CODESPECT

Multi-Asset Withdrawal Queue

SECURITY ASSESSMENT REPORT

January 14, 2026

Prepared for:

Hyperbeat

Contents

1 About CODESPECT

CODESPECT is a specialized smart contract security firm dedicated to ensure the safety, reliability, and success of blockchain projects. Our services include comprehensive smart contract audits, secure design and architecture consultancy, and smart contract development across leading blockchain platforms such as Ethereum (Solidity), Starknet (Cairo), and Solana (Rust).

At CODESPECT, we are committed to build secure, resilient blockchain infrastructures. We provide strategic guidance and technical expertise, working closely with our partners from concept development through deployment. Our team consists of blockchain security experts and seasoned engineers who apply the latest auditing and security methodologies to help prevent exploits and vulnerabilities in your smart contracts.

Smart Contract Auditing: Security is at the core of everything we do at CODESPECT. Our auditors conduct thorough security assessments of smart contracts written in Solidity, Cairo, and Rust, ensuring that they function as intended without vulnerabilities. We specialize in providing tailored security solutions for projects on EVM-compatible chains and Starknet. Our audit process is highly collaborative, keeping clients involved every step of the way to ensure transparency and security. Our team is also dedicated to cutting-edge research, ensuring that we stay ahead of emerging threats.

Secure Design & Architecture Consultancy: At CODESPECT, we believe that secure development begins at the design phase. Our consultancy services offer deep insights into secure smart contract architecture and blockchain system design, helping you build robust, secure, and scalable decentralized applications. Whether you're working with Ethereum, Starknet, or other blockchain platforms, our team helps you navigate the complexity of blockchain development with confidence.

Tailored Cybersecurity Solutions: CODESPECT offers specialized cybersecurity solutions designed to minimize risks associated with traditional attack vectors, such as phishing, social engineering, and Web2 vulnerabilities. Our solutions are crafted to address the unique security needs of blockchain-based applications, reducing exposure to attacks and ensuring that all aspects of the system are fortified.

With a focus on the intersection of security and innovation, CODESPECT strives to be a trusted partner for blockchain projects at every stage of development and for each aspect of security.

2 Disclaimer

Limitations of this Audit: This report is based solely on the materials and documentation provided to CODESPECT for the specific purpose of conducting the security review outlined in the Summary of Audit and Files. The findings presented in this report may not be comprehensive and may not identify all possible vulnerabilities. CODESPECT provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, is entirely at your own risk.

Inherent Risks of Blockchain Technology: Blockchain technology is still evolving and is inherently subject to unknown risks and vulnerabilities. This review focuses exclusively on the smart contract code provided and does not cover the compiler layer, underlying programming language elements beyond the reviewed code, or any other potential security risks that may exist outside of the code itself.

Purpose and Reliance of this Report: This report should not be viewed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. Third parties should not rely on this report for any purpose, including making decisions related to investments or purchases.

Liability Disclaimer: To the maximum extent permitted by law, CODESPECT disclaims all liability for the contents of this report and any related services or products that arise from your use of it. This includes but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services: CODESPECT does not warrant, endorse, or assume responsibility for any third-party products or services mentioned in this report, including any open-source or third-party software, code, libraries, materials, or information that may be linked to, referenced by, or accessible through this report. CODESPECT is not responsible for monitoring any transactions between you and third-party providers. We strongly recommend conducting thorough due diligence and exercising caution when engaging with third-party products or services, just as you would for any other product or service transaction.

Further Recommendations: We advise clients to schedule a re-audit after any significant changes to the codebase to ensure ongoing security and reduce the risk of newly introduced vulnerabilities. Additionally, we recommend implementing a bug bounty program to incentivize external developers and security researchers to identify and disclose potential vulnerabilities safely and responsibly.

Disclaimer of Advice: FOR AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, AND ANY ASSOCIATED SERVICES OR MATERIALS SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

3 Risk Classification

Severity Level	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Table 1: Risk Classification Matrix based on Likelihood and Impact

3.1 Impact

- **High** - Results in a substantial loss of assets (more than 10%) within the protocol or causes significant disruption to the majority of users.
- **Medium** - Losses affect less than 10% globally or impact only a portion of users, but are still considered unacceptable.
- **Low** - Losses may be inconvenient but are manageable, typically involving issues like griefing attacks that can be easily resolved or minor inefficiencies such as gas costs.

3.2 Likelihood

- **High** - Very likely to occur, either easy to exploit or difficult but highly incentivized.
- **Medium** - Likely only under certain conditions or moderately incentivized.
- **Low** - Unlikely unless specific conditions are met, or there is little-to-no incentive for exploitation.

3.3 Action Required for Severity Levels

- **Critical** - Must be addressed immediately if already deployed.
- **High** - Must be resolved before deployment (or urgently if already deployed).
- **Medium** - It is recommended to fix.
- **Low** - Can be fixed if desired but is not crucial.

In addition to High, Medium, and Low severity levels, CODESPECT utilizes two other categories for findings: **Informational** and **Best Practices**.

a) **Informational** findings do not pose a direct security risk but provide useful information the audit team wants to communicate formally.

b) **Best Practices** findings indicate that certain portions of the code deviate from established smart contract development standards.

4 Executive Summary

This document presents the security assessment conducted by CODESPECT for the smart contracts of Hyperbeat. Hyperbeat utilizes smart contracts to automate strategies that generate real and sustainable returns through Meta-Yield vaults, Delta-Neutral strategies, HYPE liquid staking, money-markets and onchain payment rails.

This audit focuses on the `WithdrawalQueueMultiAsset` contract, which serves as the withdrawal queue for the Hyperbeat vault and follows the Boring Vault design pattern. The contract is a fork of the originally deployed withdrawal queue, extended to support withdrawals in any approved asset.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of smart contracts, execution testing.

CODESPECT found three points of attention, one classified as `Medium`, one classified as `Low`, and one classified as `Best Practices`. All of the issues are summarised in Table 2.

Organisation of the document is as follows:

- **Section 5** summarizes the audit.
- **Section 6** describes the functionality of the code in scope.
- **Section 7** presents the issues.
- **Section 8** presents the additional notes raised by auditors.
- **Section 9** presents the risks raised by the auditors.
- **Section 10** discusses the documentation provided by the client for this audit.
- **Section 11** presents the compilation and tests.

Issues found:

Severity	Unresolved	Fixed	Acknowledged
Medium	0	1	0
Low	0	1	0
Best Practices	0	1	0
Total	**0**	**3**	**0**

Table 2: Summary of Unresolved, Fixed, and Acknowledged Issues

5 Audit Summary

Audit Type	Security Review
Project Name	Multi-Asset Withdrawal Queue
Type of Project	Withdrawal contract
Duration of Engagement	2 Days
Duration of Fix Review Phase	1 Day
Draft Report	January 13, 2026
Final Report	January 14, 2026
Repository	DNCoreWriter
Commit (Audit)	ebf52dcaa35d45ca70d094ff875c2304adf95c4f
Commit (Final)	258abf650f13308883958a331369e6a1d8e95fc5
Documentation Assessment	High
Test Suite Assessment	Medium
Auditors	talfao, kalogerone

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	Contract	**LoC**
1	WithdrawalQueueMultiAsset.sol	349
	Total	**349**

5.2 Findings Overview

	Finding	**Severity**	**Update**
1	Withdrawals processing only works correctly for the base asset	Medium	Fixed
2	Missing slippage protection in instant withdrawal function	Low	Fixed
3	Magic numbers should be constants	Best Practices	Fixed

6 System Overview

The Hyperbeat `WithdrawalQueueMultiAsset` contract serves as a withdrawal management system within the Hyperbeat vault infrastructure. It enables users to withdraw their vault tokens in exchange for various supported assets, providing both queued and instant withdrawal mechanisms with multi-asset support.

The system operates with two primary withdrawal modes: queued withdrawals for guaranteed execution at favorable rates, and instant withdrawals for immediate liquidity at the cost of a small fee.

6.1 Queued Withdrawal Process

Users can create withdrawal requests via the `createWithdrawalRequest(...)` function. During this process, users specify their desired output asset from a list of acceptable tokens, set slippage protection parameters, and define a deadline for execution. The system captures the current exchange rate and calculates the expected asset output amount.

```
function createWithdrawalRequest(
    address _user,
    address _assetOut,
    uint256 _amount,
    uint256 _minAssetOut,
    uint64 _deadline
) external returns (WithdrawalRequest memory);
```

Withdrawal requests are processed by authorized solvers through the `processWithdrawalRequests(...)` function. The system automatically handles exchange rate fluctuations by recalculating asset amounts if rates have lowered since request creation, while respecting users' minimum output requirements.

```
function processWithdrawalRequests(
    WithdrawalRequest[] memory _withdrawalRequestsToProcess
) external requiresAuth;
```

6.2 Instant Withdrawal Mechanism

For users requiring immediate liquidity, the `instantWithdraw(...)` function provides instant asset conversion. This mechanism burns vault tokens immediately and transfers the requested asset directly to the user, applying an `instantWithdrawalFee` to compensate for the immediate liquidity provision.

```
function instantWithdraw(
    address _user,
    address _assetOut,
    uint256 _amount
) external;
```

6.3 Multi-Asset Support Architecture

The contract integrates with a `Pricer` contract that manages exchange rates and asset configurations for multiple supported tokens. Only tokens configured in the Pricer with valid price providers can be set as acceptable withdrawal assets. The `DepositReceiver` contract is holding various assets for the Hyperbeat Vault and facilitating instant withdrawals when sufficient liquidity is available.

7 Issues

7.1 [Medium] Withdrawals processing only works correctly for the base asset

File(s): `WithdrawalQueueMultiAsset.sol`

Description: In the `WithdrawalQueueMultiAsset` contract users can redeem their `VaultToken` for a token which is not the "Base Asset". When users request a withdrawal, the `assetOutAmount` and `exchangeRate` at the time of the request are stored. The exchange rate is a variable that tracks the rate between the `VaultToken` and the "Base Asset". When requests get processed, the following logic applies according to the exchange rate at the time of the processing:

– If the request's exchange rate is greater than the `currentExchangeRate`, then the amount out is recalculated and user fairly receives less tokens than his request's `assetOutAmount`;

– If the request's exchange rate is lower than the `currentExchangeRate`, then the user receives his request's `assetOutAmount`;

```
function processWithdrawalRequests(WithdrawalRequest[] memory _withdrawalRequestsToProcess) external
 ↪  requiresAuth {
    // ...
        uint256 amountToSend = withdrawalRequest.assetOutAmount;
        uint256 exchangeRateApplied = withdrawalRequest.exchangeRate;
        if (withdrawalRequest.exchangeRate > currentExchangeRate) {
            // Recalculate with current exchange rate using the user's chosen asset
            amountToSend = Pricer(pricer).getAssetAmount(withdrawalRequest.assetOut, withdrawalRequest.amount);
            exchangeRateApplied = currentExchangeRate.toUint128();
            if (amountToSend < withdrawalRequest.minAssetOut) {
                success = _withdrawalRequests.remove(withdrawalRequestId);
                if (!success) {
                    revert WithdrawalQueue__WithdrawalRequestIdNotFound(withdrawalRequestId);
                }
                delete withdrawalRequestData[withdrawalRequestId];
                vaultToken.safeTransfer(withdrawalRequest.initiator, withdrawalRequest.amount);
                // Return the reserved asset to depositReceiver
                withdrawalRequest.assetOut.safeTransfer(depositReceiver, withdrawalRequest.assetOutAmount);
                emit AssetReturnedToDepositReceiver(withdrawalRequest.assetOut,
                 ↪  withdrawalRequest.assetOutAmount);
                emit WithdrawalRequestCancelled(
                    withdrawalRequestId, withdrawalRequest.initiator, withdrawalRequest.amount
                );
                continue;
            }
        }
    // ...
}
```

However, the price correlation for tokens that are not the "Base Asset" with the `VaultToken`, doesn't only depend on the exchange rate variable:

```
function getAssetAmount(address _token, uint256 _vaultTokenAmount) external view returns (uint256) {
    if (_token == baseAsset) {
        return _convertDecimals(
            _vaultTokenAmount.mulDiv(pricerParams.exchangeRate, 10 ** decimals),
            vaultTokenDecimals,
            baseAssetDecimals
        );
    }
    AssetConfig memory assetConfig = assetConfigs[_token];
    if (address(assetConfig.priceProvider) == address(0)) {
        revert Pricer__PriceProviderNotSet(_token);
    }
    return _convertDecimals(
        _vaultTokenAmount.mulDiv(pricerParams.exchangeRate, assetConfig.priceProvider.getPrice()),
        vaultTokenDecimals + decimals - assetConfig.priceProviderDecimals,
        assetConfig.decimals
    );
}
```

The `assetConfig.priceProvider.getPrice()` variable is also used for the calculations of the rate, meaning that a fluctuation of the exchange rate between the request and the processing doesn't mean that the overall rate of the token moved in the same direction. This can create several issues:

- The exchange rate may have gone down, so the `amountToSend` will be recalculated. It's possible that the token's price movement will result in a greater `amountToSend` than the previous one;
- The exchange rate may have gone up, so the `amountToSend` will not be recalculated. It's possible that the token's price movement would result in a lower `amountToSend` value, but it doesn't get recalculated;

Impact: Users may receive more tokens than they expected at the time of the request, or receive their expected amount while they should receive less, because of the token's price movement. Additionally, before processing requests, tokens are sent to the contract according to the requests' total amount. Since users may receive a greater amount than their requests the `processWithdrawalRequests(...)` call may revert.

Recommendation(s): Consider always calling the `getAssetAmount(...)` function and comparing that amount to the old one, instead of depending on the exchange rate.

Status: Fixed

Update from Hyperbeat: https://github.com/0xhyperbeat/DNCoreWriter/pull/20

Update from CODESPECT: We believe that this is not fixed. This is the new `if` condition:

```
if (amountToSend > currentAssetAmount || exchangeRateApplied > currentExchangeRate) {
```

It's possible that the `exchangeRateApplied > currentExchangeRate`, but the `amountToSend < currentAssetAmount` for other tokens, as they don't only rely on the exchange rate. In these cases, user withdrawals will increase. We believe that the `if` condition should only check for the asset amount, without the exchange rate.

Update from Hyperbeat: pushed

Update from CODESPECT: The latest commit in PR fixed the issue

7.2 [Low] Missing slippage protection in instant withdrawal function

File(s): `WithdrawalQueueMultiAsset.sol`

Description: The `instantWithdraw(...)` function lacks slippage protection for multi-asset withdrawals. Users can specify an `_assetOut` parameter to receive their withdrawal in different tokens, but there's no minimum amount check to protect against unfavourable exchange rates at execution time.

```
function instantWithdraw(address _user, address _assetOut, uint256 _amount) external {
    // ... validation checks ...
    VaultToken(vaultToken).burn(msg.sender, _amount);
    uint256 fee = _amount.mulDivUp(instantWithdrawalFee, 10_000);
    uint256 amountToWithdraw = _amount - fee;
    uint256 assetOutAmount = Pricer(pricer).getAssetAmount(_assetOut, amountToWithdraw); // @audit Exchange rate and
     ↪ token price could change
    DepositReceiver(depositReceiver).sendAssetsForInstantWithdrawal(_assetOut, _user, assetOutAmount);
}
```

While this issue was previously acknowledged in earlier auditing reports, the introduction of multi-asset withdrawals significantly increases the impact as different asset prices can vary substantially.

Impact: Users may receive less than expected due to price volatility when withdrawing in different assets. The lack of slippage protection exposes users to potential financial losses.

Recommendation(s): Add a `minAssetOut` parameter to the function to allow users to specify the minimum acceptable amount:

Status: Fixed

Update from Hyperbeat: https://github.com/0xhyperbeat/DNCoreWriter/pull/21

7.3 [Best Practice] Magic numbers should be constants

File(s): `WithdrawalQueueMultiAsset.sol`

Description: The `instantWithdraw(...)` function uses a magic number `10_000` for fee calculation in basis points:

```
uint256 fee = _amount.mulDivUp(instantWithdrawalFee, 10_000);
```

Magic numbers make code harder to understand and maintain. The value `10_000` represents basis points (100

Recommendation(s): Define a constant for the basis points denominator:

```
uint256 private constant BASIS_POINTS_DENOMINATOR = 10_000;

// Then use in fee calculation:
uint256 fee = _amount.mulDivUp(instantWithdrawalFee, BASIS_POINTS_DENOMINATOR);
```

Status: Fixed

Update from Hyperbeat: https://github.com/0xhyperbeat/DNCoreWriter/pull/21/commits/486e8906b333868778f88c100b503dee745d697d

8 Additional Notes

This section provides supplementary auditor observations regarding the code. These points were not identified as individual issues but serve as informative recommendations to enhance the overall quality and maintainability of the codebase.

- The `recoverERC20` function at line 338 has an incomplete comment structure that should be cleaned up for better documentation consistency, as it is missing a privileged role of this function.
- The `getActiveWithdrawals(...)` function at line 365 is missing proper NatSpec documentation comments explaining its purpose and return value.

Update from Hyperbeat: 258abf650f13308883958a331369e6a1d8e95fc5

9 Protocol Risks

This section outlines protocol-level risks as identified and understood by the CODESPECT audit team. These points reflect design choices, assumptions, or trust dependencies observed during the review and are documented to provide transparency regarding the protocol's risk profile from CODESPECT's perspective.

- **Instant Withdraw Fee Configuration**: The `instantWithdrawalFee` must be configured carefully. Ideally, the fee should always exceed any potential short-term exchange rate fluctuations. In the case of multi-asset withdrawal queues, the fee should also account for possible price movements of the output token. This ensures that malicious actors cannot exploit the instant withdrawal mechanism to arbitrage unfavourable exchange rate changes.
- **Centralisation Risk**: The overall protocol design is highly centralised and relies on a privileged `ADMIN_ROLE`. This role is responsible for modifying critical parameters, including acceptable token configurations, pricer settings, and the recovery of ERC20 tokens from the contracts. The protocol team must ensure robust key management practices are in place to mitigate the risks associated with these high-impact administrative operations.

10 Evaluation of Provided Documentation

The **Hyperbeat** documentation was provided primarily through comprehensive NatSpec comments embedded directly within the codebase:

- **NatSpec:** The in-code NatSpec comments were clear, detailed, and highly effective in explaining specific flows, design intentions, and conditional branches. Each core functionality was well documented, allowing for an efficient understanding of the system's behavior and expected invariants.

Overall, the documentation was adequate and fully sufficient for the scope of this audit. Moreover, the Hyperbeat team remained consistently available and responsive, promptly addressing all questions and clarifications raised by **CODESPECT**, which significantly streamlined the audit process.

11 Test Suite Evaluation

11.1 Compilation Output

```
> forge build src/vault/WithdrawalQueueMultiAsset.sol
[] Compiling...
[] Compiling 15 files with Solc 0.8.29
[] Solc 0.8.29 finished in 417.55ms
Compiler run successful!
```

11.2 Tests Output

```
> forge test tests/vault/WithdrawalQueueMultiAssetTest.t.sol
[] Compiling...
[] Compiling 62 files with Solc 0.8.29
[] Solc 0.8.29 finished in 2.93s
Compiler run successful with warnings:
Warning (2018): Function state mutability can be restricted to pure
  --> tests/vault/WithdrawalQueueMultiAssetTest.t.sol:76:5:
   |
76 |     function decimals() external view returns(uint8){
   |     ^ (Relevant source part starts here and spans across multiple lines).


Ran 50 tests for tests/vault/WithdrawalQueueMultiAssetTest.t.sol:WithdrawalQueueMultiAssetTest
[PASS] test_cancelWithdrawalRequestAndClaimShares_revertsIfDeadlineNotMet() (gas: 304705)
[PASS] test_cancelWithdrawalRequestAndClaimShares_success() (gas: 253153)
[PASS] test_constructor_revertsOnZeroAddress() (gas: 205442)
[PASS] test_constructor_setsCorrectValues() (gas: 29928)
[PASS] test_createWithdrawalRequest_revertsIfAmountZero() (gas: 19889)
[PASS] test_createWithdrawalRequest_revertsIfDeadlineTooSoon() (gas: 22100)
[PASS] test_createWithdrawalRequest_revertsIfMinAssetOutNotMet() (gas: 66737)
[PASS] test_createWithdrawalRequest_revertsIfPaused() (gas: 44025)
[PASS] test_createWithdrawalRequest_revertsIfTokenNotAcceptable() (gas: 20326)
[PASS] test_createWithdrawalRequest_revertsIfUserIsZeroAddress() (gas: 17479)
[PASS] test_createWithdrawalRequest_withAltAsset() (gas: 309248)
[PASS] test_createWithdrawalRequest_withBaseAsset() (gas: 308496)
[PASS] test_getActiveWithdrawalRequests_afterProcessing() (gas: 498926)
[PASS] test_getActiveWithdrawalRequests_empty() (gas: 10763)
[PASS] test_getActiveWithdrawalRequests_multiple() (gas: 572459)
[PASS] test_getActiveWithdrawalRequests_single() (gas: 310827)
[PASS] test_getActiveWithdrawals() (gas: 568780)
[PASS] test_getRequiredTokenAmounts_afterProcessing() (gas: 286048)
[PASS] test_getRequiredTokenAmounts_multipleTokens() (gas: 579297)
[PASS] test_getRequiredTokenAmounts_noWithdrawals() (gas: 12268)
[PASS] test_getRequiredTokenAmounts_nonExistentToken() (gas: 309358)
[PASS] test_getRequiredTokenAmounts_singleToken() (gas: 563044)
[PASS] test_instantWithdraw_revertsIfAmountIsZero() (gas: 17800)
[PASS] test_instantWithdraw_revertsIfInstantWithdrawalPaused() (gas: 41951)
[PASS] test_instantWithdraw_revertsIfPaused() (gas: 41872)
[PASS] test_instantWithdraw_revertsIfTokenNotAcceptable() (gas: 18278)
[PASS] test_instantWithdraw_revertsIfUserIsZeroAddress() (gas: 15460)
[PASS] test_instantWithdraw_withAltAsset() (gas: 96965)
[PASS] test_instantWithdraw_withBaseAsset() (gas: 90551)
[PASS] test_multipleUsersWithDifferentAssets() (gas: 526534)
[PASS] test_processWithdrawalRequests_cancelsIfMinAssetOutNotMet() (gas: 296175)
[PASS] test_processWithdrawalRequests_revertsIfExpired() (gas: 312501)
[PASS] test_processWithdrawalRequests_revertsIfRequestNotInStorage() (gas: 31169)
[PASS] test_processWithdrawalRequests_success() (gas: 288045)
[PASS] test_processWithdrawalRequests_withAltAsset() (gas: 289488)
[PASS] test_recoverERC20_acceptableTokenWithPendingWithdrawals() (gas: 342521)
[PASS] test_recoverERC20_doesNotRevertWhenBalanceEqualsReserved() (gas: 307843)
[PASS] test_recoverERC20_nonAcceptableToken() (gas: 74775)
[PASS] test_recoverERC20_revertsIfInvalidActiveWithdrawals() (gas: 311390)
[PASS] test_rejectWithdrawalRequest_success() (gas: 256928)
[PASS] test_setAcceptableToken_altAsset() (gas: 16387)
[PASS] test_setAcceptableToken_baseAsset() (gas: 23601)
```

11.3 Tests Output

```
[PASS] test_setAcceptableToken_emitsEvent() (gas: 18239)
[PASS] test_setAcceptableToken_revertsIfNotInPricer() (gas: 25654)
[PASS] test_setInstantWithdrawalFee() (gas: 17202)
[PASS] test_setInstantWithdrawalPaused() (gas: 34178)
[PASS] test_setMinimumDeadline() (gas: 17109)
[PASS] test_setPaused() (gas: 25952)
[PASS] test_setPricer() (gas: 17271)
[PASS] test_setPricer_revertsIfZeroAddress() (gas: 12914)
Suite result: ok. 50 passed; 0 failed; 0 skipped; finished in 7.37ms (18.79ms CPU time)

Ran 1 test suite in 52.96ms (7.37ms CPU time): 50 tests passed, 0 failed, 0 skipped (50 total tests)
```

11.4 Notes on the Test Suite

The Hyperbeat testing suite provided strong coverage across the basic core components of the contract in scope. It explored a wide range of functional flows, including the interactions between the `WithdrawalQueueMultiAsset` and the `Pricer` contract. The suite considers various edge cases that the contracts are expected to handle, validating behaviour under both normal and boundary conditions.