already been broadcast.

What I built and tested

I built a TypeScript harness around the documented lifecycle:

```
create → prepare → sign/broadcast keeper funding → intermediate confirm-broadcast →
broadcast remaining groups → final confirm-broadcast → monitor
```

The corrected harness adds the controls I expected the reference implementation to have:

- test-key-only configuration and explicit environment approval;
- strict keeper → route → orchestrator → session ordering;
- preservation checks for existing server partial signatures;
- checked HTTP responses for every API request;
- blockhash-aware Solana confirmation;
- fail-closed behavior when either RPC or API confirmation fails;
- bounded re-prepare attempts for recognized expiry/RPC failures; and
- cluster mismatch detection before broadcasting.

Test environment

- Windows
- Node.js 24.14.1
- TypeScript
- `@solana/web3.js` 1.98.4
- Vitest 4.1.9

Validation result

- TypeScript build: passed
- Regression tests: **9/9 passed**
- Dependency audit: **0 known vulnerabilities**
- Public documentation audit date: **20 June 2026**

No bounty-approved API key, funded test wallet, token balance, or RPC cluster was included in the material I received. I therefore did not create or broadcast a transfer. Findings below concern the live public documentation contract and executable reference-code behavior. I make no unsupported claim about the authenticated backend implementation.

Finding 1 — The LLM documentation index sends agents to the wrong domain

Severity: Documentation blocker

Affected surface: https://dev-docs.multihopper.com/llms.txt

What happens

The documentation tells agents to fetch `llms.txt` first and use it to discover the available pages. The index currently contains 19 page links, and all 19 point to `https://docs.multihopper.com/...` rather than the working `https://dev-docs.multihopper.com/...` host.

The target host does not serve the MultiHopper developer pages. For example, its `llms.txt` response contains unrelated Notion product content, while API Markdown and OpenAPI paths return "File not found." An agent that follows the official discovery index is therefore routed away from the documentation it is supposed to consume.

Reproduction

1. Fetch https://dev-docs.multihopper.com/llms.txt.
2. Open any linked URL, such as https://docs.multihopper.com/quickstart or https://docs.multihopper.com/api-reference/openapi.json.
3. Compare it with the same path on `dev-docs.multihopper.com`.

Automated audit result:

```
indexLinks: 19
indexLinksPointingAtWrongHost: 19
```

Expected

Every index link resolves to the corresponding MultiHopper page on the active documentation host.

Actual

Every indexed link uses a host that does not serve the referenced MultiHopper content.

Impact

This breaks the documented agent/MCP discovery workflow at its first step. It also encourages hallucinated endpoint recovery because the agent sees titles in the index but cannot retrieve their source pages.

Proposed fix

- Regenerate `llms.txt` with https://dev-docs.multihopper.com as the canonical host.

- Add documentation CI that requests every indexed link and verifies both HTTP success and the expected page title.
- Declare a canonical documentation base URL once and reuse it for sitemap, OpenAPI, MCP, and LLM-index generation.

Finding 2 — The reference TypeScript flow continues after a rejected keeper-funding confirmation

Severity: High

Affected surface: Agentic Integration guide, TypeScript `confirmBroadcast` helper

What happens

The published helper awaits `fetch()` but never checks `response.ok`. JavaScript fetch resolves normally for HTTP 400, 409, and 5xx responses. The caller therefore treats a rejected intermediate `confirm-broadcast` request as success and begins broadcasting route initialization transactions.

This contradicts the guide's stated safety rule: keeper funding must be recorded immediately, before any dependent group is broadcast, to prevent duplicate funding during resume.

Reproduction

1. Use the published helper with an HTTP response such as `400 MH_070`, `409 MH_072`, or `503 MH_091`.
2. Provide a bundle containing `keeperFundingTx` and at least one `routeInitTx`.
3. Observe that the confirmation promise resolves and the route broadcast begins.

The regression suite reproduces the behavior with a real `Response` object returning HTTP 400. The documented helper resolves instead of throwing. The corrected state machine test proves that only `keeperFundingTx` is broadcast when intermediate confirmation fails.

Expected

Any non-2xx intermediate confirmation stops the bundle before the first route transaction.

Actual

HTTP failure is silently ignored and the flow advances.

Impact

A routine validation error, idempotency conflict, transient outage, or backend error can leave keeper funding on-chain but unrecorded by the API while the agent continues into dependent deployment work. This undermines the two-step confirmation design and increases the risk of partial state or duplicate keeper funding during recovery.

Proposed fix

Use one checked request helper for every API call:

```
async function checkedJson<T>(response: Response): Promise<T> {
  const body = await response.json();
  if (!response.ok) {
    const code = body?.error?.code ?? `HTTP_${response.status}`;
    throw new Error(`${code}: ${body?.error?.message ?? "request failed"}`);
  }
  return body as T;
}
```

Add a regression test asserting that no route transaction is sent after intermediate confirmation returns non-2xx.

Finding 3 — The advertised expiry/resume examples do not preserve strict confirmation semantics

Severity: High

Affected surface: Agentic Integration guide, TypeScript full loop and Python broadcast helper

What happens

The TypeScript section labels its loop as having “retry on expiry,” but the broadcast operation is not inside a `try/catch`. A blockhash-expiry or RPC error rejects the entire function immediately. `/prepare` is called once, not again.

The Python helper has the opposite failure mode: after twelve unsuccessful confirmation polls, it returns the signature and allows dependent transactions to proceed even though it never observed `confirmed` or `finalized`.

Both behaviors conflict with the same published requirement: every group must be confirmed before its dependants are sent.

Reproduction

TypeScript path

1. Make the first broadcast reject with `blockhash expired`.
2. Run the documented loop.
3. Count calls to `/prepare`.
4. Observe one call followed by an immediate rejection.

Python path

1. Return `null` from all twelve `getSignatureStatuses` polls.
2. Observe that the helper still returns a signature.

3. The caller then proceeds to the next transaction group.

Expected

- Expiry or RPC interruption enters a bounded re-prepare path with a fresh idempotency key.
- A confirmation timeout is treated as failure/unknown, never as confirmation.

Actual

- TypeScript exits instead of retrying.
- Python proceeds without confirmed state.

Impact

Ordinary blockhash expiry or RPC lag breaks the advertised autonomous recovery behavior. The Python path can also create a predictable chain of dependent-transaction failures after a confirmation timeout.

Proposed fix

- Catch recognized expiry and transient RPC failures around the complete sign/broadcast step.
- Preserve successful intermediate signatures and re-run `/prepare`; process only the non-null groups returned by the server.
- Throw a typed confirmation-timeout error instead of returning a signature.
- Confirm with `{ signature, blockhash, lastValidBlockHeight }` and inspect the returned on-chain error.
- Cap retries and return the transfer ID plus sanitized recovery context when the limit is reached.

Finding 4 — The autonomous monitor expects the wrong GET response shape

Severity: Medium

Affected surfaces: Agentic Integration guide, Get Transfer reference, OpenAPI

What happens

The Get Transfer reference and OpenAPI schema return `TransferExtended` directly. The full autonomous example instead executes:

```
const { transfer: t } = await fetch(...).then(r => r.json());
```

With the documented response, `t` is `undefined`, and the next access to `t.status` throws. The same loop also omits the documented terminal status `refunded`, so even a corrected parser can poll forever for that state.

Reproduction

1. Return the documented body `{ "id": 42, "status": "processing", "progress": ... }` from `GET /transfers/42`.
2. Execute the sample destructuring statement.
3. Observe a TypeError on `t.status`.

The local regression test reproduces this with the exact documented direct-object shape.

Expected

The monitor reads the direct transfer object and exits for every documented terminal status.

Actual

The first successful poll crashes; `refunded` is not recognized as terminal.

Impact

The copyable “full autonomous loop” cannot reliably monitor a transfer after transactions may already have been broadcast. Agents lose completion, failure, and recovery visibility at the point where monitoring matters most.

Proposed fix

```
const t = await checkedFetch<TransferExtended>(`${API_BASE}/transfers/${id}`, { headers });
if (["completed", "failed", "expired", "refunded"].includes(t.status)) return t;
```

Generate example response types from OpenAPI and run each published example as a contract test.

Finding 5 — Required idempotency headers are absent from Quickstart and OpenAPI

Severity: Documentation blocker

Affected surfaces: Quickstart, API Introduction, OpenAPI

What happens

The API Introduction states that all POST mutations require an `Idempotency-Key` and documents `MH_070` for missing or invalid values. However:

- Quickstart contains four POST examples and zero `Idempotency-Key` headers.
- OpenAPI contains nine POST operations and zero `Idempotency-Key` parameter definitions.

The create endpoint’s dedicated page includes the header, but the primary onboarding flow and machine-readable contract do not.

Reproduction

Run the documentation audit:

```
quickstartPostCommands: 4
quickstartIdempotencyHeaders: 0
openapiPostOperations: 9
openapiPostOperationsWithIdempotencyParameter: 0
```

Expected

Every copyable POST example and generated client exposes or automatically creates a valid idempotency key.

Actual

The Quickstart and OpenAPI omit a header described as universally required.

Impact

First-run integrations fail before reaching the transfer flow. MCP- or OpenAPI-generated clients are especially likely to reproduce the omission across every mutation.

Proposed fix

- Define a reusable required OpenAPI header parameter with the documented 8–64 character constraints.
- Reference it from every operation that requires it.
- Add it to all Quickstart POST examples, including the intermediate keeper confirmation.
- If estimate is intentionally exempt, narrow the “all POST mutations” wording and document the exception.
- Add documentation CI that fails when a POST example or OpenAPI mutation lacks the header.

Finding 6 — The webhook example does not preserve the signed raw body

Severity: High

Affected surface: Webhook Events documentation

What happens

The published verifier computes HMAC over its `payload` argument, while the handler passes `req.body`. No raw-body middleware or capture step is shown.

In a standard Express JSON setup, `req.body` is a parsed object. Node’s HMAC `update()` does not accept that object and throws a `TypeError`. If an integrator re-serializes the object, verification can

still fail because whitespace, key order, Unicode escaping, or number formatting may differ from the exact bytes MultiHopper signed.

Reproduction

1. Configure a normal Express JSON parser.
2. Pass the resulting object to the documented verifier.
3. Observe that `crypto.createHmac(...).update(req.body)` throws.
4. Alternatively, re-serialize the object and observe that a legitimate signature fails whenever the byte representation differs.

The regression suite uses Node's real `crypto` implementation and confirms the parsed-object path throws.

Expected

The documentation captures the exact request bytes, verifies them first, and parses JSON only after successful authentication.

Actual

The example passes `req.body` without specifying raw-body handling.

Impact

Legitimate lifecycle events can be rejected, silently breaking webhook-based monitoring. A common workaround is to disable verification, which removes the endpoint's authenticity protection and allows spoofed transfer events.

Proposed fix

Publish a complete framework example:

```
app.post(
  "/webhooks/multihopper",
  express.raw({ type: "application/json" }),
  (req, res) => {
    const signature = req.headers["x-multihopper-signature"];
    if (!verifyWebhook(req.body, signature, process.env.MH_WEBHOOK_SECRET)) {
      return res.sendStatus(401);
    }
    const event = JSON.parse(req.body.toString("utf8"));
    // Process only after verification.
    return res.sendStatus(200);
  },
);
```

State explicitly that signature verification must happen before JSON parsing.

Finding 7 — The webhook verifier throws on malformed signature lengths

Severity: Medium

Affected surface: Webhook Events documentation

What happens

The example passes two buffers directly to `crypto.timingSafeEqual`. Node throws when the buffer lengths differ. The signature header is external input, so a missing, truncated, or malformed value turns a clean authentication failure into an unhandled exception.

The example also converts hexadecimal digests with the default UTF-8 encoding. Length-checking decoded hex buffers is clearer and rejects malformed input earlier.

Reproduction

```
const expected = crypto.createHmac("sha256", secret).update(raw).digest("hex");
crypto.timingSafeEqual(Buffer.from("deadbeef"), Buffer.from(expected));
// RangeError: Input buffers must have the same byte length
```

The regression suite confirms this using Node's real `timingSafeEqual`.

Expected

Malformed signatures return `false`/401 without throwing.

Actual

The verifier throws before it can reject the request.

Impact

Malformed delivery headers cause 5xx responses and unnecessary retries instead of a stable authentication failure. The same behavior is a small denial-of-service primitive against integrator webhook handlers copied from the documentation.

Proposed fix

```
function verifyWebhook(rawBody, signature, secret) {
  if (!Buffer.isBuffer(rawBody) || typeof signature !== "string") return false;
  const supplied = Buffer.from(signature, "hex");
  const expected = crypto.createHmac("sha256", secret).update(rawBody).digest();
  if (supplied.length !== expected.length) return false;
  return crypto.timingSafeEqual(supplied, expected);
}
```

Add negative examples for missing, non-hex, truncated, and oversized headers.

Additional inconsistencies worth cleaning up

I did not assign separate security severity to the following because authenticated test-environment behavior is still needed to determine the real server contract:

1. The Quickstart says create returns `awaiting_signature`, while the create reference says the initial status is `quote`.
2. The create reference states a flat `arrivalSeconds` minimum of 60, while the agent context says the minimum varies by hop count and references `MH_014` without publishing a formula or table.
3. The TypeScript signing example uses `VersionedTransaction.sign([keypair])`, while the copyable agent context says not to call `sign()`. In `@solana/web3.js` 1.98.4, the local partially signed transaction fixture preserved the server signature slot, but the documentation should still name the supported SDK/version and present one canonical method.

Recommended remediation order

Immediate

1. Fix the LLM-index host.
2. Make every reference HTTP call fail on non-2xx.
3. Correct the expiry/resume loop and Python confirmation timeout.
4. Replace the webhook verifier with a complete raw-body, length-safe example.

Next documentation release

1. Add `Idempotency-Key` to OpenAPI and Quickstart.
2. Correct the GET response shape and terminal-state list in the full loop.
3. Reconcile status, arrival-time, and signing instructions from one canonical schema/source.

Evidence package

The private evidence package contains:

- `src/agent.ts` — corrected fail-closed state machine;
- `src/api.ts` — checked API client with structured errors and automatic idempotency keys;
- `src/transactions.ts` — partial-signature preservation checks;
- `tests/documentation-regressions.test.ts` — five executable reproductions from published examples;
- `tests/agent-ordering.test.ts` — strict sequencing and stop-on-failure tests;
- `tests/transactions.test.ts` — partially signed v0 transaction fixture;

- `scripts/audit-docs.ts` — public-doc/OpenAPI consistency audit; and
- sanitized JSON and Markdown test summaries under `evidence/`.

Reproduction commands:

```
npm.cmd install
npm.cmd run build
npm.cmd test -- --reporter=verbose
npm.cmd run audit:docs
```

Responsible-testing statement

I used only public documentation, the public unauthenticated API error envelope, synthetic transaction fixtures, and local mocked HTTP/RPC responses. No private key, seed phrase, API key, or third-party secret was available or used. No mainnet transaction, live-user fund interaction, transfer creation, broadcast, rate-limit test, disruption, or public disclosure of an unremediated backend vulnerability was attempted.

Once bounty-approved test credentials are available, the same harness can run the minimum-value end-to-end transfer and attach sanitized API responses, transaction signatures, status transitions, and resume evidence to this report.

APPENDIX A

Test Evidence and Validation Record

This appendix contains the executable regression results, public documentation audit, sanitized API check, control verification, reproduction commands, and artifact hashes supporting the findings in the main report.

Evidence snapshot	20 June 2026, 13:46:17 +03
Automated tests	9 passed / 0 failed
TypeScript build	Passed
Dependency audit	0 known vulnerabilities

> Interpretation note: a passing regression test means the stated reference-example failure was reproduced or the corrected-client guardrail behaved as expected. Authenticated backend behavior remains pending approved test credentials.

A.1 Evidence at a glance

This appendix records the concrete evidence behind the accompanying bounty report. It separates public documentation-contract checks, executable reference-code reproductions, corrected-client guardrails, and work that remains pending authenticated test credentials.

Evidence class	Coverage	Result	Meaning
Automated regression suite	3 files / 9 tests	PASS	All expected reproductions and guardrails passed
Documentation audit	Index, Quickstart, OpenAPI, agent guide, webhooks	ISSUES	Contract inconsistencies reproduced
TypeScript compiler	Source, scripts, and tests	PASS	No compile errors
Dependency audit	164 total dependency records	PASS	0 known vulnerabilities
Authenticated API/on-chain	Approved key, wallet, RPC, balances	PENDING	Credentials were not supplied

A.2 Automated regression results

Vitest recorded 9 passed tests, 0 failed tests, and 0 pending tests across 3 test files.

Test	Evidence purpose	Duration	Result
uses the required ordering and intermediate confirmation	Verifies keeper → API confirmation → route → orchestrator → session ordering.	7.4 ms	PASSED
does not broadcast dependants when intermediate confirmation fails	Confirms the corrected harness stops before route broadcast.	3.7 ms	PASSED
does not proceed after an unconfirmed transaction	Confirms the corrected harness never treats timeout as success.	0.9 ms	PASSED
the documented confirm helper resolves even when the API returns HTTP 400	Reproduces fail-open HTTP handling in the published TypeScript helper.	41.0 ms	PASSED
the documented polling destructure crashes for the API-reference response shape	Reproduces the GET response-envelope mismatch.	4.5 ms	PASSED
the documented retry loop does not re-prepare after a broadcast expiry	Shows that the advertised expiry retry exits after one prepare call.	1.3 ms	PASSED

the documented webhook verifier cannot hash a standard parsed JSON body	Uses Node crypto to reproduce failure with a normal parsed Express body.	1.5 ms	PASSED
the documented webhook verifier throws on a malformed-length signature	Uses Node timingSafeEqual to reproduce the unhandled length error.	1.0 ms	PASSED
adds the wallet signature without changing a server partial signature	Confirms the existing server signature remains byte-for-byte unchanged.	83.2 ms	PASSED

A.3 Public documentation audit snapshot

The audit fetched the public pages directly on 20 June 2026 and compared the human-readable examples with the machine-readable OpenAPI contract. No authenticated endpoint was called.

Check	Observed	Expected	Assessment
LLM index links	19	—	Inventory
Index links pointing at the wrong host	19	0	ISSUE
Quickstart POST examples	4	—	Inventory
Quickstart Idempotency-Key headers	0	4	ISSUE
OpenAPI POST operations	9	—	Inventory
OpenAPI POST operations declaring Idempotency-Key	0	9	ISSUE
GET reference returns direct transfer object	Yes	Yes	Contract source
Autonomous loop expects a transfer wrapper	Yes	No	ISSUE
Autonomous confirm helper checks HTTP status	No	Yes	ISSUE
Autonomous loop treats refunded as terminal	No	Yes	ISSUE
Python helper proceeds after confirmation timeout	Yes	No	ISSUE
Webhook example documents raw-body capture	No	Yes	ISSUE
Webhook example checks length before timingSafeEqual	No	Yes	ISSUE

Source URLs

- **LLM index:** https://dev-docs.multihopper.com/llms.txt

- **Quickstart:** https://dev-docs.multihopper.com/quickstart
- **Agentic integration guide:** https://dev-docs.multihopper.com/guides/agentic-integration
- **OpenAPI:** https://dev-docs.multihopper.com/api-reference/openapi.json
- **Get Transfer:** https://dev-docs.multihopper.com/api-reference/transfers/get
- **Webhook Events:** https://dev-docs.multihopper.com/api-reference/webhooks/events

A.4 Sanitized public API availability check

An unauthenticated request was sent only to verify endpoint availability and the published error envelope. The service returned HTTP 401 with MH_001 as documented. The request identifier is redacted below.


```
{
  "error": {
    "code": "MH_001",
    "message": "Invalid or missing API key",
    "requestId": "[REDACTED]"
  }
}
```


This check proves only public endpoint availability and error-envelope parsing.

A.5 Corrected harness controls validated

Control	Verified behavior	Result
Strict broadcast sequencing	keeper → intermediate API confirmation → route → orchestrator → session	PASS
Intermediate API failure	No dependent transaction is broadcast after rejection	PASS
Unconfirmed RPC transaction	Timeout/unknown status is never treated as success	PASS
Server partial signature	Non-wallet signature slot remains byte-for-byte unchanged	PASS
HTTP response handling	Non-2xx responses become structured exceptions	PASS
Live-key protection	Harness refuses API keys that do not start with mh_test_	PASS
Cluster mismatch protection	Prepared blockhash must be valid on configured RPC	PASS

A.6 Reproduction commands

Run from the root of the private evidence package:

```
npm.cmd install
npm.cmd run build
npm.cmd test -- --reporter=verbose
npm.cmd run audit:docs
npm.cmd audit
```

Expected result: build succeeds; 9 tests pass; the documentation audit prints the metrics recorded in Section 3; dependency audit reports 0 vulnerabilities.

A.7 Artifact integrity record

The hashes below identify the exact source and test artifacts used for this evidence snapshot. Recalculate with Get-FileHash -Algorithm SHA256 before submission if any file is edited.

Artifact	Purpose	SHA-256
tests/documentation-regressions.test.ts	Published-example regression reproductions	721C114373713F2516049D7BBEF1D2C951784A9A368CB4E5A30E0F50F90CDB4E
tests/agent-ordering.test.ts	Strict ordering and fail-closed guardrails	78FDED518B75749ED163463DFD9D81F3CBF420611541C1662D2A38818C11783A
tests/transactions.test.ts	Server partial-signature preservation fixture	9C1FEB224553180466272B65146E88ED1D20800C6FD3E7BC5A855C9A5CC01DF0
scripts/audit-docs.ts	Public docs and OpenAPI consistency audit	07DF2BB6EF707943326340BCFE6B9344307D5775FBA2D1C00F205AB7958439D4
src/agent.ts	Corrected transfer state machine	6FC21A36371BDEA9A411C77DA465309C14AF24DD43A3A931598FD5F7A3415462
src/api.ts	Checked API client and idempotency handling	4E51AD0056079B7234F6B2E0CA96709EB2516A0B231D8ECC39962F0BBCE1F727
src/transactions.ts	Transaction signing and signature-slot checks	9CE8B90D2C8C1B2A2BFDCEE1E5521A7388450B20343AE83B6175C5ECF2040E1E

A.8 Scope boundaries and pending evidence

- No API key, private key, seed phrase, or third-party secret was stored in the evidence package.
- No mainnet transaction, live-user fund interaction, transfer creation, broadcast, rate-limit test, or disruption was attempted.
- No authenticated backend behavior is presented as confirmed.
- Live create/prepare/broadcast/resume/status evidence remains pending an approved mh_test_ key, funded test wallet, token parameters, API base, and RPC cluster.

Evidence still needed for final live validation

1. Sanitized create and prepare responses
2. Prepared transaction group counts and blockhash validity check
3. Keeper, route, orchestrator, and session transaction signatures
4. Intermediate and final confirm-broadcast responses
5. Status/phase/progress timeline through a terminal state
6. One interrupted-broadcast resume trace using only approved test funds