certora

Jito Restaking Security Assessment & Formal Verification Report – V1

September 2024

Prepared for
Jito Labs

certora

Table of content

certora

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform	Comment
Jito restaking	https://github.com/jito-foundation/restaking	108571a	Solana	Audit version
Jito restaking	https://github.com/jito-foundation/restaking/releases/tag/v0.0.3	ecbe19a	Solana	Fix version

Project Overview

This document describes the specification and verification of **Jito Restaking** using manual code review and Certora Prover. The work was undertaken from **August 8, 2024,** to **September 19, 2024**.

The following contract list is included in our scope: all files under `restaking_core/` , `restaking_program/`, `vault_core/` , and `vault_program/`.

The Certora Prover demonstrated that the implementation of the Solana contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solana contracts in the scope. During the verification process and the manual audit, the Certora team discovered bugs in the Solana contracts code, as listed on the following page.

Please note that a few more formal rules are not included in this report, as they were proven with an unreleased version of the Certora Prover. Once those rules are proven on a released version of the Certora Prover, we will add them to the next version of this document.

We have verified the fixes that are present in the commit hash of the fix version. We adapted the formal rules to that version, updated the content of some rules and removed rules that refer to non exsitant instructions. We reran the entire testbench to verify that all rules that were violated in the audit version hold in the fix version, and no new violations occur.

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	4	4	4
Medium	7	7	4
Low	6	6	6
Informational	5	5	4
Total	**22**	**22**	**18**

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

certora

Detailed Findings

High-Severity Issues

H-01 Token Denomination Mismatch in VRT Reserve Calculation

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: vault.rs	Category: Logic	Status: Fixed

Description:

The `calculate_vrt_reserve_amount` function in the Vault implementation contains a high vulnerability due to a mismatch in token denominations. This function is called in two specific instructions: `process_initialize_vault_update_state_tracker` and `process_add_delegation`. The function calculates a reserve amount in terms of the supported token (ST) but then incorrectly uses this ST amount in a function expecting VRT amounts, leading to severe miscalculations.

Impact:

- Incorrect calculation of assets needed for withdrawals during vault state updates.
- Potential over-delegation of assets, leaving insufficient reserves for VRT holders.
- Disruption of the vault's token economics, affecting the VRT/ST exchange rate.
- VRT holders may be unable to withdraw their full entitled amount.

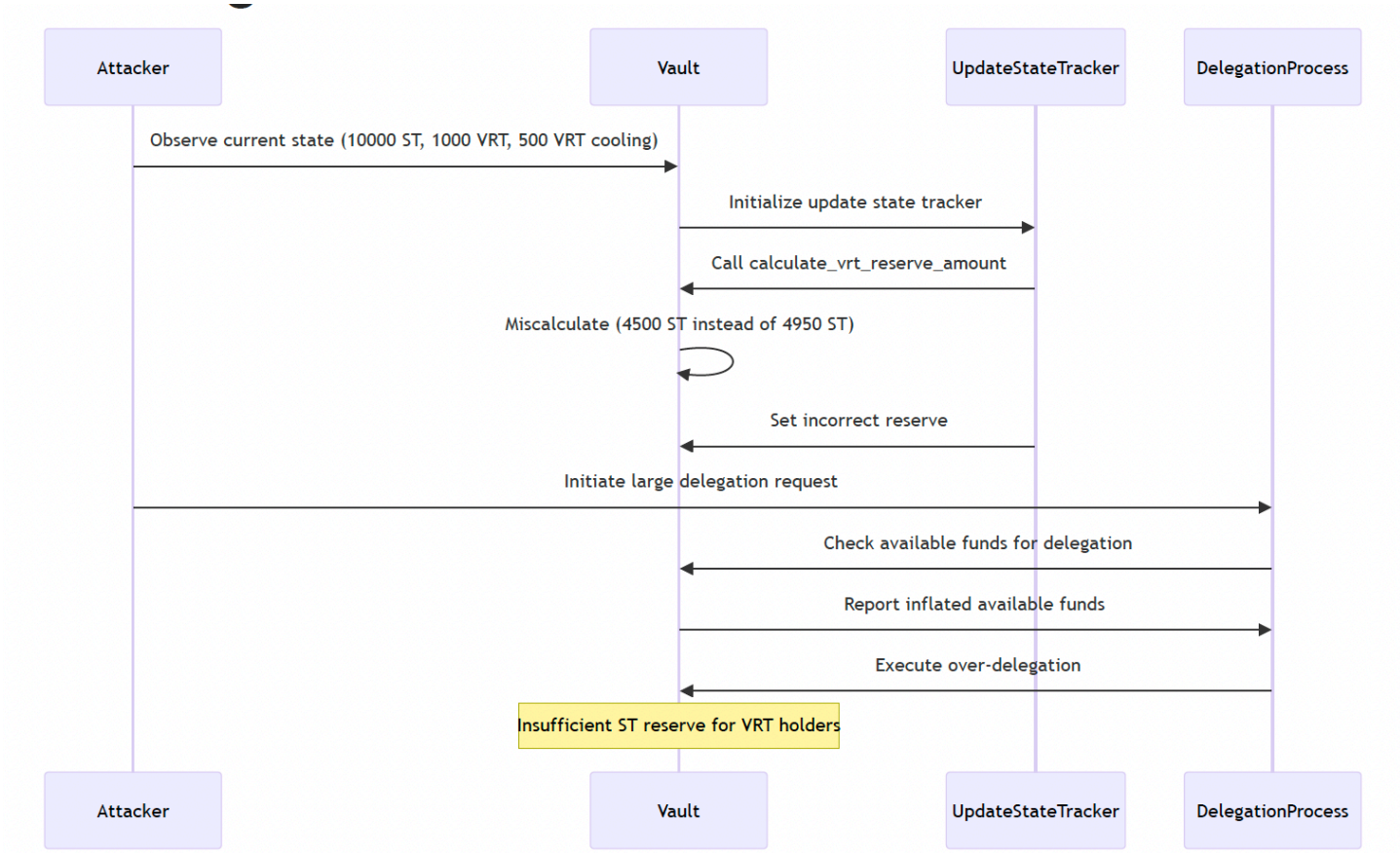
- The overall stability and trustworthiness of the vault system is compromised.

Attack Surface Analysis: Exploiting VRT Reserve Miscalculation in Delegation Process

An attacker discovers the vulnerability in the `calculate_vrt_reserve_amount` function, realizes that this miscalculation can be exploited to manipulate the vault's reserve calculations and potentially extract more value than should be possible.

Steps:

1. The attacker observes the current state of the vault:
    - 10,000 ST and 1,000 VRT in the vault (1 VRT = 10 ST)
    - 500 VRT are in the cooling down state
2. The attacker waits for the vault to initialize its update state tracker, which calls `calculate_vrt_reserve_amount`.
3. Due to the miscalculation:
    - The vault incorrectly determines it needs to reserve 4500 ST instead of 4950 ST
    - This leaves an extra 450 ST available for delegation
4. The attacker quickly initiates a large delegation request, taking advantage of the underestimated reserve.
5. The `process_add_delegation` instruction uses the miscalculated reserve amount, allowing more delegation than should be possible.
6. As a result, the vault becomes over-delegated, leaving insufficient ST to cover the VRT holders' potential withdrawals.



Recommendation:

Modify the `calculate_vrt_reserve_amount` function to consistently use VRT denominations

Customer's response: Fixed.

Fix review: The vulnerability appears properly remediated.

H-02 Delayed Vault Updates do not Properly Account for Multiple Epochs

Severity: **High**	Impact: **High**	Likelihood: **Low**

Files: vault.rs, close_update_state_tracker.rs	Category: State Management	Status: Fixed

Description:

The vault update process in the protocol contains a vulnerability where it fails to properly account for multiple epochs passing between updates. Specifically, when closing the `VaultUpdateStateTracker`, the system only moves `VRT`s one step in the cooling down process, regardless of how many epochs have actually passed. This leads to a significant delay in VRTs becoming available for claiming, potentially locking funds for longer than intended.

Impact:

- VRT holders experience extended lock-up periods beyond the expected cooldown duration.
- Reduced liquidity in the system as VRTs remain in the cooling down state longer than necessary.
- Potential economic imbalance in the protocol as the rate of VRTs becoming claimable is artificially slowed.

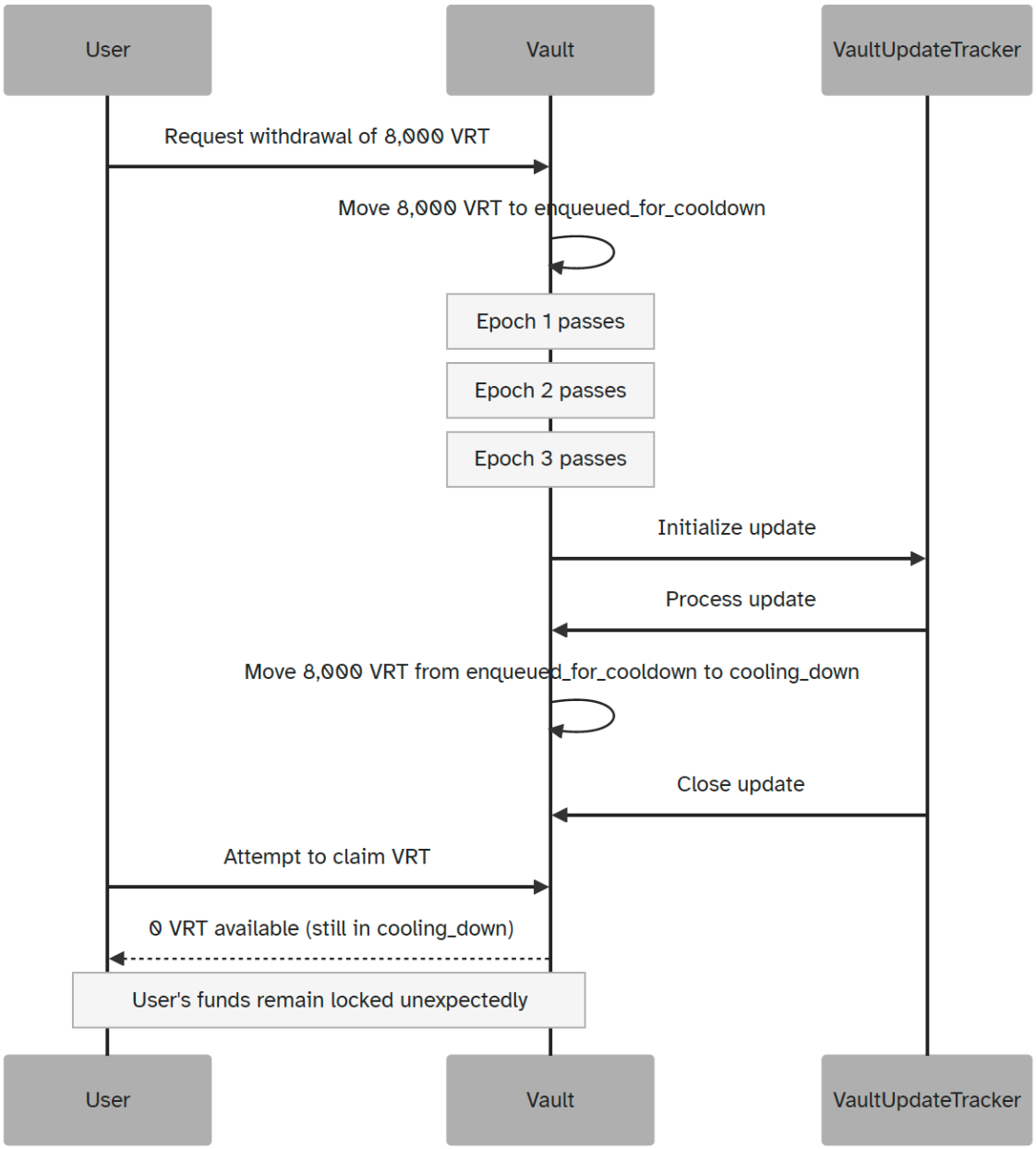
Attack Surface Analysis: Delayed Withdrawals Due to Improper Epoch Accounting

A user could be harmed by this vulnerability by expecting their funds but not getting them. This would lead to decreased integrity of the protocol and loss of trust in it.

Steps:

1. The user observes the current state of the vault:
    - 10,000 ST and 10,000 VRT in the vault (1 VRT = 1 ST)
    - No VRTs are currently in the cooldown process
2. The user initiates a large withdrawal request for 8,000 VRT, which enters the enqueued_for_cooldown state.

3. After 3 epochs, a vault update is finally processed.
4. Due to the bug, instead of all 8,000 VRT moving to the ready_to_claim state:
    - 0 VRT remain in enqueued_for_cooldown
    - 8,000 VRT move to cooling_down
    - 0 VRT move to ready_to_claim
5. The user got affected from this delay
    - Users expecting their funds to be available but will be unable to withdraw
    - VRT liquidity will differ from expected amounts



Recommendation:

- Modify the vault update process, specifically in the `close_update_state_tracker.rs` file, to account for the number of epochs that have passed since the last update. Instead of moving VRTs one step at a time, implement a mechanism to move them through multiple stages based on the elapsed time.
- Adding state tracking struct inside `VaultUpdateStateTracker.`

- Batch cranking operators instead of using multiple transactions could be used to process more without crossing the transaction size limits. This approach also helps if validators do not process in order, which might be an issue since `Init Crank(s) Close` has to be called in order, thus delaying the update.

Customer's response: Fixed.

Fix review: The vulnerability appears properly remediated.

H-03 Fee Collection Inaccuracies and Inconsistencies in Vault Reward System

Severity: **High**	Impact: **Medium**	Likelihood: **Medium**
Files: vault.rs, update_vault_balance.rs	Category: Math Error, Economic Model	Status: Fixed

Description:
The vault's fee collection system exhibits significant inconsistencies, depending on the total vault balance. These issues could lead to unexpected behavior, potential exploitation, and unfair fee distribution.

Key findings:

1. For very small rewards (0.000000001 to 0.00000001 tokens):

- Multiple small rewards result in no fees collected.
- Single large rewards always collect 1 base unit of fees, resulting in an effective fee rate much higher than intended (ranging from 100% to 10%).
- The fee collection normalizes to the expected 10% (1000 tokens out of 10000) for rewards of 10 tokens.
- ***But these values are not realistically replicable***

2. For moderate to large rewards (0.9 to 100,000 tokens):

- Multiple small rewards consistently result in slightly more fees collected compared to a single large reward.
- The discrepancy increases as the reward size grows, becoming substantial for very large rewards.

3. The fee percentage for both methods decreases as the reward size increases, deviating from the intended fee amount.

Key results:

1. For very small rewards (0.000000001 tokens):
   Multiple small rewards: 0 fees collected (0% effective rate)
   Single large reward: 1 fee unit collected (100% effective rate)
   See FV property: P-07 Assess Fees of update_vault_balance()

2. For moderate rewards (1 token):
   Multiple small rewards: 100,000,000 fee units collected
   Single large reward: 99,999,748 fee units collected (9.999975% effective rate)
   Difference: -0.000252%

3. For large rewards (10,000 tokens):
   Multiple small rewards: 995,532,755,284 fee units collected (9.955328% effective rate)
   Single large reward: 991,084,685,897 fee units collected (9.910847% effective rate)
   Difference: -0.446803%

4. For very large rewards (100,000 tokens):
   Multiple small rewards: 9,415,543,701,943 fee units collected (9.415544% effective rate)
   Single large reward: 9,030,376,438,778 fee units collected (9.030376% effective rate)
   Difference: -4.090760%

Impact:

1. **Inconsistent fee collection based on reward size and frequency, leading to potential unfairness in the system.**
2. **Disproportionate fee collection for very small rewards, potentially discouraging microtransactions.**
3. **Reduced protocol revenue from large single rewards, which could be significant for high-value transactions.**
4. **Misalignment with the protocol's intended economic incentives, potentially affecting its long-term sustainability and fairness.**

Attack Surface Analysis: Who's Affected and How

1. Large Depositors:
   - Users making large deposits (100 tokens or more) benefit from paying less in fees than intended when making single large deposits.
   - For a 100,000 token deposit, they would pay 9.030376% in fees instead of the intended 10%, saving approximately 970 tokens.
   - This creates an unfair advantage for large depositors who can make infrequent, large deposits.

2. Protocol Revenue:
   - The protocol loses revenue on large, infrequent deposits.
   - For a 100,000 token deposit, the protocol receives 9,030,376,438,778 fee units instead of the expected 10,000,000,000,000, a loss of about 9.7% of potential revenue.
   - However, the protocol gains extra revenue from very small and moderate-sized deposits.

3. Protocol Stakeholders:

- Inconsistent fee collection affects the predictability of protocol revenue.
- This could impact the stability of rewards or benefits provided to stakeholders.

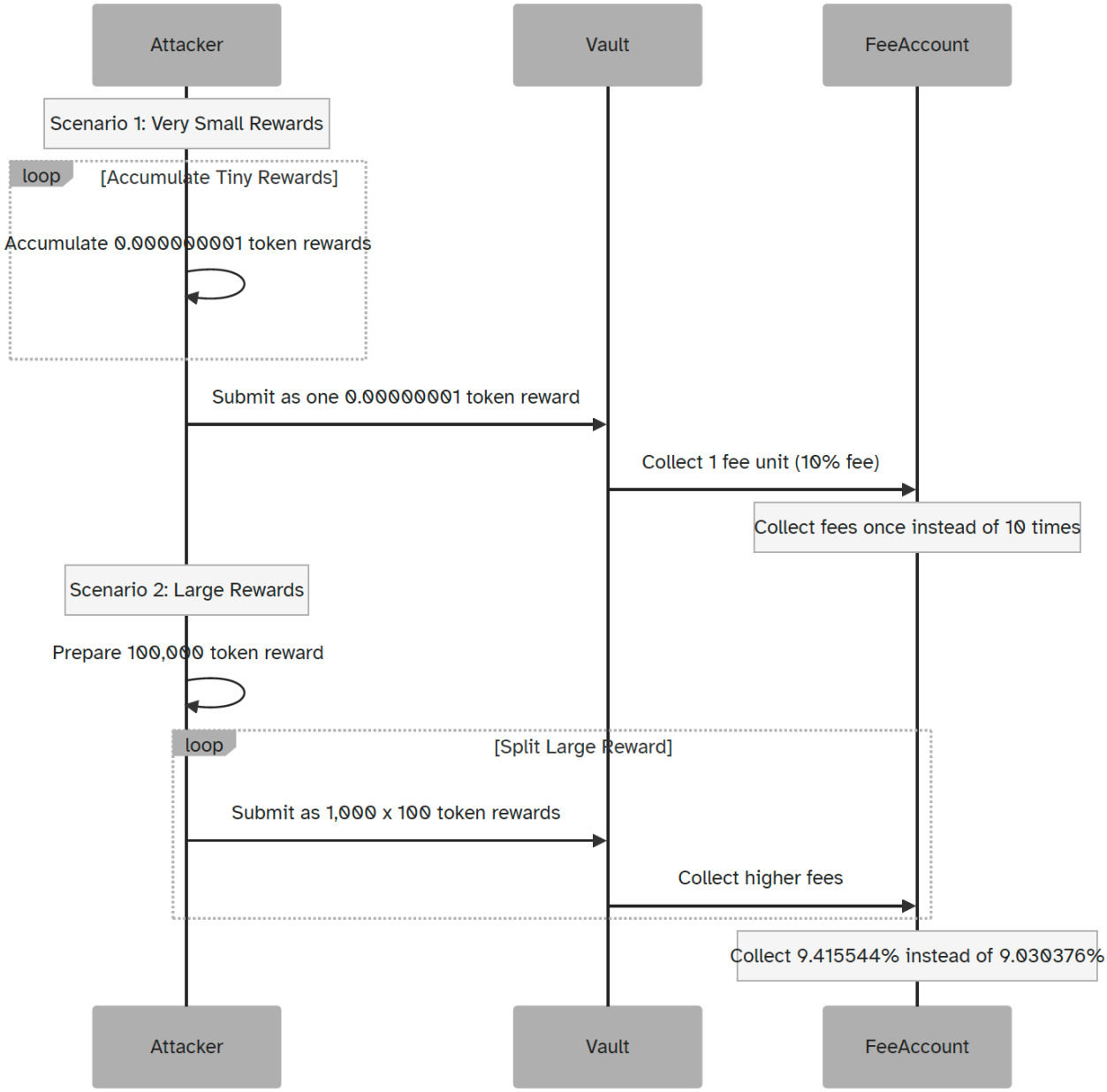
4. Vault Operators:
- The varying fee percentages based on deposit size make it difficult to accurately predict vault growth and revenue.

5. Potential Attackers:
- Sophisticated users could exploit this discrepancy by strategically sizing their deposits.
- They could split very large deposits into multiple smaller ones to minimize fees, or bundle many micro-transactions into a single larger one to avoid the high fees on tiny deposits.

Attack Surface Analysis: Exploiting Fee Collection Discrepancy

An attacker could exploit this vulnerability by manipulating the timing and size of rewards to maximize personal benefit and minimize fees paid to the protocol. Here's how this could play out:

1. The attacker observes that providing frequent smaller rewards results in lower overall fees.
2. For regular operations or smaller amounts, the attacker frequently calls the update function with small rewards.
3. When larger rewards are necessary, the attacker accumulates them off-chain and provides them as a single large reward.
4. This strategy results in the attacker paying lower fees overall, while the protocol receives less revenue than intended.



Recommendation:

1. Implement safeguards against rapid successive small rewards to prevent exploitation.
2. Update documentation to clearly explain the fee collection behavior once it's corrected, including any limitations or edge cases.
3. Add extensive unit tests covering a wide range of realistic reward sizes (e.g., 1 token to 1,000,000 tokens).

4. Consider redesigning the fee calculation mechanism to ensure consistent fee collection regardless of reward size or frequency. Or include a sliding scale fee structure that maintains fairness across different reward sizes.
5. Consider implementing a "fee correction" mechanism that accumulates and applies small fee discrepancies over time.
6. Consider conducting an economic impact assessment to understand how these inaccuracies might have affected the protocol's tokenomics and plan for any necessary adjustments post-fix.
7. Consider implementing a minimum fee threshold to ensure that even very small rewards contribute to fee collection.

Customer's response: Fixed.

Fix review: The vulnerability appears properly remediated.

H-04 Premature State Transition in SlotToggle Deactivation Process

Severity: **High**	Impact: **High**	Likelihood: **High**
Files: **slot_toggle.rs**	Category: Logical Error	Status: Fixed

Description:
The `state` method in the `SlotToggle` implementation contains a high severity vulnerability due to a premature state transition during the deactivation process. In the `Ordering::Less` case (which handles deactivation), the function allows the state to transition from Cooldown to Inactive one epoch earlier than intended. This violates the required mechanism where if a feature is ended in epoch m, the next state should only be achieved at the start of epoch m+2. This issue was uncovered by rule P-02 SlotToggle State cannot Transit from Cooldown to Inactive before an Epoch has Elapsed

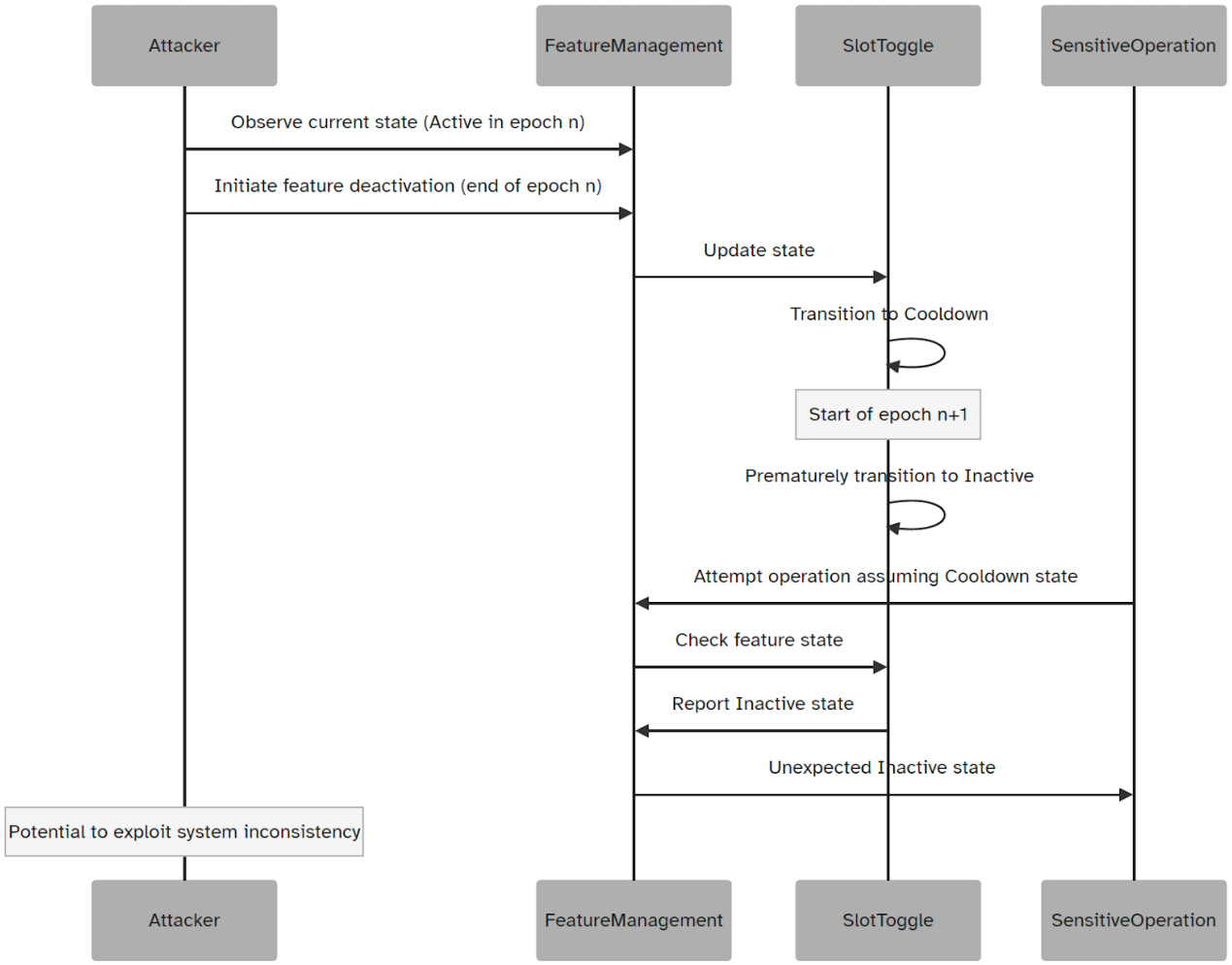
Impact:

- Allows features to transition to Inactive state one epoch earlier than intended.
- Disrupts the intended symmetry between activation and deactivation processes.
- May allow malicious actors to exploit the timing of feature activations and deactivations.
- Compromises the overall security and integrity of any system relying on SlotToggle for feature management.
- Potential for unexpected behavior in systems that depend on precise timing of feature state transitions.

Attack Surface Analysis: Exploiting Premature Deactivation in Feature Management
An attacker identifies the vulnerability in the `state` method of `SlotToggle`, realizing that this premature state transition can be exploited to manipulate the timing of feature activations and deactivations.

Steps:

1. The attacker observes a critical feature managed by SlotToggle:
    - Feature is currently active in epoch n
    - A sensitive operation is scheduled for epoch n+1, assuming the feature will still be in Cooldown
2. The attacker initiates a deactivation of the feature near the end of epoch n.
3. Due to the miscalculation in the `state` method:
    - The feature enters Cooldown state immediately
    - It transitions to Inactive state at the start of epoch n+1 instead of n+2
4. The sensitive operation occurs in epoch n+1, but the feature is already in Inactive state.
5. This unexpected state leads to system inconsistencies or security vulnerabilities.



Recommendation:
Modify the `state` method in the `SlotToggle` implementation, specifically in the `Ordering::Less` case, to ensure that the transition from Cooldown to Inactive only occurs after a full epoch has passed since deactivation.

Customer's response: Fixed.

Fix review: The vulnerability appears properly remediated.

Medium-Severity Issues

M-01 Fee Calculation is not in Favor of the Protocol

Severity: **Medium**	Impact: **Low**	Likelihood: **High**
Files: vault.rs	Category: Arithmetic	Status: Fixed

Description: calculate_deposit_fee() and calculate_withdraw_fee() round down the fee paid to the protocol. Violation of the following rules detected this issue: Additivity of calculate_deposit_fee() and P-02 Non-Zero Fee for Non-Zero Amount.

Impact: The protocol may gain less fees if a user splits the deposit. For example,

if `vault.withdrawal_fee_bps = 1` and `lrt_amount = 10_000` then the calculated fee is `1`

`calculate_deposit_fee(10_999) = 1`.

But if the deposit is split to `1` and `9_999` the total fee is zero.

`calculate_deposit_fee(1) = 0`

`calculate_deposit_fee(9_999) = 0`

Recommendation: Use ceiling division rather than floor division (checked_div).

Customer's response: Fixed.

Fix review: The issue is fixed - both functions now use div_ceil for rounding up instead of floor division.

M-02 No Upper Bounds in Vault Capacity

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: vault.rs	Category: Logic, economy	Status: Confirmed

Description: In the Vault module (vault_core/src/vault.rs), the Vault::capacity is currently treated as a non-strict upper bound. This implementation allows the amount of tokens in a vault to exceed its defined capacity, which could result in unexpected behavior and potentially compromise the integrity of fund management. The vault's capacity should act as a strict limit to prevent any overflow or mismanagement of tokens. Allowing deposits that exceed the defined capacity could lead to inconsistent states.

Impact: The current handling of the capacity parameter in the vault does not enforce a strict limit on the amount of tokens that can be deposited. This can lead to scenarios where the vault holds more tokens than its designated capacity, potentially causing issues in fund management.

Recommendation: To address this issue is it recommended to modify the set_capacity function to ensure the new capacity is never set lower than the current amount of tokens deposited in the vault. This will enforce the upper bound and prevent any overflow issues. The function can be updated as follows:

```C/C++
pub fn set_capacity(&mut self, capacity: u64) -> Result<(), VaultError> {
    if capacity < self.tokens_deposited() {
        return Err(VaultError::InvalidCapacity);
    }
    self.capacity = PodU64::from(capacity);
    Ok(())
}
```

Customer's response: Confirmed, will update documentation to more accurately reflect purpose of deposit capacity.

M-03 Inconsistent and Unchecked Basis Points (BPS) Values

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: vault.rs	Category: Arithmetic	Status: Fixed

Description: The current codebase does not consistently verify that basis points (bps) values are within the valid range of 0 to 10,000. Basis points, which are used to represent percentage values with a finer granularity (e.g., 1% = 100 bps), should always be within this range to ensure correct calculations. However, there are instances where this check is either not implemented or is inconsistent, leading to potential calculation errors. Furthermore, the codebase contains hardcoded values of 10,000 for the maximum BPS, rather than using a defined constant, which can lead to errors and reduce code maintainability.

Impact: The lack of consistent range checks for bps values introduces the risk of invalid or out-of-bounds values being used in calculations, which can lead to incorrect fee assessments, rewards, or other critical financial computations within the vault system. Hardcoded values for maximum BPS reduce the flexibility and readability of the code, making future adjustments more error-prone and less efficient.

Recommendation: It is recommended to implement the following enhancements to address this issue:

- **Replace Hardcoded Values:** Change all occurrences of the hardcoded maximum BPS value (10,000) to use the MAX_BPS constant from the configuration. This ensures consistency and easier updates in the future.
- **Implement Range Checks:** Add range checks for all bps fields to confirm values are always within 0 to MAX_BPS. This will prevent errors and maintain accurate calculations.

- **Centralize BPS Calculations:** Create a utility function for all bps calculations using MAX_BPS. This will ensure uniformity and simplify the codebase.

Customer's response: Fixed.

Fix review: The fix is properly implemented.

M-04 Inconsistent Zero-Value Handling

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: vault.rs	Category: Arithmetic	Status: Fixed

Description: There are inconsistencies in how zero values are managed across various functions in the vault codebase. Specifically, the functions burn_with_fee and mint_with_fee handle zero inputs differently. burn_with_fee explicitly checks for zero input and returns an error, while mint_with_fee does not have such a check. This inconsistency can lead to unexpected behavior or potential exploitation by allowing zero-value transactions where they might not be intended.

Impact: Inconsistent handling of zero values can lead to errors, unexpected behavior, or security vulnerabilities, especially in financial calculations where zero-value transactions might be improperly processed.

Recommendation: It is recommended to implement the following enchantments to address this issue:

- **Add Zero-Value Check to mint_with_fee:** Ensure that mint_with_fee includes a check for zero inputs similar to burn_with_fee to prevent unintended behavior and maintain consistency.

- **Modify calculate_rewards_fee:** Adjust this function to enforce a minimum fee or explicitly prevent zero fees, ensuring that all transactions contribute a meaningful amount and reducing the potential for fee avoidance.
- **Review Codebase for Zero-Value Handling:** Conduct a thorough review of all functions that deal with monetary values to ensure consistent handling of zero values throughout the codebase. This will help maintain code reliability and prevent discrepancies in financial logic.

Customer's response: Fixed.

Fix review: The vulnerability appears fixed, both mint_with_fee and burn_with_fee now handle zero values consistently.

M-05 Overflow Risk in 64-bit Calculations

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: vault.rs	Category: Arithmetic	Status: Fixed

Description: Several functions in the vault code use 64-bit integers for calculations, which can cause overflows when dealing with large numbers, particularly during multiplication before division. These overflows can result in incorrect calculations and introduce vulnerabilities into the vault's financial operations. Functions at risk include calculate_assets_returned_amount, calculate_vrt_mint_amount, calculate_deposit_fee, and calculate_withdraw_fee in vault_core/src/vault.rs.

Impact: Overflow errors in financial calculations could lead to inaccurate fund management, financial loss, and potential security vulnerabilities.

Recommendation: To prevent overflows, it is recommended to use a larger integer type, such as u128, for intermediate calculations. This will ensure that operations involving large numbers are accurately handled, preserving the integrity of the vault's calculations. Implement this change across all similar functions to avoid overflow-related issues.

Customer's response: Fixed.

Fix review: The vulnerability appears properly remediated.

M-06 Inconsistent Slasher States Between Restaking and Vault Programs

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: vault_ncn_slasher_ticket.rs , ncn_vault_slasher_ticket.rs	Category: State Management	Status: Confirmed

Description:

The Vault program's slashing mechanism, controlled by the VaultNcnSlasherTicket, operates independently of the NcnVaultSlasherTicket in the Restaking program after initialization. While this design allows for continued functionality in case of issues with the Restaking program, it also introduces a potential for state inconsistency between the two programs.

Specifically:

1. The VaultNcnSlasher can continue to function without an active NcnVaultSlasher.
2. Changes to the NcnVaultSlasherTicket, including deactivation, do not propagate to the VaultNcnSlasherTicket.

Impact:

- Potential for conflicting states between Restaking and Vault programs.
- Risk of unexpected slashing behavior if NcnVaultSlasherTicket is deactivated but VaultNcnSlasherTicket remains active.

Recommendation:

1. Implement a synchronization mechanism between NcnVaultSlasherTicket and VaultNcnSlasherTicket to ensure consistent states.
2. Add functionality to propagate critical state changes (like deactivation) from the Restaking program to the Vault program.
3. Consider adding a "master switch" that can disable slashing across both programs simultaneously.
4. Improve documentation to clearly explain the independent nature of these tickets and potential implications.

By addressing this issue, the protocol can ensure more consistent and predictable behavior in its slashing mechanism across both the Restaking and Vault programs.

Customer's response: Confirmed, will not be fixed.

M-07 Vault Capacity Limit not Enforced for Reward Minting

Severity: **Medium**	Impact: **Medium**	Likelihood: **Low**
Files: update_vault_balance.rs , vault.rs	Category: Logical Error	Status: Confirmed

Description:

The vault's capacity limit is not being enforced when processing rewards in `UpdateVaultBalance`. It is possible that vault TKN capacity is increased beyond `capacity` and

that new VRT is minted. This allows the total number of supported tokens to potentially exceed the set capacity of the vault, contradicting the expected behavior of the `capacity` field. This issue was uncovered by P-14 Capacity Limit is Respected by update_vault_balance()

Impact:

This issue could lead to an unexpected increase in the total supply of VRT tokens and supported token beyond the intended capacity of the vault. Consequences may include:

1. Violation of the vault's designed economic model

Recommendation:

1. Implement capacity checks in both the `UpdateVaultBalance` function and the `increment_vrt_supply` method to ensure the VRT supply never increases if the capacity limit is in place.
2. Create a policy for handling rewards when the vault is at or near capacity. This could involve not minting rewards, minting only up to the capacity, or implementing a queue system for pending rewards.
3. Add clear documentation explaining how the vault handles rewards in relation to its capacity.
4. Consider implementing a dynamic capacity adjustment mechanism that can be controlled by authorized administrators to handle exceptional circumstances.

Customer's response: Confirmed, will update documentation to more accurately reflect the purpose of the deposit capacity.

Low-Severity Issues

L-01 Alignment Issues in Structures used with Bytemuck

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: restaking_core/src/config.rs		Status: Fixed

Description: The structures used in `restaking_core/src/config.rs` exhibit alignment issues when utilized with the `Bytemuck` library. Specifically, the original `Config` structure uses standard Rust types, such as `u64`, which can cause alignment problems when these types are not properly aligned in memory. This misalignment may lead to undefined behavior or performance degradation when the structure is accessed.

Impact:
The potential misalignment of types can lead to undefined behavior in the program, especially when these structures are deserialized or processed in ways that assume correct memory alignment. The impact is minimized because the issue would typically manifest only under specific conditions, but it could lead to bugs that are difficult to trace and debug.

Recommendation: To address this issue, it is recommended to use Pod types provided by the SPL Pod library, which ensures correct alignment and memory safety when used with Bytemuck.

Customer's response: Fixed

Fix review: The alignment issue is fixed by using Pod types, vulnerability is fixed.

L-02 Unsafe Handling of Mint in WithdrawAssets Instruction

Severity: **Low**	Impact: **Low**	Likelihood: **Medium**
Files: restaking_program/src/lib.rs		Status: Fixed

Description: The current implementation of the `WithdrawAssets` instructions accepts the `mint` as a `Pubkey` passed in the instruction data, rather than as an account within the accounts vector. This method introduces a risk where the `mint` might not exist or could be manipulated, leading to potential errors or unintended behavior during asset withdrawals

Impact: By passing the mint as a Pubkey in the instruction data, the program does not verify that the mint account exists or is valid. This could lead to situations where withdrawals are processed using an invalid or non-existent mint, potentially causing loss of funds or other issues.

For example, in the current implementation:

```C/C++
RestakingInstruction::NcnWithdrawalAsset { token_mint, amount } => {
    msg!("Instruction: NcnWithdrawalAsset");
    process_ncn_withdraw_asset(program_id, accounts, token_mint, amount)
}
RestakingInstruction::OperatorWithdrawalAsset { token_mint, amount } => {
    msg!("Instruction: OperatorWithdrawalAsset");
    process_operator_withdrawal_asset(program_id, accounts, token_mint,
amount)
}
```

Recommendation: To address this issue, the mint can be passed as an account within the accounts vector, rather than as a Pubkey in the instruction data. This account should then be validated to ensure that it is a valid and existing mint account before processing the withdrawal.

Here's an example of how the code should be modified:

```C/C++
RestakingInstruction::NcnWithdrawalAsset { amount } => {
    msg!("Instruction: NcnWithdrawalAsset");
    let mint_account = next_account_info(account_info_iter)?;
    process_ncn_withdraw_asset(program_id, accounts, mint_account, amount)
}
RestakingInstruction::OperatorWithdrawalAsset { amount } => {
    msg!("Instruction: OperatorWithdrawalAsset");
    let mint_account = next_account_info(account_info_iter)?;
    process_operator_withdrawal_asset(program_id, accounts, mint_account,
amount)
}
```

Customer's response: Fixed.

Fix review: The fix is properly implemented.

L-03 Faulty VRT Mint Parameters in Vault Initialization

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: initialize_vault.rs		Status: Fixed

Description: The InitializeVault instruction in the current implementation has two significant issues affecting its flexibility and usability:

1. It incorrectly requires the vrt_mint to be both a signer and an empty account, which is bad design and also results in initialization failures.
2. It hardcodes the number of decimals for the VRT mint to 9. This fixed value may not be suitable for all use cases, limiting the adaptability and versatility of the vault system.

Impact: These issues can cause initialization errors and reduce the vault's flexibility, making it less suitable for varying user needs and scenarios.

Recommendation: To address this issue, the following enhancements are recommended:

- **Remove the Signer Requirement:** Adjust the InitializeVault instruction to remove the requirement for vrt_mint to be both a signer and an empty account. This change will prevent initialization errors and align the function's requirements with realistic scenarios.
- **Parameterize Decimal Setting:** Modify the instruction to accept the number of decimals as a parameter instead of hardcoding it. This will enhance the vault system's adaptability, allowing for a wider range of use cases and improving its overall versatility.

Customer's Response: Fixed decimals, ignoring vrt_mint signer.

Fix review: Initialize mint has decimals as a parameter now, issue has been fixed.

L-04 Incomplete Admin Reset in SetAdmin Function

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: set_admin.rs		Status: Fixed

Description: The current implementation of the SetAdmin function only updates the primary admin value but does not account for secondary admin values that might still contain the old admin's key.

Impact: By only resetting the primary admin and leaving secondary admin fields unchanged, the system may inadvertently allow the old admin to retain certain privileges through secondary admin roles. This creates a risk where the old admin, who should no longer have access, still has control over certain aspects of the program, leading to potential misuse or unauthorized actions.

Recommendation: To address this issue, it is recommended to update The SetAdmin function to check all secondary admin fields within the program's configuration. If any of these fields contain the old admin's key, they should also be updated to reflect the new admin's key. This ensures that all traces of the old admin are removed, maintaining consistency and security within the system.

Customer's Response: Fixed

Fix review: The issue appears remediated.

L-05 Unrecoverable Error Handling Across the Code

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: vault_sdk/src/sdk.rs	Category: Error Handling	Status: Fixed

Description: Several functions throughout the codebase use unwrap() to handle errors, rather than employing more robust error handling mechanisms such as the ? operator or match statements. Using unwrap() can cause the program to panic unexpectedly when encountering an error, leading to poor error messages and potential security vulnerabilities.

Impact: Using unwrap() leads to unexpected panics and unhelpful error messages, making debugging difficult and introducing potential vulnerabilities in the code.

Recommendation: To address this issue, it is recommended to replace all instances of unwrap() with proper error handling techniques like the ? operator or match statements. This will improve error reporting, prevent unexpected crashes, and enhance the overall stability and security of the codebase.

Customer's Response: Fixed

Fix review: The issue appears remediated.

L-06 Unnecessary call in MintTo Instruction

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: mint_to.rs	Category: Performance	Status: Fixed

Description: The MintTo instruction in the vault program unnecessarily calls `Vault::find_program_address` to derive the vault's PDA, even though this information is already available through `Vault::seeds`. This redundant computation adds unnecessary overhead to the instruction's execution.

Impact: The extra call to Vault::find_program_address slightly degrades the performance of the MintTo instruction.

Recommendation:

1. Refactor the MintTo instruction to use `Vault::seeds` instead of `Vault::find_program_address` when deriving the vault's PDA.
2. Review other instructions and functions in the vault program to identify and remove similar instances of redundant PDA derivation.

Customer's Response: Fixed.

Fix review: The issue appears remediated.

Informational-Severity Issues

I-01 Excessive Lamports and Tokens not Properly Harvested

Severity: **Informational**	Impact: **Low**	Likelihood: **Low**
Files:		Status: Confirmed

Description: The current implementation of the program does not include an instruction to harvest excessive lamports or tokens from an account. This could lead to situations where funds are left unutilized or locked in the account without the possibility of retrieval, leading to inefficiency in fund management.

Impact: Excess lamports or tokens that are not harvested could accumulate over time, resulting in unnecessary capital inefficiency. In a worst-case scenario, it could prevent the account from being effectively used for its intended purpose, especially in scenarios where precise fund management is critical.

For example, consider the following code snippet for a potential harvesting mechanism:

```C/C++
pub fn harvest_excess<'info>(
    from: &AccountInfo<'info>,
    to: &AccountInfo<'info>,
    token_from: Option<&Account<'info, TokenAccount>>,
    token_to: Option<&Account<'info, TokenAccount>>,
    token_program: Option<&Program<'info, Token>>,
) -> Result<()> {
    // Harvest excess lamports
    let from_lamports = **from.lamports.borrow();
    let to_lamports = **to.lamports.borrow();
    let rent = Rent::get()?;
    let min_rent = rent.minimum_balance(from.data_len());

    if from_lamports > min_rent {
        let lamports_to_transfer = from_lamports - min_rent;
        **from.lamports.borrow_mut() -= lamports_to_transfer;
        **to.lamports.borrow_mut() += lamports_to_transfer;
    }

    // Harvest excess tokens if token accounts are provided
```

```
    if let (Some(token_from), Some(token_to), Some(token_program)) =
(token_from, token_to, token_program) {
        let token_balance = token_from.amount;
        if token_balance > 0 {
            anchor_spl::token::transfer(
                CpiContext::new(
                    token_program.to_account_info(),
                    anchor_spl::token::Transfer {
                        from: token_from.to_account_info(),
                        to: token_to.to_account_info(),
                        authority: from.clone(),
                    },
                ),
                token_balance,
            )?;
        }
    }
```

This function demonstrates how lamports and tokens can be harvested from an account. The function transfers excess lamports above the required rent to another account.

Recommendation: To address this issue, It is recommended to implement an instruction in the program to harvest excessive lamports and tokens from accounts. This can be useful in scenarios where accounts may accumulate more funds than needed for operation.

Customer's response: Confirmed, will be fixed in future protocol upgrade.

I-02 Simplification of WithdrawAssets Instruction Implementation

Severity: **Informational**	Impact: **Low**	Likelihood: **Low**
Files:		Status: Fixed

Description: The current implementation of the WithdrawAssets instructions could be improved by utilizing token delegation (the token::Approve instruction). This approach would not only streamline the process but also enhance the program's compatibility with both the standard token program and token-2022, which might require transfer_checked instead of transfer for certain mints.

Recommendation: To address this issue, it is recommended to implement the WithdrawAssets instructions using the token delegation approach. This method will simplify the support for both token and token-2022 programs, particularly when handling transfer_checked requirements. Additionally, consider the feasibility of a multi-delegation setup to provide further flexibility within the program.

Customer's Response: Fixed.

Fix review: This issue is irrelevant in v0.0.3 since there is no WithdrawAssets instruction.

I-03 Redundant Accounts and Inconsistent Coding Patterns

Severity: **Informational**	Impact: **Low**	Likelihood: **Low**
Files:		Status: Confirmed

Description: The current implementation in the restaking_core module includes two nearly identical accounts: NcnOperatorTicket and OperatorNcnTicket. Both accounts share the same fields and serve similar purposes, leading to redundancy in the codebase. Additionally, the seeds() function for these accounts uses different coding patterns, adding unnecessary complexity and increasing the maintenance burden.

Impact: Maintaining separate but identical accounts (NcnOperatorTicket and OperatorNcnTicket) increases the complexity of the codebase and can lead to potential bugs due to inconsistent updates or logic applied to these accounts. Moreover, the different coding patterns used in the seeds() function across these accounts further complicates the code, making it harder to maintain and understand.

Recommendation: Refactor the NcnOperatorTicket and OperatorNcnTicket into a single NcnOperatorTicketCombined structure. This will reduce the number of accounts and simplify the overall design, making the program easier to maintain and less prone to errors. Additionally, standardize the seeds() function across all Ticket accounts to use a consistent coding pattern, further improving code clarity and maintainability.

Customer's Response: Confirmed, will be fixed

I-04 Non-Specific Error Handling

Severity: **Informational**	Impact: **Low**	Likelihood: **Low**
Files: vault_sdk/src/error.rs		Status: Confirmed

Description: The current error handling approach in the vault codebase uses a generic VaultOverflow error for multiple error conditions. This lack of specificity can complicate debugging and make it harder to pinpoint the exact cause of errors, reducing the effectiveness of error resolution and troubleshooting processes.

Impact: Using a single generic error type hinders precise error identification and handling, potentially slowing down the debugging process and reducing code clarity.

Recommendation: To address this issue, it is recommended to introduce more specific error types in vault_core/src/error.rs to better categorize and identify errors. For instance, define separate error types such as Overflow, DivisionByZero, and InsufficientFunds. Implement these specific error types across functions to provide more detailed and actionable error messages, enhancing debugging and error management

Customer's Response: Confirmed, will be fixed

I-05 Missing Self-Deposit Check in Vault Mint Process

Severity: **Informational**	Impact: **None**	Likelihood: **None**
Files: mint_to.rs	Category: Code Quality	Status: Confirmed

Description:

The `mint_to` function in `mint_to.rs` contains a code path that handles a scenario where the depositor and the vault have the same public key. This code updates the vault's internal accounting without performing an actual token transfer when such a condition is met. While this scenario is highly unlikely in the current implementation due to the vault being a Program Derived Address (PDA), the presence of this code introduces unnecessary complexity and potential confusion. This issue was uncovered by rule: P-02 Staker Integrity of mint().

Recommendation:

1. Add a check at the beginning of all functions that transfer tokens from vault token account to another token account (eg. `process_mint`) to ensure the depositor and vault accounts are different:

```
Unset
if depositor.key == vault_info.key {
       return Err(VaultError::InvalidDepositor.into());
}
```

2. Consider adding a post-operation check to ensure the vault's recorded state matches the actual token balances.
3. Update the `VaultError` enum to include an `InvalidDepositor` variant:

Customer's response: Confirmed, will be fixed

Formal Verification

Verification Notations

Formally Verified	The rule is verified for every state of the contract(s), under the assumptions of the scope/requirements in the rule.

Formally Verified After Fix	The rule was violated due to an issue in the code and was successfully verified after fixing the issue
Violated	A counter-example exists that violates one of the assertions of the rule.

Core properties of the Vault protocol

To guide FV, we have identified and verified the following core properties. The core properties are classified according to the different stakeholders of the protocol. Different stakeholders will have different notions of *core*.

To simplify the presentation, we use **VRT** to refer to the Vault Reserve Token. This corresponds to `vrt` in the code, and, in the previous commits was known as LRT. We use **TKN** to refer to the token holdings of the vault. This is called `supported_token` in the code.

The core properties are affected by instructions that either produce (i.e., mint) VRT, or transfer TKN into and out of the vault. We classify instructions based on their actions. Some instructions, such as `Burn`, are classified in multiple classes because they have multiple actions.

An instruction is called:

1. **TKN Out** if it transfers TKN out of the vault
    - `Burn, BurnWithdrawTicket, Slash, AdminWithdraw`
2. **VRT Mint** if it mints VRT
    - `MintTo, UpdateVaultBalance`
3. **VRT Burn** if it burns VRT
    - `Burn, BurnWithdrawTicket`
4. **TKN In** if it transfers TKN into the vault
    - `MintTo, UpdateVaultBalance`

Note that at the time of the audit, `AdminWithdraw` was not implemented. Therefore, we have not analyzed it.

We identify the following stakeholders of the protocol:

1. **Staker** – someone who is holding VRT and burns VRT for TKN
2. **Vault** – the vault itself that manages funds and the aggregate state
3. **Slasher** – a slasher that expects to be able to slash an agreed upon operator
4. **Protocol** – the protocol itself that collects fees for its operation

Core properties for the Staker

The staker redeems VRT for TKN. From its perspective, the core properties relate to the integrity of `Mint` and `Burn`, and fair accounting for the value of the VRT. Specifically:

1. The ratio VRT/TKN should never increase, except due to `Slash` instruction. For example, VRT/TNK can change from 1/10 to 1/20 (due to rewards), but should not change from 1/10 to 1/5 (making VRT worth less). This property must be preserved by every instruction that modifies VRT and TKN. We have verified it for the following instructions:
    a. `UpdateVaultBalance` with `rule_update_vault_balance_no_dilution`
    b. `Mint` with `rule_mint_no_dilution`
    c. `Burn` with `rule_burn_no_dilution`
    d. `BurnWithdrawalTicket` with `rule_burn_withdrawal_no_dilution`
2. `Burn` returns the expected amount of TKN adjusted for fees
    a. rule `rule_integrity_burn`
3. `BurnWithdrawalTicket` returns the expected amount of TKN adjusted for fees
    a. Manually inspected that uses same logic as `Burn` that is verified above
4. `EnqueueWithdrawal` schedules expected amount of VRT to burn
5. `MintTo` consumes expected TKN and produces expected VRT adjusted for fees
    a. rule `rule_integrity_mint`

Core properties for Vault

Vault maintains state. The core property for the vault is to ensure that the centralized state correctly reflects the distributed state of all accounts involved

1. `vrt_supply` and `token_deposited` reflect what is in the appropriate token accounts
    a. This is captured by `cvt_vault_inv` invariant that is checked in `UpdateVaultBalance` with rule `rule_integrity_update_vault_balance_1`, in `MintTo` with rule `rule_integrity_mint_2`, in `BurnWithdrawalTicket` with rule `rule_integrity_burn_withdrawal_2`, and for `Burn` with `rule_integrity_mint_2`
2. No new VRT is minted if vault TKN deposit is over capacity
    a. Violated by `UpdateVaultBalance` as shown by rule `rule_capacity_respected_by_update_vault_balance`

Core properties of Slasher

Slasher must be able to slash an agreed upon amount from any staked operator. The core property is that the vault always maintains sufficient supply of TKN to allow slashing:

1. There is always enough TKN in the vault to cover a slash

a. This is implied by ensuring that the vault always has at least a total_security amount of TKN. This is captured in `cvt_vault_inv` and checked in `UpdateVaultBalance`, `MintTo`, `BurnWithdrawalTicket` and `Burn`

2. Any state change between entities can only happen across epochs. This ensures that slash cannot be front run by an instruction that disables an agreement.
    a. We check that `StateToggle` logic is correct using `rule_activate` and `rule_deactivate`
    b. Manually audit that all state-change is done via `StateToggle`

Core properties of the protocol

Protocol collects fees to sustain its operation. Its core property is correct assessment of these fees.

1. If fee is expected, then it is always assessed
    a. `UpdateVaultBalance` uses `reward_fee_bps`
        i. Violated as shown by the rule `rule_update_vault_balance_assess_fees`
    b. `Burn` and `BurnWithdrawalTicket` use `withdrawal_fee_bps`
        i. Manually audit.
    c. `MintTo` uses `deposit_fee_bps`
        i. Verified with rule `rule_mint_assess_fees`

Formal Verification Properties

General Assumptions

- Loop iterations: Any loop was unrolled at most 3 times (iterations)
- We assume that TKN > 0 and VRT > 0.

vault_core/src/vault.rs:

Contract Properties

P-01 Additivity of calculate_deposit_fee()	
Status: Violated	

Rule Name	Status	Description	Links
rule_additivity_of_calculate_deposit_fee	Violated	Fees calculated by both `calculate_deposit_fee()` and `calculate_withdraw_fee()` is additive. That is, `fee(x)+fee(y) >= fee(x+y)`	Rule report Uncovered issue: M-01 Fixed rule report

P-02 Non-Zero Fee for Non-Zero Amount	
Status: Violated	

Rule Name	Status	Description	Links
rule_calculate_fee_non_zero	Violated	If VRT amount is greater than zero then some fee should be paid to the protocol. That is, `if x>0 and withdraw_fee_bps>0 then fee(x) > 0`	Rule report Uncovered issue: M-01 Fixed rule report

vault_program/src/{mint_to, update_vault_balance, burn, burn_withdrawal_ticket}.rs:

Contract Properties

P-01 Staker can not Steal Funds	
Status: Verified	

Rule Name	Status	Description	Link to rule report
rule_staker_cannot_steal_funds	Verified	A staker cannot recover more funds that she put in.	Not applicable in the fix version

P-02 Staker Integrity of mint()	
Status: Violated (at least 1 assertion does not hold)	

Rule Name	Status	Description	Links
rule_integrity_mint	Violated	1. # of vault tokens + # of depositor tokens before mint and after mint is equal. 2. Depositor's token increases by the minted value. 3. VRT supply increases (or remains the same) 4. Vault fee increase (or remains the same) 5. Minted VRT equals to the additional vault fee + the additional depositor VRTs	Rule report Uncovered issue: I-05 Fixed rule report

P-03 Staker Integrity of burn()

Status: Verified

Rule Name	Status	Description	Link to rule report
rule_integrity_burn	Verified	1. # of staker's SPL tokens increases (or stable) after burn() 2. # of staker's VRT tokens decreases (or stable) after burn() 3. # of vault fee tokens increases (or stable) after burn() 4. # of VRT tokens that the staker lost is no more than the burned amount. 5. # of vault'sSPL token increases (or stable) after burn() 6. # of vault's VRT tokens decreases (or stable) after burn() 7. The losses of the vault SPL tokens are the gain of the staker 8. The vault cannot lose more VRTs than the burned amount	Not applicable in the fix version

P-04 Vault Integrity of mint and update_vault_balance

P-05 Vault Integrity of burn and burn_withdrawal_ticket

Status: Verified

Rule Name	Status	Description	Link to rule reports
rule_integrity_update_vault_balance_1 rule_integrity_burn_withdrawal rule_integrity_mint_2 rule_integrity_burn_2	Verified	The following invariants of process_update_vault_balance(), process_mint, process_burn_withdrawal(), and process_burn() hold: 1. Vault's total security <= amount of deposited SPL tokens 2. Vault's SPL amount <= SPL supply of account vrt_mint 3. Vault's VRT amount == VRT supply of account vrt_mint	rule_integrity_update_vault_balance_1 rule_integrity_burn_withdrawal rule_integrity_mint_2 Rule_integrity_burn_2 is no applicable in the fix version

P-06 Integrity of update_vault_balance() (2)

Status: Verified

Rule Name	Status	Description	Link to rule report
rule_integrity_update_vault_balance	Verified	1. VRT supply increases (or stable) following update_vault_balance() 2. Vault's fee increases (or stable) following update_vault_balance()	report

_2		3. Vault fee increases by the same amount that minted vrt tokens following update_vault_balance()	

P-07 Assess Fees of update_vault_balance()

Status: Violated

Rule Name	Status	Description	Links
rule_update_vault_balance_assess_fees	Violated	If there are positive fees and rewards and process_update_vault_balance() does not revert then vault's vrt supply should strictly increase.	Rule report Uncovered issue: H-03 Fee Collection Inaccuracies and Inconsistencies in Vault Reward System Fixed rule report

P-08 Assess Fees of mint()

Status: Verified

Rule Name	Status	Description	Link to rule report
rule_mint_assess	Verified	If VRT supply strictly increases and	report

_fees		process_mint() does not revert then vault's fees strictly increase.	

P-09 No Dilution of update_vault_balance()

P-10 No Dilution of mint_to()

P-11 No Dilution of burn()

P-12 No Dilution of burn_withdrawal_ticket ()

Status: Verified

Rule Name	Status	Description	Link to rule report
rule_update_vault_balance_no_dilution rule_mint_no_dilution rule_burn_no_dilution rule_burn_withdrawal_no_dilution	Verified	minted vrt cannot be greater than the deposited rewards (converted to vrt)	rule_update_vault_balance_no_dilution rule_mint_no_dilution Rule_burn_no_dilution - NA rule_burn_withdrawal_no_dilution

P-13 Capacity Limit is Respected by mint_to()

Status: Verified

Rule Name	Status	Description	Link to rule report
rule_capacity_respected_by_mint_to	Verified	Total VRT supply cannot exceed capacity after mint.	report

P-14 Capacity Limit is Respected by update_vault_balance()

Status: Violated

Rule Name	Status	Description	Links
rule_capacity_respected_by_update_vault_balance	Violated	Total VRT supply cannot exceed capacity after update_vault_balance.	Rule report Uncovered issue: M-07 Vault Capacity Limit not Enforced for Reward Minting The issue is fixed by changing the documentation. token_deposited may exceed deposited_capicity. The failure is not relevant anymore.

core/src/slot_toggle.rs:

P-01 deactivate() Post Condition

P-02 SlotToggle State cannot Transit from Cooldown to Inactive before an Epoch has Elapsed

Status: Violated

Rule Name	Status	Description	Links
rule_deactivate	**Violated**	1. deactivate() post-condition is *cooldown* 2. If the state transitions from *cooldown* to *inactive* then at least one epoch has elapsed	Rule report Uncovered issue: H-04 Premature State Transition in SlotToggle Deactivation Process Fixed rule report

restaking_program/src/cooldown_operator_vault_ticket.rs:

P-01 Post Condition of process_cooldown_operator_vault_ticket()

Status: Verified

Rule Name	Status	Description	Link to rule report
rule_integrity_process_cooldown_operator_vault_ticket	Verified	The state of operator_vault_ticket after calling process_cooldown_operator_vault_ticket() is *inactive* or *cooldown.*	report

restaking_program/src/{config, ncn, operator}.rs:

P-01 Check load operations

Status: Verified

Rule Name	Status	Description	Link to rule report
rule_load_config_and_ncn rule_load_config_twice_witness rule_load_operator_and_ncn	Verified	1. A config account cannot be loaded as a ncn account 2. A config account can be loaded twice 3. An operator account cannot be loaded as a ncn account	rule_load_config_and_ncn Rule_load_config_twice_witness - this witness rule is intended to fail rule_load_operator_and_ncn

Security Rules - Restaking Program

P-01 Input Account Keys Match - process_ncn_set_admin

P-02 Input Account Keys Match - process_set_node_operator_admin

Status: Verified	

Rule Name	Status	Description	Link to rule report
rule_integrity_ncn_set_admin rule_integrity_operator_set_admin	Verified	The public keys of corresponding admins accounts received by process_ncn_set_admin(), process_ncn_set_secondary_admin(), process_set_node_operator_admin() match	rule_integrity_ncn_set_admin rule_integrity_operator_set_admin

Security Rules - Vault Program

P-01 Input Account Keys Match - process_burn

P-02 Input Account Keys Match - process_burn_withdrawal_ticket

P-03 Input Account Keys Match - process_mint

P-04 Input Account Keys Match - process_update_vault_balance

P-05 Input Account Keys Match - process_cooldown_vault_ncn_ticket

Status: Verified	

Rule Name	Status	Description	Link to rule report
rule_pubkey_checks_burn rule_pubkey_checks_burn_withdrawal_ticket rule_pubkey_chec	Verified	• The public keys of vrt_mint and vault.vrt_mint accounts sent to each of the following instructions match: process_burn(), process_burn_withdrawal_ticket(),	Rule_pubkey_checks_burn - not applicable in the fix version. rule_pubkey_checks_burn_withdrawal_ticket

ks_mint rule_pubkey_checks_update_vault_balance rule_pubkey_checks_cooldown_vault_ncn_ticket		• process_mint(),process_update_vault_balance(), and process_create_token_metadata(). • The public keys of admin and vault.ncn_admin sent to process_cooldown_vault_ncn_ticket() match.	rule_pubkey_checks_mint rule_pubkey_checks_update_vault_balance rule_pubkey_checks_cooldown_vault_ncn_ticket

P-06 Input Account Keys Match - process_create_token_metadata

Status: Violated	

Rule Name	Status	Description	Links
rule_pubkey_checks_create_token_metadata	**Violated**	The public keys of vrt_mint and vault.vrt_mint accounts sent to process_create_token_metadata() match.	Rule report This bug was already found and fixed by Jito Labs. Fixed rule report

Sanity Rules - Restaking Program

P-01 Correct Number of Accounts - process_initialize_ncn

P-02 Correct Number of Accounts - process_initialize_config

P-03 Correct Number of Accounts - process_initialize_ncn_vault_slasher_ticket

P-04 Correct Number of Accounts - process_initialize_ncn_vault_ticket

P-05 Correct Number of Accounts - process_initialize_operator

P-06 Correct Number of Accounts - process_initialize_ncn_operator_state

P-07 Correct Number of Accounts - process_initialize_operator_vault_ticket

P-08 Correct Number of Accounts - process_ncn_withdraw_asset

P-09 Correct Number of Accounts - process_operator_withdrawal_asset

P-10 Correct Number of Accounts - process_operator_warmup_ncn

P-11 Correct Number of Accounts - process_warmup_ncn_vault_slasher_ticket

P-12 Correct Number of Accounts - process_warmup_ncn_vault_ticket

P-13 Correct Number of Accounts - process_ncn_warmup_operator

P-14 Correct Number of Accounts - process_warmup_operator_vault_ticket

P-15 Correct Number of Accounts - process_ncn_cooldown_operator

P-16 Correct Number of Accounts - process_cooldown_ncn_vault_slasher_ticket

P-17 Correct Number of Accounts - process_cooldown_ncn_vault_ticket

P-18 Correct Number of Accounts - process_operator_cooldown_ncn

P-19 Correct Number of Accounts - process_cooldown_operator_vault_ticket

Status: Verified

Rule Name	Status	Description	Link to rule report
rule_sanity_initialize_ncn rule_sanity_initialize_config ...	Verified	The number of account keys received by an instruction matches the accounts specified in restaking_sdk/src/instruction.rs	

Sanity Rules - Vault Program

P-01 Correct Number of Accounts - process_burn

P-02 Correct Number of Accounts - process_mint

P-03 Correct Number of Accounts - process_add_delegation

P-04 Correct Number of Accounts - process_burn_withdrawal_ticket

P-05 Correct Number of Accounts - process_change_withdrawal_ticket_owner

P-06 Correct Number of Accounts - process_close_vault_update_state_tracker

P-07 Correct Number of Accounts - process_cooldown_delegation

P-08 Correct Number of Accounts - process_cooldown_vault_ncn_slasher_ticket

P-09 Correct Number of Accounts - process_cooldown_vault_ncn_ticket

P-10 Correct Number of Accounts - process_crank_vault_update_state_tracker

P-11 Correct Number of Accounts - process_enqueue_withdrawal

P-12 Correct Number of Accounts - process_initialize_config

P-13 Correct Number of Accounts - process_initialize_vault

P-14 Correct Number of Accounts - process_initialize_vault_ncn_slasher_operator_ticket

P-15 Correct Number of Accounts - process_initialize_vault_ncn_slasher_ticket

P-16 Correct Number of Accounts - process_initialize_vault_ncn_ticket

P-17 Correct Number of Accounts - process_initialize_vault_operator_delegation

P-18 Correct Number of Accounts - process_initialize_vault_update_state_tracker

P-19 Correct Number of Accounts - process_set_admin

P-20 Correct Number of Accounts - process_set_deposit_capacity

P-21 Correct Number of Accounts - process_set_fees

P-22 Correct Number of Accounts - process_set_secondary_admin

P-23 Correct Number of Accounts - process_slash

P-24 Correct Number of Accounts - process_update_vault_balance

P-25 Correct Number of Accounts - process_warmup_vault_ncn_slasher_ticket

P-26 Correct Number of Accounts - process_warmup_vault_ncn_ticket

Status: Verified

Rule Name	Status	Description	Link to rule report
rule_sanity_burn rule_sanity_mint ...	Verified	The number of account keys received by an instruction matches the accounts specified in vault_sdk/src/instruction.rs	

Disclaimer

The Certora Prover takes a contract and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, the guarantees of the Certora Prover are scoped to the provided specification and the Certora Prover does not check any cases not covered by the specification.

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.