Quantstamp

Dipcoin Vault

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Vault
Timeline	2025-11-24 through 2025-12-04
Language	Move
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• dipcoinlab/dipcoin-perpetual #293ae64
Auditors	• Hamed Mohammadi Auditing Engineer • Rabib Islam Senior Auditing Engineer • Tim Sigl Auditing Engineer

Documentation quality	Low
Test quality	Medium
Total Findings	22 Fixed: 12 Acknowledged: 8 Mitigated: 2
High severity findings	7 Fixed: 5 Acknowledged: 1 Mitigated: 1
Medium severity findings	5 Fixed: 3 Acknowledged: 2
Low severity findings	8 Fixed: 2 Acknowledged: 5 Mitigated: 1
Undetermined severity findings	0
Informational findings	2 Fixed: 2

Summary of Findings

The **Dipcoin Vault** module enables the creation of multiple vaults for capital allocation within the **Dipcoin** perpetual futures exchange markets. The module allows Dipcoin administrators to define a global framework and configuration parameters that any user can leverage to create and fund new vaults after paying the required vault creation fee. Once created, these vaults may be funded by other users, and the deposited capital can then be allocated across various Dipcoin futures exchanges to generate returns.

The protocol supports several enforceable constraints and configuration options:

- **Global** `max_cap` — sets the maximum amount of funds any vault can hold.
- **Global** `lock_period_ms` — prevents immediate withdrawal after deposits.
- **Global** `status` — allows pausing all operations across all vaults.
- **Per-vault** `creator_minimum_share_ratio` — ensures vault creators maintain ownership of a minimum share ratio in their own vault.
- **Per-vault** `creator_profit_share_ratio` — allows creators to charge performance-based fees.
- **Per-vault** `enable_direct_withdraw` — enables or disables asynchronous withdrawals.
- **Per-vault** `deposit_status` — allows pausing or resuming new deposits.
- **Per-vault** `min_deposit_amount` — enforces a minimum allowable deposit.

During the audit, we identified numerous issues, several of which posed a significant impact on protocol operations and security, and were classified as high severity.

Fix-Review Update 2025-12-13:
During the fix review, several issues were resolved or mitigated, while the remaining ones were acknowledged with supporting explanations. The set of mitigated and acknowledged findings includes two High severity issues, DIP-5 and DIP-6.

Overall, considerable effort was invested in improving the codebase and addressing the identified issues. The majority of concerns with potential negative impact on vault operations and safety have been handled. Looking ahead, the codebase could benefit from additional refinements, such as adopting Virtual Shares and Decimals Offset, as well as gradually strengthening areas where issues were mitigated or acknowledged, particularly those of High or Medium severity.

ID	DESCRIPTION	SEVERITY	STATUS
DIP-1	Incorrect Share Price Calculation During Withdrawal Requests	● High	Fixed
DIP-2	Vault Insolvency due to Untracked Creator Shares	● High	Fixed
DIP-3	Denial of Service in Several Functions Due to Object Limits	● High	Fixed
DIP-4	Incorrect Profit Sharing Logic in `distribute_funds()` Function Leads to Fee Leakage	● High	Fixed
DIP-5	NAV Calculation Using Oracle Spot Price Instead of Perpetual Mark Price	● High	Acknowledged
DIP-6	Stale Oracle Price Leads to System-Wide DoS	● High	Mitigated
DIP-7	Vaults Susceptible to Inflation Attacks	● High	Fixed
DIP-8	Double-Counting of Assets Allows Vault Drainage in Delisted Markets	● Medium	Fixed
DIP-9	Denial of Service on Newly Created Vaults by Manager	● Medium	Fixed
DIP-10	Vault Denial of Service if Creator Exits in Manager-Created Vaults	● Medium	Fixed
DIP-11	Unrestricted Lock Period Updates Can Break Depositor and Vault Creator Assumptions	● Medium	Acknowledged
DIP-12	Vault Name Reuse Enables Impersonation and Fund Theft	● Medium	Acknowledged
DIP-13	Changing Trader Can Revoke Operator Access	● Low	Fixed
DIP-14	`VaultNAV` Calculation Exceeds Object Limits as Market Count Grows, Causing Denial of Service for All Vault Operations	● Low	Acknowledged
DIP-15	Lack of Slippage Protection on Deposits Exposes Users to Front-Running	● Low	Acknowledged
DIP-16	Lack of Slippage Protection on Withdrawals	● Low	Acknowledged
DIP-17	Orphaned Perpetual Positions due to Incomplete Closure Check	● Low	Fixed
DIP-18	Consider Sanitizing Vault Names	● Low	Acknowledged
DIP-19	Input Parameter Validation	● Low	Mitigated
DIP-20	Lack of Deadline Protection for Withdrawal Requests	● Low	Acknowledged
DIP-21	Dust Withdrawals Allow Request Spam and Fee Avoidance	● Informational	Fixed
DIP-22	Redundant `VaultNAV` Calculation in Withdrawal Requests	● Informational	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

The scope of this audit is limited to the following file:

- `contracts/sources/vault.move`

Operational Considerations

- **Potential for Misconfiguration:** There are no hardcoded maximum or minimum limits for global configuration values; the exchange admin and exchange manager must ensure that `vaultConfig` values are set correctly.
- **Withdrawal requests:** Either the platform operator or the vault creator must gradually fulfill withdrawal requests.
- **Vault status abuse potential:** The exchange manager must not abuse the ability to freeze the entire protocol by resetting the status flag.
- **Capital allocation:** The platform operator and vault trader must not drain the vaults through malicious capital allocation.

Key Actors And Their Capabilities

Exchange Admin (`ExchangeAdminCap`)

Responsibilities:

- **Global vault configuration initialization:** Module initialization by creating global vault configurations via the `VaultConfig` object.

Trust Assumptions: Root of trust: If compromised, it can be exploited to create a malicious `VaultConfig` object, potentially leading to the initialization of new vaults with improper configurations.
Remarks: Exercises functionality through the `create_vault_config()` function, accessible from other modules within the protocol.

Exchange Manager (`ExchangeManagerCap`)

Responsibilities:

- **Global vault configurations:** Reset vault configuration properties including `operator`, `status`, `max_cap`, `creation_fee`, `fee_pool`, and `lock_period_ms`.

- **Vault creation:** Create new uncapped vaults without requiring an initial deposit or creation fee.

Trust Assumptions: High trust: Can arbitrarily update various global vault configuration parameters, which can affect all vaults.

Platform Operator (`VaultConfig.operator`)

Responsibilities:

- **Capital allocation:** Manages the allocation, transfer, and tracking of all vaults' assets across different perpetual markets through `SubAccounts` .
- **Process withdrawal requests:** Handles queued withdrawal requests for various vaults.
- **Vault Removal:** Deletes **closed** vaults to free up storage.

Trust Assumptions: High trust: If compromised, it can be exploited to allocate funds maliciously or process unauthorized withdrawal requests, potentially draining all the vaults.

Vault Creator (`Vault.creator`)

Responsibilities:

- **Vault creation:** Any user can create a new vault by submitting an initial deposit and paying the vault creation fee. The creator then assumes the role of vault creator for the newly created vault.
- **Process withdrawal requests:** Handles queued withdrawal requests for their vault.
- **Vault configurations:** Reset vault properties including `trader` , `deposit_status` , `enable_direct_withdraw` , `max_cap` , and `min_deposit_amount` .
- **Funds distribution:** Distribute funds, such as profits and dividends, proportionally to all vault depositors.
- **Vault closer:** Can close their own vault.

Trust Assumptions: Originally untrusted: anyone can create a vault; after creation, cannot be transferred.

Vault Trader (`Vault.trader`)

Responsibilities:

- **Capital allocation:** Manages the allocation, transfer, and tracking of a single vault's assets across different perpetual markets through `SubAccounts` .

Trust Assumptions: Medium trust: if compromised, it can be abused to misallocate a single vault's funds.

Vault Depositors (Regular Users)

Responsibilities:

- Can deposit and withdraw funds.

Trust Assumptions: None.

Findings

DIP-1
Incorrect Share Price Calculation During Withdrawal Requests

● High ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `230ad6bd8693581099af2dcfb2f054299b59ce4a` .

```
This issue was independently identified by both the audit team and the Dipcoin team as part of their
internal review processes.
```

File(s) affected: `vault.move`

Description: The `request_withdraw()` function immediately decrements `vault.total_shares` by the requested amount. However, the assets backing those shares remain in the vault (in the Bank or open positions) until the operator calls `fill_withdrawal_requests()` . During this interim period, `update_share_price()` calculates the price as: `price = (Bank Balance + Position Value) / total_shares` . Since `total_shares` has been reduced but the numerator (Assets) has not, the share price is artificially inflated. This allows the withdrawing user (or subsequent users) to extract more value than they are entitled to, effectively stealing from remaining shareholders.

Exploit Scenario:

1. Vault has 1000 USDC and 100 Shares. Price = 10 USDC.
2. User A requests withdrawal of 10 Shares (worth 100 USDC).
3. `total_shares` becomes 90. Assets remain 1000 USDC.
4. Operator calls `fill_withdrawal_requests()`.
5. `update_share_price()` is called: `1000 / 90 = 11.11 USDC`.
6. User A is paid: `10 shares * 11.11 = 111.11 USDC`. 7. User A steals 11.11 USDC from the vault.

Recommendation: Do not decrement `total_shares` in `request_withdraw()`. Instead, only decrement it in `fill_withdrawal_requests()` when the assets are actually paid out. Alternatively, modify `update_share_price()` to use `total_shares + requested_pending_shares` as the denominator.

DIP-2 Vault Insolvency due to Untracked Creator Shares

● High ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `230ad6bd8693581099af2dcfb2f054299b59ce4a`.

```
This issue was independently identified by both the audit team and the Dipcoin team as part of their
internal review processes.
```

File(s) affected: `vault.move`

Description: When a withdrawal request is filled via `fill_withdrawal_requests()`, the protocol calculates a profit share fee (`share_to_creator`) and adds these shares to the creator's `Position`. However, the protocol fails to increment `vault.total_shares` to reflect this new issuance.

This creates a permanent accounting mismatch where the sum of all user shares exceeds the `total_shares` tracked by the vault. `Sum(User Shares) > Vault Total Shares`

Because `total_shares` (the denominator) is smaller than the actual number of outstanding shares, the share price is permanently artificially inflated. This guarantees that the vault will eventually become insolvent, as the last users to withdraw will find there are insufficient assets to cover their shares at the inflated price.

Exploit Scenario:
1. Vault has 100 USDC and 100 Shares. Price = 1.0.
2. User withdraws. Fee calculation awards 1 Share to the Creator.
3. Creator's balance increases by 1.
4. `vault.total_shares` is NOT increased.
5. Total Claims = 101 Shares. Total Tracked Shares = 100.
6. Price = 100 USDC / 100 Shares = 1.0.
7. If everyone tries to withdraw: 101 Shares * 1.0 = 101 USDC needed.
8. Vault only has 100 USDC. It is insolvent.

Recommendation: In `fill_withdrawal_requests()`, immediately after adding shares to the creator's position, increment `vault.total_shares` by `share_to_creator`.

DIP-3 Denial of Service in Several Functions Due to Object Limits

● High ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `88027bf820e6052fa9ce956b543e6dd12bae7ebb`, `8036b79578ba123c2bf809cb68ce187d27685ab2`.

```
The distribute_funds() function was identified for removal during the audit. The Dipcoin team
confirmed that it had already been planned for removal prior to the initial report.
```

File(s) affected: `vault.move`

Description: There are several functions within the vault that perform various batch operations with potentially unbounded batch sizes.

1. The `distribute_funds()` function (and by extension `close_vault()`, which calls it) iterates over the entire `vault.users` list to distribute dividends. Inside the loop, it performs two dynamic field accesses per user:
    - `vault.user_positions.borrow(*user)`: Accesses the user's position in the Vault table.
    - `bank.transfer(...)`: Accesses the user's account in the Bank table.
2. The `set_vault_config_operator()` function iterates over all existing vaults to reset the operator address in their `SubAccounts`, which results in dynamic field accesses under the hood.

Any of these operations can be exploited by attackers or become problematic through natural protocol growth as new vaults are created and new users join.

Aside from the gas cost of these operations possibly exceeding gas limit constraints, Sui enforces a strict limit of 1,000 dynamic field accesses per transaction (see Move Book). This means that any of these functions can easily reach this 1,000 dynamic field access limit and trigger denial-of-service (DoS) issues.

Recommendation:

1. **Remove** `distribute_funds()`: As noted, users can simply withdraw to realize gains. Push-based token distribution is an anti-pattern.
2. **Refactor** `close_vault()`: Instead of pushing funds to all users, `close_vault()` should simply set the vault state to `Closed`. Users would then claim their share of the remaining assets via a new `claim_closed_vault_funds()` function (pull pattern).
3. **Refactor** the permission update logic in `set_vault_config_operator()` function to be asynchronous or paginated. Instead of updating all vaults in a single transaction, allow the operator to be updated in batches or implement a lazy update mechanism where permissions are checked and updated only when a vault is next interacted with.

DIP-4
Incorrect Profit Sharing Logic in `distribute_funds()` Function Leads to Fee Leakage

• High ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `8036b79578ba123c2bf809cb68ce187d27685ab2`.

```
The distribute_funds() function was identified for removal during the audit. The Dipcoin team
confirmed that it had already been planned for removal prior to the initial report.
```

File(s) affected: `vault.move`

Description: The `distribute_funds()` function calculates the creator's profit share by treating the distributed amount as a pro-rata slice of the user's position (containing both principal and profit in proportion to the current share price). However, unlike a withdrawal, the distribution does not reduce the user's share count or adjust their cost basis (`average_price`). This leads to a mathematical flaw where the "principal portion" of the distribution is returned tax-free, but the user's cost basis remains high. This effectively erases the "unrealized profit" from the share price without taxing it.

Exploit Scenario:

1. User deposits $100 (100 shares at $1).
2. Vault makes $100 profit. Price = $2.
3. Creator distributes $100 (the profit).
4. Code calculates: Distribution is 50% of value ($200). So it represents 50 shares.
5. Profit on 50 shares = $50. Fee = 10% * $50 = $5.
6. User receives $95.
7. Vault Share Price drops to $1 (Assets $100 / 100 Shares).
8. User's remaining unrealized profit is $0 ($1 Price - $1 Entry).
9. Total Fee Paid = $5. Correct Fee on $100 profit should be $10.
10. Creator loses 50% of their fees.

Recommendation: Since the Dipcoin team stated that `distribute_funds()` will be removed, no remediation is required.

DIP-5
NAV Calculation Using Oracle Spot Price Instead of Perpetual Mark Price

• High ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
We choose the oracle price based on the following considerations:
1. Mark price is more susceptible to manipulation than oracle price, especially when liquidity is
   insufficient in the early stages.
2. The oracle price is more transparent than the mark price.
3. The oracle price is more representative of the fair price. Any deviation of the market price from
   the oracle price is only temporary and will regress to the oracle price. When withdrawing funds
   while the market price is lower than the oracle price, the share_price refers to the price obtained
   after position closure (which already accounts for the price deviation) and will not cause losses to
   the remaining depositors.
```

File(s) affected: `vault.move`

Description: The vault's Net Asset Value (NAV) is calculated by valuing its perpetual contract positions using a price feed from the Pyth oracle. However, Pyth oracles provide the *spot price* of an underlying asset (e.g., the current market price of BTC), not the *mark price* of a specific perpetual contract trading on an exchange. In derivatives markets, the perpetual price can significantly deviate from the spot price – a difference known as the basis. By using the spot price for valuation, the contract incorrectly calculates the unrealized PnL of its positions, leading to a flawed NAV and an exploitable share price.

Exploit Scenario:

1. An attacker waits for a period of significant market backwardation, where the price of a perpetual contract (`BTC-PERP`) is trading at a substantial discount to the spot price of the underlying asset (`BTC`). For example, `BTC-PERP` might be at $48,000 while the Pyth oracle reports a spot price of $50,000.
2. The vault holds a large, long `BTC-PERP` position. The *true* market value of this position is based on the $48,000 price.
3. The attacker, who is an existing depositor, triggers a withdrawal. The `compute_perpetual_position_value` function is called and uses the *higher* spot price of $50,000 from the Pyth oracle to calculate the position's value.
4. This miscalculation artificially inflates the vault's NAV, resulting in an overstated share price. The attacker's shares are now worth more than they should be, and when they withdraw, they receive an excess amount of the underlying assets, effectively stealing the difference in value from the remaining depositors in the vault.

Recommendation: The NAV calculation in `compute_perpetual_position_value()` must be refactored to use the perpetual contract's own internal mark price for valuing positions, not the external spot price from the oracle. The spot oracle should only be used as a reference for liquidations and for calculating the funding rate, which are its correct functions in a perpetuals system.

DIP-6 Stale Oracle Price Leads to System-Wide DoS

● High ⓘ Mitigated

Update

Marked as "Mitigated" by the client.
Addressed in: `9751237437bb737c7ccc3057ce4edc0e0e5efa21`.
The client provided the following explanation:

```
Currently, our system is highly dependent on oracle prices. If the oracle of a certain market is stale,
the entire market will be unable to conduct transactions, and calculating position values based on this
state will be meaningless. In such a case, the best approach is to suspend the corresponding
operations(deposit, withdraw, etc.). Meanwhile, we have also made a modification to skip price updates
for markets with no open positions, thereby avoiding being affected by irrelevant stale markets.
```

File(s) affected: `vault.move`

Description: The vault requires that NAV be calculated using fresh prices for every single perpetual market in the system before any financial operation can proceed. If the oracle for even one, potentially minor, market becomes stale (older than 60 seconds), the `pyth::get_price_older_than` call reverts, causing the entire transaction to fail. This effectively freezes all deposits, withdrawals, and financial management for every user of the vault, regardless of whether their funds are involved with the affected market.

Recommendation: The protocol manager should have the ability to temporarily mark a specific perpetual market as "paused" or "suspended" if the oracle is unresponsive. The NAV calculation should then be modified to gracefully handle this state, for instance, by calculating the NAV based only on the remaining active markets and temporarily disallowing interactions that would be affected by the paused market. As mentioned in DIP-5, a better solution would be not to use oracle prices for NAV calculations altogether.

DIP-7 Vaults Susceptible to Inflation Attacks

● High ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `ab4b8c92315ae3ab4c2ac35efd7c968d756f2c53`, `54cfac7a02988ba86734d12207a321b748cd5128`, and `2db007f5585f8b73a46e000a630b17c097f76e8d`.
The client provided the following explanation:

```
We have a protocol-level MIN_DEPOSIT_AMOUNT limit, which is currently set at 10 USDC, and this can
prevent the issue.
```

File(s) affected: `Vault.move`

Description: The vault share price is calculated by dividing the sum of all vault assets across different perpetuals, along with the vault balance in the `Bank` module, by `vault.total_shares`. Since funds can be donated to the vault via various `Bank` functions, a malicious actor could artificially inflate the vault's assets, increasing their share value and potentially draining the vault.

Exploit Scenario:

1. Creator creates a vault with 1 MIST deposit (shares=1).
2. Creator frontruns victim's transfer to transfer 10,000 USDC to the vault's bank account.
3. Share price becomes ~10,000 USDC per share unit.
4. Victim deposits 15,000 USDC.
5. Victim receives `15,000 / 10,000 = 1` share unit.
6. Victim's position value is 10,000 USDC.
7. Victim loses 5,000 USDC (33% loss).
8. Creator withdraws their 1 share. Since they are the creator, they pay no profit sharing fees and capture the full value of the donation + the victim's loss.

Recommendation: Implement a robust defense by combining two strategies as described in OpenZeppelin ERC-4626:

1. **Virtual Shares (Offsetting the Denominator)**: Add a virtual offset to the share supply calculation to dampen the effect of early donations.
2. **Decimal Offset (Increasing Precision)**: Increase the internal precision of shares (e.g., to 1e18 or higher) relative to the asset's decimals.

This reduces the impact of rounding errors. If shares have 9 extra decimals of precision, a 1 MIST donation only inflates the price by a tiny fraction, making the attack prohibitively expensive. Implementing both provides the strongest defense against rounding losses and inflation attacks.

DIP-8 Double-Counting of Assets Allows Vault Drainage in Delisted Markets

• Medium ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `255e35036bd3437543cff9316573b8a4f5416d48` , and `6530e67a6481c59ce0cecdece079e11f5e9c7370` .

File(s) affected: `vault.move`

Description: The `VaultNAV` struct acts as a temporary snapshot of unrealized position values, while `withdraw()` adds this snapshot to the live Bank balance. This creates a race condition within a Programmable Transaction Block (PTB) where a trader can snapshot unrealized profits, close the position to realize them into the Bank, and then withdraw. This results in the same value being counted twice—once as position value and once as cash—artificially inflating the share price. This attack is only feasible in delisted markets, as `exchange::close_position()` is restricted to delisted perpetuals. For live markets, trading is permissioned (requires `SettlementCap`), preventing the vault creator from atomically closing positions in the same PTB unless they also control the settlement operator.

Exploit Scenario:
1. A perpetual market is delisted.
2. The Vault creator calls `new_vault_nav()` and `compute_perpetual_position_value()` to record the position's value in `VaultNAV` .
3. In the same PTB, the creator calls `exchange::close_position()` , which is allowed for delisted markets. This moves the position's value to the Vault's Bank.
4. Attacker calls `vault::withdraw()` . The contract calculates `NAV = VaultNAV (Position Value) + Bank (Realized Value)` .
5. The assets are counted twice, allowing the attacker to withdraw more than their fair share, potentially draining the vault.

Recommendation: Snapshot the Bank balance at the same time `VaultNAV` is created and store it within the struct. Use this snapshotted balance for share price calculations instead of `bank.get_balance()` .

DIP-9 Denial of Service on Newly Created Vaults by Manager

• Medium ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `230ad6bd8693581099af2dcfb2f054299b59ce4a` .

```
This issue was independently identified by both the audit team and the Dipcoin team as part of their
internal review processes.
```

File(s) affected: `vault.move`

Description: The `create_vault_by_manager()` function allows creating a new vault without an initial deposit, resulting in a vault with 0 shares and 0 assets. Other vault operations, such as `deposit()` , call `update_share_price()` , which performs a division by `vault.total_shares` . For vaults with 0 shares, this leads to a division-by-zero error, rendering the vault unusable.

Recommendation: Consider mitigating this by either:
1. Initializing new vaults with a small number of **virtual** shares.
2. Modifying the `update_share_price()` function, by adding a check to handle the case where total_shares is zero. If it is zero (indicating an initial deposit), the share price should be fixed at a default value (e.g., 1.0, which is `library::base_uint()`) instead of being calculated

via division.

DIP-10
Vault Denial of Service if Creator Exits in Manager-Created Vaults

● Medium ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `f615d39764979bcb1ed5e2facc3556702c0f8a11`.

File(s) affected: `vault.move`

Description: In manager-created vaults, the `creator_minimum_share_ratio` can be set to 0, allowing the creator to withdraw 100% of their shares. When a user's share balance reaches 0, their `Position` object is removed from the `user_positions` table.

However, the `withdraw_preload()` function and `fill_withdrawal_requests()` function both contain logic that unconditionally attempts to borrow the creator's position to distribute profit sharing fees:

```
if (user != vault.creator) {
    vault.user_positions.borrow_mut(vault.creator).add_shares(share_to_creator, share_price);
};
```

If the creator has exited the vault (removed their position), this `borrow_mut` call will fail, causing the transaction to abort.

Recommendation: Check if the creator's position exists before attempting to add shares. If the position is missing, handle the fee distribution gracefully (e.g., re-create the position or skip fee distribution).

DIP-11
Unrestricted Lock Period Updates Can Break Depositor and Vault Creator Assumptions

● Medium ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
For all parameter changes to VaultConfig, we will have a notification period, and will not modify
system parameters arbitrarily or abruptly.
```

File(s) affected: `vault.move`

Description: The `set_vault_config_lock_period()` function allows the exchange manager to modify `vault_config.lock_period_ms` at any time without restriction. Since this lock period applies directly to user withdrawals, sudden updates can break depositors' assumptions and result in unexpected lockups. Furthermore, because the lock period is defined globally rather than per vault, changing it can also disrupt assumptions made by vault creators, potentially affecting their strategy design or user expectations.

Recommendation: To mitigate these risks, consider the following improvements:

1. Introduce a grace period before applying the new `lock_period_ms`, allowing depositors who wish to exit to withdraw before the change takes effect.
2. Apply the updated `lock_period_ms` only to new deposits. This can be achieved by extending the `Position` struct with a `Table` that records each deposit amount alongside the lock period that was active at the time of deposit.
3. Clearly document the risks for both depositors and vault creators, explicitly stating that the manager can update `lock_period_ms` at any time and to any value.

DIP-12
Vault Name Reuse Enables Impersonation and Fund Theft

● Medium ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
For on-chain users, they are capable of and should understand that vault_id is the unique identifier.
For DAPP users, in addition to displaying vault_id, we will also present key information such as vault
creation time, TVL, and performance, and can also issue warnings about vault name reuse. Furthermore,
in our design, remove_vault is not executed immediately after a vault is closed; this is only a
reserved capability.
```

File(s) affected: `vault.move`

Description: The protocol does not prevent the reuse of a vault's name after the original vault has been closed and removed. When `remove_vault()` is called, it completely deletes the vault's name from the `name_to_id` registry. This allows a malicious actor to immediately create a new vault using the exact same name as a previously reputable and successful vault. Unsuspecting users, relying on off-chain information, social media, or outdated frontend data, may then deposit funds into the new, malicious vault, believing they are investing in the original. The attacker can then steal these deposited funds.

Recommendation: Modify the `VaultRegistry` to maintain a permanent record of all vault names that have ever been used. Instead of deleting the name from the `name_to_id` table, the remove_vault function should move the name to a new `deprecated_names` table or `VecSet`. The `create_vault()` and `create_vault_by_manager()` functions must then be updated to check against this list of deprecated names, ensuring that a vault name, once used, can never be registered again. This guarantees the uniqueness and integrity of a vault's identity throughout the protocol's entire history.

DIP-13 Changing Trader Can Revoke Operator Access

● Low ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `73a9c4c08bb0cfb9accdf2931157ca4aea0380b5`.

File(s) affected: `vault.move`

Description: The `set_trader()` function revokes the `old_trader`'s sub-account access without checking if that address is also the designated `vault_config.operator`. In the `SubAccounts` module, authorization is binary (is authorized or not) and does not track roles. If the `old_trader` address happens to be the same as the `operator` address, `set_trader()` will remove the operator from the allowlist. This locks the operator out of the vault, preventing them from performing tasks like `fill_withdrawal_requests()`.

Recommendation: In `set_trader()`, add a check to ensure `old_trader` is not the `vault_config.operator` before revoking access. Note: Simply preventing `trader == operator` is insufficient because the collision could also be created by `set_vault_config_operator()`. The check-before-revoke method handles collisions from both sources.

DIP-14 `VaultNAV` Calculation Exceeds Object Limits as Market Count Grows, Causing Denial of Service for All Vault Operations

● Low ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
For a considerable period of time, the number of contract markets we plan to launch will not exceed 50,
the current solution is sufficient for use. In addition, the solution you recommended has potential
security risks of being attacked, as traders' opening/closing positions operations are not synchronized
with the register_market process.
```

File(s) affected: `vault.move`

Description: The `VaultNAV` logic iterates through every perpetual market registered in the `ProtocolConfig` to calculate the Vault's total Net Asset Value (NAV). This occurs in `new_vault_nav()` and `compute_perpetual_position_value()`.

Sui enforces a strict limit of **1,000 unique dynamic field accesses** per transaction block (PTB).

For each perp market, `compute_perpetual_position_value()` performs the following operations:

1. `nav.position_values.contains(perp_id)` & `nav.position_values.add(perp_id, ...)`: Accesses the unique field for this perp in the NAV table.
2. `perp.has_position(vault_address)` & `perp.borrow_position(vault_address)`: Accesses the unique field for this vault in the Perpetual's position table.

Once the protocol deploys approximately **500 markets**, the transaction will hit the 1,000 unique dynamic field limit. Since `deposit()`, `withdraw()`, `distribute_funds()`, and `close_vault()` all require `VaultNAV`, all user funds will be locked in every Vault.

Recommendation: Refactor the `Vault` to track its own "Active Markets" instead of iterating the global list.

1. **Add Registry**: Add a `VecSet<ID>` field to the `Vault` struct to store `enabled_markets`.

2. **Update NAV**: Modify `VaultNAV` to iterate only over `vault.enabled_markets` .
3. **Permissionless Registration**: Add a function `register_market(vault, perp)` that adds a market to the Vault's list.
4. **Anti-Exploit Mechanism**: To prevent a malicious trader from hiding losing positions (by not registering the market), implement a permissionless `force_register()` function. This allows any user (or a keeper bot) to prove the Vault has a position in an unregistered market, forcing it into the list and ensuring the NAV remains accurate.

DIP-15 Lack of Slippage Protection on Deposits Exposes Users to Front-Running

• Low ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
Although an attacker engages in front-running a large deposit, the share price will not change, and
subsequent depositors will not receive fewer shares. Since we use the oracle spot price to calculate
the NAV, which is extremely difficult for attackers to manipulate. This finding is almost impossible to
occur.
```

File(s) affected: `vault.move`

Description: The `deposit()` function calculates the number of shares a user receives based on the vault's NAV at the start of the transaction. An attacker can exploit this by front-running a large deposit and executing a trade that artificially inflates the vault's NAV. When the victim's deposit is then processed, it will be valued against this higher NAV, causing them to receive significantly fewer shares than they expected for their assets.

Recommendation: Modify the `deposit()` function to accept a `min_shares_out` parameter. This allows the depositor to specify the minimum number of shares they are willing to accept for their deposit amount, causing the transaction to revert if MEV or other price movements result in an outcome below this threshold.

DIP-16 Lack of Slippage Protection on Withdrawals

• Low ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
Since we use the oracle spot price to calculate the NAV, which is extremely difficult for attackers to
manipulate. This finding is almost impossible to occur.
```

File(s) affected: `vault.move`

Description: When withdrawing funds, a user specifies the number of shares to burn but has no control over the minimum amount of underlying assets they receive, which is based on the `last_share_price` calculated at the start of the transaction. This exposes users to front-running attacks, where an attacker can execute a trade that negatively impacts the vault's NAV just before the withdrawal, causing the victim to receive fewer assets than expected.

Recommendation: Modify the `withdraw()` and `request_withdraw()` functions to accept an additional parameter, `min_amount_out` . This would allow users to specify the minimum amount of assets they are willing to accept for their shares, causing the transaction to revert if the final value falls below this threshold due to slippage or manipulation.

DIP-17 Orphaned Perpetual Positions due to Incomplete Closure Check

• Low ⓘ Fixed

Update
The issue was addressed in: `c5ae76b85744cf25ed6d3cb4e728981dccd3157d` .

File(s) affected: `vault.move`

Description: The `close_vault()` function intends to ensure a vault has no open positions before allowing closure, asserting `nav.position_nav == 0` . However, `nav.position_nav` is calculated in `compute_perpetual_position_value()` using `.positive_value()` , which clamps negative position values (underwater positions) to zero. This means a vault with open, underwater positions (where PnL loss > margin) can be closed and subsequently removed. While this scenario is extremely unlikely—as such positions would typically be liquidated long before a vault closure is attempted—it remains theoretically possible. When `remove_vault()` deletes the `Vault` object, the corresponding `Position` in the `Perpetual` contract remains in storage but becomes permanently inaccessible. If the market price recovers, the locked margin and potential profits in that position are lost forever as the owner (the vault) no longer exists.

Recommendation: In `close_vault()`, instead of relying on `nav.position_nav`, the function should explicitly verify that the vault has no active positions in any market. This requires iterating through all perpetual markets and checking `!perp.has_position(vault_address)`.

Alternatively, update the documentation and function comments to clarify that `close_vault()` is permitted when the vault's Net Asset Value (NAV) is zero, even if open positions exist (provided they have zero value).

DIP-18 Consider Sanitizing Vault Names

• Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.

The client provided the following explanation:

```
For on-chain users, they are capable of and should understand that vault_id is the unique identifier.
For DAPP users, in addition to displaying vault_id, we will also present key information such as vault
creation time, TVL, and performance, and can also issue warnings about similar vault names.
```

File(s) affected: `vault.move`

Description: The `create_vault()` function allows vault creators to provide a name for their vaults in string format. However, this string is not sanitized or validated. This can lead to situations where multiple vaults exist with visually similar names that differ only in capitalization or whitespace, such as `TestVault`, `TestVault`, or `test vault`.

Recommendation: Consider sanitizing vault names by:

1. Ensuring the names are alphanumeric and only allow a defined set of special characters.
2. Trimming whitespace from the beginning and end of the string.
3. Converting all characters to either uppercase or lowercase, if appropriate.

DIP-19 Input Parameter Validation

• Low ⓘ Mitigated

Update

Marked as "Mitigated" by the client.

Addressed in: `0e9a5dec3689ebfb9d60ab77d451b270c533e577`.

The client provided the following explanation:

```
We do not fix no. 5, 6, because it does not affect the usage of existing users and the error is easy to
detect. Adding hard-coded restrictions would make the configuration less flexible.
  We do not fix no. 9, we do not limit that max_cap must be greater than the current vault value. It is
acceptable for max_cap to be lower than the vault value, in which case users can only withdraw funds
and cannot make deposits.
```

File(s) affected: `vault.move`

Description: It is important to validate user-provided inputs, even in administrative functions and for values coming from trusted sources, to prevent human errors and unintended behavior.

1. In the `create_vault()` function, ensure the initial deposit `amount` does not exceed the `max_cap` set for the vault.
2. In the `create_vault()` function, ensure the initial deposit `amount` is not below `min_deposit_amount` set for the vault.
3. In the `deposit_internal()` function, ensure the deposit amount is greater than `vault.deposit_internal` instead of `0`.
4. In `set_vault_config_max_cap()`, `set_vault_config_vault_creating_fee()`, `set_vault_config_fee_pool()`, and `set_vault_config_lock_period()`, ensure the new values are not equal to the existing ones.
5. Consider adding an immutable minimum value for `VaultConfig.max_cap` to prevent very small caps.
6. Consider adding an immutable maximum value for `VaultConfig.vault_creating_fee`.
7. Consider adding an immutable maximum value for `VaultConfig.lock_period_ms`.
8. In the `set_max_cap()` function, ensure the provided `max_cap` value is properly validated against the existing `VaultConfig.max_cap` to prevent redundant or invalid updates.
9. The `set_max_cap()` function allows setting a new `max_cap` for a given vault. However, there is no mechanism in this function to ensure that the provided `max_cap` is not below the sum of existing funds in the vault. Make sure to validate this new `max_cap` value.

Recommendation: Applying above input validations can help prevent misconfigurations, human errors, and potential inconsistencies in vault behavior.

DIP-20 Lack of Deadline Protection for Withdrawal Requests

• Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
The system operator is an automated processing program. Once detecting a request_withdraw event, it
will immediately process the positions and then execute fill_withdrawal_requests. The system operator
has no interests or incentives to block or delay the processing of users' withdrawal requests. In the
current system, users need to trust the system operator, and we will add corresponding explanations in
the operation interface and documentation.
```

File(s) affected: `vault.move`

Description: The `request_withdraw()` function allows users to queue a withdrawal, but it does not allow them to specify a deadline, which causes the request to remain valid indefinitely.

This creates a "hostage" situation where a user requests to withdraw at a certain price, but the operator (or creator) can delay execution until the share price drops. The user has no way to cancel the request or enforce a minimum acceptable price.

Recommendation: Add a `deadline` parameter to `request_withdraw()`. In `fill_withdrawal_requests()`, check if conditions are met and refund the user if not.

DIP-21 Dust Withdrawals Allow Request Spam and Fee Avoidance

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `e23756921e94992791c5ba3dc5badc8ed11b1024`.

File(s) affected: `vault.move`

Description: The `request_withdraw()` and `withdraw()` functions only enforce `shares > 0`. Since shares have 9 decimals of precision (1e9), a user can operate with extremely small "dust" amounts (e.g., 1 MIST or 1e-9 shares). This lack of a minimum threshold leads to two distinct issues:

1. **Request Spam**: A malicious user with a small position (e.g., 1 USDC) can generate dust `UserWithdrawalRequest` shared objects. This floods the network state and creates clutter for off-chain indexers and the operator's UI.
2. **Fee Avoidance**: Users can avoid paying the creator's profit share fee by withdrawing small "dust" amounts. The fee calculation involves multiple integer divisions that round down. If the calculated fee is smaller than 1 share unit, it rounds to 0, allowing the user to keep 100% of the profit.

Exploit Scenario (Fee Avoidance):

- **Creator Share**: 10% (`100,000,000`)
- **Entry Price**: 1 USDC (`1,000,000,000`)
- **Current Price**: 2 USDC (`2,000,000,000`) (100% Profit)
- **Withdrawal**: 19 shares (19 MIST)

Calculation:

1. **PnL**: `(2e9 - 1e9) * 19 / 1e9 = 19` units.
2. **Fee Amount**: `19 * 10% = 1.9` → rounds to `1` unit.
3. **Fee Shares**: `1 * 1e9 / 2e9 = 0.5` → rounds to `0` shares.

Result: User withdraws 19 shares worth 38 units of USDC and pays 0 fees.

Recommendation: Enforce a minimum withdrawal size in `request_withdraw()` and `withdraw()`. This prevents both the creation of spam objects and the avoidance of fees via dust rounding.

DIP-22 Redundant `VaultNAV` Calculation in Withdrawal Requests

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `1050ac699364a3b79a85a9f84ba860e632dc11ae`.

File(s) affected: `vault.move`

Description: The `request_withdraw()` function currently calls `update_share_price()` even though `vault.last_share_price` is not required for its operation. Updating the vault share price involves constructing a `VaultNAV` and performing expensive calculations, which unnecessarily increases gas costs for users.

Recommendation: Consider removing both the `nav` parameter from `request_withdraw()` and the call to `update_share_price()` within its implementation to reduce computational overhead.

Auditor Suggestions

S1 Avoid Using Same Error for Two Different Revert Conditions

Acknowledged

Update
Marked as "Unresolved" by the client.
The client provided the following explanation:

```
This issue has no impact on users, and its impact on developers' troubleshooting is also minimal.
```

The provided explanation above actually acknowledges the issue.

File(s) affected: `vault.move`

Description: Defining distinct error types for each kind of error within a function can aid in troubleshooting by helping narrow down the trigger of the error. However, there are a number of locations in the code that reuse the same error type within a function. Below are instances where this occurs:

- `create_vault_config()`
  - `address_cannot_be_zero()`
    - `operator != @0x0`
    - `fee_pool != @0x0`
- `create_vault_by_manager()`
  - `address_cannot_be_zero()`
    - `trader != @0x0`
    - `creator != @0x0`
- `create_vault_internal()`
  - `invalid_ratio()`
    - `creator_minimum_share_ratio < base_uint`
    - `creator_profit_share_ratio < base_uint`
- `deposit_internal()`
  - `can_not_be_zero()`
    - `amount > 0`
    - `shares > 0`
- `withdraw_preload()`
  - `can_not_be_zero()`
    - `shares > 0`
    - `amount > 0`

Recommendation: Correct the above instances.

S2 `creator_loss_share_ratio` Is Unused

Acknowledged

Update
Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
This is a reserved field for future upgrading. We have documented that.
```

File(s) affected: `vault.move`

Description: The Vault struct includes a field named `creator_loss_share_ratio`, which implies that the vault's creator will share in a portion of the depositors' losses. However, this field is never used in any of the protocol's logic.

Recommendation: Either fully implement the loss-sharing logic that this field implies or remove it entirely from the Vault struct and all related documentation. If implemented, the withdrawal and NAV calculation functions would need to be modified to deduct a portion of any losses from the creator's share balance and distribute it back to the other depositors.

S3 Finalize the TODO

Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `a3a9ab0e0fa470895f912c943c1a966dfe829e42`.

File(s) affected: `vault.move`

Description: There is a TODO in `fill_withdrawal_requests()` to calculate and deduct gas fees for processing user withdrawals, which is currently not implemented.

Recommendation: Consider implementing this feature to correctly charge users for gas costs and remove the TODO from the code.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not pose an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Test Suite Results

The tests were executed using the command `sui move test -p contracts vault --skip-fetch-latest-git-deps` on Sui CLI version `1.61.2-homebrew`. The **vault** module contains 96 test cases, all of which passed successfully.

Fix Review Update During the fix review, the number of tests increased from **96** to **103**. However, despite the added tests, the coverage for the `vault` module slightly decreased from **91.87%** to **87.42%**, and the overall test quality remains unchanged.

```
sui move test -p contracts vault --skip-fetch-latest-git-deps
INCLUDING DEPENDENCY Pyth
INCLUDING DEPENDENCY Wormhole
INCLUDING DEPENDENCY Sui
INCLUDING DEPENDENCY MoveStdlib
BUILDING dipcoin
Running Move unit tests
[ PASS    ] dipcoin::vault_tests::test_close_vault_already_closed
[ PASS    ] dipcoin::vault_tests::test_close_vault_has_pending_withdrawals
[ PASS    ] dipcoin::vault_tests::test_close_vault_has_position
[ PASS    ] dipcoin::vault_tests::test_close_vault_protocol_paused
[ PASS    ] dipcoin::vault_tests::test_close_vault_success
[ PASS    ] dipcoin::vault_tests::test_close_vault_unauthorized
[ PASS    ] dipcoin::vault_tests::test_create_vault
[ PASS    ] dipcoin::vault_tests::test_create_vault_by_manager
[ PASS    ] dipcoin::vault_tests::test_create_vault_by_manager_duplicate_name
[ PASS    ] dipcoin::vault_tests::test_create_vault_by_manager_invalid_profit_share_ratio
[ PASS    ] dipcoin::vault_tests::test_create_vault_by_manager_protocol_paused
[ PASS    ] dipcoin::vault_tests::test_create_vault_by_manager_zero_creator
[ PASS    ] dipcoin::vault_tests::test_create_vault_by_manager_zero_trader
[ PASS    ] dipcoin::vault_tests::test_create_vault_config
[ PASS    ] dipcoin::vault_tests::test_create_vault_duplicate_name
[ PASS    ] dipcoin::vault_tests::test_create_vault_exceeds_max_cap
[ PASS    ] dipcoin::vault_tests::test_create_vault_initial_amount_exceeds_max_cap_should_fail
[ PASS    ] dipcoin::vault_tests::test_create_vault_invalid_creator_minimum_share_ratio
[ PASS    ] dipcoin::vault_tests::test_create_vault_invalid_creator_profit_share_ratio
[ PASS    ] dipcoin::vault_tests::test_create_vault_less_than_minimum_deposit_amount
[ PASS    ] dipcoin::vault_tests::test_create_vault_protocol_paused
[ PASS    ] dipcoin::vault_tests::test_create_vault_zero_shares
[ PASS    ] dipcoin::vault_tests::test_creator_minimum_share_ratio_boundary
[ PASS    ] dipcoin::vault_tests::test_deposit_again
[ PASS    ] dipcoin::vault_tests::test_deposit_amount_too_small
[ PASS    ] dipcoin::vault_tests::test_deposit_bank_id_mismatch
[ PASS    ] dipcoin::vault_tests::test_deposit_by_creator_when_paused
[ PASS    ] dipcoin::vault_tests::test_deposit_creator_share_ratio_below_minimum
[ PASS    ] dipcoin::vault_tests::test_deposit_exceeds_max_cap
[ PASS    ] dipcoin::vault_tests::test_deposit_nav_not_filled
[ PASS    ] dipcoin::vault_tests::test_deposit_protocol_paused
[ PASS    ] dipcoin::vault_tests::test_deposit_success
[ PASS    ] dipcoin::vault_tests::test_deposit_vault_id_mismatch
[ PASS    ] dipcoin::vault_tests::test_deposit_when_paused
[ PASS    ] dipcoin::vault_tests::test_fill_withdrawal_requests_all_not_fill
[ PASS    ] dipcoin::vault_tests::test_fill_withdrawal_requests_by_operator_success
[ PASS    ] dipcoin::vault_tests::test_fill_withdrawal_requests_partial_success
[ PASS    ] dipcoin::vault_tests::test_fill_withdrawal_requests_protocol_paused
[ PASS    ] dipcoin::vault_tests::test_fill_withdrawal_requests_unauthorized
[ PASS    ] dipcoin::vault_tests::test_min_deposit_amount_boundary
[ PASS    ] dipcoin::vault_tests::test_pnl_per_share_negative
[ PASS    ] dipcoin::vault_tests::test_pnl_per_share_positive
[ PASS    ] dipcoin::vault_tests::test_pnl_per_share_zero
[ PASS    ] dipcoin::vault_tests::test_profit_share_ratio_boundaries
[ PASS    ] dipcoin::vault_tests::test_remove_vault_account_exists
[ PASS    ] dipcoin::vault_tests::test_remove_vault_bank_id_mismatch
[ PASS    ] dipcoin::vault_tests::test_remove_vault_not_closed
[ PASS    ] dipcoin::vault_tests::test_remove_vault_success
[ PASS    ] dipcoin::vault_tests::test_remove_vault_unauthorized
[ PASS    ] dipcoin::vault_tests::test_request_withdraw_all_shares
[ PASS    ] dipcoin::vault_tests::test_request_withdraw_by_creator_failed
[ PASS    ] dipcoin::vault_tests::test_request_withdraw_insufficient_shares
[ PASS    ] dipcoin::vault_tests::test_request_withdraw_locked
[ PASS    ] dipcoin::vault_tests::test_request_withdraw_protocol_paused
[ PASS    ] dipcoin::vault_tests::test_request_withdraw_success
[ PASS    ] dipcoin::vault_tests::test_request_withdraw_user_not_exist
[ PASS    ] dipcoin::vault_tests::test_request_withdraw_zero_shares
[ PASS    ] dipcoin::vault_tests::test_set_deposit_status
[ PASS    ] dipcoin::vault_tests::test_set_deposit_status_unauthorized
[ PASS    ] dipcoin::vault_tests::test_set_direct_withdraw_status
[ PASS    ] dipcoin::vault_tests::test_set_direct_withdraw_status_unauthorized
[ PASS    ] dipcoin::vault_tests::test_set_max_cap
[ PASS    ] dipcoin::vault_tests::test_set_max_cap_below_current_value
[ PASS    ] dipcoin::vault_tests::test_set_max_cap_exceeds_global_max_cap
[ PASS    ] dipcoin::vault_tests::test_set_max_cap_invalid_status
[ PASS    ] dipcoin::vault_tests::test_set_max_cap_unauthorized
[ PASS    ] dipcoin::vault_tests::test_set_min_deposit_amount
[ PASS    ] dipcoin::vault_tests::test_set_min_deposit_amount_invalid_status
[ PASS    ] dipcoin::vault_tests::test_set_min_deposit_amount_unauthorized
[ PASS    ] dipcoin::vault_tests::test_set_trader_protocol_paused
[ PASS    ] dipcoin::vault_tests::test_set_trader_success
[ PASS    ] dipcoin::vault_tests::test_set_trader_unauthorized
[ PASS    ] dipcoin::vault_tests::test_set_trader_zero_address
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_fee_pool
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_fee_pool_failed_duplicate
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_fee_pool_zero_address
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_lock_period
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_lock_period_failed_duplicate
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_lock_period_failed_too_long
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_max_cap
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_max_cap_failed_duplicate
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_operator
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_operator_invalid
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_status_failed_duplicate_status
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_status_success
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_vault_creating_fee
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_vault_creating_fee_failed_duplicate
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_withdrawal_filling_gas_fee
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_withdrawal_filling_gas_fee_failed_duplicate
[ PASS    ] dipcoin::vault_tests::test_set_vault_config_withdrawal_filling_gas_fee_failed_too_high
[ PASS    ] dipcoin::vault_tests::test_vault_max_cap_boundary
[ PASS    ] dipcoin::vault_tests::test_withdraw_all_by_non_creator
[ PASS    ] dipcoin::vault_tests::test_withdraw_by_creator_success
[ PASS    ] dipcoin::vault_tests::test_withdraw_by_non_creator_loss
[ PASS    ] dipcoin::vault_tests::test_withdraw_by_non_creator_profit
[ PASS    ] dipcoin::vault_tests::test_withdraw_creator_share_ratio_below_minimum
[ PASS    ] dipcoin::vault_tests::test_withdraw_direct_withdraw_not_enabled
[ PASS    ] dipcoin::vault_tests::test_withdraw_insufficient_balance
[ PASS    ] dipcoin::vault_tests::test_withdraw_insufficient_shares
[ PASS    ] dipcoin::vault_tests::test_withdraw_locked
[ PASS    ] dipcoin::vault_tests::test_withdraw_protocol_paused
[ PASS    ] dipcoin::vault_tests::test_withdraw_user_not_exist
[ PASS    ] dipcoin::vault_tests::test_withdraw_zero_shares
Test result: OK. Total tests: 103; passed: 103; failed: 0
```

Code Coverage

The code coverage data was generated using the following commands with Sui CLI version `1.61.2-homebrew`:

- `sui move test -p contracts --coverage --skip-fetch-latest-git-deps` to produce the coverage report
- `sui move coverage summary -p contracts --skip-fetch-latest-git-deps` to view the summary

The resulting coverage report was manually filtered to include only the **vault** module coverage.

Module	Coverage (%)
vault	87.42
Move Coverage	83.99

Changelog

- 2025-12-05 - Initial report
- 2025-12-21 - Final report
- 2026-01-05 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

Quantstamp