CODESPECT

Credit Fund

SECURITY ASSESSMENT REPORT

December 22, 2025

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 Credit Fund strategy, which is responsible for allocating USDC and beatUSD into Morpho Vaults. The strategy also supports leveraging Hyperbeat ERC-20 wrappers to interact with permissioned Morpho vaults, enabling controlled access while maintaining composability with the broader system.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of smart contracts, execution testing.

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

Note

This report also acknowledges the review of d6c5e3cbf655c652ad9e0972b27780e54548b9f2, which included changes to permissioned wrappers that were reviewed by CODESPECT in the previous engagement with Hyperbeat.

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 risks raised by the auditors.
- **Section 8** discusses the documentation provided by the client for this audit.
- **Section 9** presents the compilation and tests.

Issues found:

Severity	Unresolved	Fixed	Acknowledged
Medium	0	1	0
Low	0	1	0
Informational	0	3	0
Total	**0**	**5**	**0**

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

5 Audit Summary

Audit Type	Security Review
Project Name	Credit Fund
Type of Project	Boring Vault Strategy
Duration of Engagement	4 Days
Duration of Fix Review Phase	1 Day
Draft Report	December 22, 2025
Final Report	December 22, 2025
Repository	DNCoreWriter
Commit (Audit)	6010261cc6a503611170f619cda33a17fb2ddd23
Commit (Final)	3b84117b5b980e133542fbbc95394ead2b0cd3a5
Documentation Assessment	Medium
Test Suite Assessment	High
Auditors	talfao, kalogerone

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	Contract	LoC
1	MorphoV2Orchestrator.sol	259
2	CreditFundExchangeUpdater.sol	164
	Total	**423**

5.2 Findings Overview

	Finding	Severity	Update
1	The total assets calculations don't include `wrappedBeatUSD` tokens in the Orchestrator contract	Medium	Fixed
2	Dynamic slippage protection does not fully prevent sandwich attacks	Low	Fixed
3	Morpho vault deployment and integration considerations	Info	Fixed
4	The Orchestrator can unwrap `HYPE` but is not able to transfer it	Info	Fixed
5	The Orchestrator has the functionality to send native tokens to a contract that can't receive them	Info	Fixed

6 System Overview

The Hyperbeat Credit Fund is one of the DeFi yield strategies which focus on the allocation of USDC and beatUSD to the Morpho Vaults.

The main components of the protocol are:

- `MorphoV2Orchestrator` — Contract responsible for the orchestration of the operations managing various Morpho V2 vaults' asset allocations.
- `CreditFundExchangeUpdater` — NAV calculation engine that aggregates total assets across all protocol locations and updates exchange rates through the Pricer contract.
- `BoringVault` — Vault that represents staked assets and accrues value through underlying strategy performance.
- `Accountant` — Price oracle provider that supplies exchange rates of the boring vault share.
- `Teller` — Multi-asset deposit/withdrawal interface that handles user interactions with the boring vault.
- `Pricer` — Exchange rate management contract that maintains vault token pricing and fee calculations.

6.1 Management of assets within orchestrator

The `MorphoV2Orchestrator` is the contract responsible for the orchestration of the assets across multiple Morpho V2 vaults. The orchestrator is responsible for managing assets transferred from the depositReceiver and it can allocate them to the specific morpho vaults:

```
function allocate(address morphoV2Vault, uint256 amount) external
    onlyRole(ALLOCATOR_ROLE) returns (uint256 shares);
```

Furthermore it is able to mint beatUSD by depositing baseAsset (initially USDC):

```
function mintBeatUSD(uint256 baseAmount) external onlyRole(ALLOCATOR_ROLE)
    returns(uint256 mintedBeatUSD);
```

And burn the beatUSD to obtain back the base asset:

```
function burnBeatUSD(uint256 shareAmount) external onlyRole(ALLOCATOR_ROLE)
    returns(uint256 baseAmount);
```

Furthermore it is able to wrap beatUSD for the management of permissioned Morpho markets:

```
function permissionedWrap(uint256 amount, address token) external
    onlyRole(ALLOCATOR_ROLE);
function permissionedUnWrap(uint256 amount, address token) external
    onlyRole(ALLOCATOR_ROLE);
```

which at the point of the review is more like future feature which will be used.

6.2 NAV Update Flow Process

The `CreditFundExchangeUpdater` calculates the total asset value across all protocol locations and updates the vault's exchange rate:

```
function updateExchangeRate() external requiresAuth;
```

The process involves:

a. **Asset Aggregation**: Calculate total beatUSD and base asset balances across deposit receiver, withdrawal queue, orchestrator, and all Morpho V2 vault positions

b. **Price Conversion**: Convert beatUSD amounts to base asset terms using the accountant's exchange rate

c. **Fee Adjustment**: Subtract accumulated fees from total assets

d. **NAV Calculation**: Compute new NAV as `(totalAssets * vaultTokenDecimals) / totalSupply`

e. **Exchange Rate Update**: Call the `Pricer` contract to update the vault's exchange rate

7 Issues

7.1 [Medium] The total assets calculations don't include `wrappedBeatUSD` tokens in the Orchestrator contract

File(s): `CreditFundExchangeUpdater.sol`

Description: The total assets calculations call the `_calculateAmountBeatUSD(...)` function which tracks the total `beatUSD` balance across multiple contracts and also if there is a MorphoV2 vault with `beatUSD` or `wrappedBeatUSD` as its asset:

```
function _calculateAmountBeatUSD(
    IMorphoV2Orchestrator orchestrator,
    address beatUSD
) internal view returns (uint256 amountBeatUSD) {
    // Direct balances
    amountBeatUSD += IERC20Metadata(beatUSD).balanceOf(depositReceiver);
    amountBeatUSD += IERC20Metadata(beatUSD).balanceOf(withdrawalQueue);
    amountBeatUSD += IERC20Metadata(beatUSD).balanceOf(morphoV2Orchestrator);

    // Get wrapped beatUSD address
    address wrappedBeatUSD = orchestrator.pWrapperMapping(beatUSD);

    // Morpho V2 vault positions
    address[] memory vaults = orchestrator.getMorphoV2Vaults();
    for (uint256 i = 0; i < vaults.length; i++) {
        address vaultAsset = IMorphoV2(vaults[i]).asset();
        if (vaultAsset == address(0)) continue;
        // Check if vault underlying is beatUSD or wrapped beatUSD
        if (vaultAsset == beatUSD || vaultAsset == wrappedBeatUSD) {
            uint256 shares = IERC20Metadata(vaults[i]).balanceOf(morphoV2Orchestrator);
            if (shares > 0) {
                amountBeatUSD += IMorphoV2(vaults[i]).convertToAssets(shares);
            }
        }
    }
}
```

However, the function doesn't track the `wrappedBeatUSD` balance of the `morphoV2Orchestrator` contract, which is the contract responsible for the MorphoV2 vaults interactions.

Impact: The total assets will return a low amount than the actual one, if `wrappedBeatUSD` is sitting in the `morphoV2Orchestrator` contract.

Recommendation(s): Also add the `wrappedBeatUSD` balance of the `morphoV2Orchestrator` contract in the total assets.

Status: Fixed

Update from Hyperbeat: fixed in : https://github.com/0xhyperbeat/DNCoreWriter/pull/13

7.2 [Low] Dynamic slippage protection does not fully prevent sandwich attacks

File(s): MorphoV2Orchestrator.sol

Description: The MorphoV2Orchestrator contract contains functionality for minting and burning BeatUSD. In both flows, a minimum expected output amount is calculated when interacting with the Teller. However, this minimum amount is calculated dynamically, meaning that the effective minimum is generally determined by what the Teller itself is willing to fulfill.

```
function mintBeatUSD(uint256 baseAmount) external onlyRole(ALLOCATOR_ROLE) returns(uint256 mintedBeatUSD){
    if(baseAmount == 0) revert ZeroAmount();
    uint256 minAmountOut = baseAmount.mulDiv(10**IBoringVault(beatUSD).decimals(),
    ↪  IAccountantWithRateProviders(accountant).getRateInQuoteSafe(baseAsset)); // @audit dynamic slippage protection
    mintedBeatUSD = ITellerWithMultiAssetSupport(teller).deposit(baseAsset, baseAmount, minAmountOut);
    emit MintedBeatUSD(baseAmount, mintedBeatUSD);
}
```

To achieve stronger slippage protection, the minimum acceptable output amount should be supplied explicitly as a function parameter. This ensures that slippage protection is enforced according to the caller's expectations rather than being derived dynamically.

Impact: The dynamic slippage protection does not provide sufficient protection against sandwich attacks. That said, the impact is limited, as the accountant rates can only be updated by a Hyperbeat-controlled party.

Recommendation(s): Consider supplying minAmountOut as an explicit parameter to the minting and burning functionality for BeatUSD.

Status: Fixed

Update from Hyperbeat: fixed here : https://github.com/0xhyperbeat/DNCoreWriter/pull/14

7.3 [Info] Morpho vault deployment and integration considerations

File(s): MorphoV2Orchestrator.sol

Description: Morpho vaults are ERC4626-compliant vaults. Morpho provides specific requirements and recommendations to ensure these vaults remain secure and function as intended. The first requirement applies at deployment time: when deploying a Morpho vault, a *dead deposit* must be made to protect against inflation share attacks. According to the documentation:

- *The dead deposit must be at least 1e9 shares (approximately equivalent to $10).*

Additionally, Morpho recommends implementing slippage protection to improve user experience. While this measure can be considered optional if a sufficient dead deposit is in place, it still provides meaningful safeguards. ERC4626 vaults do not include slippage protection by default, so it may be beneficial to update the allocate(...) and deAllocate(...) functions to accept parameters specifying the minimum expected number of shares to be minted or burned.

Impact: If a sufficient dead deposit is not implemented, the vault may be vulnerable to inflation share attacks.

Recommendation(s): Consider implementing both measures described above: enforcing a proper dead deposit at deployment and adding slippage protection to the allocation and deallocation flows.

Status: Fixed

Update from Hyperbeat: fixed here : https://github.com/0xhyperbeat/DNCoreWriter/pull/17

7.4 [Info] The Orchestrator can unwrap HYPE but is not able to transfer it

File(s): MorphoV2Orchestrator.sol

Description: The MorphoV2Orchestrator contract is able to receive native tokens and also wrap and unwrap them:

```
function wrapHype(uint256 amount) external onlyRole(WITHDRAWAL_ROLE) {
    WHYPE.deposit{ value: amount }();
}

function unwrapHype(uint256 amount) external onlyRole(WITHDRAWAL_ROLE) {
    WHYPE.withdraw(amount);
}
```

However, the contract doesn't implement functionality to transfer native tokens out of it, rendering the unwrapping function unnecessary.

Recommendation(s): Consider removing the unnecessary functionality from the contract.

Status: Fixed

Update from Hyperbeat: fixed : https://github.com/0xhyperbeat/DNCoreWriter/pull/16

7.5 [Info] The Orchestrator has the functionality to send native tokens to a contract that can't receive them

File(s): `MorphoV2Orchestrator.sol`

Description: The `releaseTokensForWithdrawals(...)` function releases tokens for the `WithdrawalQueue` contract:

```
function releaseTokensForWithdrawals(address token, uint256 amount) external onlyRole(WITHDRAWAL_ROLE) {
    // ...
    if (token == HYPE) {
        (bool success, ) = payable(withdrawalQueue).call{value: amount}("");
        require(success, "HYPE transfer failed");
    } else {
        IERC20Metadata(token).safeTransfer(withdrawalQueue, amount);
    }
    emit ReleaseTokensForWithdrawals(withdrawalQueue, token, amount);
}
```

However, the `WithdrawalQueue` contract doesn't implement a `receive()` function, thus can't accept native token transfers.

Recommendation(s): Only allow for ERC20 token transfers.

Status: Fixed

Update from Hyperbeat: fixed here : https://github.com/0xhyperbeat/DNCoreWriter/pull/15

8 Acknowledged Risks

This section summarises discussions held with the protocol team to formally acknowledge risks that are already known to the team and to accurately document CODESPECT's understanding of these points.

- **Vault and pricer parameter configuration:** Hyperbeat is aware that all vault and pricer-related parameters must be configured with care. One parameter discussed during the review concerns the lower bound of the exchange ratio during pricer updates. The Hyperbeat team stated that this lower bound will be set to `1e4`, meaning the exchange ratio cannot decrease below this value, which represents a very strict constraint. The exchange ratio could decrease in certain scenarios, for example if Morpho vaults begin applying management fees. The Hyperbeat team clarified that no management fees will be applied to Morpho vaults initially.
- **Dead deposit requirement for the credit fund boring vault:** Another discussion focused on the necessity of an initial dead deposit for the credit fund boring vault. Without a dead deposit, a scenario may arise where the pricer cannot be updated if the total supply of the boring vault reaches zero. This could, for example, prevent the correct application of management fees. The Hyperbeat team acknowledged this risk and confirmed that an initial dead deposit will be performed.
- **Centralisation risk:** The overall application design is highly centralised, as it is built around the boring vault architecture. As a result, the Hyperbeat team must carefully manage all keys associated with privileged roles. In particular, multisignature wallets should be used for high-impact actions, such as updating or assigning critical roles. Additionally, separate wallets should be used for each role to ensure a clear and correct authorization model and to reduce the risk of key compromise.

9 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.

10 Test Suite Evaluation

10.1 Compilation Output

```
> forge build src/credit_fund
[] Compiling...
[] Compiling 34 files with Solc 0.8.29
[] Solc 0.8.29 finished in 551.93ms
Compiler run successful!
```

10.2 Tests Output

```
> forge test tests/CreditFundExchangeUpdaterTest.sol

Ran 36 tests for tests/CreditFundExchangeUpdaterTest.sol:CreditFundExchangeUpdaterTest
[PASS] test_constructor_revertsOnZeroDepositReceiver() (gas: 77014)
[PASS] test_constructor_revertsOnZeroOrchestrator() (gas: 76964)
[PASS] test_constructor_revertsOnZeroPricer() (gas: 77028)
[PASS] test_constructor_revertsOnZeroVaultToken() (gas: 77070)
[PASS] test_constructor_revertsOnZeroWithdrawalQueue() (gas: 76968)
[PASS] test_constructor_setsCorrectValues() (gas: 30789)
[PASS] test_getAmountBaseAsset_includesMorphoVaultPosition() (gas: 892734)
[PASS] test_getAmountBaseAsset_withDirectBalances() (gas: 533732)
[PASS] test_getAmountBaseAsset_withZeroBalances() (gas: 46151)
[PASS] test_getAmountBeatUSD_withDirectBalances() (gas: 521824)
[PASS] test_getAmountBeatUSD_withZeroBalances() (gas: 41293)
[PASS] test_getTotalAssets_afterMintBeatUSD() (gas: 426176)
[PASS] test_getTotalAssets_withBeatUSDAndBaseAsset() (gas: 396192)
[PASS] test_getTotalAssets_withOnlyBaseAsset() (gas: 237842)
[PASS] test_getTotalAssets_withZeroBalances() (gas: 82805)
[PASS] test_setDepositReceiver_revertsOnUnauthorized() (gas: 16829)
[PASS] test_setDepositReceiver_revertsOnZeroAddress() (gas: 15281)
[PASS] test_setDepositReceiver_success() (gas: 23547)
[PASS] test_setMorphoV2Orchestrator_revertsOnUnauthorized() (gas: 16850)
[PASS] test_setMorphoV2Orchestrator_revertsOnZeroAddress() (gas: 15282)
[PASS] test_setMorphoV2Orchestrator_success() (gas: 23528)
[PASS] test_setPricer_revertsOnUnauthorized() (gas: 16853)
[PASS] test_setPricer_revertsOnZeroAddress() (gas: 15370)
[PASS] test_setPricer_success() (gas: 23590)
[PASS] test_setVaultToken_revertsOnUnauthorized() (gas: 16831)
[PASS] test_setVaultToken_revertsOnZeroAddress() (gas: 15328)
[PASS] test_setVaultToken_success() (gas: 23570)
[PASS] test_setWithdrawalQueue_revertsOnUnauthorized() (gas: 16873)
[PASS] test_setWithdrawalQueue_revertsOnZeroAddress() (gas: 15327)
[PASS] test_setWithdrawalQueue_success() (gas: 23591)
[PASS] test_updateExchangeRateView_calculatesCorrectNav() (gas: 253106)
[PASS] test_updateExchangeRateView_returnsValue() (gas: 250647)
[PASS] test_updateExchangeRateView_withFeesAccumulated() (gas: 251277)
[PASS] test_updateExchangeRateView_withMixedAssets() (gas: 578324)
[PASS] test_updateExchangeRate_revertsOnUnauthorized() (gas: 15247)
[PASS] test_updateExchangeRate_success() (gas: 257529)
Suite result: ok. 36 passed; 0 failed; 0 skipped; finished in 40.05s (64.55s CPU time)

Ran 1 test suite in 40.05s (40.05s CPU time): 36 tests passed, 0 failed, 0 skipped (36 total tests)
```

```
> forge test tests/MorphoV2Test.t.sol

Ran 42 tests for tests/MorphoV2Test.t.sol:MorphoV2Test
[PASS] test_addPermissionedWrap_revertsOnDuplicate() (gas: 860234)
[PASS] test_addWhitelistedMorphoV2Vault_revertsOnDuplicate() (gas: 135380)
[PASS] test_addWhitelistedMorphoV2Vault_revertsOnMaxReached() (gas: 144794)
[PASS] test_addWhitelistedMorphoV2Vault_revertsOnUnauthorized() (gas: 18258)
[PASS] test_addWhitelistedMorphoV2Vault_revertsOnZeroAddress() (gas: 18189)
[PASS] test_addWhitelistedMorphoV2Vault_success() (gas: 134834)
[PASS] test_allocate_3_success_3() (gas: 283559)
[PASS] test_allocate_revertsOnNotWhitelisted() (gas: 20364)
[PASS] test_allocate_revertsOnUnauthorized() (gas: 138464)
```

```
[PASS] test_allocate_success() (gas: 820023)
[PASS] test_burnBeatUSD_integration() (gas: 401294)
[PASS] test_check_vault_adapters() (gas: 283405)
[PASS] test_deAllocate_revertsOnNotWhitelisted() (gas: 20399)
[PASS] test_deAllocate_success() (gas: 1040629)
[PASS] test_getMorphoV2Vaults_returnsCorrectArray() (gas: 134721)
[PASS] test_initialize_grantsAdminRole() (gas: 16055)
[PASS] test_initialize_revertsOnZeroAddress() (gas: 3241737)
[PASS] test_initialize_setsCorrectValues() (gas: 26621)
[PASS] test_mintBeatUSD_integration() (gas: 332888)
[PASS] test_mintBeatUSD_integration_2() (gas: 335865)
[PASS] test_permissionedUnWrap_revertsOnMissingWrapper() (gas: 20743)
[PASS] test_permissionedWrap_revertsOnMissingWrapper() (gas: 20756)
[PASS] test_receive_acceptsEther() (gas: 22501)
[PASS] test_releaseTokensForWithdrawals_revertsOnQueueNotSet() (gas: 50176)
[PASS] test_releaseTokensForWithdrawals_revertsOnTokenNotAllowed() (gas: 51804)
[PASS] test_releaseTokensForWithdrawals_success() (gas: 263341)
[PASS] test_removePermissionedWrap_revertsOnNotExists() (gas: 21892)
[PASS] test_removePermissionedWrap_success() (gas: 827177)
[PASS] test_removeWhitelistedMorphoV2Vault_revertsOnNotWhitelisted() (gas: 20457)
[PASS] test_removeWhitelistedMorphoV2Vault_success() (gas: 114832)
[PASS] test_setAccountant_success() (gas: 26922)
[PASS] test_setBaseAsset_success() (gas: 49531)
[PASS] test_setBeatUSD_revertsOnUnauthorized() (gas: 19718)
[PASS] test_setBeatUSD_success() (gas: 73991)
[PASS] test_setMaxMorphoV2Vaults_revertsOnBelowCurrent() (gas: 134097)
[PASS] test_setMaxMorphoV2Vaults_revertsOnZero() (gas: 18075)
[PASS] test_setMaxMorphoV2Vaults_success() (gas: 27259)
[PASS] test_setTeller_success() (gas: 26923)
[PASS] test_setWithdrawalQueue_success() (gas: 44046)
[PASS] test_toggleWithdrawalToken_success() (gas: 34587)
[PASS] test_wrapHype_revertsOnUnauthorized() (gas: 16154)
[PASS] test_wrapHype_success() (gas: 49061)
Suite result: ok. 42 passed; 0 failed; 0 skipped; finished in 38.89s (123.84s CPU time)

Ran 1 test suite in 38.89s (38.89s CPU time): 42 tests passed, 0 failed, 0 skipped (42 total tests)
```

10.3 Notes on the Test Suite

The Hyperbeat testing suite provided strong and extensive coverage across the core components of the protocol. It thoroughly explored a wide range of functional flows, including the interactions between the Orchestrator contract and the main system or the BeatUSD token. The suite considers various edge cases that the contracts are expected to handle, validating behaviour under both normal and boundary conditions. In addition, it evaluated flows related to different roles and permissions, ensuring that access control logic behaved as intended.