IRONNODE

ironnode.io

DiversiFi

Arb Bot and On-chain Program

FINAL SECURITY AUDIT REPORT

by IronNode 12 October 2026

Executive overview

Contents

Section	Page
1 Executive Overview	4
2 Arb Bot Findings	14
3 On-chain Program Findings	51
4 Reward Sweep Review	58
5 Additional Controls Reviewed	59
6 Verification Methods and Results	60

About this report

This report presents the security assessment and remediation review of the DiversiFi arb bot and SolutioFi on-chain program. It contains the audit scope, reviewed source versions, findings, remediation results and supporting verification methods.

The finding descriptions and code excerpts refer to the audit baselines listed in Section 1.4. The remediation note at the end of each finding describes the changes in the corresponding remediation baseline. Severity describes the issue before remediation; the status describes its disposition after review.

1 Executive Overview

1.1 Audit Summary

IronNode assessed the DiversiFi arb bot and SolutioFi on-chain program, including reward sweeping, transaction accounting, pricing, settlement controls and key handling. The audit identified **35 findings: 12 Medium, 13 Low and 10 Informational**. No Critical or High findings were identified.

The remediation review records **23 verified fixes and 12 risk-reducing mitigations**. All 35 findings are marked Solved, which covers both outcomes as defined in Methodology and Status Definitions (Section 1.5). Operational Considerations (Section 1.7) explains the conditions that remain relevant to the mitigations.

Changes strengthen transaction validation, state handling, oracle checks and key management. Program changes add signed quote validation, 5% value-loss limits and removal of the bundled test key.

Reviewed arb-bot commit: `066b70d0eed89ad1654b7e364aca876aa368319c`

Reviewed program commit: `f41b2103129db93e8ffaf43bd34611025009b432`

1.2 System Overview

DiversiFi operates a Solana-based index system. Its on-chain program holds constituent assets, issues and redeems index shares, and supports portfolio rebalancing. The arb bot compares the index value with the pool price and trades when it identifies a premium or discount.

The reward-sweep instruction allows an authorized administrator to transfer eligible non-constituent reward tokens from an index vault to a fixed destination. The review of its authorization, destination and protected-asset checks is presented in Reward Sweep Review (Section 4).

1.3 Audit Scope

The assessment covers the diversifi-arb repository and the solutiofi-program repository at the source baselines listed in Section 1.4. The scope includes the bot and its local-fork tools; the Anchor program, including sweep_reward; and related configuration, test helpers, deployment runbooks and dependency lockfiles. Remediation review covers the changes addressing all 31 bot findings and four program findings, together with affected callers and controls.

1. **Transaction and escrow accounting**

Reviewed confirmation, retry decisions, stored progress, escrow-read freshness, history boundaries and completed-trade accounting.

2. **Pricing and execution limits**

Reviewed quote binding, secondary prices, cached valuations, price timestamps, route validation and top-up execution gates.

3. **Operational controls**

Reviewed endpoint encryption, mainnet confirmation, environment visibility, signer consistency, process locks, durable records and spending limits.

4. **Keys and logs**

Reviewed checksum handling, locally derived signatures, credential redaction, local-fork output and generated keypair permissions.

5. **Program and reward sweeping**

Reviewed administrator and manager authorization, canonical vaults, protected reward mints, oracle-to-asset binding, settlement and rebalance limits, test-key handling and minimum-deposit controls.

1.4 Reviewed Source Versions

The audit baseline is the source version used to identify and describe findings. The remediation baseline is the source version used to review the changes. Each entry records the repository and commit reference. The archive hash identifies the exact source package examined.

Arb bot - audit baseline

Repository: https://github.com/SolutioFi-io/diversifi-arb

Commit reference: `ea68d19588ab361c05f0325cc84126c63e011c42`

Archive: `diversifi-arb-main(2).zip`

SHA-256: `a6ec10ea41e2692d98dd100bedfc6868adf031d775bc89545251e84faf2d1958`

On-chain program - audit baseline

Repository: https://github.com/SolutioFi-io/solutiofi-program

Commit reference: `4e428ea1b5bde06f2180da8d5136921ee176dd81`

Archive: `solutiofi-program-sweep_rewards.zip`

SHA-256: `16c2eba1af7a9cbf41ab726232e7993f2cfd18386cf909b7a809c94e22c1f327`

1.4 Reviewed Source Versions - Remediation

Arb bot - remediation review

Repository: https://github.com/SolutioFi-io/diversifi-arb

Commit reference: `066b70d0eed89ad1654b7e364aca876aa368319c`

Archive: `diversifi-arb-fix-audit-findings.zip`

SHA-256: `fab915b5440237a81f8dfac3375062634a5b69b9717a981970efbc8572012560`

On-chain program - remediation review

Repository: https://github.com/SolutioFi-io/solutiofi-program

Commit reference: `f41b2103129db93e8ffaf43bd34611025009b432`

Archive: `solutiofi-program-fix-audit-findings.zip`

SHA-256: `d9f51ac1fdc331cf7b0d20a5a16bcbe8fa8dac6185255ca2e178c9d2decc23ed`

1.5 Methodology and Status Definitions

The audit used source review to examine authorization, transaction handling, accounting, pricing and operational controls. Findings were assessed against their potential impact and the conditions needed to trigger them.

Each remediation baseline was then compared with its audit baseline. All 31 bot and four program findings were traced through the affected functions, callers and controls, including persistence, settlement, quote storage and startup paths.

The bot remediation review included a passing typecheck and 78 focused unit checks. Program remediations were verified by source review against the original findings, including inspection of regression-test code and independent arithmetic checks. Verification Methods and Results (Section 6) describes these checks.

Status	Meaning
SOLVED	A verified fix or a reviewed mitigation reduces the reported risk. Conditions relevant to the mitigations are described in Operational Considerations (Section 1.7).
NOT APPLICABLE	The reported issue does not apply or was refuted by the available evidence.
RISK ACCEPTED	An unresolved non-informational issue retained as an accepted risk.
ACKNOWLEDGED	An informational item that remains unchanged.

Severity

Severity combines potential impact with the access, configuration, timing and other conditions needed to trigger an issue. Medium findings describe material risk subject to significant conditions. Low findings describe narrower impact or more restricted exposure. Informational findings identify hardening, maintainability or documentation improvements without an established direct security impact. Each rating describes the audit baseline, before remediation.

For example, the unencrypted-RPC issue is Medium because it requires a plaintext connection and an attacker on its network path. Signed transactions remain protected by their signatures. The program economic findings are Medium because they require privileged processing and adverse pricing or execution conditions. The test-key disclosure is Low because the identity differs from the compiled administrator and no active production authority was established.

1.6 Findings Summary

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	12	13	10

35 findings: 35 Solved.

31 arb-bot findings and four on-chain program findings. The table lists each finding, its pre-remediation severity and its final status.

Arb bot: 9 Medium, 12 Low, 10 Informational

Security analysis	Risk Level	Final status
2.1 Live trading permits unencrypted RPC connections	Medium	SOLVED
2.2 A completed pool swap can be submitted again	Medium	SOLVED
2.3 A new trade can use a pool quote from before the bot's own sale	Medium	SOLVED
2.4 Missing secondary prices do not stop constituent valuation	Medium	SOLVED
2.5 An empty status response can erase a recorded deposit or withdrawal result	Medium	SOLVED
2.6 A stale escrow read can finish a flow before processing occurs	Medium	SOLVED
2.7 Incomplete transaction history can mix results from different flows	Medium	SOLVED
2.8 Cached valuation can hide a new price warning	Medium	SOLVED
2.9 Top-up transactions are not checked against the quoted minimum output	Medium	SOLVED
2.10 An unconfirmed transaction error is treated as final	Low	SOLVED
2.11 The bot repeatedly pays to recreate third-party fee accounts	Low	SOLVED
2.12 A clock change can allow two bot processes to share one state directory	Low	SOLVED
2.13 The balance decoder allocates from an unchecked mint count	Low	SOLVED
2.14 Unchecked quote fields can understate trading costs	Low	SOLVED
2.15 Unrelated transactions can make escrow measurement expensive	Low	SOLVED
2.16 A power failure can restore an older loss-control state	Low	SOLVED

1.6 Arb Bot Findings Overview

Security analysis	Risk Level	Final status
2.17 RPC keys embedded in URL paths can appear in logs	Low	SOLVED
2.18 Environment names are misleading and live startup omits the selected target	Low	SOLVED
2.19 Configuration can silently disable two safety checks	Low	SOLVED
2.20 Top-up swaps bypass some trading-stop checks	Low	SOLVED
2.21 A local proxy can bypass the remote-trading confirmation	Low	SOLVED
2.22 The development signer applies different checks from the KMS signer	Info	SOLVED
2.23 Public-key checksum validation is optional during KMS setup	Info	SOLVED
2.24 Refund tracking trusts the submission endpoint's returned signature	Info	SOLVED
2.25 The saved transaction group does not separate escrow histories	Info	SOLVED
2.26 Malformed backend responses are not handled consistently	Info	SOLVED
2.27 Future price timestamps are accepted as fresh	Info	SOLVED
2.28 The bot assumes the protocol fee will remain zero	Info	SOLVED
2.29 The top-up budget resets at midnight rather than over a rolling day	Info	SOLVED
2.30 A generated fork key file uses default filesystem permissions	Info	SOLVED
2.31 Local-fork tools print complete RPC URLs	Info	SOLVED

On-chain program: 3 Medium, 1 Low

Security analysis	Risk Level	Final status
3.1 Changing constituent order can assign prices to the wrong assets	Medium	SOLVED
3.2 Rebalance lacks an independent minimum output	Medium	SOLVED
3.3 User requests do not set minimum settlement amounts	Medium	SOLVED
3.4 A reusable test signing key is stored in the repository	Low	SOLVED

1.7 Operational Considerations

The reviewed changes improve security, but some mitigations depend on integration and operating conditions. This section describes those conditions and the related checks for the release. References point to findings within this report.

Arb bot

1. **Transaction and escrow history**

Reconcile uncertain swap outcomes against complete, consistent history. Persist the latest escrow observation slot even when no initialization-status slot is available. Sections 2.2 and 2.6.

2. **Current price quality**

Decide whether the below-1% missing-price allowance is appropriate. Refresh the primary price and its age when reassessing cached valuation quality. Sections 2.4 and 2.8.

3. **Route validation**

Parse the accepted variable-length Jupiter route layouts completely and verify that the checked fields match the fields used on chain. Section 2.9.

4. **Rent and measurement limits**

Make third-party rent accounting durable and account for changes before execution. Add a deliberate total-work policy for unusually large escrow histories. Sections 2.11 and 2.15.

5. **Upgrade coordination**

Verify the bot's protocol-fee assumption against the intended program release. Section 2.28.

1.7 Operational Considerations - Program

1. **Oracle producer and integration**

The client reports that the backend producer has been updated. The reviewed program requires feed IDs derived from mint bytes, a pinned signer and a signed numeric slot. Repository examples use different feed IDs and standard Switchboard quotes, whose slot field is a hash; those examples do not describe the updated backend. Confirm that the deployed producer and program use the same quote format and provide fresh validated prices. Related finding: incorrect price-to-asset association (Section 3.1).

2. **Rebalance callers and policy**

Update transaction builders and interface definitions (IDLs) for the new oracle account; the disabled rebalance example in the repository still omits it. Test successful and rejected swaps, token extensions and rollback. The 5% oracle-value floor does not impose a separate maximum input. Confirm that this policy is intended. Related finding: rebalance execution limits (Section 3.2).

3. **User-authorized settlement**

Requests do not include user-signed minimum shares, minimum net USDC or deadlines. The 5% guard applies per leg; withdrawal checks gross USDC before fees. Settlement inputs are read from route data rather than verified as source-vault balance changes, and USDC is valued at $1. These protocol-level checks do not replace a user-specific settlement limit. Related finding: user settlement amounts (Section 3.3).

4. **Exposed legacy identity**

Removing the key does not revoke existing copies. Confirm it has no active roles or funds, rotate it wherever needed, and configure an authorized local test identity. Legacy public-address constants remain in the test helpers. Section 3.4.

Before deployment

Verify the integrated bot and program release, quote producer, generated clients, production settings, deployed program identity and key ownership.

Findings

2 Arb Bot Findings

2.1 Live trading permits unencrypted RPC connections

MEDIUM Remediation status: SOLVED

The live runner accepts a nonlocal HTTP address for its RPC connection and transaction submission endpoint. RPC_URL has no transport check, while PEG_KEEPER_SEND_URL only has to be a valid URL. An operator can therefore arm the bot with an unencrypted connection.

An attacker who can intercept that connection may read credentials carried in the URL and alter the responses delivered over it. If the read RPC is exposed, affected data can include balances, prices, transaction status, and the measurements used to calculate profit and loss. Those measurements also feed the loss controls, so they are not an independent safeguard against false RPC data.

The scope matters. An HTTP submission endpoint does not give an attacker control over a separate HTTPS read connection. The attacker also cannot change an already signed transaction without invalidating its signature. This issue is not a break of Solana signatures or of HTTPS.

2.1.1 Code location - audit baseline

`src/config.ts:225` - the RPC URL is taken with no scheme check

```
// src/config.ts:225
  rpcUrl: requireKey("RPC_URL"),
```

`src/live.ts:379-387` - the send URL only has to parse

```
// src/live.ts:379-387
  try {
    new URL(p.live.sendUrl);
  } catch {
    out.push({
      code: "send_url_unparseable",
      message: "PEG_KEEPER_SEND_URL is not a parseable URL",
    });
  }
  return out;
```

`src/live.ts:263-266` - the websocket URL is checked for scheme, and plain `ws://` is accepted

```
// src/live.ts:263-266

  try {
    const u = new URL(live.wsUrl);
    if (u.protocol !== "wss:" && u.protocol !== "ws:")
      return "PEG_KEEPER_WS_URL must be a ws:// or wss:// URL";
```

2.1.2 Impact

A deployment using an exposed plaintext read connection could make trades or record their results using false information. Losses may not trigger checks that depend on the same altered responses. Credentials sent over that connection may also be disclosed. Actual financial loss depends on the configured endpoints, the attacker's network position, and subsequent trade execution.

2.1.3 Recommendation

Require HTTPS for nonlocal RPC and transaction submission endpoints before live mode can start. Require WSS for nonlocal subscriptions as well. Keep any local-development exception explicit. For important settlement decisions, consider independent RPC checks and stop processing when the results disagree.

2.1.4 Severity rationale

Medium. The missing transport check is confirmed, but exploitation requires an operator-selected plaintext endpoint and an active attacker on its network path. No insecure endpoint default or production exposure was established. Financial consequences can be serious, and checks using the same altered read data are not independent protection. These conditions support a Medium severity rating.

Remediation

SOLVED: Client solved this issue by rejecting unencrypted RPC, send and websocket endpoints outside loopback.

Remediation source: src/live.ts:326; src/live.ts:414.

2.2 A completed pool swap can be submitted again

MEDIUM Remediation status: SOLVED

When a pool swap has no clear result, the bot checks whether its blockhash has expired and then asks for the transaction status. If the status is empty, it clears the saved swap and may build another one.

These are separate RPC reads. A load-balanced service can return a recent block height from one node and an incomplete status from another. A node that does not retain transaction history can also return no status for a swap that already completed. An empty response is therefore not proof that the swap never happened.

For a sale, the bot may sell again from other tokens in the wallet. If there are not enough tokens, it can repeatedly rebuild a failing sale without recording the first sale's proceeds. For a buy, it can close the trade with zero recorded profit or loss while the purchased tokens remain untracked.

2.2.1 Code location - audit baseline

src/sender.ts:118-129 - expiry is two separate reads, and an empty status counts as expired

```
// src/sender.ts:118-129

  export async function signatureExpired(
    connection: Pick<Connection, "getBlockHeight" | "getSignatureStatuses">,
    signature: string,
    lastValidBlockHeight: number
  ): Promise<boolean> {
    const height = await connection.getBlockHeight("confirmed");
    if (!(height > lastValidBlockHeight)) return false;
    const res = await connection.getSignatureStatuses([signature], {
      searchTransactionHistory: true,
    });
    return !res.value[0];
  }
```

src/submitter.ts:629-636 - the consumer clears the swap, so the next tick builds a new one

```
// src/submitter.ts:629-636
  if (
    verdict === "failed" ||
    (await signatureExpired(this.o.connection, swap.signature, swap.lastValidBlockHeight))
  ) {
    trade.swap = null;
    trade.lastError = `pool swap ${swap.signature} ${verdict === "failed" ? "failed" : "expired
  unconfirmed"}`;
    this.save(trade);
  }
```

2.2.2 Impact

The bot can sell additional inventory, leave purchased tokens untracked, or keep a trade open indefinitely. Its recorded result can be wrong, which also affects the loss controls.

2.2.3 Recommendation

Do not treat an empty status as proof that a transaction failed to complete. Reconcile the saved signature against a consistent RPC view and relevant account history. Require historical transaction support. Keep the trade pending and alert an operator when the available evidence is incomplete, instead of rebuilding the swap.

Remediation

SOLVED: Client added slot and history checks and holds trades when transaction status cannot be established.

Remediation source: src/sender.ts:143; src/submitter.ts:655.

2.3 A new trade can use a pool quote from before the bot's own sale

MEDIUM Remediation status: SOLVED

Each live-loop cycle reads the market, advances existing trades, and then starts new trades from the quotes it read earlier in that cycle.

Advancing an existing trade can sell newly minted index tokens into the pool and release the vault. That sale changes the pool price. The loop can then start a second trade using the quote taken before its own sale. The quote is recent enough to pass the age limit, but it no longer describes the pool.

The second trade deposits funds to capture a premium that the first sale may already have removed. When those funds become index tokens, the bot may be unable to sell them above its break-even limit.

2.3.1 Code location - audit baseline

`src/live.ts:572-588` - quotes are built before `advance()` runs and are executed after it

```
// src/live.ts:572-588

  const quotes = (await this.contain("observe", failedStages, () => this.o.observe())) ?? [];
  await this.contain("advance", failedStages, () => this.o.submitter.advance());
  await this.contain("claims", failedStages, () => this.reconcileUnbacked());
  const unbacked = this.unbacked();
  const attempts: ExecutionAttempt[] = [];
  let offered = 0;
  let unbackedRefusal = false;
  for (const quote of quotes) {
    if (!quote.actionable) continue;
    offered++;
    const attempt = await this.contain("execute", failedStages, () =>
      this.o.executor.execute({ quote })
    );
    if (!attempt) continue;
    attempts.push(attempt);
    if (
      unbacked.length > 0 &&
```

2.3.2 Impact

Up to one trade's configured limit can remain tied up in index tokens until a suitable price returns. Fees have already been paid, the position remains exposed to changes in asset value, and the claimed vault can prevent other trades.

2.3.3 Recommendation

Refresh the pool and rebuild quotes after a cycle submits a pool swap, or skip new execution in that cycle. Alternatively, reject quotes from before the bot's latest pool-changing transaction.

Remediation

SOLVED: Client solved this issue by skipping new trades when an existing trade finishes after the pool quote was read.

Remediation source: src/live.ts:638; src/live.ts:1103.

2.4 Missing secondary prices do not stop constituent valuation

MEDIUM Remediation status: SOLVED

The bot calculates net asset value from Jupiter prices and checks those prices against Birdeye. A material disagreement can mark the valuation as degraded or stale and stop trading.

For a funded constituent with no recent Birdeye price, the comparison is skipped without changing the valuation's status. The bot can therefore treat a Jupiter-only price as fresh. This is weaker than the rule used for SOL, where a current secondary price is required.

The gap matters for a thinly traded constituent whose recent trade price can move substantially. A distorted constituent price can inflate the basket value and make the bot see a discount that is not supported by the vault's settlement price.

2.4.1 Code location - audit baseline

`src/nav.ts:766-774` - a funded constituent with no current second-source price is noted and skipped

```
// src/nav.ts:766-774

  const persistReads = thresholds.divergencePersistReads;
  let pastBlock = false;
  for (const sample of params.samples) {
    const other = params.secondSource.get(sample.mint);
    if (other === undefined) {
      reasons.push(`${sample.mint} has no second-source price, so it is uncorroborated`);
      continue;
    }
    const bps = priceDivergenceBps(sample.usdPrice, other);
```

2.4.2 Impact

The bot may overpay for index tokens based on an unsupported valuation. A later settlement check may detect the loss, but it acts after the trade. Exposure remains subject to the configured trade limit and loss-control behavior.

2.4.3 Recommendation

Mark material funded constituents as degraded when no current secondary price is available. Where practical, compare against the price source used by the vault for settlement. Apply reasonable price-movement limits and require fresh supporting prices before trading.

Remediation

SOLVED: Client added a 1% materiality threshold that blocks valuation when material secondary-price coverage is missing.

Remediation source: src/nav.ts:58; src/nav.ts:839; src/nav.ts:1019.

2.5 An empty status response can erase a recorded deposit or withdrawal result

MEDIUM Remediation status: SOLVED

The bot records that an initialization transaction completed when it first confirms the escrow. Later, after the deposit or withdrawal reaches a finished state, the measurement routine asks for that transaction's status again.

If the new answer is empty or otherwise not marked as completed, the routine returns a result saying the initialization did not complete, with zero balance changes. It does not preserve the earlier confirmation.

An incomplete RPC response can therefore replace known progress with an incorrect result. This can happen with lagging nodes, missing transaction history, or a restart after the transaction leaves a node's recent-status cache.

2.5.1 Code location - audit baseline

`src/submitter.ts:645-651` - the initialize's status is re-read at the flow's end; an empty answer means "not landed"

```
// src/submitter.ts:645-651

    index: IndexState,
    flow: FlowState
  ): Promise<{ landed: boolean; usdc: bigint; dToken: bigint } | null> {
    const after = trade.vaultSignature;
    if (after === null || (await this.status(after)) !== "landed")
      return { landed: false, usdc: 0n, dToken: 0n };
    const totals = await this.meter.measure({
```

2.5.2 Impact

Spent funds or received assets can disappear from the trade's accounting even though they moved on chain. The bot may release the vault with tokens left untracked, and the loss controls may not receive the actual cost.

2.5.3 Recommendation

Preserve the flow's confirmed history. Use the recorded escrow state when deciding whether initialization completed. If later data is missing, leave measurement pending and retry rather than returning a result that contradicts an earlier confirmation.

Remediation

SOLVED: Client solved this issue by preserving recorded escrow funding when a later signature-status response is empty.

Remediation source: src/submitter.ts:700; src/flow.ts:215; src/flow.ts:498.

2.6 A stale escrow read can finish a flow before processing occurs

MEDIUM Remediation status: SOLVED

After confirming initialization, the bot immediately reads the escrow balance. Once a flow has held funds, a zero balance is interpreted as completion and the finished state is no longer updated.

The read does not require a context at least as recent as the initialization transaction. A lagging RPC node can report the account as absent, or return a balance from the previous use of that escrow. The bot can then mark the new flow as finished before the keeper has processed it.

The next measurement finds no completed processing transactions and records an incorrect result.

2.6.1 Code location - audit baseline

src/submitter.ts:539-543 - the landing is recorded, then the escrow is read with no minimum slot

```
// src/submitter.ts:539-543

  // read is still seen to have held it.
  if ((await waitForSignature(this.status, signature, this.o.confirm)) === "landed") {
    this.o.tracker.recordLanded(flow.id);
    await this.o.tracker.observe(flow.id);
  }
```

`src/flow.ts:387-413` - a zero balance after `everEscrowed` settles the flow, and a settled flow is frozen

```
// src/flow.ts:387-413
  private applyBalance(tracked: TrackedFlow, balance: bigint): void {
    const now = this.now();
    const { state, book } = tracked;
    const dust = this.dustFor(state.kind);

    if (balance !== state.escrowedBalanceRaw) {
      book.lastProgressAt = now;
      state.escrowedBalanceRaw = balance;
    }
    state.lastObservedAt = now;

    if (balance > 0n) {
      book.everEscrowed = true;
      if (balance > dust) book.everAboveDust = true;

      // Complete with dust: a residual this small is not worth the fee to refund, and an exact
      // zero may never come. A flow that was always tiny is not complete on arrival.
      if (book.everAboveDust && balance <= dust) {
        state.phase = "settled";
        return;
      }

      if (state.phase === "planned" || state.phase === "submitted") state.phase = "escrowed";
      return;
    }

    if (!book.everEscrowed) return;
```

2.6.2 Impact

Trading can stop on a false loss and the vault can be released too early. Assets received after processing can remain outside the finished trade's accounting. The problem can recur when RPC nodes return inconsistent views.

2.6.3 Recommendation

Read the escrow using minContextSlot set to at least the initialization transaction's confirmed slot. Treat an older or unavailable view as unknown. Do not finish a flow based on state that predates its own initialization.

Remediation

SOLVED: Client saved initialization slots and added checks that reject older escrow reads.

Remediation source: src/flow.ts:179; src/flow.ts:203; src/flow.ts:414; src/flow.ts:452.

2.7 Incomplete transaction history can mix results from different flows

MEDIUM Remediation status: SOLVED

The escrow measurement routine walks transaction history until it reaches the current flow's initialization signature. It adds up relevant balance changes along the way.

The RPC until argument is a stopping point, not proof that the node has indexed that signature. If the node has not indexed it, the walk can continue into older activity. Because escrows are reused, earlier deposits, withdrawals, and refunds can then be included in the current flow's result. The code does not confirm that it actually reached the intended boundary.

The resulting token amount or proceeds can be overstated. A sale based on that total can use other wallet inventory while applying a price limit calculated for only the current trade.

2.7.1 Code location - audit baseline

`src/measure.ts:229-239` - `until` is a stopping point; a node that has not indexed it lists older history

```
// src/measure.ts:229-239

  for (;;) {
    const page: ConfirmedSignatureInfo[] = await this.connection.getSignaturesForAddress(
      p.escrow,
      {
        until: walk.boundary,
        before: walk.pass?.before,
        limit: PAGE_SIZE,
        minContextSlot: p.minContextSlot,
      },
      "confirmed"
    );
```

src/measure.ts:277-282 - a short page completes the walk, with no proof the boundary was reached

```
// src/measure.ts:277-282

  if (page.length < PAGE_SIZE) {
    walk.totals = walk.totals.map((t, i) => t + pass.partial[i]!);
    walk.boundary = pass.head;
    walk.pass = null;
    return [...walk.totals];
  }
```

2.7.2 Impact

Trade sizing, proceeds, and profit or loss can be calculated from unrelated transactions. The bot may sell more inventory than the current flow produced or fail to detect a loss because its recorded proceeds are inflated.

2.7.3 Recommendation

Require proof that the history walk reached the current flow's initialization before accepting its totals. Store and use the initialization slot as additional context, but do not assume that a recent node slot proves its transaction index is complete. Keep measurement pending if the boundary is missing.

Remediation

SOLVED: Client solved this issue by requiring the escrow history to reach the recorded boundary before accepting totals.

Remediation source: src/measure.ts:292; src/measure.ts:344.

2.8 Cached valuation can hide a new price warning

MEDIUM Remediation status: SOLVED

When react-to-pool mode is enabled, the bot can reuse a recent net asset value instead of pricing every constituent again. The reuse window defaults to 20 seconds. This mode is disabled by default.

The cache also reuses the valuation's fresh status. During the window, the price-comparison routine is not called again, so a new disagreement between Jupiter and Birdeye does not change that status. The counter used to identify repeated price disagreement does not advance either.

The original read time is retained, so the normal quote-age limit still works. The missing protection is a current price-quality decision, not timestamp aging.

2.8.1 Code location - audit baseline

src/poolwatch.ts:45-75 - a fresh NAV is returned from cache, verdict and all, without consulting the NAV module

```
// src/poolwatch.ts:45-75

  export class ReusedNav implements NavModule {
    private readonly last = new Map<string, { nav: NavResult; readAt: Date; shape: string }>();

    constructor(
      private readonly inner: NavModule,
      private readonly reuseMs: number,
      private readonly now: () => Date = () => new Date()
    ) {}

    async computeNav(index: IndexState): Promise<NavResult> {
      const readAt = this.now();
      const key = index.address.toBase58();
      // A change of constituents, weights or fee is a different vault to price.
      const shape = [
        index.feeBps,
        ...index.constituents.map((c) => `${c.mint.toBase58()}:${c.targetBps}`),
      ].join(",");
      const hit = this.last.get(key);
      if (
        hit &&
        hit.shape === shape &&
        readAt.getTime() - hit.readAt.getTime() >= 0 &&
        readAt.getTime() - hit.readAt.getTime() < this.reuseMs
      ) {
        return { ...hit.nav, reusedFromReadAt: hit.readAt };
      }
      this.last.delete(key);
      const nav = await this.inner.computeNav(index);
      if (nav.staleness.level === "fresh") this.last.set(key, { nav, readAt, shape });
      return nav;
    }
```

src/oracle.ts:58-63 - the only pre-trade NAV gate reads the cached verdict

```
// src/oracle.ts:58-63

  export function staleNavCheck(): PreTradeCheck {
    return (quote) =>
      quote.nav.staleness.level === "stale"
        ? { code: "oracle_stale", message: "the NAV verdict is stale" }
        : null;
  }
```

2.8.2 Impact

During the reuse window, trades can rely on a valuation that would now fail the price-quality checks. The exposure applies only when the optional react-to-pool mode is enabled and relevant prices change during that window.

2.8.3 Recommendation

Evaluate current price quality before every trade, even if the numerical valuation is cached. Do not cache a fresh verdict as if it were still current. If fresh supporting checks are unavailable, treat the cached value as degraded and refuse execution.

Remediation

SOLVED: Client added secondary-price checks before reusing cached valuations and refreshes values when those checks fail.

Remediation source: src/poolwatch.ts:70; src/nav.ts:1059; src/index.ts:706.

2.9 Top-up transactions are not checked against the quoted minimum output

MEDIUM Remediation status: SOLVED

The top-up flow checks a Jupiter quote's assets, input amount, and minimum output against the bot's own SOL price. It then requests a transaction for that quote and reviews its instructions.

The review recognizes the Jupiter route instruction but does not decode its execution limits or fully bind its accounts and amounts to the approved quote. A transaction returned with weaker terms can therefore pass review even though the quote itself passed stronger checks.

Simulation checks the output under the simulated market state. It does not guarantee that the signed transaction will reject a worse fill when executed later. Exploitation requires a faulty or compromised swap response and adverse execution; a correct response that preserves the checked limits does not create this mismatch.

2.9.1 Code location - audit baseline

`src/topup.ts:363-367` - the Jupiter route is counted, never decoded

```
// src/topup.ts:363-367

  }
  if (program.equals(JUPITER_V6_PROGRAM_ID)) {
    routes++;
    continue;
  }
```

2.9.2 Impact

A top-up can execute at a worse price than the bot approved. Potential exposure is limited by the top-up settings, which default to $50 per swap and $150 per day. No production loss was established during the audit.

2.9.3 Recommendation

Decode the accepted Jupiter route formats and compare their input amount, minimum output, fee settings, and relevant accounts with the approved quote. Reject any mismatch. Building the transaction locally from a verified route is another option.

Remediation

SOLVED: Client added route checks that compare accounts, amounts, slippage and fees with the approved quote.

Remediation source: src/topup.ts:295; src/topup.ts:403; src/jupiter_route.ts:187; src/jupiter_route.ts:284.

2.10 An unconfirmed transaction error is treated as final

LOW Remediation status: SOLVED

The status reader requires confirmed or finalized commitment before reporting success, but it reports failure whenever an RPC status contains an error, including at processed commitment.

A processed result can still change if it belongs to a discarded fork. The bot may close a buy or clear a sale for rebuilding before the transaction's outcome is sufficiently settled. If the transaction later succeeds on the accepted chain, its assets or proceeds can be left untracked.

2.10.1 Code location - audit baseline

`src/sender.ts:105-111` - an error at any commitment, `processed` included, is final

```
// src/sender.ts:105-111
  });
  const status = res.value[0];
  if (!status) return "unknown";
  if (status.err) return "failed";
  return status.confirmationStatus === "confirmed" || status.confirmationStatus === "finalized"
    ? "landed"
    : "unknown";
```

2.10.2 Impact

A trade can be closed with the wrong result, or a replacement sale can be prepared while the original remains capable of completing. This needs a change in the chain view, so the expected frequency is limited.

2.10.3 Recommendation

Apply the same commitment requirement to failure and success. Treat processed-only errors as unknown until a confirmed or finalized result, or other conclusive expiry evidence, is available.

Remediation

SOLVED: Client solved this issue by treating errors as final only after confirmed or finalized commitment.

Remediation source: src/sender.ts:111.

2.11 The bot repeatedly pays to recreate third-party fee accounts

LOW Remediation status: SOLVED

A deposit can create missing USDC fee accounts for the manager and treasury, with the bot paying the Solana account rent. The transaction verifier permits these creations.

The manager controls its ordinary token account and can withdraw its balance and close it. A later deposit then recreates the account at the bot's expense. The checks prevent duplicate creation within one transaction but do not limit repeated creation across transactions.

2.11.1 Code location - audit baseline

src/txverify.ts:123-133 - the bot may pay to create the treasury's and the manager's USDC accounts

```
// src/txverify.ts:123-133

  // Token accounts the bot may pay rent to create, each at most once: its own USDC and dToken
  // accounts, and on a deposit the treasury and manager USDC vaults the instruction pays fees
  into.
  function creatableAccounts(expect: TxExpectation): PublicKey[] {
    const own = [ataOf(USDC_MINT, expect.user), ataOf(expect.dTokenMint, expect.user)];
    if (expect.operation !== "initialize_deposit") return own;
    return [
      ...own,
      ataOf(USDC_MINT, TREASURY),
      ...(expect.manager ? [ataOf(USDC_MINT, expect.manager)] : []),
    ];
  }
```

programs/solutiofi-program/src/contexts/initialize_deposit.rs:48-60 - the program requires both accounts to exist, so a missing one must be created in the bot-signed build

```
// programs/solutiofi-program/src/contexts/initialize_deposit.rs:48-60

  #[account(
      mut,
      associated_token::authority = index.manager,
      associated_token::mint = USDC,
  )]
  pub manager_usdc_vault: Option<Box<Account<'info, TokenAccount>>>,
  #[account(
      mut,
      associated_token::authority = TREASURY,
      associated_token::mint = USDC,
  )]
  pub treasury_usdc_vault: Box<Account<'info, TokenAccount>>,
  pub system_program: Program<'info, System>,
```

2.11.2 Impact

A cooperating or malicious manager can repeatedly collect account rent paid by the bot, approximately 0.002 SOL per deposit under the assessed conditions. The treasury account is not assumed to be similarly closable because its owner is a program address.

2.11.3 Recommendation

Require fee recipients to maintain their own accounts, or cap the rent the bot may pay for accounts it does not own. Alert an operator when repeated creation exceeds the policy.

Remediation

SOLVED: Client added a per-process guard that refuses repeated rent payments for previously recorded third-party accounts.

Remediation source: src/txverify.ts:140; src/submitter.ts:540.

2.12 A clock change can allow two bot processes to share one state directory

LOW Remediation status: SOLVED

The bot uses a lock file to prevent two live processes from sharing its state. It estimates the machine's boot time from the current wall clock minus uptime and allows a five-minute difference.

If the clock changes by more than that tolerance, a second process can interpret a live lock as belonging to a previous boot. It can take the lock without checking whether the original process is still running. The first process does not recheck ownership.

2.12.1 Code location - audit baseline

`src/executor.ts:177-188` - boot identity is derived from the wall clock

```
// src/executor.ts:177-188
  const SAME_BOOT_MS = 5 * 60_000;

  export function currentLockOwner(): LockOwner {
    return {
      pid: process.pid,
      host: hostname(),
      bootAtMs: Math.round(Date.now() - uptime() * 1000),
      instance: INSTANCE,
    };
  }

  function parseLockOwner(raw: string): LockOwner | null {
```

2.12.2 Impact

Both processes may trade and overwrite shared state. Their separate in-memory rate limits and claims do not coordinate, so caps can be exceeded and journal entries can be lost.

2.12.3 Recommendation

Use an operating-system lock held for the process lifetime and a stable boot identifier where available. Do not infer process death from wall-clock changes. Check that the recorded owner is no longer running before taking over a lock, and detect lost lock ownership during operation.

Remediation

SOLVED: Client solved this issue by using the kernel boot ID and checking lock ownership before execution.

Remediation source: src/executor.ts:192; src/executor.ts:358; src/executor.ts:624.

2.13 The balance decoder allocates from an unchecked mint count

LOW Remediation status: SOLVED

The vault-balance decoder reads the number of mints from RPC return data and loops over that count before validating the data length.

Reading beyond the buffer yields empty slices rather than immediately rejecting the count. The loop can therefore consume substantial memory and CPU before the later checks reject the response. A compromised RPC response or an incompatible program return layout could trigger the problem.

2.13.1 Code location - audit baseline

src/nav.ts:236-241 - the loop runs to the count in the data before any length check

```
// src/nav.ts:236-241

  const mintCount = readU32();
  const mints: PublicKey[] = [];
  for (let i = 0; i < mintCount; i++) {
    mints.push(new PublicKey(buf.subarray(offset, offset + 32)));
    offset += 32;
  }
```

2.13.2 Impact

Malformed data can stall or terminate the bot. Repeated delivery of the same response can cause repeated failures after restart. No direct loss of custody was established from this parsing issue.

2.13.3 Recommendation

Validate both array counts against the remaining buffer length and the program's maximum constituent count before allocating or looping. Verify that the balance and mint counts match, and reject incomplete or unexpected layouts immediately.

Remediation

SOLVED: Client solved this issue by checking constituent counts and available bytes before decoding arrays.

Remediation source: src/nav.ts:233; src/nav.ts:264.

2.14 Unchecked quote fields can understate trading costs

LOW Remediation status: SOLVED

The cost model accepts Jupiter quote fields without verifying that the returned mints and input amount match the request. It then uses the reported output to estimate each constituent trade's cost.

If a quote implies an output worth more than the input, that leg contributes a negative cost. These credits are added to the total without a separate limit or confirmation. A genuinely favorable quote may be valid, but a mismatched or faulty response can make a trade look more profitable than it is.

2.14.1 Code location - audit baseline

src/cost.ts:258-265 - the response's mints and amounts are returned as given, never compared with the request

```
// src/cost.ts:258-265

  const body = (await res.json()) as Record<string, unknown>;
  return {
    inputMint: String(body.inputMint),
    outputMint: String(body.outputMint),
    inAmountRaw: BigInt(String(body.inAmount)),
    outAmountRaw: BigInt(String(body.outAmount)),
    priceImpactPct: Number(body.priceImpactPct ?? 0),
    slippageBps: Number(body.slippageBps ?? 0),
```

2.14.2 Impact

The bot can underestimate costs, increase trade size, or accept a weaker break-even threshold. The effect depends on the returned quote and the later execution checks.

2.14.3 Recommendation

Verify the returned input mint, output mint, input amount, and swap mode against the request. Do not allow an unverified favorable quote to offset other costs; cap such credits or require independent support for them.

Remediation

SOLVED: Client solved this issue by matching quote fields to the request and preventing negative swap-cost credits.

Remediation source: src/cost.ts:274; src/cost.ts:948.

2.15 Unrelated transactions can make escrow measurement expensive

LOW Remediation status: SOLVED

The measurement routine lists transactions mentioning an escrow and fetches successful entries before checking whether they contain a relevant vault instruction. It does not impose a total work limit.

An unrelated party can add transactions that mention the escrow without contributing to its deposit or withdrawal result. The bot spends RPC requests checking those entries, even though they add nothing to the totals. The accounting remains correct, but the work needed to reach that result grows.

2.15.1 Code location - audit baseline

`src/measure.ts:250-262` - every successful transaction naming the escrow is fetched before it is filtered

```
// src/measure.ts:250-262

  if (page[i]!.err) {
    pass.before = page[i]!.signature;
    i++;
    continue;
  }
  const room = Math.min(BATCH_SIZE, this.readsPerCall - reads);
  if (room <= 0) return null;
  const batch: string[] = [];
  let j = i;
  for (; j < page.length && batch.length < room; j++) {
    if (!page[j]!.err) batch.push(page[j]!.signature);
  }
  reads += batch.length;
```

2.15.2 Impact

A sufficiently large volume of unrelated transactions can delay settlement measurement, keep the vault claimed, and consume the bot's shared RPC budget. The sender bears transaction costs; no incorrect balance total is established by this issue alone.

2.15.3 Recommendation

Limit history depth and total work per measurement. Where possible, filter against relevant bot-account activity before fetching full transactions. Alert an operator and retain the pending state when the limit is exceeded.

Remediation

SOLVED: Client added a saved read counter and an alert when escrow measurement requires excessive transaction reads.

Remediation source: src/measure.ts:21; src/measure.ts:327; src/submitter.ts:165.

2.16 A power failure can restore an older loss-control state

LOW Remediation status: SOLVED

Important state is saved to a temporary file and renamed over the previous file. This avoids exposing a partially written replacement during an ordinary process failure, but the writes are not explicitly flushed to durable storage.

A power loss or operating-system failure shortly after a write can leave an older valid file in place. Corrupt files generally stop trading, but a valid older file can be accepted without detecting that recent state was lost.

2.16.1 Code location - audit baseline

`src/outcome.ts:191-199` - temp-write and rename, with no fsync of the file or the directory

```
// src/outcome.ts:191-199
  private write(next: LossState): void {
    try {
      mkdirSync(dirname(this.opts.path), { recursive: true });
      const tmp = `${this.opts.path}.${process.pid}.tmp`;
      writeFileSync(tmp, JSON.stringify(next), "utf8");
      renameSync(tmp, this.opts.path);
    } catch (err) {
      this.fail("write", err);
```

2.16.2 Impact

The bot may restart without a recently recorded trading halt, spend entry, or transaction signature. That can weaken loss controls or recovery even though the restored file is syntactically valid.

2.16.3 Recommendation

Flush the temporary file before renaming it, then flush the containing directory. Apply durable writes to the loss state, spending ledger, executor state, and transaction journal. Test recovery from interrupted writes on the supported operating systems.

Remediation

SOLVED: Client solved this issue by flushing critical state files and their directory after atomic replacement.

Remediation source: src/utils/durable.ts:7; src/executor.ts:376; src/outcome.ts:194; src/journal.ts:158.

2.17 RPC keys embedded in URL paths can appear in logs

LOW Remediation status: SOLVED

The log redactor removes full URLs and registered secrets. When a URL is registered, it records the full path but not each individual path segment.

Some RPC providers place an API key within the URL path. If an error message contains only that key, without the surrounding path, the registered value does not match and the key can remain in the log or alert.

2.17.1 Code location - audit baseline

`src/observability.ts:124-140` - a URL contributes its whole path, never its individual segments

```
// src/observability.ts:124-140

  function urlSecretParts(u: URL): string[] {
    const path = u.pathname.replace(/^\//, "");
    const query = u.search.replace(/^\?/, "");
    const rawValues = query
      .split("&")
      .filter((kv) => kv.includes("="))
      .map((kv) => kv.slice(kv.indexOf("=") + 1));
    return [
      u.username,
      u.password,
      path,
      query,
      `${path}${u.search}`,
      ...rawValues,
      ...[...u.searchParams.values()],
    ];
  }
```

2.17.2 Impact

Anyone with access to the affected logs or alert destination may receive an RPC credential. The impact depends on that credential's permissions and the accessibility of those outputs.

2.17.3 Recommendation

Register credential-bearing path segments as secrets, including their encoded forms, while retaining full-URL redaction. Add tests for bare path keys in error text, logs, and alerts.

Remediation

SOLVED: Client solved this issue by registering individual URL path segments for log redaction.

Remediation source: src/observability.ts:136; src/observability.ts:147.

2.18 Environment names are misleading and live startup omits the selected target

LOW Remediation status: SOLVED

ENV=development selects staging, while ENV=staging selects the production program and API configuration. Although documented, this mapping is easy to misread.

The live runner's startup output does not show the selected environment, program ID, or API host. An operator can therefore select the production configuration while intending to use staging without a clear startup warning. Other arming checks still apply, so the environment value alone does not start trading.

2.18.1 Code location - audit baseline

src/config.ts:19-27 - ENV=staging selects production

```
// src/config.ts:19-27

  const ENV_TO_ENVIRONMENT: Record<string, Environment> = {
    development: "staging",
    staging: "production",
  };

  export const BASE_URLS: Record<Environment, string> = {
    staging: "https://dev-api.diversifi.trade",
    production: "https://staging-api.diversifi.trade",
  };
```

src/live.ts:1073-1076 - the signing runner's boot record carries executor settings only

```
// src/live.ts:1073-1076

  console.log("peg-keeper LIVE - signs and sends when armed");
  for (const [k, v] of Object.entries(describeExecutor(execCfg, sender.endpoint)))
    console.log(`  ${k}: ${v}`);
  console.log(`  bot: ${signer.publicKey.toBase58()}`);
```

2.18.2 Impact

An operator who satisfies the remaining arming requirements can trade against the wrong intended program or backend. The logs also omit information needed to establish which target the run selected.

2.18.3 Recommendation

Show the environment, program ID, API host, and independently checked network identity at live startup, without exposing credentials. Require explicit production confirmation. Rename environment values to match the configurations they select.

Remediation

SOLVED: Client solved this issue by showing the selected environment, program ID, network and endpoint hosts at startup.

Remediation source: src/config.ts:353; src/live.ts:1176.

2.19 Configuration can silently disable two safety checks

LOW Remediation status: SOLVED

The SOL reserve setting accepts zero. That removes the balance held back for transaction fees and recovery, while producing no warning that the protection is disabled.

The outcome-tolerance setting also accepts 10,000 basis points, or 100%. At that value, receiving nothing can still fall within the allowed shortfall, making that outcome check ineffective. The separate daily loss limit remains in place.

2.19.1 Code location - audit baseline

src/live.ts:211-217 - the reserve has an upper bound only, so 0 is accepted

```
// src/live.ts:211-217

    wholeNumber(
      env,
      "PEG_KEEPER_SOL_RESERVE_LAMPORTS",
      Number(DEFAULT_SOL_RESERVE_LAMPORTS),
      (n) => n <= 100_000_000_000
    )
  ),
```

src/live.ts:199-204 - the tolerance may be 10,000 bps, at which the check cannot breach

```
// src/live.ts:199-204

    env,
    "PEG_KEEPER_OUTCOME_TOLERANCE_BPS",
    150,
    (n) => positive(n) && n <= 10_000
  ),
  solMarkMaxAgeMs:
```

2.19.2 Impact

An unsafe configuration can leave too little SOL to process a refund or can prevent an outcome check from stopping a poor trade. These are operator-selected settings, not changes an arbitrary remote caller can make.

2.19.3 Recommendation

Require a positive operating reserve and a conservative outcome-tolerance limit. Reject values that make a control ineffective, or require an explicit override with a prominent warning that explains the disabled protection.

Remediation

SOLVED: Client solved this issue by requiring a positive SOL reserve and limiting outcome tolerance to 1,000 basis points.

Remediation source: src/live.ts:168; src/live.ts:227.

2.20 Top-up swaps bypass some trading-stop checks

LOW Remediation status: SOLVED

The top-up flow checks the execute flag and whether the executor is idle. Its idle check includes the kill switch, ledger faults, and open trades, but not the separately wired loss breaker.

It also does not apply the normal trade check that requires explicit confirmation for a nonlocal submission endpoint. A top-up swap can therefore be signed when ordinary trades are refused because the loss breaker has halted them or the remote endpoint has not been confirmed. The kill switch is still respected.

2.20.1 Code location - audit baseline

`src/live.ts:956-959` - `idle()` covers the kill switch and ledger faults, not the loss breaker

```
// src/live.ts:956-959

  idle: () =>
    executor!.haltReason() === null &&
    executor!.inFlightVaults().length === 0 &&
    journal.open().length === 0,
```

`src/live.ts:980-981` - `armed()` is the execute flag alone

```
// src/live.ts:980-981

  idle,
  armed: () => execCfg.enabled,
```

`src/executor.ts:768-773` - trades get the remote-endpoint check per trade; top-ups never do

```
// src/executor.ts:768-773

  const endpoint = endpointRefusal(
    cfg.rpcUrl,
    this.submitter.rpcEndpoint,
    cfg.remoteConfirmPhrase
  );
  if (endpoint) out.push(endpoint);
```

2.20.2 Impact

Value-moving top-up swaps can continue in states where the operator expects trading to be stopped. The configured top-up limits still apply.

2.20.3 Recommendation

Use one shared authorization check for all value-moving transactions. Top-ups should respect the loss breaker, remote-endpoint confirmation, kill switch, and relevant state faults before signing.

Remediation

SOLVED: Client solved this issue by applying the executor safety gates to top-ups, including loss and remote-endpoint checks.

Remediation source: src/live.ts:547; src/executor.ts:893; src/topup.ts:598.

2.21 A local proxy can bypass the remote-trading confirmation

LOW Remediation status: SOLVED

The executor decides whether an endpoint is local from its hostname. It does not require the remote-trading confirmation phrase for a loopback address.

A local proxy or tunnel can forward that address to a live network. With the KMS signer, the loopback name can therefore bypass the confirmation even though the submitted trades are real. The separate mainnet check applies only to the development signer. Local-fork tools use generated test keys, so their similar hostname check is not treated as a production-key exposure.

2.21.1 Code location - audit baseline

src/executor.ts:372-379 - "remote" is decided by hostname

```
// src/executor.ts:372-379

  if (!(cfgHost === wireHost || (wireLoopback && isLoopbackHost(cfgHost)))) {
    return {
      code: "endpoint_mismatch",
      message: `the submitter sends to ${wireHost}, not the configured host ${cfgHost}`,
    };
  }
  if (!wireLoopback && phrase !== REMOTE_CONFIRM_PHRASE) return loopbackVerdict(wire);
  return null;
```

src/live.ts:811 - the chain-identity check exists, but runs only for the development signer

```
// src/live.ts:811

  mainnet: live.devSigner ? await servesMainnet([cfg.rpcUrl, live.sendUrl]) : true,
```

2.21.2 Impact

The operator's deliberate confirmation of live trading can be skipped when a live network is reached through a local proxy. This requires that deployment arrangement and the other arming conditions.

2.21.3 Recommendation

Require production confirmation independently of the endpoint hostname. Check network identity for every signer and document how local forks are distinguished; a loopback address alone is not sufficient evidence that transactions are harmless.

Remediation

SOLVED: Client solved this issue by requiring confirmation for mainnet connections, including loopback proxies.

Remediation source: src/live.ts:876; src/executor.ts:429.

2.22 The development signer applies different checks from the KMS signer

INFORMATIONAL Remediation status: SOLVED

The signer interface expects the bot to be the fee payer and its signature to occupy slot zero. KmsSigner enforces that rule. LocalKeypairSigner delegates to the library's signing method without making the same fee-payer check.

Current call sites use a single-signer message with the bot as fee payer, and the development signer is refused on mainnet. No current unauthorized signing path is established. The difference could make development tests stop reflecting production behavior if new message forms are added.

2.22.1 Code location - audit baseline

src/signer.ts:39-44 - the development signer delegates to web3.js and does not write slot 0 itself

```
// src/signer.ts:39-44
  }

  async sign(tx: VersionedTransaction): Promise<VersionedTransaction> {
    tx.sign([this.keypair]);
    return tx;
  }
```

2.22.2 Impact

No current direct security impact was established. The two signer implementations enforce different assumptions.

2.22.3 Recommendation

Apply the same fee-payer and signature-position checks in both implementations, preferably through shared validation.

Remediation

SOLVED: Client solved this issue by applying the same fee-payer and sole-signer checks to both signers.

Remediation source: src/signer.ts:7; src/kms_signer.ts:139.

2.23 Public-key checksum validation is optional during KMS setup

INFORMATIONAL Remediation status: SOLVED

The documentation says KMS requests and responses carry integrity checksums. Signature responses require the checksum, but public-key setup checks pemCrc32c only when the field is present.

Other checks remain effective: the parsed public key must equal the configured bot key, and signatures are verified against that key. The missing checksum requirement does not establish a key-substitution path.

2.23.1 Code location - audit baseline

src/kms_signer.ts:120-124 - the public-key checksum is checked only when present

```
// src/kms_signer.ts:120-124

  }
  if (res.pemCrc32c != null && !checksumMatches(res.pemCrc32c, res.pem)) {
    throw new KmsSignerError("KMS public key failed its checksum");
  }
  const { key, publicKey } = ed25519KeyFromPem(res.pem);
```

2.23.2 Impact

No direct security impact was established. The implementation is less strict than the documented checksum policy.

2.23.3 Recommendation

Require the public-key checksum when the API contract supplies it, or describe the narrower behavior accurately in the documentation.

Remediation

SOLVED: Client solved this issue by requiring and checking the KMS public-key checksum.

Remediation source: src/kms_signer.ts:122.

2.24 Refund tracking trusts the submission endpoint's returned signature

INFORMATIONAL Remediation status: SOLVED

Swap and initialization paths derive a signature from the signed transaction and save it before sending. Refund recovery instead records the signature string returned by the submission endpoint after sending.

An incorrect response can make recovery track a nonexistent signature and later retry the refund. A retry fails after the escrow is closed, but can still waste fees. The endpoint could already withhold submission, which limits the additional security impact.

2.24.1 Code location - audit baseline

`src/sender.ts:86-93` - the refund path records the endpoint's reply, not `signatureOf(signed)`

```
// src/sender.ts:86-93

  export async function signAndSend(
    tx: VersionedTransaction,
    signer: Signer,
    sender: TxSender
  ): Promise<string> {
    const signed = await signer.sign(tx);
    return sender.send(signed.serialize());
  }
```

2.24.2 Impact

Recovery may wait on the wrong signature or send unnecessary refund attempts. No successful duplicate refund is established.

2.24.3 Recommendation

Derive the signature from the signed transaction and persist it before submission, using the same approach as the other transaction paths.

Remediation

SOLVED: Client solved this issue by recording the locally derived refund signature before sending the transaction.

Remediation source: src/sender.ts:96; src/refund.ts:51; src/recovery.ts:342.

2.25 The saved transaction group does not separate escrow histories

INFORMATIONAL Remediation status: SOLVED

Recovery reads a transactionGroupId from a recent memo mentioning the escrow and saves it. Any caller can write such a memo, but no accounting decision currently uses the saved field.

The field's comment suggests that it separates successive flows on a reused escrow. That separation is not implemented. Incomplete history boundaries can cause the accounting errors described in Section 2.7, Incomplete transaction history can mix results from different flows. This finding concerns the unused field and its misleading documentation.

2.25.1 Code location - audit baseline

src/flow.ts:309-321 - the group id is taken from any memo naming the escrow; nothing reads it

```
// src/flow.ts:309-321

  try {
    signatures = await anyConn.getSignaturesForAddress(pda, { limit: 30 });
  } catch {
    return null;
  }

  const tag = kind === "deposit" ? "DEPOSIT:" : "WITHDRAW:";
  for (const sig of signatures) {
    if (sig.err) continue;
    const memo = (sig.memo ?? "").replace(/^\[\d+\]\s*/, "").trim();
    if (memo.startsWith(tag)) return memo.slice(tag.length);
  }
  return null;
```

2.25.2 Impact

No direct effect from an attacker-written group value was established because the value is not used for decisions.

2.25.3 Recommendation

Remove the unused field and correct the comment, or implement flow separation using authenticated records created by the bot rather than arbitrary memos.

Remediation

SOLVED: Client solved this issue by removing the unused group ID and memo-based recovery lookup.

Remediation source: src/flow.ts:489.

2.26 Malformed backend responses are not handled consistently

INFORMATIONAL Remediation status: SOLVED

Verification and decode errors stop trading through the loss breaker. Some other malformed build responses, including an unexpected transaction count, use an ordinary error instead. The response container is also not explicitly type-checked.

These cases end the current trade without the same escalation. The reviewed paths do not pass malformed transactions to the signer, so this is a consistency and monitoring issue rather than a signing bypass.

2.26.1 Code location - audit baseline

`src/submitter.ts:494-501` - a wrong transaction count ends the trade with a plain error, not `TxVerifyError`

```
// src/submitter.ts:494-501

  const res =
    kind === "deposit" ? await this.o.client.openTx(params) : await this.o.client.closeTx(params);
  if (!res.ok) throw new Error(`the ${kind} build was refused (${res.kind}): ${res.message}`);
  if (res.data.unsignedTxs.length !== 1)
    throw new Error(
      `the ${kind} build returned ${res.data.unsignedTxs.length} transactions, not 1`
    );
  const tx = decodeBackendTransaction(res.data.unsignedTxs[0]!);
```

2.26.2 Impact

A faulty backend may repeatedly fail trades without producing the halt described in the documentation.

2.26.3 Recommendation

Validate the response structure and use the intended verification-error handling for malformed builds. Document which backend failures stop one trade and which halt trading.

Remediation

SOLVED: Client solved this issue by treating malformed accepted transaction builds as verification failures that halt trading.

Remediation source: src/submitter.ts:518; src/submitter.ts:263.

2.27 Future price timestamps are accepted as fresh

INFORMATIONAL Remediation status: SOLVED

A Jupiter slot ahead of the RPC node is assigned an age of zero, without a limit on how far ahead it may be. A future Birdeye timestamp produces a negative age and passes the freshness check.

Small clock or slot differences are expected, but an unbounded allowance can hide a faulty source timestamp. A malicious price provider already controls its reported data, so this does not establish a separate stronger attack.

2.27.1 Code location - audit baseline

`src/nav.ts:476-480` - a Jupiter `blockId` ahead of the node clamps to age zero, with no bound

```
// src/nav.ts:476-480

  export function priceAgeSlots(blockId: number | null, currentSlot: number | null): number |
  null {
    if (blockId == null || currentSlot == null) return null;
    if (!Number.isSafeInteger(blockId) || !Number.isSafeInteger(currentSlot)) return null;
    return Math.max(0, currentSlot - blockId);
  }
```

`src/birdeye.ts:49-52` - a future Birdeye time gives a negative age, which passes

```
// src/birdeye.ts:49-52

  const usd = Number(v?.value);
  const updated = typeof v?.updateUnixTime === "number" ? v.updateUnixTime : Number.NaN;
  // Written as the pass condition so a missing or NaN time drops the price.
  if (Number.isFinite(usd) && usd > 0 && nowSec - updated <= maxAgeSec) out.set(mint, usd);
```

2.27.2 Impact

A badly future-dated price can continue to appear current. The main benefit of stricter checks is detecting source errors.

2.27.3 Recommendation

Allow only a small, documented clock or slot difference. Treat timestamps beyond that allowance as unknown or degraded instead of fresh.

Remediation

SOLVED: Client solved this issue by rejecting price timestamps and slots beyond explicit future allowances.

Remediation source: src/birdeye.ts:15; src/nav.ts:502.

2.28 The bot assumes the protocol fee will remain zero

INFORMATIONAL Remediation status: SOLVED

The cost model uses a zero protocol fee. The reviewed program also sets its protocol fee constant to zero, so the current assumption is correct.

The remaining concern is upgrade coordination: a program change could introduce a fee while the bot continues calculating costs with the old assumption. Account-creation rent is a separate cost, discussed in Section 2.11, The bot repeatedly pays to recreate third-party fee accounts.

2.28.1 Code location - audit baseline

`src/cost.ts:125-134` - the bot models the protocol fee as zero

```
// src/cost.ts:125-134
  solUsdcConversionBps: 1,
  driftBufferBps: 65,
  treasuryFeeBps: 0,
  minSwapLegUsd: 1,
  withdrawalVaultCapBps: 8_000,
  maxWithdrawLegs: 5,
  botTxCount: 2,
  botTxComputeUnits: 200_000,
  priorityFeeMicroLamports: 50_000,
  expectedAtaCreations: 0,
```

`programs/solutiofi-program/src/constant.rs:22` - the program fixes it at zero

```
// programs/solutiofi-program/src/constant.rs:22
  pub const FEE: u16 = 0; // In basis points (1%)
```

2.28.2 Impact

No fee-model mismatch exists in the reviewed source. A future fee change could cause the bot to overestimate expected profit until its configuration is updated.

2.28.3 Recommendation

Make the fee assumption explicit and verify it against the intended deployed program at startup and after upgrades.

Remediation

SOLVED: Client made the protocol-fee assumption explicit and documented the required check during program upgrades.

Remediation source: src/cost.ts:126; src/cost.ts:995.

2.29 The top-up budget resets at midnight rather than over a rolling day

INFORMATIONAL Remediation status: SOLVED

The top-up ledger assigns spending to a UTC calendar day. At midnight, a new full budget becomes available. Spending immediately before and after the boundary can therefore approach twice the nominal daily cap in a short period.

This is consistent with a calendar-day limit, but differs from a limit intended to apply to every 24-hour period.

2.29.1 Code location - audit baseline

src/topup.ts:480-489 - the day is fixed at commit and resets at UTC midnight

```
// src/topup.ts:480-489
  }

  commit(usd: number, now: Date): void {
    const day = now.toISOString().slice(0, 10);
    const spentUsd = this.spentToday(now) + usd;
    mkdirSync(dirname(this.path), { recursive: true });
    const tmp = `${this.path}.${process.pid}.tmp`;
    writeFileSync(tmp, JSON.stringify({ v: 1, day, spentUsd }), "utf8");
    renameSync(tmp, this.path);
  }
```

2.29.2 Impact

With the default settings, up to $300 of top-ups can occur around a UTC day boundary. This is a budget-definition issue, not an unlimited-spending path.

2.29.3 Recommendation

Use a rolling 24-hour budget if that is the intended protection. Otherwise, clearly document the calendar-day behavior.

Remediation

SOLVED: Client solved this issue by replacing the UTC-day reset with persisted rolling 24-hour spending records.

Remediation source: src/topup.ts:540; src/topup.ts:607.

2.30 A generated fork key file uses default filesystem permissions

INFORMATIONAL Remediation status: SOLVED

The local-fork pool creation tool saves a newly generated keypair without an explicit file mode. Its permissions therefore depend on the process umask and may allow other local users to read it.

The tool generates the key for the fork and does not load a funded mainnet key in this path.

2.30.1 Code location - audit baseline

`tools/localfork/create_pool.ts:1022-1026` - the fork key file is written with the process's default mode

```
// tools/localfork/create_pool.ts:1022-1026

  if (!fs.existsSync(args.creatorPath)) {
    fs.mkdirSync(path.dirname(args.creatorPath), { recursive: true });
    const fresh = Keypair.generate();
    fs.writeFileSync(args.creatorPath, JSON.stringify(Array.from(fresh.secretKey)));
    console.log(`generated a new creator keypair at ${args.creatorPath}`);
```

2.30.2 Impact

Another local user may read the fork-only key. No production fund exposure was established.

2.30.3 Recommendation

Create the key file with owner-only permissions, such as mode 0o600 on supported systems.

Remediation

SOLVED: Client solved this issue by creating keypair files with owner-only permissions and refusing overwrites.

Remediation source: tools/localfork/lib/chain.ts:29; tools/localfork/create_pool.ts:1024; tools/localfork/fork.ts:276.

2.31 Local-fork tools print complete RPC URLs

INFORMATIONAL Remediation status: SOLVED

The fork tools print their configured RPC URL without redaction. They first reject non-loopback endpoints, so the printed address is local.

A credential would be exposed only if it were included in that local URL, such as for an authenticated local proxy. No ordinary remote provider URL is accepted by this path.

2.31.1 Code location - audit baseline

`tools/localfork/drive.ts:465-478` - the URL is printed only after `assertLocal` has refused anything that is not loopback

```
// tools/localfork/drive.ts:465-478

  assertLocal(args.rpc);
  const runDir = runDirFor(args.rpc);

  const creds = loadCreds({ credsArg: args.creds });
  const jupApiKey = creds.get("JUP_API_KEY");
  if (!jupApiKey) throw new Error(`JUP_API_KEY is unset (${creds.describe()})`);

  const trader = loadKeypair(args.keypair);
  const conn = new Connection(args.rpc, "confirmed");

  console.log("=".repeat(80));
  console.log("peg-keeper local-fork driver: the bot stays in shadow mode; only this file signs");
  console.log("=".repeat(80));
  console.log(`  rpc:     ${args.rpc}`);
```

2.31.2 Impact

A credential embedded in a loopback URL could appear in terminal output or captured logs. The exposure is limited to that configuration.

2.31.3 Recommendation

Apply the existing URL-redaction helper before printing the endpoint.

Remediation

SOLVED: Client solved this issue by redacting RPC URLs in local-fork startup output.

Remediation source: tools/localfork/check_sizing.ts:174; tools/localfork/create_pool.ts:856; tools/localfork/drive.ts:478.

3 On-chain Program Findings

3.1 Changing constituent order can assign prices to the wrong assets

MEDIUM Remediation status: SOLVED

The index manager can change the order of existing constituent mints through update_weights. The program stores that order without invalidating the existing oracle quote or recording which configuration the quote belongs to. OracleMint stores a mint and its target weight, but no expected feed identity.

Deposit and withdrawal build their vault lists in the current constituent order and then use the oracle feed at the same array position. They verify the feed count and quote age, but do not verify that each price belongs to the corresponding mint. A change in order can therefore associate an otherwise fresh price with a different asset.

The affected accounting requires at least two constituents with different prices and a later ADMIN-processed deposit or withdrawal before the existing quote exceeds the 20-slot freshness limit. Manager authorization alone does not permit these ADMIN-restricted operations. The fixed oracle account and canonical vault checks do not detect a changed price-to-mint association.

3.1.1 Code location - audit baseline

programs/solutiofi-program/src/contexts/update_weights.rs:182-185 - The new constituent order is saved without invalidating the quote.

```
// programs/solutiofi-program/src/contexts/update_weights.rs:182-185

          // Update the index with the new mints array
          let mut mints_array = [OracleMint::default(); 5];
          mints_array[..mints.len()].copy_from_slice(&mints);
          self.index.mints = mints_array;
```

programs/solutiofi-program/src/contexts/deposit.rs:278-289 - Deposit valuation uses the price at the corresponding array position.

```
// programs/solutiofi-program/src/contexts/deposit.rs:278-289

          // Calculate the total TVL by iterating over the ordered mints and vaults
          for (i, (mint, vault)) in ordered_mints.iter().zip(ordered_vaults.iter()).enumerate() {
              // Use the amount vault_amount before the swap to calculate the TVL
              let amount =  if vault.mint == self.output_mint.key() {
                  self.output_vault.amount
              } else {
                  vault.amount
              };

              let vault_value_numerator = (amount as u128)
                  .checked_mul(feeds[i].feed_value() as u128)
                  .ok_or(ProgramError::ArithmeticOverflow)?;
```

programs/solutiofi-program/src/contexts/withdraw.rs:300-311 - Withdrawal valuation uses the same positional price mapping.

```
// programs/solutiofi-program/src/contexts/withdraw.rs:300-311

          // Calculate the total TVL by iterating over the ordered mints and vaults
          for (i, (mint, vault)) in ordered_mints.iter().zip(ordered_vaults.iter()).enumerate() {
              // Use the amount before swap for input vault to match deposit behavior
              let amount = if vault.mint == self.input_mint.key() {
                  input_vault_balance_before
              } else {
                  vault.amount
              };

              let vault_value_numerator = (amount as u128)
                  .checked_mul(feeds[i].feed_value() as u128)
                  .ok_or(ProgramError::ArithmeticOverflow)?;
```

3.1.2 Impact

Incorrect asset prices can distort portfolio valuation and the number of liquidity-provider shares minted or burned. Depending on asset balances and the processed transaction, this can issue too many shares or burn too few, diluting existing holders or allocating an excessive share of portfolio value to a transaction. The size and practical reliability of this effect were not measured.

3.1.3 Recommendation

Store the approved oracle feed identity for each constituent mint and verify that every consumed price matches that identity. Bind each quote to the current constituent configuration using a version or hash, and invalidate quotes whenever the membership or order changes. Require a matching fresh quote before processing deposits or withdrawals. Add regression coverage for reordered constituents, mismatched feed identities with the correct count, and valid quotes for the current configuration.

3.1.4 Severity rationale

Medium. The accounting error can affect user value, but it requires manager access, suitable constituent prices and ADMIN processing within the 20-slot quote window. The source review did not establish how often these conditions arise in production.

Remediation

SOLVED: Client added feed-to-mint checks and restricted quote updates to a pinned signer with a signed, recent slot.

Remediation source: programs/solutiofi-program/src/guards.rs:8; programs/solutiofi-program/src/contexts/update_oracle.rs:38; programs/solutiofi-program/src/contexts/deposit.rs:203; programs/solutiofi-program/src/contexts/withdraw.rs:222.

3.2 Rebalance lacks an independent minimum output

MEDIUM Remediation status: SOLVED

Rebalance depends on ADMIN to sign an externally constructed Jupiter route. If the signed route permits an adverse exchange rate, a compromised route source or unfavorable market execution can consume constituent assets without an independent on-chain value check.

The instruction fixes the Jupiter program and event authority, requires active constituent mints and canonical index vaults, and caps an optional treasury platform fee. These guards prevent arbitrary program or destination substitution. After preparing the accepted route, the program invokes Jupiter with the index PDA's authority and returns without measuring the source-vault decrease or destination-vault increase. It enforces neither a separate maximum input nor a minimum received value.

A disabled rebalance example in the repository reads the full source-vault balance and requests 10,000 basis points (100%) of quote slippage. Its helper also enables dynamic slippage, so this setting alone does not establish the final execution threshold or the policy used in production.

3.2.1 Code location - audit baseline

programs/solutiofi-program/src/contexts/rebalance.rs:13-14 - Only the configured administrator can authorize rebalance.

```
// programs/solutiofi-program/src/contexts/rebalance.rs:13-14

      #[account(mut, address = ADMIN)]
      pub payer: Signer<'info>,
```

programs/solutiofi-program/src/contexts/rebalance.rs:222-230 - The handler returns after Jupiter execution without checking received value.

```
// programs/solutiofi-program/src/contexts/rebalance.rs:222-230

          let swap_ix = Instruction {
              program_id: self.jupiter_program.key(),
              accounts,
              data: swap_data.to_vec(),
          };

          invoke_signed(&swap_ix, &account_infos, &signer_seeds)?;

          Ok(())
```

tests/solutiofi-program.ts:509-519 - The disabled example requests the full vault balance and 10,000-bps quote slippage.

```
// tests/solutiofi-program.ts:509-519

    xit("Rebalance", async () => {
      // Extend Compute Unit Limit
      const setComputeUnitLimitIx = ComputeBudgetProgram.setComputeUnitLimit({
        units: 1_200_000,
      });

      // Swap Ray to Drift
      let amount = (await getAccount(connection, indexToken1Vault)).amount;

      const swapResult = await swap(DefiIndex[0].mint, DefiIndex[1].mint, amount.toString(),
  10_000, false, false, index, program.provider.connection);
      const remainingAccounts = extractRemainingAccountsForSwap(swapResult.swapInstruction).remain
  ingAccounts;
```

3.2.2 Impact

An ADMIN-signed route with sufficiently weak thresholds could exchange an entire constituent vault for materially inadequate output, reducing the portfolio value backing all LP holders. This is conditional financial exposure: the local handler lacks a protective value invariant, but actual loss, production use of weak thresholds, and Jupiter execution behavior were not demonstrated.

3.2.3 Recommendation

Require explicit max_input and min_output limits governed by protocol policy. Record both canonical vault balances before the Jupiter call, reload them afterward, and reject balance changes outside those limits. Derive the minimum acceptable value from an independently validated price source with freshness and deviation checks. Validate supported route fields before execution. Add regression coverage for excessive input, inadequate output, stale pricing, and valid bounded swaps across supported token programs, while preserving the existing account and platform-fee checks.

3.2.4 Severity rationale

Medium. A full constituent vault may be exposed, but ADMIN must sign a route with weak protection. The repository example is disabled and does not establish production signing policy. No adverse execution was reproduced.

Remediation

SOLVED: Client added fresh, mint-matched oracle prices and a 5% value-loss limit checked against actual vault balance changes.

Remediation source: programs/solutiofi-program/src/contexts/rebalance.rs:117; programs/solutiofi-program/src/contexts/rebalance.rs:250; programs/solutiofi-program/src/constant.rs:44.

3.3 User requests do not set minimum settlement amounts

MEDIUM Remediation status: SOLVED

When a user deposits USDC or escrows LP shares for withdrawal, the request records a balance but no user-approved minimum output or expiry. Only ADMIN can submit the later Jupiter settlement transaction. If that transaction uses an insufficient execution threshold, unfavorable execution can settle the request below the user's intended price without a user-specific limit causing rejection.

Deposit shares are calculated from tokens actually received, but settlement requires only a nonzero share amount. Withdrawals calculate the LP burn from the oracle value of the input assets and pay the actual USDC received, after fees. Neither path compares that result with a minimum authorized by the user. Canonical account and Jupiter address checks constrain the accounts used, but do not establish an acceptable execution price.

The test transaction builders in tests/solutioFi.ts request 10,000 basis points (100%) of slippage, despite comments saying 10%. The helper enables dynamicSlippage, so Jupiter may return a stronger execution threshold. These test settings do not establish the threshold used in production or imply an automatic total loss.

3.3.1 Code location - audit baseline

programs/solutiofi-program/src/state.rs:45-61 - Deposit and withdrawal requests store balances but no minimum output or deadline.

```
// programs/solutiofi-program/src/state.rs:45-61

  #[account(discriminator = [2])]
  #[derive(InitSpace)]
  pub struct UserDeposit {
      pub index: Pubkey,
      pub user: Pubkey,
      pub usdc_balance: u64,
      pub bump: [u8; 1]
  }

  #[account(discriminator = [3])]
  #[derive(InitSpace)]
  pub struct UserWithdrawal {
      pub index: Pubkey,
      pub user: Pubkey,
      pub lp_balance: u64,
      pub bump: [u8; 1]
  }
```

programs/solutiofi-program/src/contexts/deposit.rs:354 - The deposit requires nonzero shares, not a user-approved minimum.

```
// programs/solutiofi-program/src/contexts/deposit.rs:354

          require_neq!(shares_to_mint, 0, ErrorCode::InvalidAmount);
```

programs/solutiofi-program/src/contexts/withdraw.rs:415-417 - The withdrawal uses the actual received USDC without a user-specific minimum.

```
// programs/solutiofi-program/src/contexts/withdraw.rs:415-417

          let old_usdc_amount = self.output_vault.amount;
          self.output_vault.reload()?;
          let actual_usdc_amount = self.output_vault.amount.checked_sub(old_usdc_amount).ok_or(Pro
  gramError::ArithmeticOverflow)? as u128;
```

tests/solutioFi.ts:283-292 - The builder requests 10,000 basis points, despite the 10% comment.

```
// tests/solutioFi.ts:283-292

              swapResult = await swap(
                USDC_MINT,
                allocation.mint,
                allocation.usdcAmount,
                10000, // 10% slippage
                false, // exact in
                false, // only direct routes
                indexAddress,
                connection
              );
```

3.3.2 Impact

A poorly bounded ADMIN-signed settlement can give a depositor substantially fewer shares, or a withdrawing user substantially less net USDC, than intended. Significant loss is possible for the affected operation, but its size depends on the executed route and market conditions. The source does not establish a permissionless settlement path, observed fund loss, or inevitable total loss.

3.3.3 Recommendation

Record user-signed minimum shares, minimum net USDC, and an execution deadline in each request. Enforce the applicable limits during settlement so a failure reverts the swap and associated mint or burn. Define how limits apply across partial settlements. Apply conservative slippage policy in the builders and validate the returned route threshold. Add regression coverage for below-minimum and expired requests, partial settlements, and valid routes that satisfy the user's bounds.

3.3.4 Severity rationale

Medium. The missing user limits permit materially unfavorable settlement, but ADMIN must submit the transaction. The test builders do not establish the production route policy or Jupiter's effective execution threshold, so they do not establish the likelihood or amount of loss.

Remediation

SOLVED: Client added a 5% oracle-based value-loss limit to each deposit and withdrawal settlement leg.

Remediation source: programs/solutiofi-program/src/contexts/deposit.rs:355; programs/solutiofi-program/src/contexts/withdraw.rs:422; programs/solutiofi-program/src/constant.rs:40.

3.4 A reusable test signing key is stored in the repository

LOW Remediation status: SOLVED

The audit baseline contains a reusable Solana signing key in tests/admin.json. The test suite loads it with Keypair.fromSecretKey, assigns its public key to admin, and passes it to transaction signing. Anyone with access to that file can sign as the same identity; private signing material should not be distributed with source.

Offline public-key derivation matched the key to the legacy test ADMIN_ADDRESS and TREASURY_ADDRESS in tests/solutioFi.ts. It does not match the ADMIN constant in the audited program source. The key was not used to sign or submit a transaction during the review. No active production authority or funds were established for this identity.

3.4.1 Code location - audit baseline

tests/solutiofi-program.ts:75 - The test loads the saved private key. The private key bytes are not reproduced.

```
// tests/solutiofi-program.ts:75

    const adminKeypair = Keypair.fromSecretKey(new
  Uint8Array(JSON.parse(fs.readFileSync(path.join(__dirname, "admin.json"), 'utf-8'))));
```

programs/solutiofi-program/src/constant.rs:4 - The current program administrator is a different public key.

```
// programs/solutiofi-program/src/constant.rs:4

  pub const ADMIN: Pubkey = pubkey!("H1TRD2XtpMy5RJurDNtboDAUuvfzUCUZn8d82oBW3FZd");
```

3.4.2 Impact

Anyone who obtains the fixture can impersonate its signing identity wherever it remains authorized or funded. No such live use was established, and the key cannot satisfy the current source's ADMIN identity check. Low severity reflects confirmed credential disclosure with unproven production privileges or asset exposure.

3.4.3 Recommendation

Treat the key as exposed and rotate or revoke it wherever it remains trusted. Remove it from the repository and distributed artifacts; removing it alone cannot invalidate copies already obtained. Generate disposable test keypairs at runtime, or use ignored local fixtures that hold no live authority or funds. Add secret scanning for Solana keypair arrays, run the test suite with replacement fixtures, and verify that the exposed identity retains no active role or funds.

Remediation

SOLVED: Client removed the bundled signing key, added ignore rules and switched tests to a local key file or a generated keypair.

Remediation source: .gitignore:15; tests/solutiofi-program.ts:76.

4 Reward Sweep Review

Source review found no confirmed authorization or destination bypass in sweep_reward. The handler is identical in the audit and remediation baselines.

The instruction requires the configured ADMIN signer, the index program-derived address and canonical associated token accounts for the reward mint and selected token program. The destination is fixed to REWARD_SINK.

Before transferring, the function rejects active constituent mints, USDC and the index share mint. It also rejects an empty source balance. It transfers the reward-vault balance using the index signing authority. Removing an active constituent requires its vault balance to be zero, preventing a funded constituent vault from becoming eligible for sweeping through a membership update.

Source locations

`programs/solutiofi-program/src/contexts/sweep_reward.rs:12-93` and `programs/solutiofi-program/src/contexts/update_weights.rs:66-185`.

Token compatibility and operations

After rewards reach REWARD_SINK, custody, conversion and reinvestment depend on the sink operator rather than the sweep instruction. Sink ownership and key management should match the intended operating model.

Token-2022 transfer fees can reduce the destination receipt. The sweep event records the amount leaving the source, not a guaranteed net receipt. Reward mints requiring transfer-hook accounts may be incompatible with the current account list. These are token-compatibility considerations, not confirmed fund-loss findings.

The reviewed test suite has no sweep-specific execution test. Recommended coverage includes unauthorized callers, an incorrect destination, protected mints, noncanonical accounts, empty balances and each supported token extension.

5 Additional Controls Reviewed

The following controls were present in the reviewed source. They are described here to explain the budget and authorization behavior, rather than as security findings.

Daily budget release for unsigned attempts

The executor returns reserved daily budget and the rate-limit slot when it receives SubmitNotStartedError, which identifies an attempt that did not reach submission. Other error types retain the reservation. The audit baseline contains the control shown below, and the remediation baseline retains this protection.

src/executor.ts:679-691

```
// src/executor.ts:679-691

  } catch (err) {
    const message = scrubUrls(err instanceof Error ? err.message : String(err));
    if (err instanceof SubmitNotStartedError) {
      // Nothing was recorded or sent, so it spent nothing and is not a submission.
      const slot = this.submissionTimes.lastIndexOf(clockNow.getTime());
      if (slot >= 0) this.submissionTimes.splice(slot, 1);
      this.returnSpend(notionalUsd, clockNow);
      this.releaseVault(basketAddress);
      this.emit({
        level: "warn",
        cat: "operation",
        component: "executor",
        evt: "execution.not_started",
```

Manager authorization for weight updates

The program requires the signer to be the index manager or ADMIN before updating weights. This runtime constraint is enforced in the program source even though it is not fully represented in the interface definition (IDL).

programs/solutiofi-program/src/contexts/update_weights.rs:9-13

```
// programs/solutiofi-program/src/contexts/update_weights.rs:9-13

  #[account(
      mut,
      constraint = manager.key() == index.manager || manager.key() == crate::constant::ADMIN @
  crate::ErrorCode::UnauthorizedAccess
  )]
  pub manager: Signer<'info>,
```

6 Verification Methods and Results

Arb bot remediation - 10 October 2026

All 31 bot findings were reviewed against the affected code, callers and controls. TypeScript checking passed, together with 78 focused unit checks across nine test files covering pricing, configuration, durable state, endpoint checks, Jupiter routes, signing and transaction submission. The checks used local fixtures or mocked services.

Program remediation - 11 October 2026

Program remediations were verified by source review against the original findings. The review compared the audit and remediation baselines, covering the four program findings, guard functions, quote storage, deposit and withdrawal entrypoints, rebalance callers and key handling. The sweep handler and request-state layouts are unchanged between those baselines.

The assertions in 12 guard tests were inspected as source code. Four independent arithmetic checks covered the 95% acceptance boundary, rejection below that boundary, better-than-minimum output and decimal scaling.

Compatibility and related controls

The Switchboard 0.9.3 dependency source was checked against its package hash before comparing quote parsing and storage. The oracle format and producer-integration requirements are described in Operational Considerations (Section 1.7).

The minimum-deposit instruction requires the configured ADMIN signer, the index program-derived address and a positive amount. Updated index-share price events use post-settlement value. These related changes were also inspected during remediation review.

Thank you for choosing us!