Navi Vault

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Thu Mar 19 2026

Navi Vault Audit Report

1 Executive Summary

1.1 Project Information

Description	NAVI Vault is a share-based asset management vault on Sui that allocates user deposits across NAVI lending markets to generate yield, with timelocked governance, dual-track reward distribution, and share-dilution fees.
Type	Vault
Auditors	jolyon, s3cunda
Timeline	Mon Mar 02 2026 - Thu Mar 19 2026
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/naviprotocol/navi_vault
Commits	e100ac4b7d470774bf7a442d10ec4a7130dcd975 84dec6b109b59aaafeaba3a358148221d28fc581 51949069226143eed0e8b906567636c5ccd6f136 82346bffc71fe06ce3175c05b105679b7bdb188b

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
EVE1	sources/events.move	57472cb6cb1a8957395ca9a65fb89d4bb4927092
NVA	sources/navi_vault.move	cf9c1c552b681819ed5c411da3de7f6a1d8e73f0
VER1	sources/version.move	84ac82206f7119b48f25479b1a8db45157f4bc76

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	7	3	4
Centralization	1	0	1
Critical	0	0	0
Major	0	0	0
Medium	0	0	0
Minor	4	1	3
Informational	2	2	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Navi Protocol to identify any potential issues and vulnerabilities in the source code of the Navi Vault smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 7 issues of varying severity, listed below.

ID	Title	Severity	Status
NVA-2	Vault-Native Reward Index Not Settled Before Rule Disabling	Minor	Fixed
NVA-3	Centralization Risk	Centralization	Acknowledged
NVA-4	Revoked Curator's Pending Proposals Remain Executable	Minor	Acknowledged
NVA-5	The `exec_add_market` Does Not Verify `incentive_v2_address`	Informational	Fixed
NVA-6	Missing Restriction on `set_loss`	Minor	Acknowledged
NVA-7	The `convert_to_shares` Incorrect Edge Case Handling When Vault Is Empty	Informational	Fixed
NVA-9	Missing restriction when setting `penalty`	Minor	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the Navi Vault Smart Contract :

Admin

- `create_vault()` — Create a new vault.
- `add_market_by_admin()` — Add a market to the vault.
- `set_default_market_by_admin()` — Set the default deposit market.
- `set_market_status_by_admin()` — Set a market's status (Active/Disabled).
- `set_management_fee_by_admin()` — Set the management fee rate.
- `set_performance_fee_by_admin()` — Set the performance fee rate.
- `create_reward_rule_by_admin()` — Create or reactivate a reward rule.
- `disable_reward_rule_by_admin()` — Disable a reward rule.
- `deposit_reward_balance_by_admin()` — Deposit vault-native reward tokens.
- `set_reward_rate_by_admin()` — Set the vault-native reward distribution rate.
- `add_allocator()` — Authorize an allocator.
- `add_curator()` — Authorize a curator.
- `set_paused()` — Pause or unpause the vault.
- `claim_management_fee()` — Claim pending management fee shares.
- `claim_performance_fee()` — Claim pending performance fee shares.

Curator

- `propose_add_market()` — Propose adding a new market (24h timelock).
- `exec_add_market()` — Execute the add-market proposal.
- `propose_default_market()` — Propose changing the default market (24h timelock).

- `exec_default_market()` — Execute the default-market proposal.
- `propose_set_market_status()` — Propose changing a market's status (24h timelock).
- `exec_set_market_status()` — Execute the market-status proposal.
- `propose_management_fee()` — Propose changing the management fee (24h timelock).
- `exec_management_fee()` — Execute the management-fee proposal.
- `propose_performance_fee()` — Propose changing the performance fee (24h timelock).
- `exec_performance_fee()` — Execute the performance-fee proposal.
- `set_market_cap_and_penalty_by_curator()` — Update a market's deposit cap and withdrawal penalty.
- `create_reward_rule_by_curator()` — Create or reactivate a reward rule.
- `disable_reward_rule_by_curator()` — Disable a reward rule.
- `deposit_reward_balance_by_curator()` — Deposit vault-native reward tokens.
- `set_reward_rate_by_curator()` — Set the vault-native reward distribution rate.

Allocator

- `allocate()` — Move idle funds into a NAVI market.
- `deallocate()` — Withdraw funds from a NAVI market back to idle balance.

User

- `deposit()` — Deposit assets into the vault and receive shares.
- `withdraw()` — Withdraw assets from the vault by burning shares.
- `claim_reward()` — Claim accrued reward tokens.
- `sync_market_balance()` — Sync a market's balance from the underlying NAVI protocol.
- `collect_reward()` — Harvest market rewards from the NAVI protocol into the vault.

4 Findings

NVA-2 Vault-Native Reward Index Not Settled Before Rule Disabling

Severity: Minor

Status: Fixed

Code Location:

sources/navi_vault.move

Descriptions:

When `apply_disable_reward_rule()` sets `rule.is_active = false` without first calling `update_vault_native_indices()`.

sources/navi_vault.move

```
fun apply_disable_reward_rule<CoinType, RewardCoinType>(
    self: &mut Vault<CoinType>,
    navi_pool_id: address,
) {
    let reward_coin_type = type_name::with_defining_ids<RewardCoinType>().into_string();
    let len = self.reward_rules.length();
    let mut i = 0;
    while (i < len) {
        let rule = self.reward_rules.borrow_mut(i);
        if (rule.navi_pool_id == navi_pool_id && rule.reward_coin_type == reward_coin_type)
{
            if (!rule.is_active) {
                abort E_RULE_ALREADY_INACTIVE
            };
            rule.is_active = false;
            events::emit_reward_rule_disabled(
                object::id_address(self),
                navi_pool_id,
                reward_coin_type,
            );
```

```
            return
        };
        i = i + 1;
    };
    abort E_RULE_NOT_FOUND
}
```

Since `update_vault_native_indices()` skips inactive rules, the reward index growth for [last_harvest_at, disable_time] is never applied to vault_reward_index. When the rule is later re-enabled via `set_reward_rate()`, last_harvest_at is reset to the current timestamp, permanently skipping the unsettled period.

Users who held shares during this interval cannot claim the vault-native rewards they are entitled to.

Suggestion:

It is recommended to settle the vault-native reward index up to the current timestamp before disabling the rule, ensuring all accumulated rewards are recorded and claimable by users.

Resolution:

The team adopted our advice and fixed this issue by ensuring the vault-native reward index is settled before a reward rule is disabled. This ensures all accumulated rewards are accurately accounted for prior to the transition, and window of risk is mitigated by the fact that administrators can bundle reward collection and rule disabling within a single atomic transaction, which can be found at 82346bffc71fe06ce3175c05b105679b7bdb188b.

NVA-3 Centralization Risk

Severity: Centralization

Status: Acknowledged

Code Location:

sources/navi_vault.move

Descriptions:

The protocol relies on multiple privileged roles that can modify vault parameters and control fund flows.

Admin (single globally unique `AdminCap`):

- Set management fee (up to 20% APY) and performance fee (up to 40%) without timelock.
- Add markets, set default market, and change market status without timelock.
- Set bad debt (`loss`) on any market.
- Pause and unpause all user operations.
- Control vault version migration after package upgrades.
- Grant or revoke Allocator and Curator roles.

Curator (bound to a specific vault):

- Propose changes to fee rates, markets, default market, and market status (24h timelock).
- Update market deposit caps and withdrawal penalties without timelock.
- Create, disable, and configure reward rules and distribution rates.

Allocator (bound to a specific vault):

- Move funds between idle balance and NAVI lending markets.

Suggestion:

It is recommended to document these privileged roles and their capabilities transparently for users, and consider adopting a multi-sig or governance mechanism for the `AdminCap`.

Resolution:

The team acknowledged this issue.

NVA-4 Revoked Curator's Pending Proposals Remain Executable

Severity: Minor

Status: Acknowledged

Code Location:

sources/navi_vault.move

Descriptions:

The `remove_curator()` only removes the CuratorCap ID from the valid_caps registry, but does not cancel the revoked curator's pending timelock proposals:

sources/navi_vault.move

```
public fun remove_curator(_: &AdminCap, curators: &mut Curators, cap_id: ID) {
    assert!(curators.valid_caps.contains(cap_id), E_CURATOR_CAP_NOT_FOUND);
    curators.valid_caps.remove(cap_id);
    events::emit_curator_removed(cap_id.to_address());
}
```

Since exec_xxx() functions are permissionless and only verify the timelock expiry without checking any capability, any pending proposal created by the revoked curator can still be executed by anyone after the 24-hour timelock expires:

sources/navi_vault.move

```
public fun exec_management_fee<CoinType>(
    self: &mut Vault<CoinType>,
    timelocks: &mut Timelocks<CoinType>,
    clock: &Clock,
    // no CuratorCap or AdminCap required
) {
    let proposal = timelocks.proposals.remove(key);
    assert_proposal_executable(&proposal, clock);
```

```
    self.apply_management_fee(value, clock);
}
```

This applies to all timelocked operations: fee rate changes, market additions, default market changes, and market status changes. The admin can mitigate by calling cancel_proposal_by_admin() for each pending proposal, but the contract does not enforce or prompt this during curator revocation.

Suggestion:

It is recommended to cancel all pending proposals associated with the revoked curator upon removal, or to add a capability check in exec_xxx() functions to verify the original proposer is still authorized.

Resolution:

The team acknowledged this issue. The permissionless design of exec_xxx() supports automated execution. Admins can mitigate risks by revoking a curator and canceling their proposals within a single atomic transaction. The 24-hour to 7-day timelock ensures sufficient time for admins to review and override inappropriate proposals before execution.

NVA-5 The `exec_add_market` Does Not Verify `incentive_v2_address`

Severity: Informational

Status: Fixed

Code Location:

sources/navi_vault.move

Descriptions:

The `exec_add_market()` verifies the passed-in storage and incentive_v3 objects against the proposal's stored addresses, but does not take an IncentiveV2 parameter or verify `incentive_v2_address`:

sources/navi_vault.move

```
public fun exec_add_market<CoinType>(
    self, timelocks, clock,
    storage: &Storage,
    pool: &Pool<CoinType>,
    incentive_v3: &IncentiveV3,
    // IncentiveV2 is not passed in
    ctx,
) {
    assert!(object::id(storage).to_address() == storage_address, ...);      // verified
    assert!(object::id(incentive_v3).to_address() == incentive_v3_address, ...); // verified
    // incentive_v2_address is taken directly from the proposal without verification
}
```

In contrast, `add_market_by_admin()` derives `incentive_v2_address` from the actual IncentiveV2 object, ensuring correctness by construction.

Suggestion:

It is recommended to add `IncentiveV2` as a parameter to `exec_add_market()` and verify its object ID against the proposal's `incentive_v2_address`, consistent with the existing

verification for storage and `incentive_v3`.

Resolution:

The team adopted our advice and fixed this issue by adding a verification mechanism for incentive_v2_address during the proposal execution phase to prevent manual input errors by curators, which can be found at 82346bffc71fe06ce3175c05b105679b7bdb188b.

NVA-6 Missing Restriction on `set_loss`

Severity: Minor

Status: Acknowledged

Code Location:

sources/navi_vault.move

Descriptions:

`set_loss` allows the `Admin` to set an arbitrary loss value for any market with no upper bound, no timelock, and no multi-sig requirement. In `sync_market_balance`, this value is subtracted from current_balance:

sources/navi_vault.move

```
let current_balance = if (raw_balance_native > loss_native) {
    raw_balance_native - loss_native
} else { 0 };
```

A compromised or malicious `Admin` can set loss to an extremely large value, zeroing out `current_balance`, which crashes `total_assets` and devalues all user shares to near zero.

Suggestion:

It is recommended to cap loss to not exceed the market's `current_balance` at the time of setting or require a timelock or multi-sig for `set_loss`.

Resolution:

The team acknowledged this issue and state that:

"`set_loss` is an accounting adjustment tool designed to correct a vault's balance record in the event of bad debt within the NAVI protocol. The three restriction measures suggested by the auditor are not applicable:

Imposing a Cap: Actual bad debt can be any amount. If the loss is capped and the actual debt exceeds that limit, the share price would be artificially inflated. This would allow early

withdrawers to extract value from subsequent participants—a situation more dangerous than having no cap at all. Moreover, the code already includes runtime protection via `min(raw_balance, loss)` in `sync_market_balance`, using saturating subtraction. Even if the set loss exceeds current_balance, the actual effect is simply to bring the market balance to zero, without causing additional harm. The auditor's suggestion to cap the value at the time of setting is redundant—it is already automatically capped at runtime during usage.
Adding a Timelock: Bad debt requires immediate reflection. A 24-hour delay would create a 24-hour arbitrage window, during which informed parties could withdraw at inflated prices. `set_loss` is a parameter that must be adjustable in response to market conditions; imposing a timelock would defeat its intended purpose.
Requiring Multi-Signature: Multi-signature should be implemented at the wallet level (Sui multi-sig wallet) for the `AdminCap`, not enforced at the contract level. Admin authority is the foundation of trust for the entire vault. If the admin is compromised, there are far more efficient attack vectors than set_loss (such as setting a 20% management fee, freezing the vault, adding malicious markets, etc.). Restricting `set_loss` alone does not change the system's trust model. The code itself is secure—the saturating subtraction ensures that current_balance never goes below zero, preventing underflow."
, so no modifications will be made to the current version.

NVA-7 The `convert_to_shares` Incorrect Edge Case Handling When Vault Is Empty

Severity: Informational

Status: Fixed

Code Location:

sources/navi_vault.move

Descriptions:

The `convert_to_shares` returns 0 when `total_assets == 0`.

However, an actual deposit in the same state would yield amount shares (1:1 via shares_to_mint).

The early return is unnecessary since `shares_to_mint` already handles the zero case with VIRTUAL_SHARES. This only affects off-chain queries / frontend previews;

Suggestion:

It is recommended to use `shares_to_mint` handle the calculation directly.

Resolution:

The team adopted our advice and fixed this issue by adding robust handling for edge cases where the vault is empty, ensuring that `convert_to_shares` returns a consistent and mathematically sound value even when total assets or shares are zero, which can be found at 82346bffc71fe06ce3175c05b105679b7bdb188b.

NVA-9 Missing restriction when setting `penalty`

Severity: Minor

Status: Acknowledged

Code Location:

sources/navi_vault.move

Descriptions:

`set_market_cap_and_penalty_by_curator` allows a `Curator` to instantly set a market's penalty up to `WAD` (100%) without any timelock.

sources/navi_vault.move

```
public fun set_market_cap_and_penalty_by_curator<CoinType>(
    self: &mut Vault<CoinType>,
    curators: &Curators,
    curator_cap: &CuratorCap,
    pool_address: address,
    cap: u64,
    penalty: u64,
) {
    version_verification(self);
    assert!(curators.valid_caps.contains(object::id(curator_cap)),
E_CURATOR_CAP_NOT_VALID);
    self.assert_curator_cap_for_vault(curator_cap);
    self.set_market_cap_and_penalty_internal(pool_address, cap, penalty);
}

fun set_market_cap_and_penalty_internal<CoinType>(
    self: &mut Vault<CoinType>,
    pool_address: address,
    cap: u64,
    penalty: u64,
) {
    assert!(vec_map::contains(&self.markets, &pool_address), E_MARKET_NOT_FOUND);
    assert!(penalty <= WAD, E_INVALID_PENALTY);
```

```
    let info = vec_map::get_mut(&mut self.markets, &pool_address);
    info.cap = cap;
    info.penalty = penalty;
    events::emit_set_market_cap_and_penalty(object::id_address(self), pool_address, cap,
penalty);
}
```

In `shares_to_burn`, a 100% penalty doubles the shares burned for the market portion of a withdrawal.

sources/navi_vault.move

```
let penalty_shares = if (penalty > 0 && market_amount > 0) {
        let market_amount_u128 = market_amount as u128;
        let market_numerator = market_amount_u128 * shares_supply + total_assets_u128 -
1;
        let market_base_shares = (market_numerator / total_assets_u128) as u128;
        let penalty_u128 = penalty as u128;
        let wad_u128 = WAD as u128;
        assert!(market_base_shares <= (MAX_U128 - wad_u128 + 1) / penalty_u128,
E_OVERFLOW);
        let penalty_numerator = market_base_shares * penalty_u128 + wad_u128 - 1;
        let penalty_shares_u128 = penalty_numerator / wad_u128;
        assert!(penalty_shares_u128 <= (MAX_U64 as u128), E_OVERFLOW);
        penalty_shares_u128 as u64
    } else {
        0
    };
```

A malicious or compromised `Curator` could set `penalty = WAD` on a non-default market, causing users withdrawing from that market to lose approximately half their assets.

Suggestion:

It is recommended to set a maximum penalty cap or require a timelock for penalty changes above a threshold.

Resolution:

The team acknowledged this issue and state that:

"Hard Upper Limit: All code paths that set the penalty include an `assert!(penalty <= WAD)` check, where `WAD = 10^18`, representing 100%. At the contract level, it is guaranteed that the penalty cannot exceed 100%.

Curator as a Trusted Role: The `CuratorCap` is granted by the admin and is not accessible to arbitrary external users. Setting the penalty is part of the curator's regular operational duties. If a curator is compromised, the appropriate response is to revoke the `CuratorCap`, rather than adding a timelock to every curator action.

Timelock Reduces Flexibility: Penalty parameters may need to be adjusted in response to market conditions. Introducing a timelock would delay such routine parameter updates, while offering limited security benefits—since a compromised curator could still manipulate other parameters such as allocation or market status.

Penalty Only Applies to Non-Default Market Withdrawals: Withdrawals from the default market are not subject to any penalty. Users can review the current penalty before choosing a withdrawal path, making it a transparent protocol parameter rather than an implicit risk."

, so no modifications will be made to the current version.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.