Security Assessment

Final Report

Navi Vault

April 2026

Prepared for Navi Protocol

Table of Contents

Project Summary

Project Scope

Project Name	Repository Link	Commit Hash	Platform
Navi Vault	https://github.com/naviprotocol/navi_vault	82346bf (audit) 878557f (fix)	Sui, Move

Project Overview

This document describes the security review of **Navi Vault**. The work was undertaken from **March 20th, 2026** to **April 10th, 2026**.

The following contract list is included in our scope:

- `sources/events.move`
- `sources/navi_vault.move`
- `sources/version.move`

The team performed a manual audit of all the files in scope**.** During the manual audit, the Certora team discovered bugs in the code, as listed on the following page.

Protocol Overview

Navi is a vault-style yield protocol where users deposit their funds into a shared pool, and the protocol allocates those funds across approved lending markets to seek higher returns. In exchange, users receive a claim on their share of the vault, allowing them to withdraw their portion later along with any yield the vault has generated. At a high level, Navi operates as a managed on-chain treasury: users supply the capital, while protocol-authorized managers allocate it within predefined rules, limits, and risk controls. The system also supports rewards, fees, and administrative safeguards such as pausing and market restrictions, so assets are managed in a structured and controlled manner.

Assessment Methodology

Our assessment approach combines design level analysis with a deep review of the implementation to ensure that a protocol is secure, economically sound, and behaves as intended under realistic conditions.

At the design level, we evaluate the architecture, the economic assumptions behind the protocol, and the safety properties that should hold independently of a specific chain or environment. This process includes reviewing internal and cross protocol interactions, state transition flows, trust boundaries, and any mechanism that could be exploited to extract value, deny service, or alter core system behavior. At this stage, a focused threat modelling exercise helps identify key attack surfaces and adversarial capabilities relevant to the system. Design level issues often relate to incentive structures, governance implications, or systemic behavior that emerges under adversarial conditions.

Implementation analysis focuses on the concrete behavior of the code within the execution model of the target chain. This involves reviewing the correctness of logic, access control, state handling, arithmetic behavior, and the nuanced behaviors of the chain environment. Familiar classes of vulnerabilities such as reentrancy conditions, faulty permission checks, precision issues, or unsafe assumptions often surface at this layer. These findings require context aware reasoning that takes into account both the code and the architectural intent.

To support this analysis, the codebase is examined through repeated manual passes and supplemented by automated tools when appropriate. High-risk logic areas receive deeper scrutiny, invariants are validated against both design intent and actual implementation, and potential vulnerability leads are thoroughly investigated. Automated techniques such as static analysis, fuzzing, or symbolic execution may be used to complement manual review and provide additional insight.

Collaboration with the development team plays an important role throughout the audit. This helps confirm expected behaviors, clarify design assumptions, and ensure an accurate understanding of the protocol's intended operation. All findings are documented with clear reasoning, reproducible examples, and actionable recommendations. A follow up review is conducted to validate the applied fixes and verify that no regressions or secondary issues have been introduced.

Threat and Security Overview

Access control

A central focus of the review was the protocol's access-control model, since Navi relies on a set of privileged roles to administer the vault, manage allocations, configure markets, and control certain operational parameters. The threat model therefore treated privileged-role integrity as a core security assumption: if any of these roles were misused, the protocol's intended safeguards could be bypassed. For that reason, the audit paid particular attention to role separation, capability scoping, and whether sensitive actions were appropriately restricted to authorized actors.

Share pricing mechanism

Share pricing was treated as one of the most security-critical components in the protocol, because the share price determines how much value users receive when entering or exiting the vault. In any vault-based system, the ability to manipulate share pricing can directly enable value extraction from the protocol and its users, making this a key area of review. The audit therefore focused on whether deposits, withdrawals, fee accrual, balance synchronization, and accounting updates preserve a fair and tamper-resistant conversion between assets and shares. A specific point of attention was the set_loss function, since any privileged mechanism that affects reported asset value can also affect share pricing. This makes the associated role a security-sensitive role, and it should be secured accordingly.

Fair reward distribution

Another key area of review was whether rewards are distributed fairly and consistently across participants according to their economic position in the vault. This includes scenarios where an attacker attempts to manipulate the reward-accounting mechanism, timing of state updates, or claim flow in order to receive a disproportionate share of rewards and thereby divert significant value from other users. The audit therefore paid close attention to reward index updates, harvesting flows, claim logic, and sequencing assumptions to verify that rewards cannot be improperly concentrated, accelerated, or double-counted in a way that would allow material reward theft.

Rounding issues

Rounding behavior was also included in the threat model because even small arithmetic edge

cases can accumulate into measurable value transfer in financial systems. The review therefore examined whether rounding in share minting, share burning, fee accounting, and reward distribution remained consistent with the protocol's intended economic design. The goal in this area was to verify that rounding choices do not create exploitable imbalances or allow participants to systematically gain value at the expense of the vault.

Denial of service (DoS)

Availability was treated as an important part of protocol security, especially because vault operations depend on coordinated accounting updates and interactions with configured markets and reward flows. The threat model therefore considered whether an attacker, operator failure, or edge-case state condition could prevent normal user actions such as deposit, withdrawal, or reward claiming. This part of the review focused on whether the protocol remains operational under realistic stress conditions and whether its dependencies and control flows could be used to disrupt service against the intended design.

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	2	2	2
Low	13	13	9
Informational	8	8	4
Total	23	23	15

Severity Matrix

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

Detailed Findings

ID	Title	Severity	Status
M-01	Stale supply_index in sync_market_balance Causes Mispricing of Shares	Medium	Fixed
M-02	Market's loss field can be used to manipulate price share and steal funds	Medium	Fixed
L-01	Set-loss doesn't sync the market balance, allowing users to not account for it in the current block	Low	Fixed
L-02	Depositors Can Front-Run set_loss to Withdraw Before Loss Is Applied	Low	Acknowledged
L-03	accrue_interest() might overflow the total shares, which would DoS the vault	Low	Acknowledged
L-04	A curator can propose multiple different default market simultaneously	Low	Fixed
L-05	Insufficient Virtual Share Offset Enables Multi-Victim Donation Attack	Low	Fixed
L-06	Last Reward Claimers May Be Unable to Claim Due to Cumulative Half-Up Rounding in Reward Calculations	Low	Fixed

L-07	All Vault Operations Blocked by Underfunded NAVI Reward Rule	Low	Fixed
L-08	Inconsistent Harvest Enforcement in Fee Functions Causes Reward Dilution	Low	Fixed
L-09	Share Price Inflation from Penalty Logic Enables Front-Running Profit and Penalty Self-Reduction	Low	Acknowledged
L-10	Allocation-Based Cap Enforcement Drifts from Reality	Low	Fixed
L-11	Event emission functions are public, allowing anybody to emit false events	Low	Fixed
L-12	Loss set by the admin is fixed, causing share pricing issues	Low	Acknowledged
L-13	Lack of slippage protection during withdrawal	Low	Fixed
I-01	No Mechanism to Withdraw Remaining Reward Funds After Disabling a Vault-Native Rule	Informational	Fixed
I-02	Limiting the set-loss admin function to loss-increase can enhance security	Informational	Acknowledged
I-03	Unnecessary Iteration Over Irrelevant Reward Rules	Informational	Acknowledged
I-04	Incorrect reward_rule_created Event Emitted When Reactivating an Existing Reward Rule	Informational	Fixed

I-05	Misleading pool_address in Withdraw Event When Withdrawal Is Served Entirely from Idle Balance	Informational	Fixed
I-06	Default Market Can Be Disabled via apply_set_market_status	Informational	Fixed
I-07	`apply_deposit_reward_balance()` doesn't check that the rule isn't disabled	Informational	Acknowledged
I-08	Deposit Reverts When Default Market Cap Is Fully Consumed	Informational	Acknowledged

Medium Severity Issues

M-01: Stale supply_index in sync_market_balance Causes Mispricing of Shares

Severity: Medium	Impact: Medium	Likelihood: Medium
Files: sources/navi_vault.move	Status: Fixed	

Description:

The vault's sync_market_balance reads NAVI's supply_index via storage.get_index() - a pure storage read that returns whatever index was last written. NAVI's lending protocol only recalculates this index inside logic::update_state, which is triggered by state-mutating operations (deposit, withdraw, borrow, repay, etc.), not by reads.

If no one has interacted with that NAVI market since the last such operation, the index remains stale and the vault under-counts its current_balance. This feeds into total_assets, which directly affects share pricing on deposits and withdrawals:

- New depositors are favored. They mint shares against a deflated `total_assets`, receiving more shares than they should. An attacker could intentionally time deposits into the vault during periods of market inactivity to exploit the stale index, then call update_state_of_all afterward to realize the accrued interest on their now-overweighted share position.
- Existing holders are diluted. Their share value is understated because accrued interest is not yet reflected in the index.
- Withdrawing users incur a loss. They redeem shares at a price that does not account for interest earned since the last index update, receiving fewer tokens than their actual entitlement.

The degree of staleness depends on how long ago someone last interacted with the specific NAVI lending market the vault deposits into. On high-activity markets (e.g., USDC, SUI), the index is refreshed frequently by organic activity and the mispricing window is small. On low-activity markets, the index can lag significantly, and the mispricing becomes meaningful - especially at higher vault TVL where even small share price deviations translate to non-trivial token amounts.

NAVI exposes a public permissionless function lending::update_state_of_all(clock, storage) that forces a fresh index recalculation. The vault does not currently call this before reading the index.

Recommendations:

Invoke lending::update_state_of_all(clock, storage) (or the equivalent per-market update) within the vault's sync_market_balance flow - or document that it must be included in the PTB before any vault deposit/withdraw call - to ensure the index is always fresh before share pricing calculations.

Customer Response:

Fixed in fd3b5a4

Fix Review: Fix looks good

M-02: Market's loss field can be used to manipulate price share and steal funds

Severity: Medium	Impact: High	Likelihood: Low
Files: sources/navi_vault.move	Status: Fixed	

Description:

The vault allows withdrawing from a market with loss (including the lost amount), as long as the underlying market allows that. This can be used to manipulate the share price and steal funds from users. Consider the following scenario:

- Vault has 100 tokens and 100 shares, a market has a loss of 20 tokens
    - Share price is 80/100=0.8
    - For the sake of simplicity, assume the total balance (before loss) in the market with loss is 20
    - Penalty is set to 0 (for simplicity, can also work if penalty isn't too high)
- Eve deposits 100 tokens and gets 125 shares
- Eve withdraws 20 tokens (for 25 shares) from the market with loss
    - Market now has 180 tokens (loss isn't accounted any more) and 200 shares
    - Share price is now 0.9
- Eve withdraws the remaining 100 shares, and gets 90 tokens

Eve started with 100 tokens, and now has 110, stealing 10 tokens from the vault

This can also be exploited by the allocator role, and in that case they can repeat this attack multiple times and empty the vault.

Recommendations:

Don’t allow withdrawing or deallocating funds if it causes the market’s balance to be below the loss

Customer Response:

Fixed in fd3b5a4

Fix Review: Fix looks good

Low Severity Issues

L-01: Set-loss doesn't sync the market balance, allowing users to not account for it in the current block

Severity: Low	Impact: Medium	Likelihood: Low
Files: sources/navi_vault.move	Status: Fixed	

Description:

The set_loss() function sets the loss for the market, but doesn't ensure this would reflect in the market's balance for the current block. Consider a scenario where in the same block there's:

- A tx to sync the market (e.g. deposit by a user)
- `set_loss()` called by the admin
- A tx to withdraw (without syncing the market)

The withdrawal would succeed without accounting for the loss, since the market is considered synced for the current block.

Recommendations:

Either update the balance by subtracting the loss or set last_sync_at to 0 when setting loss.

Customer Response:

Fixed in fd3b5a4

Fix Review: Fix looks good

L-02: Depositors Can Front-Run set_loss to Withdraw Before Loss Is Applied

Severity: Low	Impact: Medium	Likelihood: Low
Files: sources/navi_vault.move	Status: Acknowledged	

Description:

The set_loss function, which allows an admin to record a loss event for a market, can be front-run by depositors who observe the pending transaction. A depositor who sees the set_loss call can withdraw their funds before the loss is applied, exiting at the pre-loss share price. Once set_loss executes, the recorded loss is distributed across a smaller remaining share base, meaning the remaining depositors absorb a disproportionately larger share of the loss.

```rust
public fun set_loss<CoinType>(
    self: &mut Vault<CoinType>,
    _: &AdminCap,
    pool_address: address,
    loss: u64,
) {
    version_verification(self);
    assert!(vec_map::contains(&self.markets, &pool_address), E_MARKET_NOT_FOUND);
    let market_info = vec_map::get_mut(&mut self.markets, &pool_address);
    market_info.loss = loss;
    events::emit_set_loss(object::id_address(self), pool_address, loss);
}
```

Recommendations:

Since automatic loss realization is not possible, consider a two-step loss application: first pause the vault, then apply the loss.

Customer Response:

Acknowledged

L-03: accrue_interest() might overflow the total shares, which would DoS the vault

Severity: Low	Impact: Medium	Likelihood: Low
Files: sources/navi_vault.move	Status: Acknowledged	

Description:

The additions to total_shares within accrue_interest (for fee shares) and within deposit lack explicit overflow guards. Although individual minted shares might fit in a u64, the running total_shares accumulator is not verified against MAX_U64. In extreme loss scenarios where total_assets drops significantly below total_shares, new deposits generate shares at a massive ratio. An attacker can deposit iteratively to push total_shares near MAX_U64. The next legitimate interaction triggers accrue_interest, minting fee shares and overflowing the accumulator, which permanently bricks all core vault operations.

Recommendations:

Check if overflow is expected, and cap the fees shares to the max amount that won't overflow

Customer Response:

Acknowledged

L-04: A curator can propose multiple different default market simultaneously

Severity: Low	Impact: Low	Likelihood: Medium
Files: sources/navi_vault.move	Status: Fixed	

Description:

The Timelock system uses the target pool_address as the subject in the ProposalKey for the DefaultMarket. Because the key incorporates the target address, a malicious or compromised Curator can submit multiple conflicting DefaultMarket proposals simultaneously. Both will be successfully queued and can be executed sequentially in arbitrary order, undermining timelock predictability.

Recommendations:

Change the subject field for ProposalType::DefaultMarket to a constant value like @0x0 to ensure only a single default market proposal can be pending at a time.

Customer Response:

Fixed in fd3b5a4

Fix Review: Fix looks good

L-05: Insufficient Virtual Share Offset Enables Multi-Victim Donation Attack

Severity: Low	Impact: Low	Likelihood: Medium
Files: sources/navi_vault.move	Status: Fixed	

Description:

The vault uses a virtual share offset of +1 to mitigate inflation/donation attacks on the share price. While this is sufficient to make a single-victim attack net-negative for the attacker, it does not adequately protect against a multi-victim variant.

The vault tracks its assets via sync_market_balance, which reads the vault's actual lending pool balance (user_supply * supply_index). NAVI's incentive_v3 exposes a public entry function entry_deposit_on_behalf_of_user that lets anyone deposit tokens into the lending pool credited to any address - including the vault's lending account. The vault's account address is publicly discoverable from on-chain DepositEvent emissions.

This means an attacker can inflate total_assets (picked up on the next sync_market_balance) without going through the vault's deposit function, so no shares are minted for the donated amount.

Attack Flow:

1. Deposit 1 token to receive 1 share.

2. Call `entry_deposit_on_behalf_of_user` to donate D tokens directly into the lending pool on behalf of the vault's lending account. No shares are minted. Cost so far: D+1.

3. The share mint formula floors: `amount * (S+1) / (A+1)`. After inflation, any victim depositing just under 2× the share price gets floored from ~2 shares to 1, losing up to ~50% of their deposit to rounding. That rounding accrues to existing shareholders (including the attacker).

With a single victim, the attack is net-negative for the attacker (loses ~D/3) — the +1 virtual offset is sufficient there. But the math changes with multiple victims.

With D=1000 donation, each victim who hits this rounding cliff adds ~167 to the attacker's share value. After 3 such victims the attacker breaks even; after 5, they profit +334 on a 1001 cost. Profit scales linearly: 1001 × (k-3) / 6 where k = number of max-rounding victims.

Example:

Setup: deposit 1, donate 1000. S=1, A=1001. Cost=1001.

Deposits:

Victim	A before	V deposited	Mint calculation	Got	A after	S after
V1	1001	1001	2002/1002 = 1.998	1	2002	2
V2	2002	1335	4005/2003 = 1.999	1	3337	3
V3	3337	1668	6672/3338 = 1.998	1	5005	4
V4	5005	2002	10010/5006 = 1.999	1	7007	5
V5	7007	2335	14010/7008 = 1.999	1	9342	6

Each victim deposits just under 2 shares' worth, gets floored to 1.

Withdrawals:

Who	S	A	floor((A+1)/(S+1))	Gets	S after	A after

V5	6	9342	9343/7 = 1334	1334	5	8008
V4	5	8008	8009/6 = 1334	1334	4	6674
V3	4	6674	6675/5 = 1335	1335	3	5339
V2	3	5339	5340/4 = 1335	1335	2	4004
V1	2	4004	4005/3 = 1335	1335	1	2669
Attacker	1	2669	2670/2 = 1335	1335	0	1334

P&L:

Person	**Deposited**	**Withdrawn**	**P&L**
Attacker	1001	1335	+334
Victim 1	1001	1335	+334
Victim 2	1335	1335	0
Victim 3	1668	1335	-333

Victim 4	2002	1334	-668
Victim 5	2335	1334	-1001

1334 tokens end up permanently trapped in the virtual share accounting, but the attacker is in profit.

In any scenario, over the long term, due to rounding (because of donation) in share calculation, the later users would incur loss, even when the attacker is not profitable.

Recommendations:

Consider increasing the virtual offset significantly (e.g., to 1e6 or higher) so that the rounding loss per victim becomes economically negligible relative to realistic deposit sizes.

Customer Response:

Fixed in dfa0c7e

Fix Review: Fix looks good

L-06: Last Reward Claimers May Be Unable to Claim Due to Cumulative Half-Up Rounding in Reward Calculations

Severity: Low	Impact: Low	Likelihood: Low
Files: sources/navi_vault.move	Status: Fixed	

Description:

Two places in the reward calculation logic use half-up rounding:

1. **Index increment:** `ray_div(reward_amount, total_shares)` - rounds up the per-share index growth.
2. **Per-user accrual:** `ray_mul(shares, index_diff)` - rounds up the reward owed to each user.

Both rounding steps favor the claimant, meaning slightly more rewards are attributed to each user than the exact mathematical entitlement. Over multiple users and multiple collection cycles, these small overages accumulate. As a result, the last claimants for a given RewardCoinType may find insufficient funds remaining in the reward balance and be unable to claim their entitled rewards.

Recommendations:

Use round-down (truncation) for both ray_div in index increment and ray_mul in per-user accrual, ensuring the total attributed rewards never exceed the actual deposited reward funds.

Customer Response:

Fixed in fd3b5a4

Fix Review: Fix looks good

L-07: All Vault Operations Blocked by Underfunded NAVI Reward Rule

Severity: Low	Impact: Medium	Likelihood: Low
Files: sources/navi_vault.move	Status: Fixed	

Description:

In NAVI's lending protocol, set_reward_rate_by_rule_id and deposit_incentive_v3_reward_fund are completely decoupled. When setting a reward rate, the only constraint is rate <= max_rate. When depositing reward funds, there is no enforcement that deposited funds >= rate * duration.

Ideally the admin would be responsible for depositing sufficient funds, but in the worst case it is possible that when the vault tries to call collect_rewards, the transaction reverts due to insufficient reward funds. Since assert_all_rules_harvested is validated in almost every vault function (deposit, withdraw, claim_fee, etc.), a reverting collect_rewards would effectively brick the entire vault - no core operation could execute.

Disabling the problematic reward rule at the vault level is also not a viable escape path, because the disable function itself enforces the last_harvest_at validation, which requires a successful harvest, creating a circular dependency.

Recommendations:

Add an emergency admin function that bypasses all validations (including last_harvest_at) solely to disable a reward rule. This provides a recovery path when an external reward fund is depleted, preventing the vault from being bricked by a dependency outside its control.

Customer Response:

Fixed in fd3b5a4

Fix Review: Fix looks good

L-08: Inconsistent Harvest Enforcement in Fee Functions Causes Reward Dilution

Severity: Low	Impact: Low	Likelihood: Low
Files: sources/navi_vault.move	Status: Fixed	

Description:

Both apply_performance_fee and apply_management_fee call accrue_interest, which inflates total_shares via fee share minting. However, neither function enforces assert_all_rules_harvested beforehand. Every other function that touches share accounting (deposit, withdraw, claim_fee) enforces this check before calling accrue_interest.

This is problematic because accrue_interest increases total_shares. If collect_reward hasn't been called beforehand, rewards accumulated since the last collection are distributed against the post-mint (inflated) total_shares, diluting existing depositors' reward index growth.

Note: the code comments mention that all markets should be synced in the current PTB, but make no mention of collect_reward needing to be called as well.

The severity depends on compounding factors:

- **Duration since last** `accrue_interest`**:** If significant time has passed, the fee shares minted could be considerable, making the dilution non-negligible.
- **Duration since last** `collect_reward`**:** If rewards haven't been collected in a while, a relatively large amount of accumulated rewards gets divided by the inflated `total_shares`, amplifying the dilution effect.

Recommendations:

Enforce assert_all_rules_harvested in both apply_performance_fee and apply_management_fee before calling accrue_interest, consistent with every other function that mints shares. Alternatively, at minimum, document clearly that collect_reward must be called for all active reward rules in the same PTB before invoking these fee functions, so that admin and curators are aware of the required ordering.

Customer Response:

Fixed in fd3b5a4

Fix Review: Fix looks good

L-09: Share Price Inflation from Penalty Logic Enables Front-Running Profit and Penalty Self-Reduction

Severity: Low	Impact: Low	Likelihood: Medium
Files: sources/navi_vault.move	Status: Acknowledged	

Description:

The penalty logic for non-default market withdrawals burns more shares than the assets returned, effectively increasing the share price in a single step. This creates two exploitable scenarios:

1. **Front-running for profit:**

An attacker who observes a user's pending withdrawal from a non-default market can front-run it with a large deposit, capture the share price spike caused by the penalty burn, and withdraw at a profit.

Example: asset = 1000, share = 1000 (1:1 ratio), penalty = 20%. A user is withdrawing 500 tokens from a non-default market.

- Attacker front-runs by depositing 10,000 tokens. `asset = 11000`, `shares = 11000`.
- User's withdrawal executes — 600 shares burned for 500 assets due to penalty. `asset = 10500`, `shares = 10400`.
- Attacker withdraws 10,000 shares, receives ~10,096 tokens. Profit: ~96 tokens.

2. **Self-reduction of penalty:**

The withdrawer themselves can reduce their effective penalty within the same PTB: deposit a large amount into the default market (or idle balance if no default market is set), execute the penalized withdrawal from the non-default market, then withdraw from the default market. Since the withdrawer holds a large share position during the penalty burn, they recapture a portion of the penalty they just paid, effectively reducing it.

Recommendations:

Consider dripping the penalty over time - similar to how rewards are distributed - rather than applying it as an instantaneous share price spike. This would smooth the increase and eliminate the single-transaction arbitrage window for both front-runners and self-reducing withdrawers.

Customer Response:

Acknowledged

L-10: Allocation-Based Cap Enforcement Drifts from Reality

Severity: Low	Impact: Low	Likelihood: High
Files: sources/navi_vault.move	Status: Fixed	

Description:

We believe the cap is meant to limit risk exposure to a market, but given the current design, it may not fully serve that purpose. The core issue is that allocation never accounts for interest or loss, but during withdraw/deallocate, the full amount (which includes interest) is subtracted from it, which creates a growing divergence over time. Consider the following scenarios:

1. **Over-allocation (interest accrual):**

• Two users deposit 50 each. `allocation = 100`, `current_balance = 100`.

• Interest accrues: `current_balance = 150`, `allocation = 100`.

• User A withdraws their share (75): `allocation = 25`, `current_balance = 75`.

• User B's original principal of 50 is still in the market, but `allocation` only shows 25. This systematically under-reports exposure, allowing more capital into the market than the cap intended.

1. **Phantom allocation (loss):**

• `allocation = 100`, `current_balance = 100`.

• Admin sets 30 loss: `current_balance = 70`, `allocation = 100`.

• The user withdraws all 70: `allocation = 30`, but nothing is actually in the market anymore. That phantom 30 permanently occupies cap space and blocks legitimate deposits.

Impact:

• **Cap bypass:** Interest accrual causes `allocation` to drift below actual exposure, allowing more capital into a market than the cap intended.

- **Permanent cap space exhaustion:** After admin-set losses, phantom `allocation` remains that can never be cleared through normal operations, permanently blocking legitimate deposits.
- Both effects compound over time - the longer the vault operates, the more inaccurate cap enforcement becomes.

Recommendations:

Remove allocation as a separate tracking variable and use current_balance for cap enforcement instead, since it already reflects the true state of funds in a market including interest and losses.

Customer Response:

Fixed in fd3b5a4

Fix Review: Fix looks good

L-11: Event emission functions are public, allowing anybody to emit false events

Severity: Low	Impact: Low	Likelihood: Low
Files: sources/events.move	Status: Fixed	

Description:

The event emission helper functions in the navi_vault::events module are declared as public. Any external smart contract can invoke these functions to emit authentic-looking vault events with arbitrary, malicious, or fabricated payload values. Off-chain analytics, indexers, and front-ends that trust these event signatures could be tricked into recording fake deposits, withdrawals, or governance actions, poisoning off-chain infrastructure.

Recommendations:

Restrict the visibility of all event emission helper functions from public to public(package)or internal to ensure that only authorized modules within the package can emit vault events.

Customer Response:

Fixed in fd3b5a4

Fix Review: Fix looks good

L-12: Loss set by the admin is fixed, causing share pricing issues

Severity: Low	Impact: Low	Likelihood: Low
Files: sources/navi_vault.move	Status: Acknowledged	

Description:

The admin-defined loss is a static value denominated in the vault's base asset, changing only when manually updated. In contrast, the vault's actual loss from bad debt in the underlying market is dynamic—it is calculated proportionally based on the vault's share of the total market. This discrepancy between a fixed administrative loss and a dynamic actual loss can lead to pricing inaccuracies, as total market shares constantly fluctuate due to user deposits and withdrawals, or the vault adjusting its allocations.

Recommendations:

Ideally, bad debt should be resolved directly at the underlying lending protocol level. If that is not feasible, the vault's loss should be calculated dynamically, as outlined above. Also make sure to not allow allocating more funds to a market with loss, in order to not drive the price share down.

Customer Response:

Acknowledged

L-13: Lack of slippage protection during withdrawal

Severity: Low	Impact: Low	Likelihood: Low
Files: sources/navi_vault.move	Status: Fixed	

Description:

During withdrawal the outcome might be different than the user has expected due to 2 reasons:

- Idle balance decreasing due to withdrawal/allocation - this would increase the amount withdrawn from the market and therefore the penalty as well (if the default market is empty and the user is withdrawing from another market)
- Changes in the penalty parameter, given that even the curator can set it instantly without a timelock then this change might happen after the user has signed the transaction and before it's executed

Recommendations:

Consider adding a slippage check. Given that the withdrawn amount can also change (due to the availability of funds in the market), then this check can be in the form of the ratio between assets withdrawn and shares burned

Customer Response:

Fixed in fd3b5a4 and 4070181

Fix Review: Fix looks good

Informational Severity Issues

I-01: No Mechanism to Withdraw Remaining Reward Funds After Disabling a Vault-Native Rule

Files:

sources/navi_vault.move

Description:

If a vault-native reward rule is disabled mid-way through its distribution period, any undistributed reward funds remain locked in the vault with no withdrawal mechanism available to the admin. The only option to resume distributing those funds is to reactivate the rule, there is no way to recover the remaining balance if the admin genuinely wants to discontinue the incentive.

Recommendations:

If the likelihood of an admin wanting to permanently discontinue a vault-native reward and reclaim remaining funds is realistic, consider adding an admin function to withdraw undistributed reward funds from a disabled rule.

Customer Response:

Fixed in fd3b5a4

Fix Review: Fix looks good

I-02: Limiting the set-loss admin function to loss-increase can enhance security

Files:

sources/navi_vault.move

Description:

The set_loss() function can be abused to manipulate the share price and steal funds from the vault if the admin cap falls into the wrong hands. If this function is intended to be used only to increase loss then it's best to restrict it only to do that, this can enhance the security of the protocol - even if the admin cap falls into the wrong hands, it can only be used to increase the loss and not decrease it back (which is necessary to steal funds from the vault).

Recommendations:

Consider limiting the function for loss increase only

Customer Response:

Acknowledged

I-03: Unnecessary Iteration Over Irrelevant Reward Rules

Files:

sources/navi_vault.move

Description:

Two frequently called functions iterate over the entire reward_rules vector but only operate on a subset:

- `assert_all_rules_harvested` is invoked in nearly every core vault operation (`deposit`, `withdraw`, `claim_fee`, etc.) and iterates over all reward rules, but only market rules are relevant - vault-native rules are always skipped via the `!rule.is_vault_native` condition.
- `update_vault_native_indices` similarly iterates over all reward rules, but only vault-native rules are relevant - market rules are skipped via the `rule.is_vault_native` condition.

In both cases, every core operation pays the cost of iterating over entries that will never be acted upon, and this overhead grows as more reward rules of either type are added.

Recommendations:

Consider separating reward rules into two distinct vectors, one for vault-native rules and one for market rules. assert_all_rules_harvested would only iterate over the market rules vector, update_vault_native_indices would only iterate over the native rules vector, and other logic specific to either type can operate on its own collection.

Customer Response:

Acknowledged

I-04: Incorrect reward_rule_created Event Emitted When Reactivating an Existing Reward Rule

Files:

sources/navi_vault.move

Description:

In apply_create_reward_rule, when a previously disabled reward rule is being reactivated, the function still emits a reward_rule_created event. This is semantically incorrect - the rule is not being created, it is being re-enabled. This can mislead off-chain indexers and monitoring systems that track rule lifecycle events.

Recommendations:

Emit a distinct reward_rule_reactivated event (or equivalent) when reactivating an existing rule, reserving reward_rule_created for genuinely new rule creation.

Customer Response:

Fixed in fd3b5a4

Fix Review: Fix looks good

I-05: Misleading pool_address in Withdraw Event When Withdrawal Is Served Entirely from Idle Balance

Files:

sources/navi_vault.move

Description:

During withdrawal, when the idle balance fully covers the requested amount and no funds are pulled from any market, the emitted event still reports the default market as the pool_address. This is misleading since no interaction with the default market occurred. The pool_address should be @0x0 to accurately reflect that the withdrawal was served from idle balance

```rust
let pool_address = if (from_market > 0) {
        // ...
    } else {
        self.default_market
    };

    events::emit_withdraw(
        vault_address,
        ctx.sender(),
        receipt_id,
        pool_address,
        actual_total,
        total_shares_to_burn,
    );
```

Recommendations:

Emit @0x0 as the pool_address in the withdraw event when the withdrawal is fully served from idle balance.

Customer Response:

Fixed in fd3b5a4

Fix Review: Fix looks good

I-06: Default Market Can Be Disabled via apply_set_market_status

Files:

sources/navi_vault.move

Description:

When setting a market as the default market, the vault validates that the target market is not disabled. However, in apply_set_market_status, there is no reciprocal check preventing a market that is currently set as the default from being disabled. Both are admin/curator-gated actions, so the risk is limited, but the asymmetry in validation is inconsistent - one path enforces the invariant that the default market must be active, while the other does not.

Recommendations:

Add a check in apply_set_market_status that prevents disabling a market if it is currently set as the default market.

Customer Response:

Fixed in fd3b5a4

Fix Review: Fix looks good

I-07: `apply_deposit_reward_balance()` doesn't check that the rule isn't disabled

Files:

sources/navi_vault.move

Description:

The apply_deposit_reward_balance function deposits reward tokens into the bag and credits the rule's budget, but it does not verify that the rule's is_active flag is true. Tokens deposited to a disabled rule will not be distributed by update_vault_native_indices. While the admin can safely reactivate the rule later via set_reward_rate, the initial deposit provides no feedback that funds are currently stagnant.

Recommendations:

Add an is_active check in apply_deposit_reward_balance to ensure funds cannot be deposited to rules that will not distribute them.

Customer Response:

Acknowledged

I-08: Deposit Reverts When Default Market Cap Is Fully Consumed

Files:

sources/navi_vault.move

Description:

Currently, when a user deposits into the vault and the default market's cap has been fully consumed, the entire deposit transaction reverts. There is no fallback mechanism to handle this scenario gracefully.

Ideally, when the default market's cap is reached, the deposit should still be accepted and the funds should be held in the vault's idle balance. The allocator role could then later allocate those idle funds into different markets with available capacity.

Impact:

- Users are unable to deposit into the vault whenever the default market's cap is consumed, even though the vault could productively hold and later allocate those funds.
- In high-demand periods, this could effectively lock out new depositors entirely despite the vault having alternative allocation paths available through the allocator role.

Recommendations:

When the default market cap is consumed, accept the deposit, fill any remaining cap space first, and route the overflow to idle balance for later allocation by the allocator.

Customer Response:

Acknowledged

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.