JELLEO | AUTONOMOUS CODE REVIEW

AUDIT CYCLE · JUNE 05, 2026

OpenDevTool — Security & Correctness Review

AUDITOR	Kirill Sakharuk · kirill@jelleo.com
TARGET	OpenDevTool — OpenSubmissionn/Open_DevTool (TypeScript)
AUDIT DATE	June 05, 2026
CYCLE	20260605-opendevtool
ENGINE SHA	1536ef5
WRAPPER SHA	?
GENERATED	2026-06-05T16:33:09+00:00

0	0	4	4	1
CRITICAL	HIGH	MEDIUM	LOW	INFO

CONFIRMED · DISCLOSED · FIXED · VERIFIED

SIGNED · ED25519

(see jelleo.com/keys/)

verify with `audit-pipeline sign verify <file> <file>.sig --pubkey jelleo.ed25519.pub`
public key at
https://jelleo.com/keys/jelleo.ed25519.pub

PLATFORM · V0.1

JELLEO · The underwriting layer for Solana DeFi.

Methodology	jelleo.com/methodology.html
Disclosure	jelleo.com/security.html
Source	github.com/Copenhagen0x/audit-pipeline-cli

Apache-2.0 · contact security@jelleo.com

00 — EXECUTIVE SUMMARY

This report documents an independent security & correctness review by Jelleo of the `OpenDevTool — OpenSubmissionn/Open_DevTool (TypeScript)` codebase at commit 1536ef5. A multi-agent sweep across 8 dimensions, followed by adversarial verification that rejected 13 of 25 candidates, surfaced 4 Medium, 4 Low, and 1 informational finding. Seven were reproduced with executable proofs-of-concept against the tool's own code; three correctness findings were additionally proven by property-based tests (one verified live against a real mainnet transaction); and confirmed fix patches accompany five of them.

00.1 — SCOPE

IN-SCOPE SOURCE SET	
Target workspace	`OpenDevTool — OpenSubmissionn/Open_DevTool (TypeScript)`
Protocol	TypeScript — read-only Solana-transaction analysis tool
Engine commit	`1536ef5`
Source files	`services/ · cli/ · api/ · web/ (TypeScript monorepo)`
Hypothesis library	25 candidate hypotheses across 8 review dimensions (code-execution, CI/CD, secret handling, web/API, parser robustness, analysis correctness, dependencies, CLI robustness)
Out of scope	Off-chain components (indexers, frontends, oracles); deployment scripts; framework / standard-library code; dependencies pinned in `package.json` beyond their declared interfaces.

01 — PER-FINDING ANALYSIS · CONTENTS

Each finding below begins on its own page. Numbering matches the `FINDING NN / NN` banner in the body. Click any row to jump.

FINDING 01 / 9

MEDIUM F1

Compute-unit accounting is wrong on real multi-program transactions

INVARIANT Compute-unit accounting is wrong on real multi-program transactions

AFFECTED CODE

- `services/src/analysis/cuProfiler.ts:27-53` — regex `/consumed (\d+) of (\d+) compute units/` with `totalConsumed += consumed` for every match (no CPI-depth awareness, no `^Program` anchor).
- `services/src/analysis/cpiTreeBuilder.ts:120-131` — `trace.totalComputeUnits += consumed` for every matched node (parent and child).
- Surfaced as the headline KPI in `cli/src/renderers/terminal/renderer.ts:1012,1031,1041,1065` (the big "{totalConsumed/1000}k of {limit}k" card, CU-used %, and budget bar). `renderer.ts:1012` does **not** prefer the canonical `raw.computeUnitsConsumed`.

DESCRIPTION

The RPC returns the authoritative total in `meta.computeUnitsConsumed`. The tool ignores it and re-derives a total from log lines:

- **Overcount (double-count):** a tx where Squads (`consumed 75855`) CPIs into Token (`consumed 6200`) — the 75855 already contains the 6200 — is summed to 82,055.
- **Undercount:** programs with no `consumed` line (ComputeBudget) are silently dropped.
- **Zero:** for a tx whose logs don't contain a line matching the regex, the total is reported as 0 even though the tx really consumed CU.

For `opendev simulate`, there is no canonical `computeUnitsConsumed` at all, so this derived (wrong) number is the *only* figure the user sees.

IMPACT

A developer using OpenDevTool to understand or optimize CU usage is shown numbers that disagree with Solscan / the validator. The flagship feature reports incorrect data. Not a security issue — a **correctness** issue that goes to the core value of the product.

REPRODUCTION

```
=== realSquadsTx.json ===
  CU log lines (2):
     Token... consumed 6200 of 21272 compute units
     SQDS...  consumed 75855 of 89700 compute units
  canonical meta.computeUnitsConsumed = 76155   <-- ground truth
  profileCU().totalConsumed           = 82055
  buildCPITree().totalComputeUnits    = 82055
  -> profiler error  = +5900 (+7.75%)     (nested CPI double-counted)

=== realMagicEdenTx.json ===
  canonical = 84453 ; reported = 84153  -> -300 (-0.36%)   (ComputeBudget CU missed)

=== mockDeepCpiTx.json ===
  canonical = 178900 ; reported = 0  ->  -178900 (-100%)   (no log line matched the regex)
```

(Harness: `services/repro-cu.ts`.)

```
live signature: 4weawkYSaVczCw4R5H1egR5hp6SuHRdWXT3UqTLJrUUP9SAj2gJK3UhQ3cSGXEARcjX5JrE4RBqqSLGJxvmRniBx  (slot
424467976)
  CU "consumed" log lines: 21
  canonical meta.computeUnitsConsumed (chain truth) = 168487
  profileCU().totalConsumed (CLI headline)          = 302576   ->  +134089 (+79.58%)
```

On a real Jupiter swap the CLI's headline CU figure is **~80% too high** (reports ~303k for a tx that used ~168k). (Harness: `services/repro-live.ts`.)

RECOMMENDED FIX

Stop summing nested CU lines. Either (a) make the profiler CPI-aware — track invoke depth like `cpiTreeBuilder` and count only each program's *own* (leaf) CU, i.e. subtract child CU from parent; or (b) report the canonical `bundle.computeUnitsConsumed` as the transaction total and use per-line values only for relative per-instruction attribution. At minimum, anchor the regex to the real RPC format (`^Program <id> consumed …`) and treat the outermost program's line as the subtree total. Add a fixture that asserts `total === meta.computeUnitsConsumed`.

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

```
index 4e00e0e..fdc56bf 100644
--- a/cli/src/renderers/terminal/renderer.ts
+++ b/cli/src/renderers/terminal/renderer.ts
@@ -1009,7 +1009,10 @@ function renderDashboard(
     : (((analyzed as any).cpiTree?.root ?? []) as CPINodeView[]);
   const bottleneckTarget = collectBottleneckTarget(analyzed);
   const cuProfile = analyzed.cuProfile;
-  const totalConsumed = cuProfile?.totalConsumed ?? analyzed.cuCost?.cuConsumed ?? 0;
+  // Prefer the chain's canonical CU (exact; includes programs that emit no CU
+  // log line, e.g. ComputeBudget). Fall back to the (now CPI-aware) profiler.
+  const totalConsumed =
+    (analyzed as any).raw?.computeUnitsConsumed ?? cuProfile?.totalConsumed ??
analyzed.cuCost?.cuConsumed ?? 0;
   const totalLimit =
     cuProfile?.totalLimit && cuProfile.totalLimit > 0 ? cuProfile.totalLimit : BUDGET_LIMIT_DEFAULT;
   const utilization = totalLimit > 0 ? totalConsumed / totalLimit : 0;
diff --git a/services/src/analysis/cpiTreeBuilder.ts b/services/src/analysis/cpiTreeBuilder.ts
index fd173bc..7fdb948 100644
--- a/services/src/analysis/cpiTreeBuilder.ts
+++ b/services/src/analysis/cpiTreeBuilder.ts
@@ -126,7 +126,8 @@ export function buildCPITree(logMessages: string[]): ExecutionTrace {
       const consumed = parseInt(cuMatch[2], 10);
       if (matchedNode.computeUnitsConsumed === undefined) {
         matchedNode.computeUnitsConsumed = consumed;
-        trace.totalComputeUnits += consumed;
+        // Do NOT accumulate here: a parent's reported CU already includes its
+        // CPI children. The transaction total is summed from root nodes below.
       }
       continue;
     }
@@ -161,5 +162,13 @@ export function buildCPITree(logMessages: string[]): ExecutionTrace {
     while (stack.length > 0) markTruncated(stack.pop()!);
   }

+  // A root program's reported CU already includes its entire CPI subtree, so the
+  // transaction total is the sum of the ROOTS' consumed CU — not every node
+  // (which would double-count nested CPI).
+  trace.totalComputeUnits = trace.roots.reduce(
+    (sum, root) => sum + (root.computeUnitsConsumed ?? 0),
+    0
+  );
+
   return trace;
 }
diff --git a/services/src/analysis/cuProfiler.ts b/services/src/analysis/cuProfiler.ts
index 5e4cda1..1cdd178 100644
--- a/services/src/analysis/cuProfiler.ts
+++ b/services/src/analysis/cuProfiler.ts
@@ -23,32 +23,53 @@ export function profileCU(logMessages: string[]): CUProfile {
```

```
   // Tracks the highest CU consumer found so far.
   let bottleneck: CUEntry | null = null;

-  // Matches log lines like: "consumed X of Y compute units".
-  const cuRegex = /consumed (\d+) of (\d+) compute units/;
+  // Anchored to the real RPC format so we can read the program id + invoke depth.
+  // A parent program's reported CU ALREADY INCLUDES every program it CPI-invokes,
+  // so only TOP-LEVEL (depth 1) lines may contribute to the transaction total —
+  // otherwise nested CPI is double-counted.
+  const consumedRegex = /^Program (\S+) consumed (\d+) of (\d+) compute units$/;
+  const invokeRegex = /^Program (\S+) invoke \[(\d+)\]$/;
+  const endRegex = /^Program \S+ (?:success|failed)/;
+  const depthStack: number[] = [];

   for (const log of logMessages) {
-    const match = log.match(cuRegex);
-    if (match) {
-      const consumed = parseInt(match[1], 10);
-      const limit = parseInt(match[2], 10);
+    const inv = log.match(invokeRegex);
+    if (inv) {
+      depthStack.push(parseInt(inv[2], 10));
+      continue;
+    }

-      totalConsumed += consumed;
-      totalLimit += limit;
+    const match = log.match(consumedRegex);
+    if (match) {
+      const programId = match[1];
+      const consumed = parseInt(match[2], 10);
+      const limit = parseInt(match[3], 10);
+      const depth = depthStack.length ? depthStack[depthStack.length - 1] : 1;

-      // Program metadata is not extracted in this step yet.
       const currentEntry: CUEntry = {
         cuConsumed: consumed,
         cuLimit: limit,
-        utilizationPercent: (consumed / limit) * 100,
-        programId: 'Unknown Program ID',
-        programName: 'Unknown Program',
-        depth: 0,
+        utilizationPercent: limit > 0 ? (consumed / limit) * 100 : 0,
+        programId,
+        programName: programId,
+        depth,
       };
       perInstruction.push(currentEntry);

       if (!bottleneck || currentEntry.cuConsumed > bottleneck.cuConsumed) {
         bottleneck = currentEntry;
       }
```

```
+
+      // Only top-level invocations contribute to the transaction total.
+      if (depth === 1) {
+        totalConsumed += consumed;
+        totalLimit += limit;
+      }
+      continue;
+    }
+
+    if (endRegex.test(log)) {
+      depthStack.pop();
     }
   }

@@ -68,6 +89,6 @@ export function profileCU(logMessages: string[]): CUProfile {
       programId: 'N/A',
       programName: 'N/A',
       depth: 0,
-    }, // <-- CORRIGIDO
+    },
   };
```

FINDING 02 / 9

MEDIUM F2

`simulate` leaks the user's stored API keys to executed code

INVARIANT `simulate` leaks the user's stored API keys to executed code

AFFECTED CODE

- `services/src/solana/sourceRunner.ts:155-162` — `spawn(..., { env: { ...process.env, ...options.env } })` (full env inherited; `simulationService.ts:269` passes no `env`).
- `cli/src/config/credentials.ts:108-116` — `applyCredentialsToEnv()` copies `~/.opendev/credentials.json` keys into `process.env.ANTHROPIC_API_KEY` / `GROQ_API_KEY` at startup (`cli/bin/open.ts:40` → `loader.ts:30`).
- Also exposes `HELIUS_RPC_URL` (a URL embedding an API key) and `HELIUS_API_KEY`.

DESCRIPTION

Plain code execution is intentional (it's how `simulate` builds a tx). The avoidable part is the **silent handoff of secrets**: a tx-building snippet has no need for your LLM API keys, yet inherits them and has network access. The marginal harm is "runs my code" → "runs my code *and hands it my credentials*".

> Note: an earlier reviewer's `SECRET_ENV_PATTERN` / `buildChildEnv` allowlist fix is **not** on `main` (`HEAD 1536ef5`) — `git grep` finds it in neither source nor the shipped `cli/dist/open.js` (still `env: { ...process.env, ...options.env }`). `install.sh` clones `--branch main`, so installed users get the vulnerable version.

IMPACT

Theft of billable Anthropic/Groq keys (attacker runs up charges) and the Helius RPC key, from a developer who merely ran `opendev simulate sample.ts` on a snippet shared in a gist/issue/tutorial — exactly the advertised use case.

REPRODUCTION

```
victim ran: opendev simulate jelleo-preview-leak.ts
[attacker-snippet] EXFILTRATED {"anthropic":"sk-ant-FAKE...","groq":"gsk_FAKE...","helius":"...?api-key=FAKE-HELIUS-KEY"}
runSourceFile resolved OK → victim sees a normal base64 tx (len 120).
LEAK CONFIRMED = true  (all three secrets were readable by the executed snippet)
```

(Harness: `services/repro-env.ts`. Fake placeholder keys used.)

RECOMMENDED FIX

Build the child env from a small allowlist (`PATH` , `HOME` , `TMPDIR` , rust `RUSTUP_HOME` / `CARGO_HOME`) and strip anything whose name matches `/(API[_-]?KEY|SECRET|TOKEN|PASSWORD|PRIVATE[_-]?KEY|MNEMONIC|SEED)/i` by default. Offer an explicit `--inherit-env` opt-in with a warning banner for users who truly need it. (This is exactly the unmerged fix — it just needs to land on `main` and be rebuilt into `cli/dist` .)

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

```
index 666dd72..0603bd7 100644
--- a/services/src/solana/sourceRunner.ts
+++ b/services/src/solana/sourceRunner.ts
@@ -28,6 +28,28 @@ const BASE64_LINE_REGEX = /^[A-Za-z0-9+/]+={0,2}$/;
 const MIN_BASE64_LEN = 100;
 const STDERR_TAIL_LINES = 20;

+// Names whose VALUE is a secret even though the name doesn't match the pattern.
+const SECRET_ENV_NAMES = new Set(['HELIUS_RPC_URL', 'MCP_ENDPOINT_URL']);
+// Strip anything that looks like a credential by name.
+const SECRET_ENV_PATTERN =
+  /(API[_-]?KEY|APIKEY|SECRET|TOKEN|PASSWORD|CREDENTIAL|PRIVATE[_-]?KEY|ACCESS[_-]?
KEY|SESSION|MNEMONIC|SEED)/i;
+
+// Build a scrubbed environment for untrusted user code: inherit non-secret vars
+// only, so a `simulate` snippet cannot read the user's stored API/RPC keys.
+// An explicit `--inherit-env` opt-in (via options.env) can re-add what's needed.
+function buildChildEnv(extra?: NodeJS.ProcessEnv): NodeJS.ProcessEnv {
+  const out: NodeJS.ProcessEnv = {};
+  for (const [key, value] of Object.entries(process.env)) {
+    if (value === undefined) continue;
+    if (SECRET_ENV_PATTERN.test(key) || SECRET_ENV_NAMES.has(key)) continue;
+    out[key] = value;
+  }
+  if (extra) {
+    for (const [key, value] of Object.entries(extra)) out[key] = value;
+  }
+  return out;
+}
+
 export function detectSourceKind(input: string): SourceKind | null {
   if (!fs.existsSync(input)) return null;
   const stat = fs.statSync(input);
@@ -155,7 +177,7 @@ export async function runSourceFile(
   return new Promise((resolve, reject) => {
     const child = spawn(spawnTarget, spawnArgs, {
       cwd,
-      env: { ...process.env, ...options.env },
+      env: buildChildEnv(options.env),
       stdio: ['ignore', 'pipe', 'pipe'],
       shell: isWindows,
       windowsHide: true,
```

FINDING 03 / 9

MEDIUM F4

Unauthenticated `/api/analyze` + wildcard CORS → cost amplification

INVARIANT Unauthenticated `/api/analyze` + wildcard CORS → cost amplification

AFFECTED CODE

`api/analyze.ts:28-42` ; `web/server.ts:1674-1708` (handler), `:1620` (wildcard CORS in the `send()` helper), `:1417-1457` (RPC + `McpInsightProvider.fetchInsights()` fan-out); `services/src/mcp/client.ts:92-119` (per-request LLM call); `vercel.json` (`maxDuration: 60`).

DESCRIPTION

An anonymous client (or any third-party site, via wildcard CORS, from a victim's browser) can exhaust the operator's Helius RPC quota, run up unbounded LLM spend on the operator's key (financial DoS), and consume Vercel function-seconds (up to 60s each). No on-chain write or data theft → cost/availability, hence Medium.

IMPACT

Hardening issue or invariant violation requiring a privileged signer / improbable state.

REPRODUCTION

```
OpenDevTool server up at http://127.0.0.1:3399 (NO api key configured)
  OPTIONS /api/analyze  (Origin: https://evil.example) → 204  ACAO=*
  POST /api/analyze     (NO auth header)               → 500  ACAO=*
  fired 30 rapid UNAUTHENTICATED POSTs → 30 reached the pipeline, 0 rate-limited (429)
  F4 CONFIRMED = true   (no-auth=true, wildcard-CORS=true, no-rate-limit=true)
```

(Harness: `services/repro-api.ts` . The `500` is the dummy RPC failing — it proves the unauthenticated request passed every gate into `analyze()` . The actual $ burn needs the operator's real key, so it isn't quantified here.)

RECOMMENDED FIX

Add IP-keyed rate limiting (Vercel KV / Upstash token bucket) on `/api/analyze` and `/api/latest-tx` . Restrict CORS to your own origin(s) — the demo is same-origin, so `*` is unnecessary. Cache LLM results by signature (a confirmed tx's analysis is immutable) and/or default anonymous web traffic to rule-based-only insights.

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

```
# Proposed fix - web/server.ts (add per-IP rate limit + same-origin CORS on /api/analyze)
+const RL = new Map<string, { n: number; reset: number }>();
+function rateLimited(ip: string, limit = 30, windowMs = 60_000): boolean {
+  const now = Date.now(); const e = RL.get(ip);
+  if (!e || now > e.reset) { RL.set(ip, { n: 1, reset: now + windowMs }); return false; }
+  e.n += 1; return e.n > limit;
+}
 // inside the /api/analyze handler, BEFORE analyze():
+  const ip = (req.headers['x-forwarded-for'] as string)?.split(',')[0]?.trim()
+             ?? req.socket.remoteAddress ?? 'unknown';
+  if (rateLimited(ip)) return send(res, 429, JSON.stringify({ error: 'rate limited' }));
 // in send(): restrict CORS to the app's own origin instead of '*'
-    'Access-Control-Allow-Origin': '*',
+    'Access-Control-Allow-Origin': APP_ORIGIN,   // e.g. https://opendev-kappa.vercel.app
```

FINDING 04 / 9

MEDIUM F5

Unbounded on-chain IDL fetch + disk persistence per program ID → DoS

INVARIANT Unbounded on-chain IDL fetch + disk persistence per program ID → DoS

AFFECTED CODE

`services/src/analysis/txParser.ts:191-223,499-524` (`prefetchIdls` over all `uniqueProgramIds`, no filter); reachable remotely via `api/analyze.ts:42` → `analyze()` → `mergeAnalysis` → `parseTransaction`. IDL persisted by `idlcache.ts:233-250` (`set()` does `fs.writeFileSync(..., JSON.stringify(entry))` with no size/shape check).

DESCRIPTION

A malicious program author can `anchor idl init` an IDL account holding a small zlib blob that inflates to hundreds of MB (a **decompression bomb**), and reference many such program IDs in one transaction. Analyzing that signature drives concurrent unbounded inflations into memory plus uncontrolled disk writes — DoS / resource exhaustion. Remotely triggerable on the serverless backend by anyone who supplies such a signature.

IMPACT

Hardening issue or invariant violation requiring a privileged signer / improbable state.

REPRODUCTION

```
distinct program ids fetched & persisted from ONE tx = 20   (no count cap)
each IDL written verbatim with NO size cap / NO shape validation
total written to ~/.open-cli/cache/idls (here: temp) = 40.0 MB
F5 (disk half) CONFIRMED = true
```

(Harness: `services/repro-idl.ts`. The stub stands in for an inflated on-chain IDL. The **full memory-exhaustion DoS additionally needs an on-chain zlib decompression-bomb IDL, which I did not detonate** — that would require deploying a hostile program on-chain.)

RECOMMENDED FIX

Cap the number of distinct program IDs fetched per tx (or restrict to your existing `program-registry.json` allowlist); wrap each `Program.fetchIdl` in a timeout + size guard, rejecting oversized account data *before* inflating; enforce a max decompressed size before `cache.set`, and a max total cache size; validate each IDL is an object with a bounded `instructions` array before use.

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

```
index 9246382..7c9d926 100644
--- a/services/src/solana/idlcache.ts
+++ b/services/src/solana/idlcache.ts
@@ -234,6 +234,14 @@ export class IdlCache {
     if (this.noCache) return;

     try {
+      // Reject implausibly large (attacker-controlled, possibly zlib-bomb) IDLs
+      // before touching memory/disk. Real Anchor IDLs are tens of KB.
+      const serializedIdl = JSON.stringify(idl);
+      const MAX_IDL_BYTES = 512 * 1024; // 512 KB
+      if (serializedIdl.length > MAX_IDL_BYTES) {
+        verboseLog(this.verbose, `skip oversized ${programId} (${serializedIdl.length} bytes)`);
+        return;
+      }
       this.ensureCacheDirSync();
       const entry: IdlCacheEntry = {
         idl,
```

FINDING 05 / 9

LOW F3

`simulate <file.rs>` runs the whole enclosing Cargo project, not the file you named

INVARIANT `simulate <file.rs>` runs the whole enclosing Cargo project, not the file you named

AFFECTED CODE

`services/src/solana/sourceRunner.ts:45-67` (`findCargoRoot` upward walk → `{ cmd:'cargo', args:['run','--release','--quiet'], cwd: cargoRoot }`; `absInput` is never used in the rust branch).

DESCRIPTION

The "EXECUTING USER CODE" banner (`cli/src/commands/simulate.ts:58-66`) echoes the single file path you named, implying *that file* runs. In reality the whole repo the file lives in is compiled and its default binary run. Pointing `simulate` at any `.rs` buried inside a cloned/untrusted repo silently runs that repo's `build.rs`.

IMPACT

Surprise scope: compile-time execution of build scripts you never named, and execution of the wrong target (correctness — wrong tx produced). Local-only (CLI), and the user already opted into running Rust, so **Low** — but the executed scope and code both exceed what "run this file" implies.

REPRODUCTION

a throwaway project `evil/` with `build.rs`, default `src/main.rs`, and a named `src/print_tx.rs`; ran the tool's `runSourceFile('evil/src/print_tx.rs')`:

```
cargo command actually run : cargo run --release --quiet   (cwd=...\evil)
base64 first char          : A   (A ⇒ src/main.rs default bin ran; B ⇒ print_tx.rs)
build.rs marker exists     : true    ⟵ compile-time exec the user never named
default-bin marker exists  : true    ⟵ src/main.rs (the default bin) ran
NAMED-file marker exists   : false   ⟵ print_tx.rs (what the user pointed at) did NOT run
```

(Harness: `services/repro-cargo.ts` + `evil/`.)

RECOMMENDED FIX

For a lone `.rs`, compile/run only that file in an isolated temp dir (`rustc --edition 2021 <file>`), or refuse and instruct the user to pass `cargo run --bin <name>`. Only walk to `Cargo.toml` when the user passed a *directory*. When running a project, run the named target (`cargo run --bin <derived>`) and make build-script execution explicit in the banner.

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

```
# Proposed fix - services/src/solana/sourceRunner.ts (buildCommand, rust branch)
# Run only the file the user named; walk to Cargo.toml ONLY for a directory input.
   if (kind === 'rust-source') {
-    const cargoRoot = findCargoRoot(absInput);
-    if (!cargoRoot) throw new Error(...);
-    return { cmd: 'cargo', args: ['run', '--release', '--quiet'], cwd: cargoRoot };
+    if (fs.statSync(absInput).isFile()) {
+      // Lone .rs: compile & run ONLY that file in an isolated temp dir.
+      const out = path.join(os.tmpdir(), `odt-${path.parse(absInput).name}`);
+      return { cmd: 'sh', args: ['-c', `rustc --edition 2021 "${absInput}" -o "${out}" && "${out}"`],
+               cwd: path.dirname(absInput) };
+    }
+    // Directory = explicit project intent: run the NAMED target and state in the
+    // EXECUTING USER CODE banner that build scripts + the default bin will run.
+    const cargoRoot = findCargoRoot(absInput)!;
+    return { cmd: 'cargo', args: ['run', '--bin', deriveBin(absInput), '--release', '--quiet'], cwd:
cargoRoot };
```

FINDING 06 / 9

LOW F6

`accountDiff` mislabels ALT-loaded accounts on versioned transactions

INVARIANT `accountDiff` mislabels ALT-loaded accounts on versioned transactions

AFFECTED CODE

`services/src/analysis/accountDiff.ts:62-92`; data shape from `services/src/solana/rpc.ts:41-43`.

IMPACT

Wrong role labels in the account-diff panel for any v0 tx using an ALT with a balance change on a loaded account — common for Jupiter/Pump.fun/aggregator swaps. The SOL/token delta numbers themselves are still correct; this is a labeling/sort-order bug → Low.

REPRODUCTION

a v0 bundle with 2 static + 2 ALT-loaded keys; account C is in `meta.loadedAddresses.writable` and has a +2,000,000-lamport change:

```
tool labelled C role = "readonly"   (ground truth: "writable")
F6 CONFIRMED = true   ⟵ loaded writable account mislabelled
```

(Harness: `services/repro-diff-decoder.ts`.)

RECOMMENDED FIX

When parsed `accountKeys` carry per-key `signer` / `writable` flags, use them directly. Otherwise combine the header (static keys) with `meta.loadedAddresses.writable/readonly` lengths to place the loaded boundaries per the canonical v0 ordering.

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

```
index a6e4953..d7e4668 100644
--- a/services/src/analysis/accountDiff.ts
+++ b/services/src/analysis/accountDiff.ts
@@ -189,6 +189,20 @@ export function computeAccountDiffs(bundle: RawTransactionBundle): AccountDiff[]
   const header = getHeaderFromTransaction(bundle.transaction);
   const tokenDeltaByAccount = getTokenDeltas(bundle);

+  // Versioned (v0) transactions append address-lookup-table accounts AFTER the
+  // static keys, but the message header only describes the static keys. Use the
+  // RPC's authoritative meta.loadedAddresses for loaded accounts, and apply the
+  // header/index inference only across the static range.
+  const loaded = (bundle as { loadedAddresses?: { writable?: string[]; readonly?: string[] }
}).loadedAddresses;
+  const loadedWritable = new Set<string>(loaded?.writable ?? []);
+  const loadedReadonly = new Set<string>(loaded?.readonly ?? []);
+  const staticCount = Math.max(totalAccounts - loadedWritable.size - loadedReadonly.size, 0);
+  const roleFor = (index: number, pubkey: string): AccountDiff['role'] => {
+    if (loadedWritable.has(pubkey)) return 'writable';
+    if (loadedReadonly.has(pubkey)) return 'readonly';
+    return getAccountRole(index, staticCount, header);
+  };
+
   const diffs: Array<AccountDiff & { _index: number }> = [];

   for (let index = 0; index < totalAccounts; index += 1) {
@@ -205,7 +219,7 @@ export function computeAccountDiffs(bundle: RawTransactionBundle): AccountDiff[]
     diffs.push({
       _index: index,
       pubkey: accountKeys[index] ?? `unknown-account-${index}`,
-      role: getAccountRole(index, totalAccounts, header),
+      role: roleFor(index, accountKeys[index] ?? `unknown-account-${index}`),
       solDelta,
       tokenDeltas,
     });
```

FINDING 07 / 9

LOW F7

Claude CI workflows: write + secret on externally-openable triggers, no in-workflow author gate

INVARIANT Claude CI workflows: write + secret on externally-openable triggers, no in-workflow author gate

AFFECTED CODE

`claude.yml:3-37`, `claude-code.yml:3-41`.

DESCRIPTION

Today this is held safe **only** by `anthropics/claude-code-action@v1`'s built-in default that refuses non-write-access actors (these workflows don't set `allowed_non_write_users`, and the triggers run from the default branch, so untrusted PR code isn't checked out). It is therefore a **defense-in-depth gap, not a live RCE** — but the entire boundary rests on one external action's default. If that default is ever weakened (someone adds `allowed_non_write_users: '*'`, or a version bump changes behavior), arbitrary external users could drive a write-capable, key-bearing agent. `pr-checks.yml` is safe (`pull_request`, no secrets).

> Not PoC'd: proving this would require opening an `@claude` issue/comment on **your** repo to trigger the workflow — which I won't do. The evidence is the workflow YAML itself (above).

IMPACT

Minor issue with no plausible path to fund loss.

RECOMMENDED FIX

Add an explicit `author_association ∈ {OWNER,MEMBER,COLLABORATOR}` gate to each job's `if:`. Drop `write` perms to `read` unless needed; remove `id-token: write` unless OIDC is used. Pin the action to a full commit SHA. Never set `allowed_non_write_users` on a public repo. Consider deduplicating the two near-identical bot workflows.

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

```
# Proposed fix - .github/workflows/claude.yml + claude-code.yml
# Gate each job on author_association so an external @claude trigger can't reach
# the write-capable, key-bearing job. Also drop perms to read + pin the action SHA.
     if: |
-      (github.event_name == 'issue_comment' && contains(github.event.comment.body, '@claude'))
+      ( (github.event_name == 'issue_comment' && contains(github.event.comment.body, '@claude'))
        || ... )
+      && contains(fromJSON('["OWNER","MEMBER","COLLABORATOR"]'),
+                  github.event.comment.author_association || github.event.issue.author_association)
     permissions:
-      contents: write
-      pull-requests: write
-      issues: write
-      id-token: write
+      contents: read         # raise individually only where an automation needs it
-      uses: anthropics/claude-code-action@v1
+      uses: anthropics/claude-code-action@<full-commit-sha>   # pin, don't float @v1
```

FINDING 08 / 9

LOW F8

Vulnerable transitive dependencies in the production runtime

INVARIANT Vulnerable transitive dependencies in the production runtime

AFFECTED CODE

`services/package.json:17-23` , `cli/package.json:52-58` .

IMPACT

Low in practice: `uuid` 's bug needs caller-supplied buffers (web3.js doesn't), and `ws` is only reachable if the tool is pointed at a hostile WebSocket RPC (the core path uses HTTP). Listed for completeness; the dev-only highs don't affect installed users.

REPRODUCTION

```
full audit       : {"moderate":9,"high":5,"critical":0,"total":14}
prod (--omit=dev): {"moderate":6,"high":0,"critical":0,"total":6}
```

RECOMMENDED FIX

Add `overrides` to force `uuid ⩾11.1.1` / `ws ⩾8.20.1` where the graph allows (you already use `overrides` for `rpc-websockets → uuid`). Wire `npm audit --omit=dev` into CI so the prod signal isn't drowned by dev-only highs. Track upstream `@solana/web3.js` releases.

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

```
# Proposed fix - services/package.json + cli/package.json
   "overrides": {
+    "uuid": "⩾11.1.1",
+    "ws": "⩾8.20.1"
   },
   "scripts": {
+    "audit:prod": "npm audit --omit=dev"   // wire into CI so prod signal isn't drowned by dev-only highs
```

FINDING 09 / 9

INFO F9

Native SPL/System decoders read instruction data with the wrong encoding

INVARIANT Native SPL/System decoders read instruction data with the wrong encoding

AFFECTED CODE

`services/src/analysis/decoders/spl-token.ts:19-67`, `decoders/system-program.ts:37`. (`decoders/anchor-idl.ts` does it correctly — tries hex then base64.)

IMPACT

None today (dead code in the shipped pipeline). If these decoders are ever wired in, they'd display wrong SPL/System amounts/mints/authorities. Flagging so it's fixed before they go live.

REPRODUCTION

```
instruction data (hex, exactly as txParser.normalizeDataToHex produces) = 0c40420f000000000006
CORRECT decode → type=12 amount=1000000 decimals=6
TOOL decode    → {"instructionName":"unknown","rawData":"0c40420f000000000006"}
F9 CONFIRMED = true   ⟵ a valid TransferChecked decodes to unknown/garbage
```

(Harness: `services/repro-diff-decoder.ts`.)

RECOMMENDED FIX

Make the native decoders encoding-agnostic like `anchor-idl.ts` (try hex first, then base64). Add a fixture feeding `normalizeDataToHex` output through `decodeSPLInstruction`.

LAYER P3 — PROPOSED STRUCTURAL FIX (PATCH DIFF)

```
index 1a82a80..8309bd5 100644
--- a/services/src/analysis/decoders/spl-token.ts
+++ b/services/src/analysis/decoders/spl-token.ts
@@ -1,6 +1,18 @@
 import type { ParsedInstruction, TokenInstruction } from '../types.js';
 import { Buffer } from 'buffer';

+/**
+ * txParser normalises instruction data to lowercase hex, but RPC jsonParsed
+ * paths can still hand us base64. Decode hex when the string is valid even-length
+ * hex, otherwise fall back to base64 (mirrors decoders/anchor-idl.ts).
+ */
+function decodeInstructionBytes(data: string): Buffer {
+  if (data.length % 2 === 0 && /^[0-9a-fA-F]+$/.test(data)) {
+    return Buffer.from(data, 'hex');
+  }
+  return Buffer.from(data, 'base64');
+}
+
 /**
  * Decodes SPL Token program instructions.
  * Returns null if instruction is not SPL Token related or cannot be decoded.
@@ -17,8 +29,8 @@ export function decodeSPLInstruction(ix: ParsedInstruction): TokenInstruction |
   }

   try {
-    // Decode base64 payload into a binary buffer.
-    const dataBuffer = Buffer.from(ix.data, 'base64');
+    // Decode the payload (hex from txParser, or base64 from jsonParsed) into bytes.
+    const dataBuffer = decodeInstructionBytes(ix.data);

     if (dataBuffer.length < 1) {
       return null;
diff --git a/services/src/analysis/decoders/system-program.ts b/services/src/analysis/decoders/system-program.ts
index 1d894fe..33cd8ee 100644
--- a/services/src/analysis/decoders/system-program.ts
+++ b/services/src/analysis/decoders/system-program.ts
@@ -1,5 +1,17 @@
 import type { ParsedInstruction, SystemInstruction } from '../types.js';

+/**
+ * txParser normalises instruction data to lowercase hex, but RPC jsonParsed
+ * paths can still hand us base64. Decode hex when valid even-length hex,
+ * otherwise fall back to base64 (mirrors decoders/anchor-idl.ts).
+ */
+function decodeInstructionBytes(data: string): Buffer {
+  if (data.length % 2 === 0 && /^[0-9a-fA-F]+$/.test(data)) {
+    return Buffer.from(data, 'hex');
```

```
+  }
+  return Buffer.from(data, 'base64');
+}
+
 /**
  * System Program instruction discriminators (u32 little-endian).
  * Reference: https://github.com/solana-labs/solana/blob/master/sdk/program/src/system_instruction.rs
@@ -33,8 +45,8 @@ export function decodeSystemInstruction(ix: ParsedInstruction): SystemInstructio
   }

   try {
-    // Decode base64 data to buffer
-    const buffer = Buffer.from(ix.data, 'base64');
+    // Decode the data (hex from txParser, or base64 from jsonParsed) to buffer
+    const buffer = decodeInstructionBytes(ix.data);

     // Ensure buffer has at least 4 bytes for discriminator
     if (buffer.length < 4) {
```

A — SEVERITY RUBRIC

TIER	DEFINITION
CRITICAL	Direct loss of user funds or full protocol takeover with no meaningful preconditions. Reachable from a permissionless instruction by any signer. Must be patched immediately.
HIGH	Significant loss of user funds or protocol invariant violation under realistic preconditions (specific market state, signer with limited but obtainable role). Patch should ship in next release.
MEDIUM	Hardening issue, partial loss possible, or invariant violation requiring privileged signer or improbable state. Worth fixing in normal cadence.
LOW	Minor issue with no plausible path to fund loss. Code-quality or defense-in-depth concern.
INFO	Informational. No security impact. Documentation or style suggestion.

B — METHODOLOGY

Layer overview

LAYER	FUNCTION
`Layer 1`	Multi-agent recon. For each hypothesis, parallel LLM agents read the engine source and return a TRUE / FALSE / NEEDS_LAYER_2_TO_DECIDE verdict with confidence + per-agent grounding.
`Layer 1.5`	Adversarial debate. Contested verdicts (NEEDS_L2 or split verdicts) are promoted through a single-round attacker / defender debate, with a separate judge resolving the final verdict.
`Layer 2`	Concrete proof-of-concept. An inverted-assertion test is authored in Rust and run via `cargo test`. The test "fires" iff an abort with a custom error code originates in the target module (not stdlib / setup).
`Layer 2.5`	Triage. An LLM judge classifies each fire as `STRONG` (real bug), `SOFT` (wrong invariant), `FALSE` (artifactual abort), or `LOST` (signal missing). STRONG fires are clustered by (engine_function, target_file) so the same code-site bug under multiple hypothesis IDs collapses to one root cause.
`Layer 3`	Symbolic verification. Kani-based bounded model checking. The harness asserts the violated invariant; Kani either finds a counterexample within the bounded depth or proves safety.
`Layer 4`	On-chain BPF reproduction. The Solana program is deployed into LiteSVM and the PoC re-executed through the deployed instructions, confirming the wrapper-side defenses don't catch the bug.
`Layer P3`	Fix-bundle pipeline. The LLM authors a structural patch against the confirmed root cause and verifies it through a 6-gate machine check (well-formed diff, single-function scope, PoC fails pre-patch, PoC passes post-patch, existing tests still pass, and a language-specific symbolic/runtime check — Kani for Solana, Move Prover for Aptos, Halmos for Solidity, CBMC for C). Gates auto-skip when the language doesn’t apply (the symbolic / runtime gates of one toolchain skip on cycles authored against another, with that language’s verdict already reported under Layer 3 / Layer 4); the test-suite gate skips for eval targets that ship without a unified runner. Operator authorization is required before any upstream PR is opened.

Cycle execution

This cycle was produced by Jelleo's continuous, multi-agent security review loop. Every finding originates as a falsifiable invariant claim from a per-protocol hypothesis library, dispatched to Layer 1 multi-agent recon, promoted on contested verdicts via Layer 1.5 adversarial debate, and confirmed empirically through a Layer 2 `cargo test` proof-of-concept. Layer 2.5 triage classifies each fire as `STRONG` / `SOFT` / `FALSE` / `LOST`; only STRONG cluster representatives advance to `confirmed` and appear in §01 above. SOFT and STRONG duplicates land in `triaged`; FALSE fires return to `new`. Lifecycle: `new → triaged → confirmed → disclosed → fixed → verified`. Every cycle is signed Ed25519 against the platform key — see the cover-page receipt.

§ B.1 — Cycle funnel. Hypotheses tested → PoC fires → Layer 2.5 judge filters out artifactual / mis-invariant fires → surviving STRONG fires cluster by code site → cluster representatives become published findings.

C — AUDIT ARTIFACTS

All cycle artifacts are persisted on disk and verifiable independently of this report. The table below lists the canonical paths under the cycle workspace so a reviewer can re-execute every layer or recompute the cycle Merkle root.

ARTIFACT	PATH (RELATIVE TO WORKSPACE)
Cycle summary (manifest of every step)	`hunts/<cycle>/hunt_summary.json`
Per-step event log	`hunts/<cycle>/hunt.log.jsonl` (absent in this cycle)
Layer 2.5 triage verdicts	`hunts/<cycle>/triage.jsonl` (absent in this cycle)
Layer 2 PoC sources (Rust)	`hunts/<cycle>/poc/test_<slug>.rs` (absent in this cycle)
Layer 2 PoC run logs	`hunts/<cycle>/poc/cargo_<slug>.log` (absent in this cycle)
Layer 3 Kani harnesses + verdicts	`hunts/<cycle>/kani/<slug>/` (absent in this cycle)
Layer 4 LiteSVM exploit tests	`hunts/<cycle>/litesvm/<slug>/` (absent in this cycle)
Layer P3 fix bundles (patch.diff + evidence/ + manifest.json)	`hunts/<cycle>/bundles/<finding_id>/`
Narrative writeups (per finding)	`hunts/<cycle>/narratives/<hyp_id>.md`
Cycle Merkle root (tamper-evidence)	`hunts/<cycle>/merkle.json` (absent in this cycle)
Findings DB (SQLite)	`findings.db`
Ed25519 public key for receipt verification	`https://jelleo.com/keys/jelleo.ed25519.pub` (platform-wide key — served from jelleo.com/keys/)

D — DISCLAIMERS

Findings in this report reflect the state of the engine source at the commit hash on the cover page. Subsequent changes to the codebase are not analyzed. The report is not a guarantee of code correctness or security: it documents invariants that fired (or held) under the hypothesis library applied during this cycle. Out-of-scope items are listed in §00.1 (Scope).

§03 reflects bundle-level state. A row is treated as a confirmed finding when the bundle's machine verification gates (PoC fails pre-patch + PoC passes post-patch + tests still pass) all hold, even if the Layer 2.5 LLM judge initially classified the fire as `SOFT` / `FALSE` / `LOST` — the verifier's empirical patch-defuses-bug evidence supersedes the judge. Rows that did not reach a confirmed lifecycle state are retained in §03 as audit-trail evidence but are not published findings; the authoritative set is whatever appears in §01.

Communication channel: security@jelleo.com (PGP key on jelleo.com/security.html). Coordinated disclosure follows the timeline published in our security policy; pre-disclosure leak protections are enforced at the report level (the `--public` renderer suppresses confirmed-but-not-disclosed findings).

Methodology spec: docs/methodology/ · Live reference: jelleo.com/methodology.html · Source: github.com/Copenhagen0x/audit-pipeline-cli