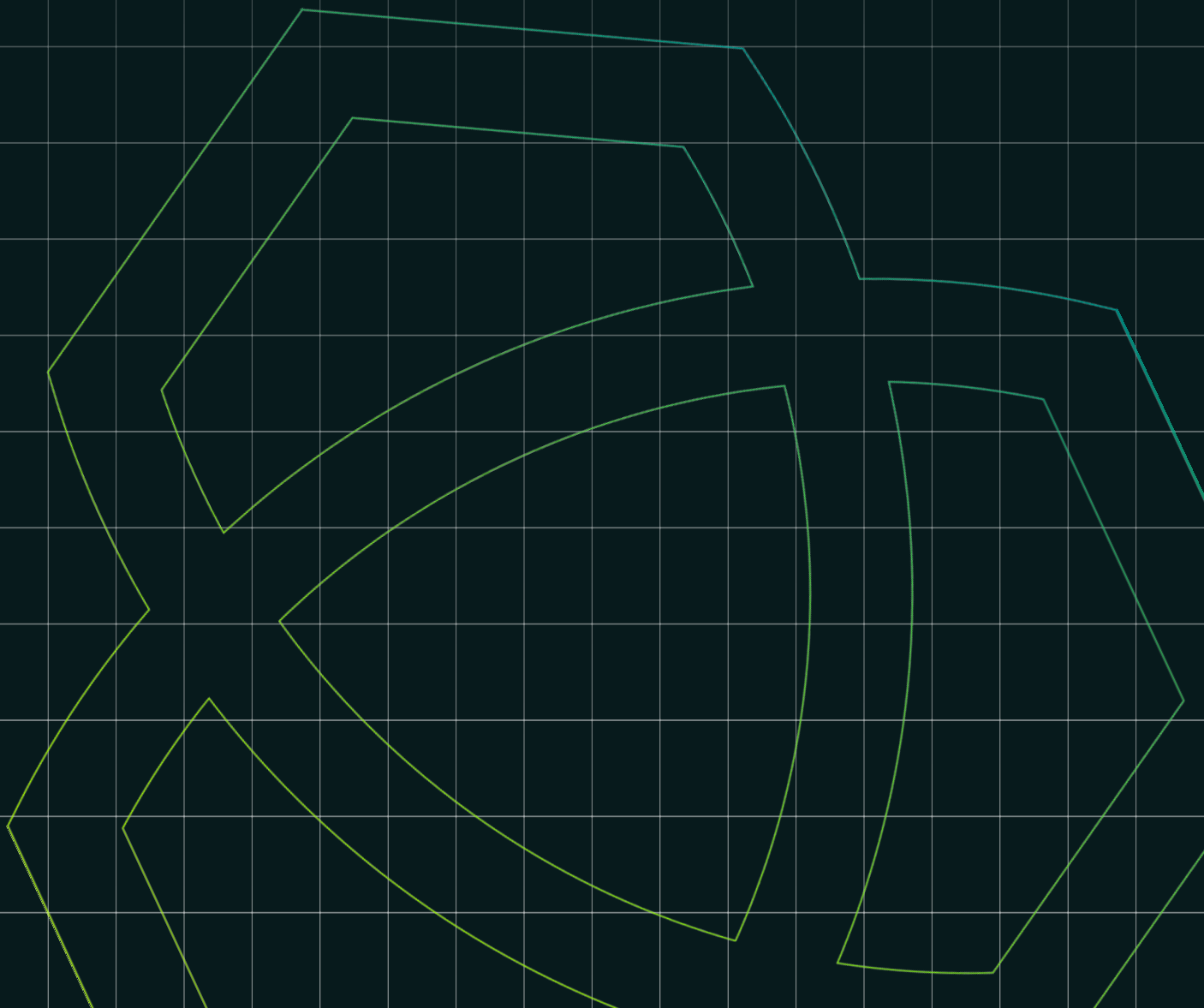
FAILSAFE

10th October 2025

# Valiant - Vortex Audit

## Solana Audit Report



# Table of Contents

## Executive Summary

Vortex is Valiant's concentrated-liquidity market maker, designed around sessionized authorization via Fogo, a streamlined governance surface (feature flags and token-badge attributes removed), and several product-driven behaviors in swap/liquidity flows. Most of the implementation aligns with the published Vortex specification and documentation; however, we identified 1 High, 6 Medium, 8 Low, and 3 Informational issues that either: deviate from the intended protocol and business invariants, or create an under-specified trust on the external session layer.

The most critical risks are: Direction-blind transfer-fee accounting in swap_v2 (High): using fixed (mint_a, mint_b) for fee calculations after removing the direction guard can mischarge/cap fees on Token-2022 fee mints.

# Project Details

| | |
|---|---|
| **Project** | Valiant (Vortex) |
| **Website** | https://valiant.trade |
| **Documentations** | https://docs.valiant.trade |
| **Source Code** | https://github.com/valiant-trade/vortex |
| **Blockchain** | Fogo (SVM Compatible) |
| **Initial Commit** | a31fc4cb7249e9ebed268df4de5d3176f3a70e15 |
| **Final Commit** | cf57b03e38d89bba05b95fdf01944b9e97f4ca42 |
| **Timeline** | Initial Report - 8th September 2025 - 16th September 2025<br>Final Report - 10th October |

## Structure & Organization of Audit Report

Issues are tagged as "Open", "Acknowledged", "Partially Resolved", "Resolved" or "Closed" depending on whether they have been fixed or addressed.

- Open: The issue has been reported and is awaiting remediation from developer team.
- Acknowledged: The developer team has reviewed and accepted the issue but has decided not to fix it.
- Partially Resolved: Mitigations have been applied, yet some risks or gaps still remain.
- Resolved: The issue has been fully addressed and no further work is necessary.
- Closed: The issue is deemed no longer pertinent or actionable.

Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

| | |
|---|---|
| **Critical** | The issue affects the Solana Program in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss. |
| **High** | The issue affects the ability of the Solana Program to compile or operate in a significant way. |
| **Medium** | The issue affects the ability of the Solana Program to operate in a way that doesn't significantly hinder its behavior. |
| **Low** | The issue has minimal impact on the Solana Program's ability to operate. |
| **Info** | The issue is informational in nature and does not pose any direct risk to the Solana Program's operation. |

## Audit Approach

The following are areas of concern will be investigated during the audit, along with any similar potential issues:

- Correctness of the implementation;
- Adversarial actions and other attacks on the network;
- Potential misuse and gaming of the Solana Programs;
- Attacks that impacts funds, such as the draining or the manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution of the Solana Programs;
- Vulnerabilities in the Solana Programs code;
- Protection against malicious attacks and other ways to exploit Solana Programs;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

The following schedule will be followed:

- Code review completed and delivery of Initial Audit Report
- Client responds with fixes and/or acknowledgments for all findings
- Security team validates the fixes and/or acknowledgments
- Verification completed
- Delivery of Final Audit Report

## Methodology

In-scope code: programs/vortex/src/** and anchor-generated IDL. Reference baseline: Vortex business and protocol specification, public documentation, and the canonical on-chain IDL and source structure.

Audit Methodology:

- Structural comparison of IDL (instruction names, account metas, types) against the intended Vortex API surface.
- Manual comparison of instruction handlers (instructions/**), math (math/**), managers (manager/**), and core states (state/**) against the specified behavior.
- Fogo session integration density trace (util/sessions, util/shared, util/token), including all call sites (swap/liquidity/close/collect).
- Reason through invariants: ownership, price/tick bounds, fee caps, protocol/creator fees, and session-path equivalence.

Out-of-scope: Fogo SDK internal correctness, external programs/libraries, and Cyclone launchpad full audit (Covered in a separate report).

## System Overview

Vortex implements a concentrated-liquidity core designed by Valiant, while:

- Introducing Fogo session keys (program PDA "program_signer" and CPI to fogo_sessions_sdk::token::*) for SPL token transfer/burn.
- Streamlining governance by removing feature flags, token-badge attribute governance, adaptive fee/oracle code paths, and position range reset from the protocol.

Where traditional CLMM designs rely on a direct wallet signer as owner/delegate, Vortex permits a session signer; session validation extracts the underlying wallet "user pubkey" and compares it to the position NFT owner or delegate.

### Threat Model & Trust Assumptions

Assume Anchor/SPL invariants hold (checked via account constraints & system CPI). Assume fogo_sessions_sdk performs: (a) robust signature and expiry/constraints validation, (b) proper source-

ATA ownership/delegate checks when a session signer is presented to the token program CPI via program_signer. Assume no cross-program privilege escalation (i.e., Fogo never treats a session key as "owner" for an arbitrary SPL account unless it belongs to that session user per policy).

These assumptions are necessary because Vortex's SPL token moves in session paths do not pre-validate ATA ownership in-program (they rely on the Fogo program to reject invalid authorities).

### Protocol Invariants & Security Properties

Minimum bar we expect Vortex to maintain (divergences documented later):

- Ownership: All mutations to a position require either NFT owner or valid delegate==signer with delegated_amount==1.
- Tick/Price bounds: swaps/liquidity ops keep ticks within [-443636, 443636] and use spacing multiples.
- Fee caps: MAX_FEE_RATE=60_000 (hundredths of a basis point), MAX_PROTOCOL_FEE_RATE=2,500 (25%), MAX_CREATOR_FEE_RATE=2,500 (25%).
- Accounting symmetry: direction (a_to_b) must control which mint accrues transfer fees and which vault is charged/credited.
- Session parity: session path must produce identical post-state to non-session path, modulo the token-account close limitation.

### Codebase Inventory & Versioning

- Anchor =0.29.0, spl-token =4.0.1, solana-program =1.17.22.
- fogo-sessions-sdk =0.4.0 (with "token-program" feature).
- By design removed: adaptive_fee, oracle accessors, dynamic tick-array updates in modify-liquidity, feature flags & token-badge attribute governance.

### Accounting & Fee Semantics Overview

- Fees: FEE_RATE_MUL_VALUE=1_000_000 (hundredths of bps); protocol/creator fees are bps of fee, not notional.
- Creator fees: Added by Vortex (accumulated per-side), see programs/vortex/src/state/vortex.rs::update_after_swap.

- Token-2022 transfer-fees: Must account by direction (a_to_b) — the input mint pays input transfer fee, the output mint applies output-side fee if applicable.

### Compatibility Notes for Integrators

- Reward authority model: Vortex uses per-index authorities instead of a single packed authority, which affects how integrators model reward configuration.
- Token-badge governance: Feature flags & attribute toggles are removed; delete_token_badge is a no-op in v2.
- No reset_position_range: Clients must not assume a built-in position range reset operation; any such behavior must be implemented off-protocol or via separate tooling.
- Modify-liquidity tick arrays: Vortex's path doesn't call update_tick_array_accounts; treat arrays as fixed.
- Swap API drift: fee_rate typed as u16 (vs u32 upstream).

**Design Decisions & Compatibility Considerations**

The Vortex implementation makes several explicit governance and execution choices relative to other concentrated-liquidity deployments. Below is a narrative that ties each choice to its likely operational and security consequences.

Feature flags & token-badge governance removed. The Vortex configuration does not include the feature-flag and token-badge gating mechanisms present in some other CLMM deployments; Vortex deletes the feature flag machinery entirely (no feature_flags state, no verify_enabled_feature guard) and ships an empty v2 delete_token_badge handler. Governance that might otherwise be enforced on-chain is instead a social/process convention. This increases policy drift risk: tools or bots calling a "deletion" ix will succeed with no state change, while operators cannot toggle protections at runtime. If this is the intended product policy (lean governance), it should be documented as such; otherwise, stronger on-chain controls should be considered.

Non-transferable position requirement dropped. The protocol no longer exposes a configuration to require non-transferable position NFTs for particular pools; Vortex removes both the attribute and the runtime gate in open-position paths. This expands composability and secondary trading but deliberately weakens a safety rail used by some deployers to constrain position mobility. It must be communicated to integrators who assume non-transferable-position semantics by default.

Reward authority semantics (single -> per-index). Earlier designs often stored a single reward authority; Vortex instead exposes per-reward-index authorities and checks the index in account constraints/handlers. This is a reasonable product choice (finer-grained control), but it raises the bar for consistency: every reward-mutating path must validate the same index-specific authority or a privilege hole opens. It also breaks SDK assumptions that "there is only one reward authority."

Modify-liquidity without dynamic tick-array updates. In other CLMM implementations, a helper like update_tick_array_accounts dynamically resizes/initializes adjacent tick arrays and then re-loads them to apply updates. Vortex skips this and operates directly on provided arrays via loaders. The security impact is subtle: if callers don't pre-create arrays at boundaries, edge-condition accounting can drift from those reference designs. It's a conscious simplification but moves responsibility to clients.

Vault initialization & authority wiring. Some reference implementations initialize vault accounts via a helper that enforces authority wiring and program owner. Vortex initializes SPL vaults directly with authority=vortex. Security is equivalent if the wiring is correct, but devOps scripts expecting a helper abstraction will need to adapt. This should be highlighted for integrators.

Swap interface drift (fee parameter type) and fee logic coupling. Reference AMM implementations commonly accept a u32 fee domain for compute_swap; Vortex narrows this to u16. Separately, Vortex removed

the direction-dependent mint selection in v2 swap fee handling and hard-codes (mint_a, mint_b) for Token-2022 transfer-fee accounting. The latter is a functional correctness gap (covered as a High issue below); the former is an API compatibility change that can surprise tooling that assumes a wider domain.

Two-hop tick-sequence builder API change. Some earlier designs use SparseSwapTickSequence-Builder::new(…) to construct the sequence; Vortex uses ::try_from(vortex, a_to_b, vec![…]). If the new constructor subtly changes search-range shifting at boundaries, actual pathing through ticks can diverge even when the provided arrays are identical. This needs consistency tests; otherwise, deviations at array edges are easy to miss.

Lock-position constraints tightened to full-range only. Other deployments only forbid locking empty positions; Vortex additionally forbids locking any non-full-range position. This is a business constraint rather than a security bug, but downstream protocols that rely on locking narrow ranges will find this incompatible without modification.

Open-position rent prepay removed. Some prior implementations pre-pay rent for tick coverage when opening positions to reduce later rent failures. Vortex removes that call, which shifts rent costs to later operations. Not a vuln, but a UX/operability deviation that can surface as “random rent errors” during liquidity changes.

Oracle/adaptive fee & related accessors removed. Vortex excises adaptive-fee and oracle accessor logic. That’s a conscious simplification with two consequences: (1) fewer moving parts to audit; (2) less dynamic defense against toxic order flow or price manipulation. If pools are expected to behave like plain constant-product concentrated AMMs, this is fine—document the change so integrators don’t expect TWAP-style protections or dynamic fee schedules.

IDL/API surface reshaped. Beyond the above, several IDL items move, disappear, or change constraints (e.g., missing transfer_locked_position; reward-index authority checks in setters; optional program_signer accounts added to many ixs to drive Fogo session paths). The API is coherent but should be treated as an independent protocol surface; SDKs must treat Vortex as an independent protocol with its own API surface, not as a drop-in replacement for any other CLMM runtime.

### In-scope Files

- programs/vortex/src/** (all source files under the vortex program)
- Anchor-generated IDL for Vortex

### Out of Scope

- Fogo SDK internal correctness (fogo_sessions_sdk internals)
- External programs/libraries outside the Vortex program
- Cyclone launchpad full audit (covered in separate report)
- Front-end client code and SDKs

## Summary of Findings

| Severity | Total | Open | Acknowledged | Partially Resolved | Resolved | Closed |
|---|---|---|---|---|---|---|
| Critical | - | - | - | - | - | - |
| High | 1 | - | - | - | 1 | - |
| Medium | 4 | - | 2 | - | 2 | - |
| Low | 8 | - | 8 | - | - | - |
| Info | 5 | - | 3 | 1 | 1 | - |
| Total | 18 | 0 | 13 | 1 | 4 | 0 |

| # | Findings | Severity | Status |
|---|---|---|---|
| 1 | Direction-blind Token-2022 transfer-fee accounting in v2 swap | High | Resolved |
| 2 | Session close burns but does not close token account | Medium | Acknowledged |
| 3 | .unwrap() on token-support checks can panic (initialize pool/reward v2) | Medium | Resolved |
| 4 | Ambiguous selection between session and non-session paths | Medium | Acknowledged |
| 5 | Reliance on Fogo to enforce source-ATA ownership (no local pre-check) | Medium | Resolved |
| 6 | No explicit PDA equality check before invoke_signed (session helpers) | Low | Acknowledged |
| 7 | Non-transferable position requirement removed from open-position paths | Low | Acknowledged |
| 8 | Modify-liquidity omits dynamic tick-array update/resize | Low | Acknowledged |
| 9 | Reward authority changed to per-index — consistency risk | Low | Acknowledged |
| 10 | Narrowed fee_rate domain (u32 -> u16) in compute_swap | Low | Acknowledged |
| 11 | Two-hop tick sequence builder semantic drift | Low | Acknowledged |
| 12 | Open-position rent prepay removed (operational drift) | Low | Acknowledged |
| 13 | transfer_locked_position instruction missing | Low | Acknowledged |
| 14 | Feature flags removed; delete_token_badge v2 is a no-op | Info | Acknowledged |
| 15 | Session error taxonomy is too coarse | Info | Partially Resolved |
| 16 | get_reversed_sqrt_price semantics unclear (added helper) | Info | Resolved |
| 17 | Initialize-tick-array type And seeding drift | Info | Acknowledged |
| 18 | delete_token_badge is present in IDL but functionally inert | Info | Acknowledged |

### Finding 1: Direction-blind Token-2022 transfer-fee accounting in v2 swap

**Severity:** High

**Status:** Resolved

**Source:** programs/vortex/src/instructions/v2/swap.rs — fee calculation section after removing direction-dependent mint selection.

**Description:**

The Vortex specification assumes that `(token_mint_input, token_mint_output)` are selected based on `a_to_b` so that Token-2022 transfer-fees follow the actual swap direction. The current implementation removed that selection and now calls transfer-fee helpers on `token_mint_a` and `token_mint_b` unconditionally. When swapping B->A, fees are still computed as if A->B, producing wrong deltas.

Technical Analysis: Transfer-fee logic is directional: the input pays input-side fees; the output may apply output-side fee exclusions. Hard-coding `(mint_a, mint_b)` breaks the invariant that fee attribution follows `a_to_b`.

**Impact:**

Mis-settlement: users can be over-charged/under-charged, AMM vault balances drift, and min/max constraints can be violated without detection. Any pool with Token-2022 transfer-fee mints is affected on one of the directions.

**Code:**

```
// Reference design (direction-aware fee accounting)
let (token_mint_input, token_mint_output) = if a_to_b {
    (&ctx.accounts.token_mint_a, &ctx.accounts.token_mint_b)
} else {
    (&ctx.accounts.token_mint_b, &ctx.accounts.token_mint_a)
};


// Vortex (directional guard removed)
let input_transfer_fee =
    calculate_transfer_fee_excluded_amount(&ctx.accounts.token_mint_a, input_amount)?.transfer_fee;
let output_transfer_fee =
    calculate_transfer_fee_excluded_amount(&ctx.accounts.token_mint_b, output_amount)?.transfer_fee;
```

**Remediation:**

Restore direction-aware mint selection and pass `(token_mint_input, token_mint_output)` into fee helpers. Add an assertion that for `a_to_b==false`, the “input” mint equals `token_mint_b`. If this behavior is intentionally different, document the business rule and its implications (e.g., only fee-mint on side A is supported) and treat B->A swaps as unsupported or pre-checked.

**Developer Response:**

Fixed by adding direction-aware mint selection so that B->A swaps are handled correctly.

**Auditor Response:**

Fixed at commit cf57b03.

### Finding 2: Session close burns but does not close token account

**Severity:** Medium

**Status:** Acknowledged

**Source:** programs/vortex/src/util/token.rs::burn_and_close_user_position_token (session branch) — Callers -> instructions/close_position.rs (passes optional program_signer)

**Description:**

In session mode, Vortex calls Fogo's burn CPI to destroy the position token but explicitly comments that "Close Account instruction not supported in sessions." The SPL account remains allocated and rent-bearing.

Technical Analysis: `close_account` requires the token account's close authority. The session CPI provides only a burn authority flow; without an additional authority hop or a follow-up ix, closing is impossible on-chain in the same path.

**Impact:**

Rent leakage and account lifecycle ambiguity. Wallets and indexers will observe "zombie" accounts (burned balance, still allocated), confusing users and automation. Attackers can't steal funds here, but permanent rent loss and operational inconsistency are serious.

**Code:**

```
// Vortex session path (util/token.rs)
// ...
// Burn position token via sessions
session_token_burn(...)?;

// NOTE: Close Account instruction not supported in sessions
```

**Remediation:**

Introduce a second ix that, when called in the same transaction, closes the burned account via a PDA close authority. Or forbid session closes unless a non-session signer is present to close. If you intentionally accept "burn without close," document that the wallet must close the account off-chain and surface this in UX.

**Developer Response:**

Intentional design choice. Sessions prioritize UX simplicity over rent reclamation. Power users can use non-session instructions to close accounts and reclaim rent.

**Auditor Response:**

Acknowledged without further refute.

### Finding 3: .unwrap() on token-support checks can panic (initialize pool/reward v2)

**Severity:** Medium

**Status:** Resolved

**Source:** programs/vortex/src/instructions/v2/initialize_pool.rs, programs/vortex/src/instructions/v2/initialize_reward.rs

**Description:**

Both handlers call `is_supported_token_mint(...).unwrap()` and then `require!(ok, UnsupportedTokenMint)`. If the helper returns `None` for unknown Token-2022 attributes or sentinel code paths, the program will panic.

Technical Analysis: `Option<bool>` must be mapped to a program error, not unwrapped. Upstream consistently avoids `.unwrap()` in instruction handlers for precisely this reason.

**Impact:**

Hard abort -> transaction failure with no controlled error code. Attackers can DoS pool initialization with malformed mints; legitimate deployers receive opaque failures.

**Code:**

```
// Vortex
let is_badge = is_token_badge_initialized(...)?;
if !is_supported_token_mint(&ctx.accounts.reward_mint, is_badge).unwrap() {
    return Err(ErrorCode::UnsupportedTokenMint.into());
}
```

**Remediation:**

Replace `.unwrap()` with:

```
let supported = is_supported_token_mint(&mint, is_badge)
    .ok_or(ErrorCode::UnsupportedTokenMint)?;
require!(supported, ErrorCode::UnsupportedTokenMint);
```

If you choose to accept panics as “unexpected platform states,” document that only specific Token-2022 configurations are supported and treat others as undefined behavior.

**Developer Response:**

Fixed by replacing .unwrap() calls with proper error handling using the ? operator and require! macro for cleaner error propagation.

**Auditor Response:**

Fixed at commit cf57b03.

### Finding 4: Ambiguous selection between session and non-session paths

**Severity:** Medium

**Status:** Acknowledged

**Source:** programs/vortex/src/util/token.rs::transfer_from_owner_to_vault, programs/vortex/src/util/sessions/mod.rs::{session_token_transfer, session_token_burn} — Callers across swap/liquidity/close.

**Description:**

Handlers accept `Option<AccountInfo> program_signer`. If Some, Vortex routes to Fogo session CPI; otherwise, it uses standard SPL. There's no explicit guard that the actual signer is a session signer when `program_signer` is Some, or that `program_signer` is None when a normal wallet signer is used.

Technical Analysis: The code should determine session vs non-session via `validate_session(owner_account_info, program_id)` and then enforce the matching `program_signer` condition. Passing the wrong branch should fail early with a clear program error.

**Impact:**

Integration errors cause inconsistent failure modes deep in CPI, complicating debugging. In the worst case, a future code path may assume "session == stronger checks," while callers accidentally use the non-session branch.

**Code:**

```
pub fn transfer_from_owner_to_vault(..., program_signer: Option<&AccountInfo<'info>>) -> Result<()> {
    match program_signer {
        None => token::transfer(...),
        Some(program_signer) => session_token_transfer(..., program_signer),
    }
}
```

**Remediation:**

Add an early guard: if `validate_session` returns a session user, require `program_signer == expected PDA`; else require `program_signer.is_none()`. If you intentionally allow mixed use, document that the caller chooses the path and bears responsibility for correctness.

**Developer Response:**

Documented design choice. The caller chooses the execution path and is responsible for correctness. Session validation is handled at the instruction level before calling this utility function.

**Auditor Response:**

Acknowledged without further refute.

### Finding 5: Reliance on Fogo to enforce source-ATA ownership (no local pre-check)

**Severity:** Medium

**Status:** Resolved

**Source:** programs/vortex/src/util/sessions/mod.rs::session_token_transfer — Callers -> util/token.rs::transfer_from_owner_to_vault and SPL movements under session.

**Description:**

For sessionized transfers, Vortex does not pre-assert `source_token_account.owner == session_user`. It defers entirely to Fogo's token program wrapper to validate that the session signer is authorized for the source ATA.

Technical Analysis: A single equality check on-chain is cheap insurance. It also improves error clarity ("source owner mismatch") vs opaque CPI failures.

**Impact:**

If Fogo policy changes or a bug slips in, Vortex has no defense-in-depth. The trust boundary is acceptable for this specification-focused audit but should be explicit.

**Remediation:**

Add an optional pre-check comparing session user to ATA owner in Vortex (behind a feature flag).

**Developer Response:**

Fixed by adding local pre-check validating source_token_account.owner matches session user before transfer.

**Auditor Response:**

Fixed at commit cf57b03.

### Finding 6: No explicit PDA equality check before invoke_signed (session helpers)

**Severity:** Low

**Status:** Acknowledged

**Source:** programs/vortex/src/util/sessions/mod.rs::{session_token_transfer, session_token_burn}

**Description:**

The helpers compute the PDA with `Pubkey::find_program_address([PROGRAM_SIGNER_SEED], &crate::ID)` and use the bump in `invoke_signed`, but never assert that the passed `program_signer.key()` equals the derived PDA.

Technical Analysis: A simple `require_keys_eq!` gives earlier, clearer errors and prevents accidentally signing with mismatched seeds.

**Impact:**

Passing the wrong `program_signer` produces confusing CPI errors instead of a crisp program-level failure. It's a correctness smell and a hard-to-debug integration foot-gun.

**Code:**

```
let (_address, bump) = Pubkey::find_program_address(&[PROGRAM_SIGNER_SEED], &crate::ID);
invoke_signed(
    &instruction,
    &account_infos,
    &[&[PROGRAM_SIGNER_SEED, &[bump]]],
)?;
```

**Remediation:**

Assert `program_signer.key() == _address` and return `SessionMissingProgramSigner` if not. If you rely on the downstream CPI to fail instead, document that guarantee and keep a comment explaining the trade-off.

**Developer Response:**

The program_signer account is already validated by Anchor constraints in instruction handlers using #[account(seeds = [...], bump)]. No additional validation needed in utility functions.

**Auditor Response:**

We accept the team's rationale that Anchor seed constraints at all call-sites validate the program_signer PDA. Given this is defense-in-depth rather than an exploitable condition, we're downgrading to Low and

closing as Acknowledged, with the requirement that handlers consistently annotate program_signer with #[account(seeds=[…], bump)], a CI check to prevent drift, and explicit helper-level docs on the precondition.

### Finding 7: Non-transferable position requirement removed from open-position paths

**Severity:** Low

**Status:** Acknowledged

**Source:** programs/vortex/src/instructions/open_position*.rs (gate removed) — prior reference design exposed `is_non_transferable_position_required`

**Description:**

The runtime check that enforces non-transferable position NFTs for certain pools is removed. This is likely a deliberate product choice, but it directly affects downstream compliance or farming strategies that assumed non-transferable positions.

Technical Analysis: The badge attribute and control flag are no longer available, so there is no straightforward way to opt back in per pool.

**Impact:**

Positions become tradable where some prior designs intended them to be bound to owners; this alters incentive models and can enable borrow-and-dump patterns.

**Remediation:**

Reintroduce the check (behind a config bit) Otherwise, clearly document that Vortex positions are always transferable and recommend alternative mitigations if needed.

**Developer Response:**

Upstream divergence

**Auditor Response:**

Acknowledged without further refute.

### Finding 8: Modify-liquidity omits dynamic tick-array update/resize

**Severity:** Low

**Status:** Acknowledged

**Source:** Vortex -> instructions/increase_liquidity.rs, decrease_liquidity.rs — some reference implementations use manager::tick_array_manager::update_tick_array_accounts(…).

**Description:**

Vortex loads tick arrays and applies updates without the dynamic update/resize step used in some prior designs. This can diverge at boundaries where new arrays should be brought into scope.

Technical Analysis: Reference design: load -> compute update -> drop -> update arrays -> re-load -> sync. Vortex: load -> compute -> sync, no intermediate update.

The difference matters when arrays need resizing/initialization.

**Impact:**

Edge-case liquidity ops near array edges can fail unexpectedly or under-update accounting versus the behavior assumed in those reference designs.

**Remediation:**

Port the update step or require callers to pre-create arrays (and document that requirement). If you prefer the simpler model, mark it as a conscious deviation and add tests covering boundary ticks.

**Developer Response:**

Upstream divergence

**Auditor Response:**

Acknowledged without further refute.

### Finding 9: Reward authority changed to per-index — consistency risk

**Severity:** Low

**Status:** Acknowledged

**Source:** programs/vortex/src/state/vortex.rs (per-index authority), programs/vortex/src/instructions/v2/set_reward_authority.rs

**Description:**

The model is valid but increases the chance of inconsistent checks across reward-related instructions. If any mutation path validates a different thing (e.g., global authority), an attacker with control over index i could touch index j.

Technical Analysis: Centralize authority verification in a helper and call it from all reward ix. Unit-test that every ix rejects mismatched index authority.

**Impact:**

Privilege escalation over reward configuration for other indices.

**Remediation:**

Introduce `require_reward_authority(index, signer)` and reuse. If the looser model is intentional, document the rationale and require multi-sig off-chain controls.

**Developer Response:**

Upstream divergence

**Auditor Response:**

Acknowledged without further refute.

### Finding 10: Narrowed fee_rate domain (u32 -> u16) in compute_swap

**Severity:** Low

**Status:** Acknowledged

**Source:** programs/vortex/src/math/swap_math.rs::compute_swap

**Description:**

Vortex narrows the parameter type to `u16`. This is an API break relative to earlier designs that used `u32` and can reject otherwise valid values or cause different rounding behavior when composing fees across components expecting `u32`.

Technical Analysis: The internal constant `FEE_RATE_MUL_VALUE=1_000_000` remains; constraining the type reduces headroom and may cause unexpected overflows in callers before they reach the program.

**Impact:**

Integration failures or silent differences in fee math (especially in SDKs that assume a `u32` fee domain).

**Remediation:**

Accept `u32` and range-check to `<= MAX_FEE_RATE`. If `u16` is a deliberate simplification, document the supported range in the IDL and public docs.

**Developer Response:**

Upstream divergence

**Auditor Response:**

Acknowledged without further refute.

### Finding 11: Two-hop tick sequence builder semantic drift

**Severity:** Low

**Status:** Acknowledged

**Source:** programs/vortex/src/instructions/two_hop_swap.rs

**Description:**

Changing the constructor may alter "shifted search range" semantics around array boundaries. Even a one-tick difference in next-tick discovery can change slippage and fee accrual outcomes.

Technical Analysis: Verify that `try_from` implements the same `in_search_range` shift `(!a_to_b)` used upstream. Without parity tests, this is a fragile area.

**Impact:**

Hard-to-reproduce discrepancies in routing across two pools, surfacing as price differences or unexplained min/max failures in edge cases.

**Remediation:**

Add tests covering boundary behaviors for both directions. If the search is intentionally different, document the new semantics.

**Developer Response:**

Upstream divergence

**Auditor Response:**

Acknowledged without further refute.

### Finding 12: Open-position rent prepay removed (operational drift)

**Severity:** Low

**Status:** Acknowledged

**Source:** programs/vortex/src/instructions/open_bundled_position.rs, open_position.rs, open_position_with_metadata.rs — (Removed call to collect_rent_for_ticks_in_position)

**Description:**

Without rent prepay, later liquidity ops may fail due to rent requirements around newly touched ticks, yielding surprising UX outcomes.

Technical Analysis: The prepay routine was a design band-aid upstream; removing it is a valid simplification, but the trade-off is real.

**Impact:**

“It worked yesterday, now fails with rent errors” under churny liquidity providers; increased support load.

**Remediation:**

Either restore prepay or document the new requirement: users must fund rent at modify-liquidity time; SDKs should surface rent predictions.

**Developer Response:**

Upstream divergence

**Auditor Response:**

Acknowledged without further refute.

### Finding 13: transfer_locked_position instruction missing

**Severity:** Low

**Status:** Acknowledged

**Source:** Absent in Vortex (present in some prior CLMM designs)

**Description:**

Integrators relying on transferring locked positions (e.g., in governance flows) cannot do so in Vortex.

Technical Analysis: Not a security vulnerability, but an API parity break that can cascade into ad-hoc workarounds.

**Impact:**

Broken workflows and unexpected “feature not found” errors.

**Remediation:**

Implement if needed, or document the absence and suggest alternatives.

### Finding 14: Feature flags removed; delete_token_badge v2 is a no-op

**Severity:** Info

**Status:** Acknowledged

**Source:** programs/vortex/src/state/vortex_config.rs (no feature flags), programs/vortex/src/instructions/v2/delete_token_badge.rs (empty handler)

**Description:**

Governance that similar concentrated-liquidity deployments often enforce on-chain via feature flags is not present in Vortex. The v2 delete handler returns `Ok(())` without mutating state, which can mislead automation or operators who expect actual deletion.

Technical Analysis: A no-op instruction that succeeds is an API hazard. Either implement deletion semantics or remove the instruction from the IDL to avoid misinterpretation.

**Impact:**

Policy drift and false assurances; badge-dependent behaviors won't change even after a "successful" deletion transaction.

**Code:**

```
// Vortex v2 delete badge
pub fn handler(_ctx: Context<DeleteTokenBadge>) -> Result<()> {
    Ok(())
}
```

**Remediation:**

Implement guarded deletion, or remove the ix from IDL. If keeping as a stub, document clearly that it is a placeholder and should not be used.

**Developer Response:**

The instruction does perform deletion via Anchor's automatic account closure and rent refund. The handler itself is minimal but functional - it closes the token_badge account as intended.

**Auditor Response:**

We confirm the instruction behavior is functionally valid via Anchor's auto-close semantics. On-chain governance flags were intentionally removed; no further action required beyond documenting this policy shift. Downgrading to Informational and closing as Acknowledged.

### Finding 15: Session error taxonomy is too coarse

**Severity:** Info

**Status:** Partially Resolved

**Source:** programs/vortex/src/util/shared.rs::validate_session_and_owner, programs/vortex/src/errors.rs (SessionValidationFailed, SessionMissingProgramSigner)

**Description:**

Distinct conditions (invalid session signature; session user != expected owner; owner not a signer; delegate mismatch; delegated_amount != 1) collapse into a small number of errors. This weakens observability, makes SOC triaging harder, and obscures exploit attempts in telemetry.

Technical Analysis: Split errors: `SessionInvalid`, `SessionUserMismatch`, `OwnerNotSigner`, `DelegateMismatch`, `InvalidDelegatedAmount`. Propagate them from helpers to instruction handlers.

**Impact:**

Harder incident response and poorer UX (users get generic failures with no actionable message).

**Remediation:**

Add granular errors and use them in `require!` statements. If you prefer minimal error surface to avoid leakage, document that decision and provide off-chain logs/metrics to compensate.

**Developer Response:**

Added a couple Session errors for clarity, but the Fogo Sessions UI packages rely on propagating errors from the fogo-sessions-sdk methods.

**Auditor Response:**

Downgrading to Informational and closing with Partially Fixed at commit cf57b03.

### Finding 16: get_reversed_sqrt_price semantics unclear (added helper)

**Severity:** Info

**Status:** Resolved

**Source:** programs/vortex/src/math/tick_math.rs::get_reversed_sqrt_price

**Description:**

The helper computes `u128::MAX / sqrt_price` and checks bounds. The name suggests reciprocal price; in Q64.64 fixed-point, a true reciprocal requires scaling. Misuse later would be a math bug.

Technical Analysis: Keep it internal/private or rename to reflect intent (“bounded_div_max”).

**Impact:**

Future maintainers can misinterpret and introduce price math errors.

**Code:**

```
pub fn get_reversed_sqrt_price(sqrt_price: u128) -> Result<u128, ErrorCode> {
    let quotient = u128::MAX / sqrt_price;
    if quotient > MAX_SQRT_PRICE_X64  || quotient < MIN_SQRT_PRICE_X64 {
        return Err(ErrorCode::SqrtPriceOutOfBounds.into());
    }
    Ok(quotient)
}
```

**Remediation:**

Clarify docstring and add unit tests asserting only bounds behavior. If you intend a reciprocal, fix scaling.

**Developer Response:**

See issue 1 of Cyclone - Launchpad report

**Auditor Response:**

Fixed as per commit 6d84f44.

### Finding 17: Initialize-tick-array type And seeding drift

**Severity:** Info

**Status:** Acknowledged

**Source:** instructions/initialize_tick_array.rs

**Description:**

Type/size names differ. If SDKs hard-code `FixedTickArray`, they will mis-size accounts in Vortex.

Technical Analysis: The structure contents look equivalent; the naming change is the main source of incompatibility.

**Impact:**

Deployment friction and rent mis-calculation.

**Code:**

```
// Vortex:
// instructions/initialize_tick_array.rs
// uses TickArray::LEN and
// seeds [b"tick_array", vortex.key(), start_tick_index.to_string()]

// Reference implementation: same seeds with FixedTickArray::LEN.
```

**Remediation:**

Document the account size and provide a helper to compute it from IDL. If you intend long-term divergence, keep it as is and warn integrators.

**Developer Response:**

Upstream divergence

**Auditor Response:**

Acknowledged without further refute.

### Finding 18: delete_token_badge is present in IDL but functionally inert

**Severity:** Info

**Status:** Acknowledged

**Source:** programs/vortex/src/instructions/v2/delete_token_badge.rs (returns Ok(()))

**Description:**

Duplicative with Finding #7 but captured here as an API smell: IDL exposes an instruction with no effect.

Technical Analysis: Shipping inert instructions is error-prone even if not dangerous per se.

**Impact:**

Confuses automation; breaks expectations in governance scripts.

**Remediation:**

Remove from IDL or implement semantics; otherwise, document as a stub and advise integrators to avoid it.

**Developer Response:**

See response to issue 6

**Auditor Response:**

Acknowledged without further refute.

## Disclaimer

This audit report (“Report”) is provided by FailSafe (“Auditor”) for the exclusive use of the client (“Client”). The audit scope is limited to a technical review of the Solana smart contract program code supplied by the Client. This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without FailSafe’s prior written consent in each instance.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts FailSafe to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. FailSafe’s position is that each company and individual are responsible for their own due diligence and continuous security. FailSafe’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by FailSafe is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND “AS AVAILABLE” AND

WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, FAILSAFE HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, FAILSAFE SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, FAILSAFE MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

WITHOUT LIMITATION TO THE FOREGOING, FAILSAFE PROVIDES NO WARRANTY OR DISCLAIMER UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER FAILSAFE NOR ANY OF FAILSAFE'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. FAILSAFE WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT FAILSAFE'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST FAILSAFE WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF FAILSAFE CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST FAILSAFE WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.