CODESPECT

TEMPEST

SECURITY ASSESSMENT REPORT

27 November, 2024

Prepared for

Contents

1 About CODESPECT 2

2 Disclaimer 2

3 Risk Classification 3

4 Executive Summary 4

5 Summary of the Audit 5
- 5.1 Scope - Audited files . . . 5
- 5.2 Summary of Issues . . . 5

6 System Overview 6
- 6.1 LST/LRT Vault . . . 6
- 6.2 Symmetric Strategy . . . 6
- 6.3 Features of Vaults . . . 7

7 Issues 9
- 7.1 [Medium] `setFees(...)` inside LST strategies should claim fees before updating the rate . . . 9
- 7.2 [Low] Insufficient checks to confirm the correct status of the `sequencerUptimeFeed` . . . 10
- 7.3 [Best Practices] The `redeem(...)` function lacks a complimentary `previewRedeem(...)` function . . . 10

8 Additional Notes 11

9 Evaluation of Provided Documentation 12

10 Test Suite Evaluation 13
- 10.1 Compilation Output . . . 13
- 10.2 Tests Output . . . 13
- 10.3 Notes about Test suite . . . 14

1 About CODESPECT

CODESPECT is a specialized smart contract security firm dedicated to ensuring 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 building 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 Tempest. Tempest is the Large Liquidity Model for Ambient Finance, enabling users to manage liquidity in a non-custodial and automated manner.

This audit focuses on Tempest's LST/LRT arbitrage vaults and symmetric rebalancing strategies (vaults). Both of these contracts operate similarly to ERC-4626 vaults, where users deposit assets in exchange for shares. The majority of the assets are then deployed to Ambient, where they are managed based on the specific vault type to optimize efficiency and returns.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of smart contracts, execution testing.

c) Creation of test cases.

CODESPECT found 3 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.

Organization of the document is as follows:

- **Section 5** summarizes the audit.
- **Section 6** describes the system overview.
- **Section 7** presents the issues.
- **Section 8** contains additional notes for the audit.
- **Section 9** discusses the documentation provided by the client for this audit.
- **Section 10** 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 Summary of the Audit

Audit Type	Security Review
Project Name	Tempest
Type of Project	Management Protocol
Duration of Engagement	2 Week
Duration of Fix Review Phase	1 day
Draft Report	Nov 26, 2024
Final Report	Nov 27, 2024
Repository	tempest_smart_contract
Commit (Audit)	b7f80c391582b19a6e30bf59bb2e28122c9b9d8d
Commit (Final)	ace7ee5e32e9acbe1ee683796097aa3065cfb444
Documentation Assessment	Medium
Test Suite Assessment	High
Auditors	Talfao and Kalogerone

Table 3: Summary of the Audit

5.1 Scope - Audited files

	Contract	LoC	Comments	Ratio	Blank	Total
1	RswEthStrategy.sol	35	22	62.9%	15	72
2	SymetricAmbientStrategy.sol	72	7	9.7%	12	91
3	libraries/VaultLibrary.sol	141	34	24.1%	27	202
4	ambient-lp/SymetricAmbientStorageLayout.sol	7	1	14.3%	2	10
5	ambient-lp/BaseAmbientStorageLayout.sol	5	1	20.0%	2	8
6	ambient-lp/BaseAmbientOperator.sol	38	17	44.7%	15	70
7	ambient-lp/AmbientUser.sol	423	180	42.6%	89	692
8	ambient-lp/AmbientStorageLayout.sol	24	1	4.2%	3	28
9	ambient-lp/AmbientCommon.sol	92	36	39.1%	18	146
10	ambient-lp/AmbientLiquidity.sol	166	76	45.8%	25	267
11	ambient-lp/AmbientInterface.sol	313	100	31.9%	46	459
12	ambient-lp/SymetricAmbientOperator.sol	83	15	18.1%	19	117
13	ambient-lp/AmbientOperator.sol	66	29	43.9%	10	105
14	lst/LstAmbientLiquidity.sol	233	61	26.2%	28	322
15	lst/LstAmbientStorageLayout.sol	26	1	3.8%	3	30
16	lst/LstAmbientInterface.sol	299	88	29.4%	51	438
17	lst/LstAmbientUser.sol	175	85	48.6%	41	301
18	lst/LstAmbientCommon.sol	88	45	51.1%	22	155
19	lst/LstAmbientOperator.sol	118	38	32.2%	31	187
20	lst/rsweth/RswEthAmbientStorageLayout.sol	13	1	7.7%	4	18
21	lst/rsweth/RswEthAmbientUser.sol	8	1	12.5%	2	11
22	lst/rsweth/RswEthAmbientCommon.sol	26	10	38.5%	9	45
23	lst/rsweth/RswEthAmbientOperator.sol	89	18	20.2%	22	129
24	utils/Errors.sol	20	1	5.0%	1	22
25	utils/RswETHOracle.sol	17	3	17.6%	7	27
26	utils/Variables.sol	55	1	1.8%	5	61
27	utils/BridgeOracle.sol	108	22	20.4%	23	153
28	utils/DepositWithdrawPausable.sol	29	1	3.4%	8	38
29	utils/Constants.sol	8	1	12.5%	1	10
	Total	**2777**	**896**	**32.3%**	**541**	**4214**

5.2 Summary of Issues

	Finding	Severity	Update
1	`setFees(...)` inside LST strategies should claim fees before updating the rate	Medium	Fixed
2	Insufficient checks to confirm the correct status of the `sequencerUptimeFeed`	Low	Fixed
3	The `redeem(...)` function lacks a complimentary `previewRedeem(...)` function	Best Practices	Fixed

6 System Overview

Tempest introduces a Large Liquidity Model for Ambient Finance, enabling non-custodial, automated liquidity management while adhering to ERC-4626 standards. Tempest's vaults are designed to optimize returns for users through two distinct strategies, each tailored to a specific purpose.

6.1 LST/LRT Vault

The LST/LRT vault strategy focuses on exploiting arbitrage opportunities. Users deposit a single asset into the vault, and Tempest's current implementation supports the **ETH/rswETH** strategy.

In this strategy, users deposit assets (e.g., `ETH`) via the `deposit(...)` function:

```
function deposit(
    uint256 amount,
    address receiver,
    bytes memory merkleProofs
) external payable depositActive nonReentrant returns (uint256 shares)
```

Upon deposit, users receive an equivalent number of shares representing their claim on the vault's assets. A portion of these assets, determined by the `investedPercentage` parameter, is allocated to Ambient Finance. This percentage is adjustable by Tempest administrators.

The allocated assets are distributed to active knockout positions initialized by Tempest. These positions act as limit orders: when the ETH/rswETH price reaches a target value, the ETH is exchanged for rswETH. These knockout positions enable arbitrage by purchasing rswETH at a discount, which can later be redeemed through the official rswETH contract. Upon full redemption, the ETH is returned to the vault for reinvestment. Over time, the value of user shares should increase unless a significant slashing event occurs for rswETH.

The vault applies a performance fee whenever the total assets exceed their previously stored value. This fee is collected in the form of shares, which Tempest can later redeem.

Users can withdraw their assets or redeem their shares using the `redeem(...)` or `withdraw(...)` functions, respectively:

```
function redeem(
    uint256 shares,
    address receiver,
    address owner,
    bytes memory merkleProofs
) external withdrawActive nonReentrant returns (uint256 assets)

function withdraw(
    uint256 assets,
    address receiver,
    address owner,
    bytes memory merkleProofs
) external withdrawActive nonReentrant returns (uint256 shares)
```

During withdrawals or redemptions, the vault first checks if sufficient assets are available to fulfil the request. If not, liquidity from knockout positions is burned to meet the demand. This process continues iteratively, burning liquidity positions until the required amount of ETH is obtained. If some positions have already been knocked out, rswETH will be sent to the vault, but the user can only withdraw ETH. If the vault cannot provide enough ETH, the transaction reverts.

6.2 Symmetric Strategy

The symmetric strategy is another type of vault designed to manage liquidity positions in Ambient Finance while maintaining a 50:50 token ratio. Users can deposit assets into this vault through two methods: `deposit(...)` or `deposits(...)`.

```
function deposit(
    uint256 amount,
    address receiver,
    bool checkSlippage
) external payable depositActive nonReentrant returns (uint256 shares)

function deposits(
    uint256[] memory amounts,
    address receiver,
    bool checkSlippage
) external payable depositActive nonReentrant returns (uint256 shares)
```

The purpose of both functions is to deposit assets into the vault, which are then invested in an initialized liquidity position on Ambient Finance. The difference lies in the number of tokens deposited:

a. Single-Sided Deposit (`deposit(...)`): Users deposit only the primary token of the vault, defined by `assetIDx`. A portion of this token is swapped on Ambient Finance for the secondary token, and both tokens are added as liquidity to the position.

b. Dual-Sided Deposit (`deposits(...)`): Users deposit both tokens directly into the vault, which are then used to provide liquidity without any swapping.

In both cases, users receive shares equivalent to the total value of the deposited assets, expressed in terms of the main token.

Shares can later be redeemed or withdrawn using one of three functions: `redeem(...)`, `withdraw(...)`, or `redeemWithoutSwap(...)`.

```
function redeem(
    uint256 shares,
    address receiver,
    address owner,
    uint256 minimumReceive,
    bool checkSlippage
) external withdrawActive nonReentrant returns (uint256 assets)

function withdraw(
    uint256 assets,
    address receiver,
    address owner,
    uint256 minimumReceive,
    bool checkSlippage
) external withdrawActive nonReentrant returns (uint256 shares)

function redeemWithoutSwap(
    uint256 shares,
    address receiver,
    address owner,
    bool checkSlippage
) external withdrawActive nonReentrant returns (uint256 amount0, uint256 amount1)
```

a. `redeem(...)` and `withdraw(...)`: These functions provide similar functionality to the LST/LRT vaults, where users receive the main token. However, since liquidity positions include both tokens, the secondary token is first swapped for the main token before being sent to the user.

b. `redeemWithoutSwap(...)`: This function directly returns both tokens from the liquidity position to the user without performing any swaps.

Additionally, upon every deposit or withdrawal, the vault collects earned swap fees, a portion of which is sent to Tempest's wallet as a management fee.

6.3 Features of Vaults

Tempest administrators are responsible for managing both types of vaults. Their responsibilities include rebalancing liquidity positions and ensuring the security mechanisms operate effectively.

- Rebalancing: Vaults can be rebalanced by closing all active positions on Ambient Finance and creating new positions based on updated parameters.

- LST/LRT Redeeming: For LST vaults, administrators are also tasked with requesting the redemption of `rswETH`. This process can take several days to complete, requiring careful planning and monitoring.

Additionally, all deposit and withdrawal operations are **pausable**. This feature allows Tempest administrators to temporarily halt interactions with the vaults in response to potential attacks or issues within the Ambient Finance protocol.

7 Issues

7.1 [Medium] `setFees(...)` inside LST strategies should claim fees before updating the rate

File(s): `LstAmbientInterface.sol`

Description: The `setFees(...)` function allows a user with the `GOVERNANCE_ROLE` to update the `fee` rate, which represents the percentage taken from the profits.

```
function setFees(uint16 _fee) external nonReentrant onlyRole(GOVERNANCE_ROLE) {
    if (_fee > BASE) {
      revert(BAD_SETUP);
    }
    // @audit-issue Fee should be taken from the profits with the old rate here.
    fee = _fee;
    emit FeesSet(_fee);
}
```

The issue lies in how the fee is applied. The profits generated before the `fee` update should be subject to the old `fee` rate, but the current implementation applies the new rate immediately, even to those profits.

Impact: Incorrect profit distribution, as the fees are calculated with the new rate on profits that were generated under the old rate.

Recommendation(s): Consider applying the fee on the existing profits using the old `fee` rate before updating to the new value.

Status: Fixed

Update from the Tempest: Fixed PR: pr-197. This fix will be upgraded later when we have a big upgrade, currently setFees can be called by only Tempest's operator, we will solve it off-chain by collecting all fee before setting new fee.

7.2 [Low] Insufficient checks to confirm the correct status of the `sequencerUptimeFeed`

File(s): `BridgeOracle.sol`

Description: For `Layer 2` network deployments, the `_checkSequencerDowntime()` function inside the `BridgeOracle`contract is responsible for correctly check if the `sequencer` of the network is up and `revert` if it's down. This check always happens before retrieving a price from the Chainlink Aggregator to ensure up-to-date prices.

```
function _checkSequencerDowntime() internal view {
  if (isL2) {
    // check sequencer downtime
    (, int256 answer, uint256 startedAt, , ) = IOracle(sequencerUptimeOracle).latestRoundData();

    bool isSequencerUp = answer == 0;
    if (!isSequencerUp) {
      revert(SEQUENCER_DOWN);
    }

    // Make sure the grace period has passed after the
    // sequencer is back up.
    uint256 timeSinceUp = block.timestamp - startedAt;
    if (timeSinceUp <= sequencerDowntimeLimit) {
      revert(SEQUENCER_DOWN);
    }
}
```

There is an important check missing. It is possible that `answer` = `0` which means that the sequencer is up and `startedAt` = `0` which means that the round is invalid.

Normally, when a round starts, `startedAt` is recorded, and the initial status (`answer`) is set to `0`. Later, both the `answer` and the time it was updated (`updatedAt`) are set at the same time after getting enough data from oracles, making sure that `answer` only changes from `0` when there's a confirmed update different from the start time.

It is possible when a round starts, the beginning `startedAt` is recorded to be `0`, and `answer`, the initial status is set to be `0`. In this case here, `answer` and `startedAt` can be `0` initially, until after all data is gotten from oracles and update is confirmed then the values are reset to the correct values that show the correct status of the sequencer [Ref].

Impact: Inadequate checks to confirm the correct status of the `sequencerUptimeFeed` will cause the `latestAnswer()` and `numeraireLatestAnswer()` functions to not revert even when the sequencer uptime feed is not updated or is called in an invalid round.

Recommendation(s): Consider checking if the `startedAt` value is equal to `0` in the `_checkSequencerDowntime()` function.

Status: Fixed

Update from the Tempest: Fixed PR: pr-192. This fix will be upgraded for current vaults.

7.3 [Best Practices] The `redeem(...)` function lacks a complimentary `previewRedeem(...)` function

File(s): `LstAmbientInterface.sol`, `AmbientInterface.sol`

Description: Both the `LstAmbientInterface` and the `AmbientInterface` have complimentary `previewDeposit(...)` and `previewWithdraw(...)` functions for their respective `deposit(...)` and `withdraw(...)` functions. However, the `redeem(...)` functions don't have a complimentary `previewRedeem(...)` function.

Impact: Vault users are unable to preview and calculate the results of calling the `redeem(...)` function and possibly set the slippage more accurately.

Recommendation(s): Consider adding a `previewRedeem(...)` function.

Status: Fixed

Update from the Tempest: Fixed PR: pr-196. This fix will be upgraded later when we have a big upgrade, because currently 'convertToAssets' can be used on behalf.

8 Additional Notes

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

- The call to `IERC20Metadata(lParams.token1).safeIncreaseAllowance(address(crocsSwapDex), type(uint256).max)` within the `initialize(...)` function of the LST Vault is redundant [code]. The token in question is exclusively received from Ambient after completing the knockout position and will never be deposited back into Ambient. Consequently, its allowance for Ambient can safely remain set to 0. [Resolved in pr-198]
- In the `lstAmbient` vaults, the fee is collected in the `_updateStratPnLAndFees` function during every **deposit** or **withdrawal** if there is a positive PnL. If the fee is 1% and the profit is `<100`, the fee collected will be `0` and lost, due to rounding down [code]. [Acknowledged]

9 Evaluation of Provided Documentation

The Tempest documentation was provided in four forms:

- **Official Documentation Website:** The Gitbook contains high-level use cases for each vault type, providing an overview of the protocol's purpose for both users and auditors.
- **Natspec Comments:** Most of the code included Natspec comments, which explained the purpose of complex functionality in detail and facilitated understanding of individual functions. However, some functionalities lacked comments, and expanding documentation coverage would enhance the overall comprehensibility of the code.
- **Inheritance and Interaction Diagrams:** Non-public inheritance and high-level interaction diagrams were provided. These effectively clarified the contracts' complex inheritance structures and interactions, aiding CODESPECT's analysis significantly.
- **Mathematical Equations:** The Tempest team provided the mathematical equations used within the protocol. These equations were instrumental in understanding the protocol's calculations and logic.

The documentation provided by Tempest offered valuable insights into the protocol, significantly aiding CODESPECT's understanding. However, the public technical documentation could be further improved to better present the protocol's overall functionality and facilitate the understanding of each component.

Additionally, the Tempest team was consistently available and responsive, promptly addressing all questions raised by CODESPECT during the evaluation process.

10 Test Suite Evaluation

10.1 Compilation Output

```
> forge compile
[] Compiling...
[] Compiling 123 files with Solc 0.8.23
[] Solc 0.8.23 finished in 46.07s
Compiler run successful!
```

10.2 Tests Output

```
> forge test
[] Compiling...
No files changed, compilation skipped

Ran 1 test for test/RswETHClaimTestToken.t.sol:RswClaimTestToken
[PASS] test_ClaimCmd() (gas: 77995)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 373.00ms (17.57ms CPU time)

Ran 1 test for test/WstETHClaimTestToken.t.sol:LSTClaimTestToken
[PASS] test_ClaimCmd() (gas: 77994)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 389.41ms (10.17ms CPU time)

Ran 2 tests for test/FeeCollection.t.sol:FeeCollectionTest
[PASS] testFuzz_FeeCollection() (gas: 2259345)
[PASS] testFuzz_SetFee() (gas: 2091634)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 636.13ms (271.33ms CPU time)

Ran 7 tests for test/DepositSymTestToken1.t.sol:DepositSymTestToken1
[PASS] testFuzz_Deposit1(uint256) (runs: 1003, : 724373, ~: 732984)
[PASS] testFuzz_DepositAndUpdateWidth(uint256) (runs: 1003, : 1121428, ~: 1148752)
[PASS] testFuzz_DepositSymAndRebalance1(uint256) (runs: 1003, : 1115605, ~: 1136055)
[PASS] testFuzz_FeeCollection() (gas: 1827893)
[PASS] testFuzz_SetFee() (gas: 1641005)
[PASS] testFuzz_deposit1MultipeUser(uint256,uint256) (runs: 1003, : 1236483, ~: 1236488)
[PASS] testFuzz_depositAndWithdraw(uint256) (runs: 1003, : 931534, ~: 944492)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 77.99s (127.72s CPU time)

Ran 5 tests for test/DepositTestToken1.t.sol:DepositTestToken1
[PASS] testFuzz_Deposit1(uint256) (runs: 1003, : 864295, ~: 905522)
[PASS] testFuzz_DepositAndRebalance1(uint256) (runs: 1003, : 1163246, ~: 1193866)
[PASS] testFuzz_deposit1MultipeUser(uint256,uint256) (runs: 1003, : 1407356, ~: 1407356)
[PASS] testFuzz_depositAndWithdraw(uint256) (runs: 1003, : 1205835, ~: 1304280)
[PASS] testSimple1() (gas: 835444)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 81.07s (135.79s CPU time)

Ran 4 tests for test/DepositTestToken0.t.sol:DepositTestToken0
[PASS] testFuzz_Deposit0(uint256) (runs: 1003, : 709259, ~: 775044)
[PASS] testFuzz_deposit0AndWithdraw(uint256) (runs: 1003, : 918227, ~: 1012681)
[PASS] testFuzz_depositMultipeUser(uint256,uint256) (runs: 1003, : 1240367, ~: 1291733)
[PASS] testSimple0() (gas: 685074)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 88.27s (87.73s CPU time)
```

```
Ran 13 tests for test/WstETHTestToken.t.sol:WstETHTestToken
[PASS] testFuzz_DepositAndRebalanceLST(uint256,uint256[3]) (runs: 1003, : 1421094, ~: 1421150)
[PASS] testFuzz_DepositLST(uint256) (runs: 1003, : 1058690, ~: 1058690)
[PASS] testFuzz_WithdrawAllFromLido(uint256) (runs: 1003, : 2654790, ~: 2654790)
[PASS] testFuzz_WithdrawFromLido(uint256) (runs: 1003, : 3717252, ~: 3717268)
[PASS] testFuzz_claimFromLido(uint256) (runs: 1003, : 3169226, ~: 3169237)
[PASS] testFuzz_depositLSTAndRedeem(uint256) (runs: 1003, : 1197288, ~: 1194419)
[PASS] testFuzz_depositLSTAndWithdraw(uint256) (runs: 1003, : 1398339, ~: 1398350)
[PASS] testFuzz_depositLSTInvestAndWithdraw(uint256) (runs: 1003, : 1050969, ~: 1050124)
[PASS] testFuzz_depositLSTMultipeUser(uint256,uint256) (runs: 1003, : 1302450, ~: 1302511)
[PASS] testFuzz_getWithdrawalDatas(uint256) (runs: 1003, : 2290412, ~: 2290424)
[PASS] test_depositWstWhenVaultHavingKnockedOutPositions() (gas: 2359159)
[PASS] test_totalAssetsWithIdle() (gas: 1407974)
[PASS] test_unstake() (gas: 1521331)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 109.37s (463.44s CPU time)

Ran 9 tests for test/DepositSymTestToken0.t.sol:DepositSymTestToken0
[PASS] testFuzz_DepositSym0(uint256) (runs: 1003, : 583672, ~: 592289)
[PASS] testFuzz_DepositSymAndUpdateWidth0(uint256) (runs: 1003, : 1176374, ~: 1176374)
[PASS] testFuzz_FeeCollection() (gas: 1892790)
[PASS] testFuzz_SetFee() (gas: 1706928)
[PASS] testFuzz_depositSym0AndWithdraw(uint256) (runs: 1003, : 784121, ~: 787739)
[PASS] testFuzz_depositSymMultipeUser(uint256,uint256) (runs: 1003, : 2996066, ~: 3031880)
[PASS] test_Deposits() (gas: 658405)
[PASS] test_Deposits2() (gas: 658387)
[PASS] test_DepositsAndWithdraw() (gas: 804900)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 114.88s (120.18s CPU time)

Ran 12 tests for test/RswETHTestToken.t.sol:RswTestToken
[PASS] testFuzz_DepositAndRebalanceRsw(uint256,uint256[3]) (runs: 1003, : 1303288, ~: 1303418)
[PASS] testFuzz_DepositRsw(uint256) (runs: 1003, : 904809, ~: 904809)
[PASS] testFuzz_WithdrawAllFromSwell(uint256) (runs: 1003, : 4714244, ~: 4714244)
[PASS] testFuzz_WithdrawFromSwell(uint256) (runs: 1003, : 5324710, ~: 5324710)
[PASS] testFuzz_claimFromSwell(uint256) (runs: 1003, : 6247818, ~: 6247831)
[PASS] testFuzz_depositRswAndRedeem(uint256) (runs: 1003, : 1047654, ~: 1038952)
[PASS] testFuzz_depositRswAndWithdraw(uint256) (runs: 1003, : 1339446, ~: 1339460)
[PASS] testFuzz_depositRswInvestAndWithdraw(uint256) (runs: 1003, : 928067, ~: 923725)
[PASS] testFuzz_depositRswMultipeUser(uint256,uint256) (runs: 1003, : 1113578, ~: 1113634)
[PASS] testFuzz_getWithdrawalDatas(uint256) (runs: 1003, : 5161314, ~: 5161314)
[PASS] test_depositRswWhenVaultHavingKnockedOutPositions() (gas: 2074096)
[PASS] test_unstake() (gas: 1407910)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 150.66s (660.68s CPU time)

Ran 9 test suites in 150.69s (623.64s CPU time): 54 tests passed, 0 failed, 0 skipped (54 total tests)
```

10.3 Notes about Test suite

The Tempest team delivered a robust and comprehensive test suite, showcasing a well-structured approach to ensuring the protocol's correctness and resilience. Key highlights of the test suite include:

- **Fuzzing Tests:** The extensive use of fuzzing tests enabled coverage of numerous edge cases, particularly for core functionalities such as deposits, withdrawals, and fee collection. These tests validate the behaviour of the system under a wide range of inputs, uncovering potential vulnerabilities or inconsistencies that could emerge in unexpected conditions.
- **Complex Scenarios:** Beyond basic operations, the test suite included sophisticated scenarios such as rebalancing positions across vaults. These tests verified the correctness and stability of the system in handling advanced operational flows, reinforcing confidence in the protocol's ability to operate effectively under dynamic conditions.
- **Coverage of Multiple Functional Areas:** Separate test cases targeted specific vault types (e.g., LST and Symmetric strategy vaults), token interactions, and withdrawal mechanics. This segmented testing approach helped isolate issues and provided clear insights into how different components of the system operate and interact.

Overall, the test suite reflects a mature development process and significantly enhances the reliability of the protocol. While the suite's depth is commendable, ongoing improvements, such as extending fuzzing ranges and incorporating even more complex operational scenarios, could further solidify its effectiveness.

CODESPECT also recommends explicitly defining strict invariants that the protocol must uphold. Incorporating tests to validate these invariants would ensure that critical assumptions about the system's behaviour are consistently maintained across all functionalities, further bolstering the protocol's security and stability.