CANTINA

Multiliquid solana swap program

Security Review

Cantina Managed review by:
Sujith s, Lead Security Researcher
Red-swan, Security Researcher

December 17, 2025

Contents

1 Introduction **2**
1.1 About Cantina 2
1.2 Disclaimer 2
1.3 Risk assessment 2
1.3.1 Severity Classification 2

2 Security Review Summary **3**
2.1 Scope 3

3 Overview **4**
3.1 Overall Security Posture & Maturity 4
3.2 Major Findings and Risks 4
3.3 General Business Impact 4
3.4 Review Outcome 4

4 Findings **5**
4.1 Medium Risk 5
4.1.1 `U64DynamicAddress` Pricing cannot distinguish between assets 5
4.2 Low Risk 5
4.2.1 Global config initialization can be frontrun 5
4.2.2 Asset type modification post-initialization causes denial of service on swaps 5
4.2.3 No validation for combined fees causes transaction panic when total exceeds 100% 6
4.2.4 Missing `protocol_fee_bps` upper bounds validation in update global config 6
4.2.5 Vault token accounts not closed in `close_pair()` instruction 6
4.3 Gas Optimization 7
4.3.1 Unnecessary CPI call when protocol fees are zero 7
4.4 Informational 7
4.4.1 Unused macros and error codes 7
4.4.2 Missing token account type constraints in `spl_helpers::transfer()` 7
4.4.3 Unused version field in `AssetConfig` struct 8
4.4.4 Error code `MathOverflow` used instead of `MathUnderflow` for `checked_sub()` operations 8
4.4.5 Typographical errors 8
4.4.6 NavData enum documentation is incomplete and misleading 8
4.4.7 Integer overflow in `min_acceptable_nav` calculation causes transaction failure for large NAV values 9
4.4.8 Function `set_new_admin()` does not check for duplicate assignment 9
4.4.9 Swap input amount limited to ~18.45 tokens for 18-decimal tokens due to `u64` type 10
4.4.10 Redundant `mint_address` assignment in `update_asset_config_account()` 10
4.4.11 Redundant pause state initialization to default value 10
4.4.12 Use `UncheckedAccount` instead of `AccountInfo` 10
4.4.13 Unchecked `.unwrap()` in `SwapExecuted` event emission 11

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Multiliquid is a decentralized protocol that enables institutional-grade atomic swaps between multiple permissioned Real World Asset (RWA) tokens and multiple stablecoins.

From Nov 23rd to Nov 24th the Cantina team conducted a review of Multiliquid - Solana - swap - program on commit hash f316df4c. The team identified a total of **20** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	5	4	1
Gas Optimizations	1	0	1
Informational	13	11	2
Total	**20**	**16**	**4**

2.1 Scope

The security review had the following components in scope for Multiliquid - Solana - swap - program on commit hash f316df4c:

```
programs/swap-program
├── Cargo.toml
├── src
│   ├── constants.rs
│   ├── error.rs
│   ├── instructions
│   │   ├── add_liquidity.rs
│   │   ├── claim_fees.rs
│   │   ├── close_pair.rs
│   │   ├── confirm_new_admin.rs
│   │   ├── init_asset_config_account.rs
│   │   ├── init_global_config.rs
│   │   ├── init_pair.rs
│   │   ├── mod.rs
│   │   ├── remove_liquidity.rs
│   │   ├── set_new_admin.rs
│   │   ├── set_paused_for_asset.rs
│   │   ├── set_paused_for_lp_stable_config.rs
│   │   ├── swap.rs
│   │   ├── update_asset_config_account.rs
│   │   ├── update_global_config.rs
│   │   └── update_pair.rs
│   ├── lib.rs
│   ├── macros.rs
│   ├── math.rs
│   ├── spl_helpers.rs
│   └── state
│       ├── asset_config.rs
│       ├── global_config.rs
│       ├── lp_stable_config.rs
│       ├── mod.rs
│       ├── pair.rs
│       └── user_vault_info.rs
└── Xargo.toml
```

3 Overview

3.1 Overall Security Posture & Maturity

The security review of the Multiliquid Solana swap program indicates a moderate level of security maturity. While no critical or high-risk vulnerabilities were identified, several areas for improvement were noted across different risk categories. The project team demonstrated responsiveness by promptly addressing the most significant medium-risk finding and the majority of low-risk and informational issues identified during or shortly after the review period. This proactive approach reflects a commitment to enhancing the protocol's security.

3.2 Major Findings and Risks

- **U64DynamicAddress Pricing cannot distinguish between assets** This issue meant the system could incorrectly determine the value of assets, potentially causing users to swap at inaccurate prices. This could lead to financial losses for users and erode trust in the platform's fairness and reliability.
- **Global config initialization can be frontrun** A malicious actor could potentially initiate the system's global configuration with their own parameters before the legitimate administrators. If undetected, this could lead to a loss of operational control, financial misdirection, such as diverting fees, and significant reputational damage to the Multiliquid platform.

3.3 General Business Impact

The identified issues primarily posed risks to the financial integrity and operational stability of the Multiliquid swap program. Incorrect asset pricing, if unaddressed, could lead to financial losses for users, directly impacting customer trust and potentially drawing negative public attention. The risk of a malicious actor frontrunning the system's initialization could lead to unauthorized control over critical parameters, resulting in financial theft or operational disruption, with severe financial and reputational consequences for Multiliquid. While many issues have been resolved, ongoing vigilance and careful deployment procedures are crucial to maintain a robust and trustworthy platform.

3.4 Review Outcome

The security review concluded with most identified findings, including the highest-severity medium risk and several low and informational items, being successfully addressed by the development team. The significant risk related to global configuration initialization being frontrun was acknowledged. This indicates the team is aware of this potential vulnerability and must implement robust deployment safeguards and validation procedures to mitigate this risk effectively, ensuring a secure launch and ongoing operation.

4 Findings

4.1 Medium Risk

4.1.1 `U64DynamicAddress` Pricing cannot distinguish between assets

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: When a `U64DynamicAddress` type is processed in the `get_nav` function, it reads the nav from the first account in the unchecked `remaining_accounts` array. There are a few issues with how this is done.

1. Only the first of the user-provided `remaining_accounts` is ever read from and used for all dynamic prices of the asset.
2. The only requirement for this account is that it be owned by a specified `nav_program_id` account. Should this `nav_program_id` own multiple accounts then any one of them could be passed and read as a price for any other assets in this process.

Recommendation:

1. Either iterate through the remaining_accounts or don't allow multiple dynamic pricing accounts for a single asset.
2. Determine if supporting dynamic accounts is required and whether one could not ask the data account owner for the price of the asset rather than directly querying the data account. Another possibility could include having the pricing data account owned by the swap program itself and tracking which account would go to which asset.

4.2 Low Risk

4.2.1 Global config initialization can be frontrun

Severity: Low Risk

Context: init_global_config.rs#L27

Description: There is no access control on the global config initialization. A malicious actor can frontrun the initialization with their own parameters. If this goes undetected, user funds could be at risk.

Recommendation: Consider one of the following:

1. Restrict initialization to privileged accounts.
2. Always ensure initialization scripts validate configurations after the transaction submission.

4.2.2 Asset type modification post-initialization causes denial of service on swaps

Severity: Low Risk

Context: update_asset_config_account.rs#L51

Description: The `update_asset_config_account()` function allows the protocol admin to modify the asset_type field of an AssetConfig account after initialization. This will break the swap implementation, causing swaps to fail.

The `swap()` function enforces strict asset type validation at `swap.rs`:

```
// check that the asset types are correct
require!(ctx.accounts.rwa_config.asset_type == AssetType::Rwa,
↪   ErrorCode::InvalidAssetType);
require!(ctx.accounts.stable_config.asset_type == AssetType::Stable,
↪   ErrorCode::InvalidAssetType);
```

When an admin changes the asset_type of an asset from its original designation (e.g., changing a Stable to an Rwa, or vice versa), all swaps involving that asset will permanently revert with an InvalidAssetType error.

This effectively bricks all liquidity pools using that asset. However, LPs are not affected, as such validations do not happen in the withdrawal liquidity-related functions.

Recommendation: Make asset_type immutable after initialization by removing it from the `update_asset_config_account()` function:

```
pub fn update_asset_config_account(
     ctx: Context<UpdateAssetConfigAccount>,
     nav_data: Vec<NavData>,
     price_difference_bps: u16,
     // Remove asset_type parameter
 ) -> Result<()> {
     // ....
     // Remove: asset_config.asset_type = asset_type;

     Ok(())
 }
```

If asset type updates must be supported for legitimate reasons, add comprehensive validation:

1. Check that no active pairs exist using this asset before allowing type changes.
2. Add a time-lock mechanism requiring a delay before the change takes effect.
3. Emit events to warn LPs of the upcoming change.
4. Consider adding a separate "migration" function with stricter controls.

4.2.3 No validation for combined fees causes transaction panic when total exceeds 100%

Severity: Low Risk

Context: math.rs#L90-L97

Description: The protocol validates individual fee parameters but fails to validate that combined fees (pair fees + protocol fees) don't exceed 100%, causing transaction panics during swaps.

1. Init_pair / update_pair.

```
require!(redemption_fee_bps <= 9900, ErrorCode::OutOfRange);  // Up to 99%
require!(discount_rate_bps <= 9900, ErrorCode::OutOfRange);    // Up to 99%
```

2. Init_global_config.

```
require!(protocol_fees_bps <= 9900, ErrorCode::OutOfRange);  // Up to 99%
```

As a result, the total fees calculated exceed the amount_in parameter during stable-to-asset swaps, resulting in an underflow.

Proof of Concept: Consider the following scenario:

1. Admin sets protocol_fees_bps = 200 (2%).
2. LP sets redemption_fee_bps = 9900 (99%).
3. User attempts swap: 9900 + 200 = 10100 > 10000.
4. Transaction panics with a MathOverflow error instead of meaningful message.

Recommendation: Add validations in all fee configuration functions to ensure the total_fee_bps remains less than 10,000. Or consider adding a validation in the `calculate_swap_results()` function.

4.2.4 Missing `protocol_fee_bps` upper bounds validation in update global config

Severity: Low Risk

Context: update_global_config.rs#L43-L45

Description: The `update_global_config()` function allows updating protocol_fees_bps without any validation, unlike `init_global_config()`, which enforces a maximum of 9900 BPS.

As a result, the function allows setting protocol_fees_bps to any u16 value (up to 65535), breaking protocol_-fee_bps assumptions throughout the protocol.

Recommendation: Consider adding the same validation used in `init_global_config()`:

```
   if let Some(protocol_fees_bps) = protocol_fees_bps {
+      require!(protocol_fees_bps <= 9900, ErrorCode::OutOfRange);
       new_protocol_fees_bps = protocol_fees_bps;
  }
```

4.2.5 Vault token accounts not closed in `close_pair()` instruction

Severity: Low Risk

Context: close_pair.rs#L39-L59

Description: The close_pair instruction fails to close the SPL token vault accounts (stable_coin_vault_-token_account and asset_token_vault_token_account) when the reference counter (used) reaches zero.

While the instruction properly:

1. Transfers remaining tokens back to the LP.
2. Closes the UserVaultInfo PDA accounts.

It does not close the actual SPL token vault accounts themselves, resulting in orphaned token accounts that permanently lock rent on-chain.

Recommendation: Add SPL token account closure to the `close_pair()` instruction `when used == 0`:

```
close_account(
  CpiContext::new_with_signer(
     ctx.accounts.token_program_stable.to_account_info(),
        CloseAccount {
          account: ctx.accounts.stable_coin_vault_token_account.to_account_info(),
          destination: ctx.accounts.admin.to_account_info(),
          authority: ctx.accounts.program_authority.to_account_info(),
        },
       &[signer_seeds],
   )
)?;
```

4.3 Gas Optimization

4.3.1 Unnecessary CPI call when protocol fees are zero

Severity: Gas Optimization

Context: swap.rs#L264-L273, swap.rs#L285-L294

Description: The `swap()` function in swap.rs makes CPI calls to transfer protocol fees even when the fee amount is zero, unnecessarily wasting compute units.

Recommendation: Consider adding a conditional check to skip the transfer CPI when protocol fees are zero (only if protocol_fees can be zero). If it will never be set to zero, then adding an additional check is redundant and this finding could be safely ”acknowledged”.

4.4 Informational

4.4.1 Unused macros and error codes

Severity: Informational

Context: error.rs#L4, macros.rs#L1

Description: The codebase contains several instances of dead code that is defined but never used throughout the program:

1. Unused Macro. The assert_range macro is defined and exported with #[macro_export] but has zero usages across the entire codebase.
2. Unused Error Code Variants. Seven error code enum variants are defined, but never referenced in any instruction or validation logic:

- InvalidNewAdmin.
- NotAdmin.
- InvalidMintAddress.
- MustProvidePriceDecimals.
- MustProvideNavProgramIdOrNavAccountDiscriminator.
- MustProvideNavPriceOffset.
- LpFeeMustBePositive.

Recommendation:

1. Remove the unused assert_range macro from `macro.rs`.
2. Remove all seven unused error code variants from `error.rs` to maintain a clean error enumeration that only includes actively used error cases.

4.4.2 Missing token account type constraints in `spl_helpers::transfer()`

Severity: Informational

Context: spl_helpers.rs#L6-L7

Description: The `spl_helpers::transfer()` function accepts generic AccountInfo parameters for the from and to accounts instead of strongly-typed `InterfaceAccount<'info, TokenAccount>` parameters.

Current state:

- All nine usage instances (swap.rs, claim_fees.rs, close_pair.rs, add_liquidity.rs, remove_liquidity.rs) properly validate token accounts at the instruction level with `InterfaceAccount<'info, TokenAccount>` and appropriate constraints.
- Accounts are downgraded to AccountInfo when passed to the helper via `.to_account_info()`.
- The helper relies entirely on the SPL Token Program validation during CPI.

While the current implementation is secure, it weakens Rust's type-safety guarantees. Future code modifications or new call sites could accidentally pass incorrect account types, with errors only caught at runtime during CPI execution rather than at compile-time or during Anchor validation.

Recommendation:

1. Change the helper function signature to enforce strong typing and remove downgrading to type AccountInfo while passing it to the helper in all usage instances.

```
from: &InterfaceAccount<'info, TokenAccount>,
to: &InterfaceAccount<'info, TokenAccount>
```

2. Alternatively, add relevant documentation in the `spl_helpers::transfer` function to ensure this behavior is properly documented for future developers.

4.4.3 Unused version field in `AssetConfig` struct

Severity: Informational

Context: asset_config.rs#L70-L71

Description: The `AssetConfig` struct contains a version field that is defined but never used anywhere in the codebase. The version field is documented as ”version of the NavData struct”.

Recommendation: Consider removing the unused field (or) implement version tracking if required and start from version 1 instead of 0.

4.4.4 Error code `MathOverflow` used instead of `MathUnderflow` for `checked_sub()` operations

Severity: Informational

Context: math.rs#L106-L112, math.rs#L180-L182, math.rs#L193-L196

Description: The math.rs module inconsistently uses error codes for arithmetic operations. Multiple instances of `checked_sub()` (subtraction) incorrectly return MathOverflow instead of MathUnderflow, making debugging significantly harder.

Recommendation: Update all `checked_sub()` operations to use MathUnderflow error code.

4.4.5 Typographical errors

Severity: Informational

Context: asset_config.rs#L4, asset_config.rs#L81

Description:

```
- /// NavData is used to store info about the methpds to retrieve the NAV data for a
↪   specific asset
+ /// NavData is used to store info about the methods to retrieve the NAV data for a
↪   specific asset


- // DRY function to read the price from an account (expecting the price to be stored as
↪   au64)
+ // DRY function to read the price from an account (expecting the price to be stored as
↪   u64)
```

4.4.6 NavData enum documentation is incomplete and misleading

Severity: Informational

Context: asset_config.rs#L5-L11

Description: The inline documentation for the `NavData` enum in `asset_config.rs` states that "Currently we support three methods" for retrieving NAV data. However, the actual implementation supports four distinct methods. The fourth variant, `PythPush`, is completely omitted from the documentation comment.

Recommendation: Update the inline documentation to reflect all four supported NAV data retrieval methods accurately.

4.4.7 Integer overflow in `min_acceptable_nav` calculation causes transaction failure for large NAV values

Severity: Informational

Context: asset_config.rs#L260-L264

Description: The `get_nav()` function in `asset_config.rs` performs a price difference validation check that is vulnerable to arithmetic overflow when NAV values are significant (has less price decimals).

When `max_nav` is sufficiently large (e.g., 1e18 = 1e9 with zero decimals), the multiplication operation `max_nav.checked_mul(10000 -  price_difference_bps as u64)` overflows the `u64` type and returns `None`. The subsequent `.unwrap()` call then panics with the error `Option::unwrap()` on a None value.

This issue results in:

1. Denial of Service: Any asset with a NAV above approximately 1.84 × 10^15 (with 9 decimals) will cause all transactions to fail.

2. Unpredictable Failures: The issue only manifests when NAV reaches certain thresholds, making it difficult to diagnose.
3. No Graceful Error Handling: The transaction panics rather than returning a proper error code.

Proof of Concept:

- `max_nav` = 1_000_000_000_000_000_000 (1 quintillion, representing a price of 1e9 asset with 0 decimals) - `price_difference_bps` = 500 (5% tolerance).
- Calculation: 1_000_000_000_000_000_000 × (10000 - 500) = 1_000_000_000_000_000_000 × 9500 - Result: 9.5 × 10^21, which exceeds `u64::MAX` (18,446,744,073,709,551,615 ≈ 1.84 × 10^19) - The `checked_mul()` returns `None`, causing a panic.

Recommendation:

1. Replace all `.unwrap()` calls with proper error handling using `.ok_or(ErrorCode::MathOverflow)?` which will return meaningful error.
2. Add comprehensive tests with edge cases: Maximum safe NAV values, Minimum price_difference_bps values, and Various combinations that approach u64 limits.
3. Document maximum supported NAV values in the code comments.

Multiliquid: Fixed by 4cf5446b3adfa8d12d8dc7f86a7007d6f3ffbec5 (for now, we've added error handling with MathOverflow).

The likelihood is really close to 0 (most of the stables will have a value around $1, and for RWAs, we can't imagine having such a NAV. Since it is not something a random user (the NAV) could set, we don't see a security issue here either.

Cantina: While the newly added error checks will detect and revert on overflows with more explicit error messages, the protocol's capacity to safely handle large NAV values remains constrained. It may still result in unintended or unpredictable behavior.

4.4.8 Function `set_new_admin()` does not check for duplicate assignment

Severity: Informational

Context: set_new_admin.rs#L23

Description: The `set_new_admin()` function lacks validation to ensure that the new admin address differs from the current admin address. This allows the current admin to unnecessarily set themselves as the pending admin again, wasting transaction fees and emitting misleading events without actual state changes.

Recommendation: Add validation to prevent setting the new admin to the current admin address:

```
require!(new_admin != global_config.admin, ErrorCode::AlreadyAdmin);
```

4.4.9 Swap input amount limited to ~18.45 tokens for 18-decimal tokens due to `u64` type

Severity: Informational

Context: add_liquidity.rs#L56, remove_liquidity.rs#L58, swap.rs#L165

Description: The swap instruction uses u64 to represent token amounts, which significantly restricts the maximum swappable amount for tokens with high decimal precision.

For tokens with 18 decimals, the maximum value of u64 (18,446,744,073,709,551,615) translates to only approximately 18.45 tokens in human-readable terms:

u64::MAX / 10^18 = 18,446,744,073,709,551,615 / 1,000,000,000,000,000,000 ≈ 18.45.

This limitation severely restricts the protocol's usability for:

- High-value swaps involving tokens with 18 decimals.
- Liquidity provision operations that require larger amounts.
- Any operation where users need to transact more than ~18 tokens.

The issue affects not just swap operations but potentially all instruction parameters that handle token amounts using u64, including `add_liquidity()`, `remove_liquidity()`, and related functions.

Recommendation: Replace u64 with u128 for all token amount parameters throughout the program. The u128 type provides sufficient range to handle tokens with up to 38 decimals at scale:

u128::MAX / 10^18 ≈ 340 trillion tokens.

4.4.10 Redundant `mint_address` assignment in `update_asset_config_account()`

Severity: Informational

Context: update_asset_config_account.rs#L48

Description: The `update_asset_config_account()` function contains unnecessary computation where mint_address is assigned or validated.

Since the asset_config account is a PDA derived with mint_address as a seed, the Anchor framework already enforces that the account can only be accessed with the correct mint_address used in derivation.

Any assignment or validation of this field is redundant and wastes compute units.

Recommendation: Remove the redundant mint_address assignment or validation from the update_-asset_config_account function.

4.4.11 Redundant pause state initialization to default value

Severity: Informational

Context: init_asset_config_account.rs#L79, init_pair.rs#L144

Description: In the `init_pair()` and `init_asset_config_account()` functions, the pause state is explicitly set to false, which is already the default value for boolean fields in newly initialized accounts. This redundant assignment wastes compute units without providing any functional benefit.

Recommendation: Remove the explicit pause state assignment to false in both `init_pair()` and `init_asset_config_account()` functions.

Multiliquid: Fixed in e8625298 and 7f3591ab

4.4.12 Use `UncheckedAccount` instead of `AccountInfo`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Multiple instructions across the entire program use `AccountInfo` instead of `UncheckedAccount`, against the anchor's suggestion.

Recommendation: Consider replacing all instances of `AccountInfo` in-favor of `UncheckedAccount`.

4.4.13 Unchecked `.unwrap()` in `SwapExecuted` event emission

Severity: Informational

Context: swap.rs#L331

Description: The `amount_in_for_vault.checked_add(protocol_fees).unwrap()` could panic if the addition overflows.

Recommendation: Consider using `.ok_or` at the end of the checked_add to ensure the overflow is caught and reverted accordingly.