CODESPECT

Tempest DepositIdle Feature

SECURITY ASSESSMENT REPORT

18 March, 2025

Prepared for

TEMPEST

Contents

1 About CODESPECT 2

2 Disclaimer 2

3 Risk Classification 3

4 Executive Summary 4

5 Audit Summary 5
5.1 Scope - Audited Files 5
5.2 Findings Overview 5

6 System Overview 6

7 Issues 7
7.1 [Medium] Tempest vault not fully compatible with Royco IAM Vault 7

8 Evaluation of Provided Documentation 8

9 Test Suite Evaluation 9
9.1 Compilation Output 9
9.2 Tests Output 9
9.3 Notes about Test suite 11

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 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 the review of the new feature `depositIdle(...)` function, which allows users to deposit base or quote assets into the idle balance of the Vault. The Vault Idle is then allocated to Ambient during the rebalancing process. Furthermore, this feature was introduced as part of the planned integration of Tempest Vault with Royco's protocol, specifically the IAM Vault. Consequently, the audit also considers potential integration challenges with this protocol.

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 1 point of attention, classified as `Medium`. 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** 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
Total	**0**	**1**	**0**

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

5 Audit Summary

Audit Type	Security Review
Project Name	Tempest
Type of Project	Vault/Royco Integration
Duration of Engagement	3 Days
Duration of Fix Review Phase	2 Days
Draft Report	March 12, 2025
Final Report	March 18, 2025
Repository	tempest_smart_contract
Commit (Audit)	5d5dd30c5b33e533f95c42aa421a136f0e6e320f
Commit (Final)	85ed7344cd2d874ea32203109c993565cd6338f7
Documentation Assessment	High
Test Suite Assessment	High
Auditors	Talfao, Kalogerone

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	Contract	**LoC**
1	AmbientUser.sol	460
2	AmbientCommon.sol	114
3	AmbientInterface.sol	338
	Total	**912**

Scope information

Only the code changes from the main branch (a613ef5a7e8d41573705d6d2a8996340192d604b) were considered within the scope of this audit, along with components related to the Royco integration. As a result, the audit focused solely on the modified and relevant parts of the code rather than reviewing entire files.

5.2 Findings Overview

	Finding	**Severity**	**Update**
1	Tempest vault not fully compatible with Royco IAM Vault	Medium	Fixed

6 System Overview

Tempest introduces a new feature, `depositIdle(...)`, which is intended for future integration with Royco's Vault.

```
function depositIdle(uint8 assetIdx_, uint256 amount, address receiver)
   external payable depositActive nonReentrant returns (uint256 shares)
```

This function enables users to deposit base or quote assets into the Vault's idle balance, which is later utilized in Ambient during the rebalancing process. The implementation of this feature led to the change of the `_totalAssets(...)` function, as the new internal function `_toAssetAmount(...)` was introduced. This function extracts part of the logic from original `_totalAssets(...)`.

```
function _toAssetAmount(
    uint8 _assetIdx,
    address[2] memory _tokenAddresses,
    uint256 amount0,
    uint256 amount1,
    uint256 oracleLatestAnswer,
    uint8 oracleDecimals
    ) internal view returns (uint256)
```

The `_toAssetAmount(...)` function calculates the value of a base asset in terms of a quote asset (or vice versa), depending on the configuration of the Tempest vault defined by `assetIdx`. This function plays a crucial role in `depositIdle(...)`, ensuring that deposits are accounted for in a single asset type. Additionally, `_totalAssets(...)` relies on this function to determine the total value of all assets within the protocol.

Furthermore, Tempest plans to integrate its Vault (Symmetric Strategy) with the Royco protocol on the Plume network. The Royco IAM vault will incorporate the Tempest vault and introduce an incentive mechanism to attract more assets to the Tempest ecosystem.

7 Issues

7.1 [Medium] Tempest vault not fully compatible with Royco IAM Vault

File(s): `AmbientInterface.sol`

Description: The Tempest team has introduced a new feature, `depositIdle`, designed to incentivize users to deposit into Tempest via Royco on the Plume network. However, deposits must be processed through Royco's Vault IAM, which requires partial ERC-4626 compatibility. The Tempest Vault, however, does not fully comply with this standard.

To successfully integrate the Tempest Vault with Royco's IAM Vault, the following minimum requirements must be met [ref]:

- `asset()` function must not revert → *Not implemented in Tempest.*;
- `deposit()` function must not revert → *Tempest intends to use `depositIdle()` for Royco instead.*;
- `convertToAssets()` function must not revert → *Implemented.*;
- `previewDeposit()` function must return an accurate value → *Implemented.*;
- ERC-20 functionality must remain intact → *Implemented.*;

Additional compatibility issues

There are also differences in the parameter structure of deposit and withdrawal functions between Tempest and Royco Vaults:

- Tempest's `withdraw(...)` function takes **five** parameters, whereas Royco's `withdraw(...)` uses **three**:;

```
VAULT.withdraw(uint256, address, address)
```

- Tempest's `depositIdle(uint8, uint256, address)` (even if renamed to `deposit(...)`) differs from Royco's deposit function:;

```
VAULT.deposit(uint256, address)
```

- Royco's `mint(...)` function will not work as Tempest does not implement any `mint(...)` function;

Withdrawal reversion risk

Additionally, Royco's `withdraw(...)` function is likely to revert due to its built-in slippage check. Royco verifies that:

`expectedShares == shares`

Where:

- **`expectedShares`** is derived from `previewWithdraw(...)`;
- **`shares`** is returned from `withdraw(...)`;

However, Tempest collects fees from Ambient during the execution of `withdraw(...)`, which may cause the returned `shares` to be lower than `expectedShares`, triggering a revert.

A potential workaround is to exclusively use Royco's `redeem(...)` function, which does not include the slippage check.

Impact: The Tempest Vault is currently incompatible with Royco's Vault IAM, which could prevent successful integration and disrupt the planned incentive mechanism.

Recommendations:

- Rename `depositIdle(...)` to `deposit(...)`, ensuring an equivalent function remains for the existing `deposit(...)` implementation;
- Implement the `asset(...)` getter function to meet compatibility requirements;
- Modify deposit and withdrawal function definitions to align with Royco Vault's structure;
- Consider using only the `redeem(...)` function, as it bypasses Royco's slippage check;

Status: Fixed

Update from Tempest:

8 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.

9 Test Suite Evaluation

9.1 Compilation Output

```
> forge compile
[] Compiling...
[] Compiling 126 files with Solc 0.8.23
[] Solc 0.8.23 finished in 25.37s
Compiler run successful!
```

9.2 Tests Output

```
> forge test
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 212.61ms (2.87ms CPU time)

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 232.39ms (998.81µs CPU time)

Ran 2 tests for test/FeeCollection.t.sol:FeeCollectionTest
[PASS] testFuzz_FeeCollection() (gas: 2258976)
[PASS] testFuzz_SetFee() (gas: 2091464)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 259.82ms (41.25ms CPU time)

Ran 4 tests for test/DepositTestToken0.t.sol:DepositTestToken0
[PASS] testFuzz_Deposit0(uint256) (runs: 1000, : 708847, ~: 775124)
[PASS] testFuzz_deposit0AndWithdraw(uint256) (runs: 1000, : 914052, ~: 1012653)
[PASS] testFuzz_depositMultipeUser(uint256,uint256) (runs: 1000, : 1237271, ~: 1291403)
[PASS] testSimple0() (gas: 684846)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 43.26s (43.06s CPU time)

Ran 5 tests for test/DepositTestToken1.t.sol:DepositTestToken1
[PASS] testFuzz_Deposit1(uint256) (runs: 1000, : 874261, ~: 929094)
[PASS] testFuzz_DepositAndRebalance1(uint256) (runs: 1000, : 1164649, ~: 1210898)
[PASS] testFuzz_deposit1MultipeUser(uint256,uint256) (runs: 1000, : 1407200, ~: 1407200)
[PASS] testFuzz_depositAndWithdraw(uint256) (runs: 1000, : 1207562, ~: 1302245)
[PASS] testSimple1() (gas: 835192)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 57.70s (67.98s CPU time)

Ran 13 tests for test/DepositSymTestToken0PlumeChronicle.t.sol:DepositSymTestToken0PlumeChronicle
[PASS] testFuzz_DepositSym0Plume(uint256) (runs: 1000, : 587022, ~: 593341)
[PASS] testFuzz_DepositSymAndUpdateWidth0Plume(uint256) (runs: 1000, : 1190068, ~: 1190068)
[PASS] testFuzz_FeeCollectionPlume() (gas: 1603676)
[PASS] testFuzz_SetFeePlume() (gas: 1419592)
[PASS] testFuzz_depositSym0AndWithdrawPlume(uint256) (runs: 1000, : 758268, ~: 758673)
[PASS] testFuzz_depositSymMultipeUserPlume(uint256,uint256) (runs: 1000, : 2980999, ~: 3014423)
[PASS] test_DepositIdle0Plume() (gas: 1636341)
[PASS] test_DepositIdle0PlumeWithSlippageCheck() (gas: 616916)
[PASS] test_DepositIdle1Plume() (gas: 1718334)
[PASS] test_DepositIdle1PlumeWithSlippageCheck() (gas: 677261)
[PASS] test_Deposits2Plume() (gas: 695873)
[PASS] test_DepositsAndWithdrawPlume() (gas: 835092)
[PASS] test_DepositsPlume() (gas: 695870)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 59.37s (65.57s CPU time)

Ran 7 tests for test/DepositSymTestToken1.t.sol:DepositSymTestToken1
[PASS] testFuzz_Deposit1(uint256) (runs: 1000, : 726305, ~: 735037)
[PASS] testFuzz_DepositAndUpdateWidth(uint256) (runs: 1000, : 1123388, ~: 1151090)
[PASS] testFuzz_DepositSymAndRebalance1(uint256) (runs: 1000, : 1119051, ~: 1138393)
[PASS] testFuzz_FeeCollection() (gas: 1829298)
[PASS] testFuzz_SetFee() (gas: 1642543)
[PASS] testFuzz_deposit1MultipeUser(uint256,uint256) (runs: 1000, : 1238411, ~: 1238415)
[PASS] testFuzz_depositAndWithdraw(uint256) (runs: 1000, : 934065, ~: 946416)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 63.76s (76.65s CPU time)
```

```
Ran 9 tests for test/DepositSymTestToken0.t.sol:DepositSymTestToken0
[PASS] testFuzz_DepositSym0(uint256) (runs: 1000, : 585851, ~: 594344)
[PASS] testFuzz_DepositSymAndUpdateWidth0(uint256) (runs: 1000, : 1178714, ~: 1178714)
[PASS] testFuzz_FeeCollection() (gas: 1894233)
[PASS] testFuzz_SetFee() (gas: 1708485)
[PASS] testFuzz_depositSym0AndWithdraw(uint256) (runs: 1000, : 787435, ~: 790455)
[PASS] testFuzz_depositSymMultipeUser(uint256,uint256) (runs: 1000, : 2997753, ~: 3035863)
[PASS] test_Deposits() (gas: 660424)
[PASS] test_Deposits2() (gas: 660428)
[PASS] test_DepositsAndWithdraw() (gas: 807459)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 69.65s (71.35s CPU time)
Ran 7 tests for test/DepositSymTestToken1PlumeChronicle.t.sol:DepositSymTestToken1PlumeChronicle
[PASS] testFuzz_Deposit1Plume(uint256) (runs: 1000, : 762870, ~: 780337)
[PASS] testFuzz_DepositAndUpdateWidthPlume(uint256) (runs: 1000, : 1208019, ~: 1261979)
[PASS] testFuzz_DepositSymAndRebalance1Plume(uint256) (runs: 1000, : 1190368, ~: 1217229)
[PASS] testFuzz_FeeCollectionPlume() (gas: 1650908)
[PASS] testFuzz_SetFeePlume() (gas: 1487888)
[PASS] testFuzz_deposit1MultipeUserPlume(uint256,uint256) (runs: 1000, : 1236383, ~: 1236383)
[PASS] testFuzz_depositAndWithdrawPlume(uint256) (runs: 1000, : 965777, ~: 996801)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 74.86s (59.17s CPU time)
Ran 13 tests for test/WstETHTestToken.t.sol:WstETHTestToken
[PASS] testFuzz_DepositAndRebalanceLST(uint256,uint256[3]) (runs: 1000, : 1421070, ~: 1421144)
[PASS] testFuzz_DepositLST(uint256) (runs: 1000, : 1058690, ~: 1058690)
[PASS] testFuzz_WithdrawAllFromLido(uint256) (runs: 1000, : 2654790, ~: 2654790)
[PASS] testFuzz_WithdrawFromLido(uint256) (runs: 1000, : 3717252, ~: 3717266)
[PASS] testFuzz_claimFromLido(uint256) (runs: 1000, : 3169226, ~: 3169237)
[PASS] testFuzz_depositLSTAndRedeem(uint256) (runs: 1000, : 1197673, ~: 1194419)
[PASS] testFuzz_depositLSTAndWithdraw(uint256) (runs: 1000, : 1398340, ~: 1398352)
[PASS] testFuzz_depositLSTInvestAndWithdraw(uint256) (runs: 1000, : 1050760, ~: 1050121)
[PASS] testFuzz_depositLSTMultipeUser(uint256,uint256) (runs: 1000, : 1302449, ~: 1302511)
[PASS] testFuzz_getWithdrawalDatas(uint256) (runs: 1000, : 2290411, ~: 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 74.86s (269.66s CPU time)
Ran 9 tests for test/DepositSymTestToken0Plume.t.sol:DepositSymTestToken0Plume
[PASS] testFuzz_DepositSym0Plume(uint256) (runs: 1000, : 621896, ~: 630421)
[PASS] testFuzz_DepositSymAndUpdateWidth0Plume(uint256) (runs: 1000, : 1110564, ~: 1110564)
[PASS] testFuzz_FeeCollectionPlume() (gas: 1637798)
[PASS] testFuzz_SetFeePlume() (gas: 1448993)
[PASS] testFuzz_depositSym0AndWithdrawPlume(uint256) (runs: 1000, : 797725, ~: 795862)
[PASS] testFuzz_depositSymMultipeUserPlume(uint256,uint256) (runs: 1000, : 3010564, ~: 3040047)
[PASS] test_Deposits2Plume() (gas: 733419)
[PASS] test_DepositsAndWithdrawPlume() (gas: 877374)
[PASS] test_DepositsPlume() (gas: 733460)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 100.35s (62.87s CPU time)
Ran 7 tests for test/DepositSymTestToken1Plume.t.sol:DepositSymTestToken1Plume
[PASS] testFuzz_Deposit1Plume(uint256) (runs: 1000, : 777582, ~: 785871)
[PASS] testFuzz_DepositAndUpdateWidthPlume(uint256) (runs: 1000, : 1127041, ~: 1134858)
[PASS] testFuzz_DepositSymAndRebalance1Plume(uint256) (runs: 1000, : 1120164, ~: 1123037)
[PASS] testFuzz_FeeCollectionPlume() (gas: 1686571)
[PASS] testFuzz_SetFeePlume() (gas: 1513196)
[PASS] testFuzz_deposit1MultipeUserPlume(uint256,uint256) (runs: 1000, : 1249720, ~: 1249720)
[PASS] testFuzz_depositAndWithdrawPlume(uint256) (runs: 1000, : 975774, ~: 993124)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 100.35s (55.95s CPU time)

Ran 12 tests for test/RswETHTestToken.t.sol:RswTestToken
[PASS] testFuzz_DepositAndRebalanceRsw(uint256,uint256[3]) (runs: 1000, : 1303284, ~: 1303388)
[PASS] testFuzz_DepositRsw(uint256) (runs: 1000, : 904809, ~: 904809)
[PASS] testFuzz_WithdrawAllFromSwell(uint256) (runs: 1000, : 4714244, ~: 4714244)
[PASS] testFuzz_WithdrawFromSwell(uint256) (runs: 1000, : 5324710, ~: 5324710)
[PASS] testFuzz_claimFromSwell(uint256) (runs: 1000, : 6247820, ~: 6247834)
[PASS] testFuzz_depositRswAndRedeem(uint256) (runs: 1000, : 1047393, ~: 1038952)
[PASS] testFuzz_depositRswAndWithdraw(uint256) (runs: 1000, : 1339444, ~: 1339457)
[PASS] testFuzz_depositRswInvestAndWithdraw(uint256) (runs: 1000, : 928671, ~: 923730)
[PASS] testFuzz_depositRswMultipeUser(uint256,uint256) (runs: 1000, : 1113571, ~: 1113634)
[PASS] testFuzz_getWithdrawalDatas(uint256) (runs: 1000, : 5161314, ~: 5161314)
[PASS] test_depositRswWhenVaultHavingKnockedOutPositions() (gas: 2074096)
[PASS] test_unstake() (gas: 1407910)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 110.67s (354.06s CPU time)
Ran 13 test suites in 110.69s (755.53s CPU time): 90 tests passed, 0 failed, 0 skipped (90 total tests)
```

9.3 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.