Security Assessment

Hyperbeat Vault Infrastructure

October 2025

Prepared for Hyperbeat

Table of contents

Project Summary......3
Project Scope......3
Project Overview......3
Findings Summary......5
Severity Matrix......5
Detailed Findings......6
Medium Severity Issues......8
M-01 Management fees can be inflated on liquidity spikes......8
Low Severity Issues......10
L-01 Fees should be updated under timelock......10
L-02 Attackers can steal yield by sandwiching the update of the exchange rate......11
L-03 Unbounded withdrawal requests can DoS recoverERC20 and slow processWithdrawalRequests......13
L-04 Missing slippage protection in instantWithdraw......14
L-05 OracleAggregator lacks price validation and staleness check......15
L-06 WithdrawalQueue.recoverERC20 should also reserve vaultToken......16
L-07 Withdrawal requests can be both processed and canceled at deadline block......17
L-08 Fee rounded down in deposit......18
L-09 depositFee, depositFeeRecipient and depositCap shall be set in the Depositor constructor......19
Informational Issues......20
I-01. Depositor.setDepositCap should allow reset to 0......20
I-02. ExchangeRateUpdater misses withdrawalQueue setter......21
I-03. Depositor and DepositReceiver lack input validation for several admin setters......21
I-04. Native tokens are only partially supported......22
I-05. Implicit Depositor.togglePaused is error-prone......22
I-06. Withdrawal batches can be DoS'ed through blacklisted addresses......23
I-07. Unused ExchangeRateUpdater.InitializationParams.withdrawalQueue......23
I-08. Redundant storage variables......24
Disclaimer......25
About Certora......25

Project Summary

Project Scope

Project Name	Repository (link)	Initial Commit Hash	Platform
DNCoreWriter	0xhyperbeat/DNCoreWriter	12ad406	EVM

Project Overview

This document describes the verification of **Hyperbeat Vault Infrastructure** using manual code review. The work was undertaken from **October 15th, 2025** to **October 28th, 2025**.

The team performed a manual audit of the following Solidity contracts:

- `src/vault/WithdrawalQueue.sol`
- `src/vault/ExchangeRateUpdater.sol`
- `src/vault/VaultToken.sol`
- `src/vault/Pricer.sol`
- `src/vault/DepositReceiver.sol`
- `src/vault/interfaces/IPriceProvider.sol`
- `src/vault/Depositor.sol`
- `src/OracleAggregator.sol`

During the manual audit, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.

Protocol Overview

The Hyperbeat Vault Infrastructure is a protocol that runs on the Hyperliquid HyperEVM. It allows users to deposit EVM ERC20 assets like USDC or USDT (`Depositor.sol`, `DepositReceiver.sol`) in exchange for protocol token (`VaultToken.sol, Pricer.sol`). Funds are then bridged to the Hyperliquid HyperCore (`DnCoreWriterVault.sol`, out of scope) where they can be actively managed.

Users can withdraw funds (`WithdrawalQueue.sol`) either instantly or after a short amount of time that allows funds to be withdrawn from HyperCore.

The protocol keeps track of the exchange rate between vault tokens by calculating the protocol TVL respective to the vault token supply (`ExchangeRateUpdater.sol, OracleAggregator.sol`).

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	0	-	-
High	0	-	-
Medium	1	1	1
Low	9	9	0
Informational	8	8	0
Total	18	18	1

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	Management fees can be inflated on liquidity spikes	Medium	Fixed
L-01	Fees should be updated under timelock	Low	Acknowledged
L-02	Attackers can steal yield by sandwiching the update of the exchange rate	Low	Acknowledged
L-03	Unbounded withdrawal requests can DoS recoverERC20 and slow processWithdrawalRequests	Low	Acknowledged
L-04	Missing slippage protection in instantWithdraw	Low	Acknowledged
L-05	OracleAggregator lacks price validation and staleness check	Low	Acknowledged
L-06	WithdrawalQueue.recoverERC20 should also reserve vaultToken	Low	Acknowledged
L-07	Withdrawal requests can be both processed and canceled at deadline block	Low	Acknowledged
L-08	Fee rounded down in deposit	Low	Acknowledged
L-09	depositFee, depositFeeRecipient and depositCap should be set in the Depositor	Low	Acknowledged

	constructor		
I-01	Depositor.setDepositCap should allow reset to 0	Informational	Acknowledged
I-02	ExchangeRateUpdater misses withdrawalQueue setter	Informational	Acknowledged
I-03	Depositor and DepositReceiver lack input validation for several admin setters	Informational	Acknowledged
I-04	Native tokens are only partially supported	Informational	Acknowledged
I-05	Implicit Depositor.togglePaused is error-prone	Informational	Acknowledged
I-06	Withdrawal batches can be DoS'ed through blacklisted addresses	Informational	Acknowledged
I-07	Unused ExchangeRateUpdater.InitializationParams.withdrawalQueue	Informational	Acknowledged
I-08	Redundant storage variables	Informational	Acknowledged

Medium Severity Issues

M-01 Management fees can be inflated on liquidity spikes

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: Pricer.sol#L322-326	Status: Fixed	

Description: The `_calculateManagementFee` function in `Pricer` calculates management fees with the following formula:

JavaScript

```
File: src/vault/Pricer.sol
322: uint256 timeDelta = currentTime - lastUpdateTimestamp;
323: uint256 totalAssets =
324:   _convertDecimals(totalSharesSupply * newExchangeRate, vaultTokenDecimals + decimals,
baseAssetDecimals);
325: uint256 managementFeesAnnual = totalAssets.mulDiv(managementFee, 1e4);
326: managementFeesAccumulatedInBaseAsset = managementFeesAnnual.mulDiv(timeDelta, 365 days);
```

This formula has two problems:

- The exchange rate used is `newExchangeRate` and not the old one. This means that if exchange rate was `1e8` for 2 years, and suddenly it spikes to `2e8` at the moment of the update, the calculations make fees as if it was `2e8` for 2 years
- the same goes for `totalSharesSupply` : if supply was `100e8` for 2 years, and suddenly spikes to `200e8`, management fees are calculated as if they were `200e8` for 2 years

This is true in both directions (so also for decreases of supply and exchange rate), however it's reasonable to assume that increases happen more often than decreases.

Recommendations: In order to accurately track fees, we recommend accumulating a time-weighted TVL tracker.

Customer's response: Fixed with 59744d5

Fix review: Fix confirmed. Management fees are now calculated conservatively, taking the lower value of both rate and supply between the current and the previous update.

Low Severity Issues

L-01 Fees should be updated under timelock		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: WithdrawalQueue.sol#L295 Depositor.sol#L138	Status: Acknowledged	

Description: Both the `depositFee` in `Depositor.sol` and the `instantWithdrawalFee` in `WithdrawalQueue.sol` allow instant updates with a simple setter.

Because users may be depositing or withdrawing concurrently with an update - and more specifically, an increase of these fees, the outcome of their transaction may be less favorable than what they expected.

Recommendations: We recommend the general practice of applying a time-lock to changes that are sensitive like fees.

Customer's response: Acknowledged

L-02 Attackers can steal yield by sandwiching the update of the exchange rate

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: Depositor.sol#L68-L90 ExchangeRateUpdater.sol#L179-L192 WithdrawalQueue.sol#L173	Status: Acknowledged	

Description: The `updateExchangeRate()` function calculates the new rate as `totalAssets / totalSupply`, distributing yield proportionally to all shareholders regardless of when they deposited.

An attacker can front-run the keeper's `updateExchangeRate()` transaction with a large `deposit` at the old rate, capture proportional yield when the rate updates, then immediately exit via `instantWithdraw()` at the new rate. The attacker captures yield proportional to their share of total supply despite holding shares for only seconds and contributing nothing to earning the yield.

With greater risk, but higher profit margin, this same attack is possible through a combination of deposit and delayed (non-instant) withdrawal around the exchange rate update transaction.

Recommendations:

- Ensure fees(`instantWithdrawalFee + depositFee`) are higher than `allowedExchangeRateChangeUpper` so a `deposit + instantWithdraw` attack of this sort is not profitable.
- In case `depositFee` alone remains below `allowedExchangeRateChangeUpper` we also recommend off-chain monitoring to ensure that the non-instant withdrawals that follow a deposit + withdraw attack pattern are rejected

Customer's response: Acknowledged. The team will pause deposits for 2-3 minutes before price updates.

L-03 Unbounded withdrawal requests can DoS recoverERC20 and slow processWithdrawalRequests		
Severity: **Low**	Impact: **Low**	Likelihood: **Medium**
Files: WithdrawalQueue.sol#L177-L222 WithdrawalQueue.sol#L266-L278	Status: Acknowledged	

Description: A malicious user can open many small withdrawal requests, impacting two key functions:

`recoverERC20()`: iterates over all active withdrawals to compute excess tokens and can be DoSed through gas exhaustion.

`processWithdrawalRequests()`: avoids DoS but can become increasingly inefficient off-chain when handling numerous small requests.

Recommendations: Enforce a minimum withdrawal amount. Additionally, independently track the total locked base asset/vault token, then use these values in the `recoverERC20` function to make the recovery process `O(1)`.

Customer's response: Acknowledged

L-04 Missing slippage protection in instantWithdraw

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: WithdrawalQueue.sol#L165	Status: Acknowledged	

Description: As compared to the withdraw function, `instantWithdraw` does not allow the caller to specify a minimum amount of `baseAsset` that they are willing to accept in return:

JavaScript

```
File: src/vault/WithdrawalQueue.sol
165:     function instantWithdraw(address _user, uint256 _amount) external {
166:         if (isPaused || isInstantWithdrawalPaused) {
167:             revert WithdrawalQueue__Paused();
168:         }
---
175:     }
```

while this protection is less relevant for instant than regular withdrawals – which have it – it's still a good practice since users don't control when the exchange rate is updated.

Recommendations: Consider letting users specify a minimum amount of `baseAsset`, below which the `instantWithdraw` call would revert.

Customer's response: Acknowledged

L-05 OracleAggregator lacks price validation and staleness check		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: OracleAggregator.sol#L23-L29	Status: Acknowledged	

Description: `OracleAggregator._getPrice()` directly returns values from RedStone Price Feeds without validating that prices are positive and up-to-date. This can allow invalid or stale prices to be used.

Recommendations: Add validation logic to reject stale and invalid price data. Price Feed Heartbeat's info can be found here.

Customer's response: Acknowledged

L-06 WithdrawalQueue.recoverERC20 should also reserve vaultToken		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: WithdrawalQueue.sol#L264-L283	Status: Acknowledged	

Description: The `WithdrawalQueue.recoverERC20` function has specific logic that prevents moving out `baseAsset` tokens below what's needed to make the active withdrawals solvent.

However, `baseAsset` is not the only token held by `WithdrawalQueue` - there is also `vaultToken` ready to be burned or refunded when withdrawals are either processed or cancelled.

This means that if at any point in time `recoverERC20(vaultToken)` is called, all withdrawals in the queue can no longer be cancelled or processed.

Recommendations: Update the `_activeWithdrawals` loop to also count the `vaultTokens` escrowed and disallow recovering these.

Customer's response: Acknowledged

L-07 Withdrawal requests can be both processed and canceled at deadline block		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: WithdrawalQueue.sol#L188-L190 WithdrawalQueue.sol#L229-L231	Status: Acknowledged	

Description: In the `WithdrawalQueue` contract, the `processWithdrawalRequests` function treats withdrawal requests where `deadline == block.timestamp` as valid (not expired). However, the `cancelWithdrawalRequestAndClaimShares` function also allows canceling requests at `deadline == block.timestamp`. This creates a conflict where users can cancel withdrawals that are still processable.

This enables an attack where a user cancels their withdrawal in the same block that `processWithdrawalRequests` attempts to process it, causing the transaction to revert and other withdrawals to be delayed by one block.

Recommendations: Update the check in `cancelWithdrawalRequestAndClaimShares` to fail when `deadline ≥ block.timestamp`.

Customer's response: Acknowledged

L-08 Fee rounded down in deposit

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: Depositor.sol#L77	Status: Acknowledged	

Description: In the `Depositor.deposit` function the fee is calculated using the following approach:

```javascript
File: src/vault/Depositor.sol
76:         if (depositFee > 0 && depositFeeRecipient != address(0)) {
77:             feeAmount = _amount.mulDiv(depositFee, BASE);
```

However, `mulDiv()` will actually round the fee down. As a result, users may split their deposits into smaller portions in order to avoid paying small portions of the associated fees.

Recommendations: Consider rounding the fee up.

Customer's response: Acknowledged

L-09 depositFee, depositFeeRecipient and depositCap shall be set in the Depositor constructor

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: Depositor.sol#L48-L59 Depositor.sol#L75-L86	Status: Acknowledged	

Description: When the contract is deployed, the `depositFee`, the `depositFeeRecipient` and the `depositCap` are not set in the constructor; they are instead expected to be set by the owner using separate calls.

As a result, depositors who enter before these values are set can avoid fees and potentially exceed the intended deposit cap.

Recommendations: Set default values for `depositFee` and `depositCap` in the constructor, from either input parameters, or non-zero, reasonable defaults.

Customer's response: Acknowledged

Informational Issues

I-01. Depositor.setDepositCap should allow reset to 0

Description: The initial state of `Depositor.depositCap` is 0 (no cap).

Once set to any value > 0, it can't be brought back to zero, because the check below at L158 disallows setting a value lower than the vault token supply:

JavaScript

```
File: src/vault/Depositor.sol
157:     function setDepositCap(uint256 _depositCap) external requiresAuth {
158:         if (_depositCap < VaultToken(vaultToken).totalSupply()) {
159:             revert Depositor__InvalidDepositCap();
160:         }
161:         depositCap = _depositCap;
162:         emit DepositCapSet(_depositCap);
163:     }
```

Recommendation: We recommend making an exception to the L158 check, in that the new value of 0 should be allowed regardless of supply.

Customer's response: Acknowledged

I-02. ExchangeRateUpdater misses withdrawalQueue setter

Description: At creation, `ExchangeRateUpdater` derives its own `withdrawalQueue` from that configured in `DnCoreWriterVault` :

```
JavaScript
File: src/vault/ExchangeRateUpdater.sol
109:         withdrawalQueue =
DnCoreWriterVault(payable(_params.dnCoreWriterAddress)).withdrawalQueue();
```

After this happens, admins are allowed to change the `withdrawalQueue` in `DnCoreWriterVault` by calling its `setWithdrawalQueue` setter.

However, there is no entry point to propagate this change to `ExchangeRateUpdater` which misses a setter for its `withdrawalQueue`, that is set only at creation despite being a mutable storage variable.

Recommendation: We recommend adding a setter that either takes the raw address or queries once more `DnCoreWriterVault` to update the storage value.

Customer's response: Acknowledged

I-03. Depositor and DepositReceiver lack input validation for several admin setters

Description: The following functions lack input sanitization for input addresses (zero address check):

- Depositor: setDepositReceiver, setPricer, setDepositFeeRecipient, setDepositReceiver
- DepositReceiver: setDnCoreWriter, setWithdrawalQueue, setPricer
- WithdrawalQueue: setPricer

Recommendation: We recommend adding zero-address checks for the above functions.

Customer's response: Acknowledged

I-04. Native tokens are only partially supported

Description: Although the protocol currently does not handle native tokens, some functionality in DnCoreWriterVault (`receive`, `bridgeHYPE`, `releaseTokensForWithdrawals`, `wrapHype`, `unwrapHype`) suggests these may be in the future.

Recommendation: If/when this happens, the following functionality needs adaptation:

- `ExchangeRateUpdater.getIdleTokenBalances` should be updated to handle `0xEEE…`
- `WithdrawalQueue` should be updated to handle native tokens, including a `receive` function
- (lower priority) `DnCoreWriterVault` `bridgeHYPE` and `wrapHype` may become `payable`

Customer's response: Acknowledged

I-05. Implicit Depositor.togglePaused is error-prone

Description: `Depositor.togglePaused` does not receive a desired final state in input, but instead inverts the current paused status.

This is undesirable because in this setup, in case there are several addresses / processes that are authorized to pause, they can act concurrently, and undo each other's action without noticing.

Recommendation: We recommend changing `togglePaused` to receive `bool paused` in input.

Customer's response: Acknowledged

I-06. Withdrawal batches can be DoS'ed through blacklisted addresses

Description: Some tokens like USDC and more importantly USDT implement blacklists, that is a list of addresses that cannot send or receive tokens.

When initiating a withdrawal, the initiator specifies a `user` who will receive tokens when the withdrawal is filled.

One can initiate a withdrawal and indicate a blacklisted `user`, which will cause a full batch to fail.

Recommendation: We recommend either:

- skipping withdrawals whose `baseAsset.transfer` reverted instead of failing the whole batch
- implementing defensive simulation in the off-chain logic that builds the `processWithdrawalRequests` call to not include failing withdrawals

Customer's response: Acknowledged

I-07. Unused ExchangeRateUpdater.InitializationParams.withdrawalQueue

Description: The `withdrawalQueue` field of `ExchangeRateUpdater.InitializationParams` is unused, because `withdrawalQueue` is instead sourced from querying from `dnCoreWriterAddress` .

Recommendation: We recommend removing this field from `InitializationParams` for clarity.

Customer's response: Acknowledged

I-08. Redundant storage variables

Description: The `vaultToken`, `baseAsset`, `depositReceiver` fields of `WithdrawalQueue` and `baseAssetDecimals` in `VaultToken` are set only in the contract constructor. Also in `VaultToken`, the storage variables `minter` and `burner` are never used.

Recommendation: Consider removing `minter` and `burner`, and making the other variables immutable to save gas when accessing their values.

Customer's 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.

The review was conducted over a limited timeframe and may not have uncovered all relevant issues or vulnerabilities.

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.