Security Assessment
Final Report

Fragmetric restaking v0.6.3

May - July 2025

Prepared for Fragmetric

Table of content

Project Summary **3**
Project Scope 3
Project Overview 3
Protocol Overview 3
Findings Summary 5
Severity Matrix 5
Detailed Findings **6**
Medium Severity Issues **8**
M-01 Array index mismatch causes withdrawal processing panic 8
M-02 The OrcaDEXLiquidityPool pricing source is vulnerable to price manipulation 9
M-03 Incorrect assert in set_supported_token_redelegating_amount allows for wrong redelegating amount setting 11
M-04 Missing validation allows same token swap strategy registration leading to DoS 12
Low Severity Issues **13**
L-01 Inconsistent command discriminants and module file numbering 13
L-02 Users can lose rewards when settlement blocks exceed maximum capacity 14
L-03 Token swap strategies cannot be updated or removed 15
L-04 Missing mint account validation for distributing reward token 16
L-05 Missing account ownership validation for vault operator delegation 17
Informational Severity Issues **19**
I-01. Undocumented magic values for minimum operation thresholds 19
I-02. Unused command imports in DenormalizeNT module 20
I-03. Hardcoded seeds string should use SPL library constant for extra account metas 21
I-04. Unused desired_account_size parameter for user_create_fund_account_idempotent 22
I-05. Inconsistent event emission across the codebase 23
I-06. Unimplemented vault types create silent failures 24
I-07. Insufficient vector capacity reservation in resolve_underlying_assets 25
I-08. Redundant error handling in ClaimUnstakedSOLCommand::execute_prepare 26
I-09. Typos in variable and function names 27
Disclaimer **28**
About Certora **28**

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Fragmetric restaking v0.6.3	/fragmetric-labs/fragmetric-contracts	581422f	Solana

Project Overview

This document describes the verification of the Fragmetric staking program using manual code review. The work was undertaken from May 22nd 2025 to July 18th 2025..

The following files are considered in scope for this review:

- All files under **/programs/restaking/src**

The team performed a manual audit of all the contracts**.** During manual audit, the Certora team discovered issues in the code, as listed in the following pages.

Protocol Overview

The Fragmetric program is an on-chain protocol for the Solana blockchain that helps users and fund managers manage staking, restaking, liquidity, and fund operations across multiple supported protocols. It makes it easier to move and allocate assets between different staking pools and vaults, so users can earn rewards and manage risk without having to interact with each protocol separately.

Fragmetric uses a command-based workflow, where each operation—such as staking, unstaking, restaking, delegating, withdrawing, or harvesting rewards—is handled as a series of commands. Each command represents a specific step or action, and the workflow ensures that these steps are executed in order, can be paused and resumed, and that progress is tracked throughout the process.

The program uses token2022 hooks to track rewards. By using these hooks, Fragmetric can monitor and account for rewards earned through staking and restaking activities, ensuring fair and correct accurate reward distribution.

Fragmetric uses restaking to optimize rewards by reallocating assets that have already been staked in one protocol into another staking or restaking vault. This approach enables more efficient use of assets and the potential for higher returns. Currently, restaking in Fragmetric is only supported through Jito, so only Jito restaking vaults are used for this process.

Fragmetric supports a variety of token types and vaults, including Marinade, SPL Stake Pool, Sanctum, Jito Restaking Vaults, and its own internal vaults. The protocol keeps careful track of all assets, including how much is reserved, pending, or available for each token and SOL. It also manages fees, making sure that any protocol or external fees are applied correctly.
Fragmetric is designed to batch actions and allow partial execution, so large operations can be split up and completed over several transactions.

Special attention was given during the audit to the following:

- Command-based workflow and execution of staking, unstaking, restaking, delegation, withdrawal, and reward harvesting operations.
- State machine logic for command sequencing, step tracking, and safe resumption of interrupted processes.
- Account validation, asset tracking, and prevention of out-of-bounds or misaligned access.
- Integration and handling of multiple token pricing sources, including supported, unsupported, and internal types.
- Use of token2022 hooks for tracking and accounting of staking and restaking rewards.
- Calculation and application of protocol and external fees, including rounding and precision handling.
- Batch processing, partial execution, and logic for resuming operations after transaction or compute limits are reached.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	0	0	0
High	0	0	0
Medium	4	4	4
Low	5	5	4
Informational	9	9	3
Total	18	18	11

Severity Matrix

Impact					
	Critical	Medium	Medium/High	High	Critical
	High	Low/Medium	Medium	Medium/High	High
	Medium	Low	Low/Medium	Medium	Medium/High
	Low	Informational	Low	Low/Medium	Medium
		Rare	Unlikely	Likely	Very Likely
			Likelihood		

certora

Fragmetric

Detailed Findings

ID	Title	Severity	Status
M-01	Array index mismatch causes withdrawal processing panic	Medium	Fixed in 1d84067
M-02	The OrcaDEXLiquidityPool pricing source is vulnerable to price manipulation	Medium	Fixed in ae0d816
M-03	Incorrect assert in set_supported_token_redelegating_amount allows for wrong redelegating amount setting	Medium	Fixed in 54bc48c
M-04	Missing validation allows same token swap strategy registration leading to DoS	Medium	Fixed in 5134c8e
L-01	Inconsistent command discriminants and module file numbering	Low	Fixed in 9f7054c
L-02	Users can lose rewards when settlement blocks exceed maximum capacity	Low	Will not be fixed
L-03	Token swap strategies cannot be updated or removed	Low	Fixed in 5134c8e
L-04	Missing mint account validation for distributing reward token	Low	Fixed in 1d391ef
L-05	Missing account ownership validation for vault operator delegation	Low	Fixed in e75b816

I-01	Undocumented magic values for minimum operation thresholds	Informational	Fixed in 1881a84
I-02	Unused command imports in DenormalizeNT module	Informational	Fixed in e75b816
I-03	Hardcoded seeds string should use SPL library constant for extra account metas	Informational	Not fixed
I-04	Unused desired_account_size parameter for user_create_fund_account_idempotent	Informational	Planned for next release
I-05	Inconsistent event emission across the codebase	Informational	Will not be fixed
I-06	Unimplemented vault types create silent failures	Informational	Fixed in 1a4f22f
I-07	Insufficient vector capacity reservation in resolve_underlying_assets	Informational	Planned for next release
I-08	Redundant error handling in ClaimUnstakedSOLCommand::execute_prepare	Informational	Planned for next release
I-09	Typos in variable and function names	Informational	Planned for next release

Medium Severity Issues

M-01 Array index mismatch causes withdrawal processing panic		
Severity: **Medium**	Impact: **High**	Likelihood: **Unlikely**
Files: cmd6_process_withdrawal_batch.rs	Status: Fixed in 1d84067	

Description:
There is an issue in *cmd6_process_withdrawal_batch*.rs where the code incorrectly assumes array indices match token iteration indices, leading to runtime panics when *PeggedToken* or other skipped token types are present in the fund's token list.

At the very beginning, only specific token types increment the counter during *Execute*. For example, *PeggedToken* returns without incrementing, while *SPLStakePool* increments it.

Later, the array is allocated with size equal to the count of specific token types only. Right after, in *#290* loop, it uses enumerate which provides indices for all tokens once again, while the array contains only entries for the specific token types (let's assume *SPLStakePool* only). That means that if there were some other not counted token types initially, for example *[PeggedToken, SPLStakePool, PeggedToken, SPLStakePool]*, the index will not be aligned to the array elements, panicking when accessed.

Consequently, when there will be only two elements, accessing *supported_token_pricing_sources[3]* will fail. As this will panic, the entire transaction will be reverted, effectively preventing withdrawals.

Recommendations:
We recommend introducing a separate index variable that will track position within the filtered array.

Customer response:

Fixed in commit 1d84067**.** For transparency: with today's fund line-up (fragSOL, fragJTO, fragBTC, FRAG²) the problematic index pattern cannot occur, so user funds were not at risk.

Fix review:

The update correctly handles the issue. Account index i is only incremented for supported tokens that use a pricing source account.

M-02 The *OrcaDEXLiquidityPool* pricing source is vulnerable to price manipulation

Severity: **Medium**	Impact: **Critical**	Likelihood: **Unlikely**
Files: orca_dex_liquidity_pool_value_provider.rs	Status: Fixed in ae0d816.	

Description:

The *TokenPricingSource::OrcaDEXLiquidityPool* implementation directly uses the pool's *sqrt_price* for token price calculations without any manipulation resistance mechanisms. While this vulnerability currently has limited impact due to the specific configuration of supported tokens, as all tokens in affected pools like *fragBTC* and *fragJTO* use the same oracle through pegged relationships, however it presents a security risk if the system evolves in the future.

Currently, pools like *fragBTC* maintain price stability through an invariant where all supported assets (*zBTC, cbBTC, wBTC*) share the same price oracle - either directly (*zBTC* via *OrcaDEXLiquidityPool*) or indirectly (*cbBTC*/*wBTC* via *PeggedToken* pointing to *zBTC*).

This creates a ratio that remains stable regardless of the volatile underlying price.

However, if any pool were to add a token with an independent pricing source (not pegged to the pool's primary asset), this invariant would break. An attacker could then manipulate the Orca DEX pool price to create arbitrage opportunities between the manipulated price and the true market price of the newly added token.

Recommendations:
We recommend implementing a Time Weighted Average Price (TWAP) mechanism for *OrcaDEXLiquidityPool* instead of direct *sqrt_price*, making price manipulation significantly more expensive and difficult. Additionally, enforce strict on-chain validation to ensure only tokens with the same pricing source or explicitly pegged tokens can be added to pools using *OrcaDEXLiquidityPool*.

Customer response:
Fixed in commit ae0d816. The fix will assert that if an Orca DEX pricing source is registered, only tokens pegged to that source may be added. TWAP alone is insufficient,
because even a thin, manipulated pool could still be exploited.

Fix review:
An extra check validate_new_supported_token_pricing_source is added to process_add_supported_token. This check correctly ensures that OrcaDex pricing source can only be set for the first token in a fund account with fund_account.sol.depositable disabled. If a token with OrcaDex pricing source is set, only supported tokens with PeggedToken pricing source can be added.
This check ensures that when OrcaDex is used, only tokens with PeggedToken pricing can be used. As any price manipulation would impact all tokens of the fund account vault equally, this nullifies any price manipulation effects.

M-03 Incorrect assert in *set_supported_token_redelegating_amount* allows for wrong redelegating amount setting

Severity: **Medium**	Impact: **Medium**	Likelihood: **Likely**
Files: fund_account_restaking_vaults.rs	Status: Fixed in 54bc48c	

Description:
In the *fund_account_restaking_vault*.rs, the *set_supported_token_redelegating_amount* function contains a critical logic error in its validation check. The function is responsible for setting the amount of tokens to be redelegated from a restaking vault delegation. The current implementation uses *require_gte!(token_amount, self.supported_token_delegated_amount, ErrorCode::FundInvalidConfigurationUpdateError)*, which enforces that the redelegating amount must be greater than or equal to the currently delegated amount.

This is logically incorrect because it's impossible to redelegate more tokens than what is currently delegated. The parameters in the *require_gte!* call are inverted.

This validation could lead to wrong delegating amount setting, causing fund accounting errors and downstream calculation issues in other parts of the system that rely on the relationship between delegated and redelegating amounts.

Recommendations:
The validation parameters should be swapped to ensure that the redelegating amount cannot exceed the delegated amount.

Customer response:
Fixed in commit 54bc48c. The redelegation feature has been removed in the current release; this code path no longer exists.

M-04 Missing validation allows same token swap strategy registration leading to DoS

Severity: **Medium**	Impact: **Medium**	Likelihood: **Unlikely**
Files: fund_account.rs	Status: Fixed in 5134c8e	

Description:
The *add_token_swap_strategy* function in *modules/fund/fund_account.rs* lacks validation to ensure that *from_token_mint* and *to_token_mint* are different tokens. Currently, the function only checks if a swap strategy for the *from_token_mint* already exists and validates the maximum number of strategies, but it does not prevent registering a swap strategy where the source and destination tokens are identical.

This missing validation can lead to a Denial of Service (DoS) scenario. When an automated token swap is attempted with identical source and destination tokens, the operation will fail at the system program level, disrupting the normal operation of the fund.

Recommendations:
We recommend adding a validation check in the *add_token_swap_strategy* function to ensure that *from_token_mint* and *to_token_mint* are not equal before initializing the token swap strategy. This can be implemented by adding a *require_keys_neq!* check or similar validation after the existing checks and before the strategy initialization.

Customer response:
Fixed in commit 5134c8e. Now it rejects identical from_mint ↔ to_mint pairs and validates the underlying Orca pool.

Low Severity Issues

L-01 Inconsistent command discriminants and module file numbering		
Severity: **Low**	Impact: **Low**	Likelihood: **Likely**
Files: mod.rs	Status: Fixed in 9f7054c	

Description:
There is an inconsistency in *module/fund/commands/mod.rs* where the discriminant values assigned to commands do not match their module file naming convention. The *discriminant* function assigns *OperationCommand::StakeSOL(_) => 10* and *OperationCommand::HarvestReward(_) => 11*, while the module files are named *cmd10_harvest_reward.rs* and *cmd11_stake_sol.rs*, suggesting the opposite assignment.

This mismatch does not cause functional issues because serialization and deserialization remain internally consistent - both operations use the same *discriminant* function. When *HarvestRewardCommand* is serialized with discriminant 11 and later deserialized, the verification check compares the stored discriminant (11) with the command's discriminant (11), allowing successful deserialization.

The execution flow is determined by what each command returns in its *execute* function, not by discriminants or file names. For instance, *HarvestRewardCommand* transitions to *StakeSOLCommand* regardless of their discriminant values. While this inconsistency does not impact protocol functionality, it could lead to developer errors if someone assumes file numbers correspond to discriminant values when adding new commands or debugging the protocol code.

Recommendations:
We recommend aligning the discriminant values with the module file names by updating the *discriminant* function to assign *OperationCommand::StakeSOL(_) => 11* and *OperationCommand::HarvestReward(_) => 10*, or alternatively renaming the files to match the current discriminant assignment.

Customer response:
Fixed in commit [9f7054c]

L-02 Users can lose rewards when settlement blocks exceed maximum capacity		
Severity: **Low**	Impact: **Medium**	Likelihood: **Unlikely**
Files: user_reward_settlement.rs	Status: Will not be fixed	

Description:
The reward settlement system maintains a maximum of 64 settlement blocks per reward account. When this limit is reached and new blocks need to be added, the *force_clear_settlement_block* function removes the oldest block and transfers its remaining amount to *reward_settlement.remaining_amount*.

However, if a user does not call *user_update_reward_pools* for an extended period (during which more than 64 new settlement blocks are created), they will permanently lose access to rewards from the removed blocks. The issue occurs because:

1. When processing missed blocks during *user_update_reward_pools*, the contribution is added to *total_settled_contribution* but not to *total_settled_amount*
2. The *claim* function only uses *total_settled_amount* to determine claimable rewards
3. Rewards from force-cleared blocks become inaccessible to users who haven't updated in time

This creates a scenario where active reward distribution can inadvertently penalize inactive users by making their accrued rewards unclaimable if they don't update within the 64-block window.

Given the current on-chain data shows only 5 blocks maximum being used per settlement, the severity of the issue was reduced, but it should be addressed to prevent potential future reward losses as the protocol scales.

Recommendations:
Implement a mechanism to preserve user rewards even when settlement blocks are force-cleared.

Customer response:
Will not be fixed. We need to limit the storage and believe this should be handled as a matter of operation policy. We will document the 64-block cap and enforce periodic settlement in the fund-operation policy. At weekly settlements, the cap covers ≈1–2 years of history, which we deem sufficient.

L-03 Token swap strategies cannot be updated or removed

Severity: **Low**	Impact: **Medium**	Likelihood: **Unlikely**
Files: fund_account.rs	Status: Fixed in 5134c8e	

Description:
There is a functional limitation in the token swap strategy management where the *fund_manager_add_token_swap_strategy* function only supports adding new strategies without providing mechanisms to update or remove existing ones. Once a token swap strategy is added to a fund, it becomes permanent and cannot be modified or deleted.

This one way operation creates operational challenges. If a swap strategy is configured incorrectly, it cannot be corrected. Also, if a DEX pool becomes deprecated or malicious, the strategy cannot be removed, and if swap parameters need adjustment due to market conditions, no update mechanism exists.

Additionally, this design pattern contradicts standard administrative practices where configuration parameters can typically be updated or removed by authorized parties to maintain system health and adapt to changing requirements.

It is worth mentioning that the maximum number of existing strategies, defined as *FUND_ACCOUNT_MAX_TOKEN_SWAP_STRATEGIES*, is 30. When reached, no new strategies could be added.

Recommendations:
We recommend reconsidering implementation of complementary functions to manage swap strategies. These functions should validate that no pending operations depend on the strategy being modified or removed.

Customer response:
Fixed in commit 5134c8e. fund_manager_remove_token_swap_strategy has been added. The 30-strategy cap aligns with the reward-account size limit.

L-04 Missing mint account validation for distributing reward token

Severity: **Low**	Impact: **Low**	Likelihood: **Unlikely**
Files: fund_configuration_service.rs	Status: Fixed in 1d391ef	

Description:
There is a missing account type validation in *fund_manager_add_restaking_vault_distributing_reward_token* where the function accepts *distributing_reward_token_mint* without verifying it is actually a valid mint account. The function allows any account to be registered as a reward token mint without checking that the account contains valid mint data or is owned by the token program.

This validation gap means a fund manager could accidentally or intentionally register a non-mint account, such as a token account, system account, or arbitrary data account, as a reward token. When the protocol later attempts to distribute rewards using this registered "mint", operations will fail because the account lacks the expected mint structure.

This would break reward distribution for the affected vault, locking accumulated rewards and disrupting the expected reward flow to users. The issue is particularly problematic because the

error would only show up during reward distribution, not at configuration time. By then, the vault may have accumulated significant rewards that become unclaimable due to the invalid mint configuration.

Recommendations:
We recommend adding validation to verify the account is a valid mint by attempting to deserialize it as *InterfaceAccount<Mint>* or checking that the account owner is either *anchor_spl::token::ID* or *anchor_spl::token_2022::ID*. This validation should occur before storing the mint address in the vault's configuration.

Customer response:
Fixed in commit 1d391ef

L-05 Missing account ownership validation for vault operator delegation

Severity: **Low**	Impact: **Low**	Likelihood: **Unlikely**
Files: jito_restaking_vault_service.rs	Status: Fixed in e75b816	

Description:
There is a missing account ownership validation in *modules/restaking/jito_restaking_vault_service.rs* where the vault operator delegation account is not verified to be owned by the Jito vault program before being used. The validation function accepts delegation accounts without confirming they are owned by the expected Jito program.

This allows an attacker to create a spoofed account with an identical data layout but owned by a different program, manipulating delegation states or amounts that the system trusts. While the risk is reduced because this function is only callable by the fund manager (a trusted role), the lack of ownership verification violates Solana's security best practices.

Even trusted callers should not be able to pass accounts owned by incorrect programs, as this could lead to state corruption or enable attack vectors if the trust model changes in the future.

Recommendations:
We recommend adding an ownership check to verify that the *vault_operator_delegation* account is owned by the expected Jito vault program before processing its data. This should be implemented using *require_keys_eq!(vault_operator_delegation.owner, expected_jito_program_id)* or similar validation pattern used elsewhere in the codebase.

Customer response:
Fixed in commit e75b816. Deserialization now asserts that the account is owned by the Jito vault program, regardless of account type.

Informational Severity Issues

I-01. Undocumented magic values for minimum operation thresholds

Description:
There are undocumented magic values in *cmd7_unstake_lst.rs*, *cmd11_stake_sol.rs* and *cmd13_restake_vst.rs* where threshold checks use hardcoded numbers without explanation of their purpose or origin.

In *cmd13_restake_vst.rs*, the code checks *if allocated_token_amount >= 1_000_000* before adding items to the restaking list. Similarly, in *cmd11_stake_sol.rs*, the threshold *if allocated_sol_amount >= 1_000_000_000* is used to filter staking operations. The same pattern appears in *cmd7_unstake_lst.rs* with *if allocated_token_amount >= 1_000_000*.

These values appear to represent minimum amounts for operations - the 1_000_000_000 value equals 1 SOL (since 1 SOL = 10^9 lamports), suggesting it's a minimum SOL staking threshold. The 1_000_000 value likely represents a minimum token amount in the smallest unit, possibly to avoid dust transactions.

Without documentation, this makes it difficult to assess whether the values need adjustment when integrating new tokens with different decimals or when protocol requirements change. The lack of clarity also prevents proper validation of whether these thresholds are appropriate for all token types, particularly when different tokens may have vastly different decimal configurations.

Recommendations:
We recommend defining these values as named constants with clear documentation explaining their purpose.

Customer response:
Fixed in commit 1881a84

I-02. Unused command imports in *DenormalizeNT* module

Description:
There are unused imports in *modules/fund/commands/cmd4_denormalize_nt.rs* where the code imports specific *OperationCommand* variants that are never referenced in the file's implementation.

The import statement *use crate::modules::fund::commands::OperationCommand::{ClaimUnstakedSOL, UndelegateVST};* brings in two command types that are not utilized anywhere in the *DenormalizeNTCommand* logic.

This unused import does not affect the functionality or security of the contract, as Rust's compiler optimizes away unused code. The presence of unused imports can mask actual dependencies and make it harder to track which modules truly depend on which commands.

Recommendations:
We recommend removing the unused import statement for *ClaimUnstakedSOL* and *UndelegateVST* from the file, keeping only the imports that are actually utilized in the implementation.

Customer response:
Fixed in commit e75b816

I-03. Hardcoded seeds string should use SPL library constant for extra account metas

Description:
The *extra_account_meta_list* account uses hardcoded seed strings *b"extra-account-metas"* in three different contexts across the codebase:

- *AdminReceiptTokenMintExtraAccountMetaListInitialContext*
- *AdminReceiptTokenMintExtraAccountMetaListUpdateContext*
- *UserReceiptTokenTransferContext*

Each occurrence directly uses the inline string literal instead of leveraging the existing constant *EXTRA_ACCOUNT_METAS_SEED* from the SPL Transfer Hook Interface library. This approach reduces code maintainability and increases the risk of typos or inconsistencies if the seed value needs to be modified in the future.

Recommendations:
Replace all hardcoded *b"extra-account-metas"* seed strings with the *EXTRA_ACCOUNT_METAS_SEED* constant from the SPL library. Import the constant at the beginning of the relevant files and update all account declarations to use it.

I-04. Unused *desired_account_size* parameter for *user_create_fund_account_idempotent*

Description:
The *user_create_fund_account_idempotent* function accepts a *desired_account_size* parameter that is not utilized in the implementation. The parameter is passed through to *UserFundConfigurationService::process_create_user_fund_account_idempotent*, where it is prefixed with an underscore (*_desired_account_size*) and marked with a comment "*// reserved*", indicating it is intentionally unused.

This creates confusion for API consumers who may expect this parameter to influence the account size allocation, when in reality it has no effect on the function's behavior. The function always initializes accounts with a fixed size of *8 + UserFundAccount::INIT_SPACE*, regardless of the provided parameter value.

Recommendations:
Either remove the unused parameter from the public API to avoid confusion, or properly document its reserved status for future use.

Customer response:
Seems like the parameter can be deprecated; It will be removed in the next minor release.

I-05. Inconsistent event emission across the codebase

Description:
The codebase implements inconsistent event emission patterns across different instruction types, particularly for privileged operations. Analysis reveals that fund manager instructions lack event emission except for *fund_manager_add_normalized_token_pool_supported_token* and *fund_manager_remove_normalized_token_pool_supported_token*. Similarly, among admin instructions, only *admin_create_user_reward_account_idempotent* emits events, while critical administrative operations like *admin_initialize_fund_account*, *admin_update_fund_account_if_needed*, *admin_initialize_reward_account*, and *admin_update_reward_account_if_needed* do not emit any events.

This is problematic, as such approach might be problematic for frontend applications, monitoring systems, and data aggregators that rely on events to track state changes and user interactions. The absence of events for privileged operations is particularly problematic as these actions often involve critical configuration changes that external systems need to monitor for security, compliance, and user experience purposes.

Recommendations:
We recommend adding event emission to all instructions, especially admin and fund manager operations. Each event should include the action taken, accounts involved, and any state changes. This will help frontends and monitoring tools track what's happening in the system. Consider using a consistent event format for similar operations to make integration easier.

Customer response:
Design rationale: admin instructions merely initialize structures; activation requires fund-manager calls, which do emit events. Normalized-token-pool ops are internal. We will document this behaviour.

I-06. Unimplemented vault types create silent failures

Description:
There are multiple unimplemented vault type handlers across *modules/fund/commands/cmd1_initialize.rs*, *modules/restaking/mod.rs*, *modules/fund/commands/cmd13_restake_vst.rs*, and *modules/fund/commands/cmd14_delegate_vst.rs* where *SolvBTCVault* and other vault types have placeholder implementations that skip actual functionality. Throughout these files, *match* statements handle various *TokenPricingSource* variants, but *SolvBTCVault* consistently returns empty results or skips to the next item without performing any operations.

For example, in restaking operations, the code simply continues to the next vault when encountering *SolvBTCVault* rather than executing restaking logic. This creates a silent failure mode where operations involving these vault types appear to succeed but perform no actual work. Users might add *SolvBTC* vaults expecting them to participate in restaking, delegation, or initialization operations, but these vaults remain inactive without any error indication.

This incomplete implementation is particularly problematic because the code compiles and runs without errors, giving no indication that these vault types are non-functional.

Recommendations:
We recommend either completing the implementation for *SolvBTCVault* and other TODO-marked vault types, or explicitly returning errors when these types are encountered to prevent silent failures. If these implementations are planned for future releases, add clear error messages indicating the feature is not yet available rather than silently skipping the logic.

Customer response:
SolvBTC vault integration has been added in commit 1a4f22f

I-07. Insufficient vector capacity reservation in *resolve_underlying_assets*

Description:
In the *fund_receipt_token_value_provider.rs* file, the *resolve_underlying_assets* method reserves insufficient capacity for the *numerator* vector, causing unnecessary memory reallocation.

The code currently reserves exactly *TokenValue::MAX_NUMERATOR_SIZE (33)* items. However, the subsequent code can potentially add up to 34 items:

- 1 SOL asset
- Up to 16 supported tokens (*FUND_ACCOUNT_MAX_SUPPORTED_TOKENS = 16*)
- Up to 1 normalized token
- Up to 16 restaking vaults (*FUND_ACCOUNT_MAX_RESTAKING_VAULTS = 16*)

When all asset types are present (1 + 16 + 1 + 16 = 34), pushing the 34th item will trigger a vector reallocation, doubling the capacity to 66 items. This results in unnecessary memory allocation and data copying operations.

Recommendations:
Update the *TokenValue::MAX_NUMERATOR_SIZE* constant to 34 to match the actual maximum number of assets that can be added. Alternatively, calculate the exact capacity needed dynamically based on the fund account's configuration.

Customer response:
This will be fixed in a next release

I-08. Redundant error handling in *ClaimUnstakedSOLCommand::execute_prepare*

Description:
In the *execute_prepare* function of *ClaimUnstakedSOLCommand*, there are duplicate error checks for token pricing sources that create unreachable code. The first match statement at line 199 already errors out for any pricing source that isn't *SPLStakePool*, *MarinadeStakePool*, *SanctumSingleValidatorSPLStakePool*, or *SanctumMultiValidatorSPLStakePool*.

This means the second match statement at line 263 will only ever receive one of these four valid pricing sources, making its error arms for *JitoRestakingVault*, *FragmetricNormalizedTokenPool*, and other variants unreachable dead code.

Recommendations:
Remove the redundant error handling in the second match statement since these cases are already handled by the first match. The second match can be simplified to only handle the four valid pricing source types without error arms, improving code clarity and eliminating dead code.

Customer response:
This will be fixed in a next release

I-09. Typos in variable and function names

Description:
Throughout the *PricingService* implementation, there is a consistent typo where "micro" is misspelled as "mirco" in several key identifiers. Specifically, the *token_value_as_mirco_lamports* field and the *get_token_value_as_mirco_lamports* function both contain this typo.

This is particularly confusing because the code correctly uses "micro" in local variable names like *micro_lamports*, *to_micro_token*, and *from_micro_lamports*, creating an inconsistent naming convention within the same module. While this doesn't affect functionality, it impacts code readability and could lead to confusion for developers working with the codebase.

Recommendations:
Refactor all instances of "mirco" to "micro" for consistency, including renaming *token_value_as_mirco_lamports* to *token_value_as_micro_lamports* and *get_token_value_as_mirco_lamports* to *get_token_value_as_micro_lamports*.

Customer response:
This will be fixed in a next release

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.