Quantstamp

Tempest Finance

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Strategy Vaults
Timeline	2024-08-21 through 2024-09-05
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• https://github.com/Tempest-Finance/tempest_smart_contract #775b386
Auditors	• Jennifer Wu Auditing Engineer • Roman Rohleder Senior Auditing Engineer • Rabib Islam Senior Auditing Engineer

Documentation quality	Medium
Test quality	Medium
Total Findings	12 Fixed: 8 Acknowledged: 2 Mitigated: 2
High severity findings	0
Medium severity findings	2 Fixed: 2
Low severity findings	5 Fixed: 4 Mitigated: 1
Undetermined severity findings	1 Fixed: 1
Informational findings	4 Fixed: 1 Acknowledged: 2 Mitigated: 1

Summary of Findings

Quantstamp conducted an audit of the Tempest Finance protocol, which includes four strategy vaults built with Ambient Finance concentrated liquidity and knockout positions: `SymetricAmbientStrategy`, `BaseAmbientStrategy`, `RswEthStrategy`, and `WstEthStrategy`.

The `SymetricAmbientStrategy` is a passive rebalancing strategy that uses base and limit positions to optimize liquidity provision. It balances liquidity between deposited tokens through limit orders and base positions.

The `BaseAmbientStrategy` is a vault that accepts single-token deposits from users. Guardians can use the `rebalance()` function to deploy concentrated liquidity positions into the vault, allowing them to actively manage user funds.

The `RswEthStrategy` and `WstEthStrategy` are liquid staking token (LST) vaults. They leverage knockout liquidity positions to arbitrage between the pool price and the LST rate, creating opportunities for fee income and arbitrage profits.

The audit uncovered 12 findings and 7 suggestions for code improvements. We recommend the client to consider all identified issues.

Fix Review: The client has addressed all findings and suggestions. During the fix review, the client refactored the codebase to eliminate redundancy and improve code quality, while also making the contracts upgradeable. The client also implemented a solution to mitigate the price impact from large deposits by supporting dual deposits, and a mechanism has been added to claim fees from Ambient Finance knockout liquidity positions. Furthermore, the protocol also allows the governor to pause deposits or withdrawals from the vault.

ID	DESCRIPTION	SEVERITY	STATUS
TEM-1	Incorrect Liquidity Provision when Investing Knockout Positions	● Medium	Fixed
TEM-2	Improper Use of Oracle May Lead to Incorrect Outcomes	● Medium	Fixed
TEM-3	Miscalculation of Profit and Fees when Depositing	● Low	Fixed
TEM-4	Rounding Direction Can Lead to Loss	● Low	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
TEM-5	First Depositor Vulnerable to Inflation Attack	● Low	Mitigated
TEM-6	Rebalancing May Be Blocked by Excess Token Balance	● Low	Fixed
TEM-7	Incorrect Emission of Withdrawal Amount in `_claimFromLido()`	● Low	Fixed
TEM-8	Redundant Ambient Finance LP Burn User Command	● Informational	Fixed
TEM-9	Missing Input Validation	● Informational	Mitigated
TEM-10	Loop / Gas Concerns	● Informational	Acknowledged
TEM-11	Inconsistent Liquidity Allocation Due to Dynamic Recalculation of LP Weights	● Informational	Acknowledged
TEM-12	Anyone Can Re-Invest `investedPercentage` Knockout Liquidity	● Undetermined	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer
Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

The following files were in-scope during the initial audit:

Files

- `12b...c5a ./src/BaseAmbientStrategy.sol`
- `ed4...7bb ./src/RswEthStrategy.sol`
- `0b6...1e4 ./src/WstEthStrategy.sol`
- `10c...c08 ./src/SymetricAmbientStrategy.sol`
- `8e5...e96 ./src/utils/Errors.sol`
- `9ed...d98 ./src/utils/WstETHOracle.sol`
- `114...970 ./src/utils/Constants.sol`
- `5ac...6bc ./src/utils/RswETHOracle.sol`
- `f37...0b1 ./src/libraries/VaultLibrary.sol`
- `768...8ef ./src/libraries/LiquidityMath.sol`
- `4ae...f99 ./src/libraries/LiquidityAmountsNative.sol`
- `09a...8e5 ./src/libraries/LiquidityAmounts.sol`
- `fe3...cf3 ./src/libraries/FixedPoint.sol`
- `f97...0c4 ./src/libraries/TickMath.sol`
- `112...058 ./src/libraries/CompoundMath.sol`
- `9dc...43b ./src/libraries/CurveMath.sol`
- `00c...870 ./src/libraries/SafeCast.sol`
- `1b6...7fd ./src/libraries/PoolSpecs.sol`

Tests

- `a99...2c1 ./test/DepositSymTestToken1.t.sol`
- `f48...1cf ./test/FeeCollection.t copy.sol`
- `9f3...128 ./test/DepositSymTestToken0.t.sol`
- `0cf...08e ./test/RswETHTestToken.t.sol`
- `428...adf ./test/DepositTestToken1.t.sol`
- `aa7...8f2 ./test/WstETHTestToken.t.sol`
- `070...cc3 ./test/DepositTestToken0.t.sol`

The Files (Appendix) section was updated following the fix review.

Operational Considerations

Managing user funds within the strategy involves several key risks that must be carefully addressed to ensure effective and secure fund management.

Consider the role of `GUARDIAN_ROLE` (strategy managers), which is responsible for rebalancing and optimizing fund usage. There is a risk that funds could be mismanaged, either through malicious actions or inefficient strategies. Additionally, `GOVERNANCE_ROLE` holds significant power, as holders of the role can use the `execute()` function to transfer all funds to any arbitrary address, as well as interact with any external contract. This level of control introduces risks that should be carefully managed to prevent misuse.

Fix Review: The client removed the `execute()` function.

The strategy contract grants unlimited approval (`type(uint256).max`) to the Ambient Finance contract (`crocsSwapDex`) for token transfers. While this reduces the need for repeated approvals and saves gas costs, it presents a potential security vulnerability. If the Ambient Finance contract were compromised, unauthorized token transfers could occur. Ensuring that adequate safeguards are in place to mitigate this risk should be considered.

`WstEthStrategy` and `RswEthStrategy` contracts are specifically designed to handle ETH deposits. It is important to ensure that `_tokenAddresses[_assetIdx]` is always set to `address(0)` to represent ETH. Any deviation from this configuration could result in miscalculations, as the contract assumes funds are transmitted via `msg.value`. This could lead to errors in how the portfolio's performance is updated, so verifying correct setup is critical.

In the `withdraw()` and `redeem()` functions of `BaseAmbientStrategy` and `SymetricAmbientStrategy`, the `minimumReceive` argument specifies the minimum amount of assets the receiver is expected to obtain. This value should be carefully set during each function call, as slippage is not otherwise accounted for. Failure to account for this could result in users receiving fewer assets than anticipated, so setting this parameter with care is essential.

Fix Review: The client updated the withdraw and redeem functions to check for slippage. The users can opt for the force alternatives to skip slippage checks.

Consideration should also be given to the `_claimFromLido()` and `_claimFromSwell()` functions, where the withdrawal process could result in suboptimal outcomes. If the difference between `maxWithdrawAmount` and `minWithdrawAmount` is less than `minWithdrawAmount`, the second-to-last withdrawal request may result in receiving less than intended. It is important to ensure that `maxWithdrawAmount - minWithdrawAmount` is always greater than or equal to `minWithdrawAmount` to avoid this situation.

Consideration should also be given to the risk of price impact when making large deposits into the protocol, particularly during operations that involve LPing and swapping within the same pool. Large deposits can shift the price before liquidity is added, resulting in suboptimal outcomes for the protocol.

Fix Review: The client updated the strategy vault to allow dual deposits to minimize price impact.

Lastly, consider the lack of a mechanism to collect fees from knockout liquidity positions. Implementing such a mechanism could enhance the strategy's profitability and should be considered as a potential improvement.

Fix Review: The client updated the strategy vault to claim fees collected from knocked-out liquidity positions. Note that during a deposit, it is advantageous for the user to provide the Merkle proof for the claim process because the shares are computed before the fee claim. This was acknowledged by the client after discussions; this can be an incentive for a user to supply Merkle proofs for the claim process.

Fix Review: Following the fix review, the client refactored the codebase to use upgradeable contracts. The contracts can be upgraded at any time by the contract owner. For future update considerations, we recommend adding storage gaps in upgradeable abstract contracts to avoid storage collisions when introducing new variables. This pattern is used in OpenZeppelin's upgradeable contracts. Please note that this audit does not guarantee the behavior of future contract upgrades.

Key Actors And Their Capabilities

The strategy contracts use `AccessControl` to manage permissions, dividing responsibilities across two primary roles: `GOVERNANCE_ROLE` and `GUARDIAN_ROLE`.

Governance Role

The `GOVERNANCE_ROLE` is the highest-level role in the system and has admin control over both itself and the `GUARDIAN_ROLE`. This role can access all permissioned functions and set important parameters of the contract.

Due to the significant power associated with this role, it is advisable that the `GOVERNANCE_ROLE` be administered by a multisig wallet or a DAO to ensure proper checks and balances in decision-making and fund movement.

Guardian Role

The `GUARDIAN_ROLE` has a more operational role within the strategy and is primarily responsible for functions that are regularly used in the operation of the protocol. For example, it can call `rebalance()`, which updates liquidity ranges within each strategy. Guardians help manage the strategies by rebalancing funds and ensuring that the strategy operates according to its parameters.

Actor Responsibilities

- Users: Deposit funds into the strategy, aiming to earn interest or gains.
- Guardians: Manage deposited funds by rebalancing and optimizing strategy performance within the defined parameters.
- Strategies: Serve as the custodians of deposited funds, which are managed by guardians and interact with external, strategy-specific protocols. Governors: Control key variables of strategies and guardians and may arbitrarily interact with other 3rd party contracts via `execute()`.

Strategy-Specific Permissions and Functions

`BaseAmbientStrategy`

- `GOVERNANCE_ROLE`:
  - `renounceRole()`: Renounce the role, disabling future access to the functions listed below.
  - `grantRole()`: Grant the `GOVERNANCE_ROLE` or `GUARDIAN_ROLE` to any address.
  - `revokeRole()`: Revoke the `GOVERNANCE_ROLE` or `GUARDIAN_ROLE` from another address.
  - `setFees()`: Adjust the protocol fee (any value between 0% and 100%).
  - `setFeeRecipient()`: Set the address that receives protocol fees.
  - `setInvestedPercentage()`: Adjust the percentage of available funds to invest in the strategy.
  - `setSwapSlippage()`: Modify acceptable slippage for swaps.
  - `setLiqSlippage()`: Adjust slippage for liquidity provisioning.
  - `setOracle()`: Update the address for the asset oracle.
  - `setPadding()`: Arbitrarily change liquidity padding.
- `GUARDIAN_ROLE`:
  - `rebalance()`: Reinvest and adjust the strategy's parameters.
  - `investDust()`: Invest residual liquidity into the strategy.
  - `toggleDepositPause()`: Pause deposit.
  - `toggleWithdrawPause()`: Pause withdraw.

`SymetricAmbientStrategy`

- Inherits the same permissions and functions as `BaseAmbientStrategy`, with additional `GUARDIAN_ROLE` functions:
  - `updateBaseAndLimit()`: Adjust the strategy's base and limit parameters and perform a rebalance.

`RswEthStrategy`

- `GOVERNANCE_ROLE` :
  - `renounceRole()` : Renounce the role, disabling future access to the functions listed below.
  - `grantRole()` : Grant the `GOVERNANCE_ROLE` or `GUARDIAN_ROLE` to any address.
  - `revokeRole()` : Revoke the `GOVERNANCE_ROLE` or `GUARDIAN_ROLE` from another address.
  - `execute()` : Execute any function on an arbitrary external contract.
  - `setFees()` : Adjust the protocol fee (any value between 0% and 100%).
  - `setFeeRecipient()` : Set the address that receives protocol fees.
  - `setInvestedPercentage()` : Adjust the percentage of available funds to invest in the strategy.
  - `setOracle()` : Update the asset oracle address.
  - `setPadding()` : Modify liquidity padding arbitrarily.
- `GUARDIAN_ROLE` :
  - `rebalance()` : Reinvest and adjust the strategy's parameters.
  - `claimQueued()` : Claim queued exit requests.
  - `claimFromSwell()` : Create withdrawal requests from the Swell protocol.
  - `investBalance()` : Invest residual liquidity into the strategy.
  - `toggleDepositPause` : Pause deposit.
  - `toggleWithdrawPause` : Pause withdraw.

`WstEthStrategy`

- Inherits permissions and functions from `RswEthStrategy` , but substitutes:
  - `claimFromLido()` in place of `claimFromSwell()` to handle withdrawals from Lido.

Findings

TEM-1 Incorrect Liquidity Provision when Investing Knockout Positions

● Medium ⓘ Fixed

Update

The client fixed the issue in `874df55a21ec46745eeb35699fa50459bfc25ce4` by correcting the condition when providing knockout liquidity. The client added validation to ensure that the asset `assetIdx` used in the LST strategy is zero.

File(s) affected: `WstETHStrategy.sol` , `RswEthStrategy.sol`

Description: The `_provideLiquidity()` function is intended to provide liquidity when the price is outside the defined tick range (i.e., above the upper tick for buying or below the lower tick for selling) when investing in knockout positions. The `_rebaseWeights()` function sums the liquidity provision weights when the current tick is outside the defined tick range. However, the condition in `_provideLiquidity()` used to determine whether to add knockout liquidity is misaligned with the logic in `_rebaseWeights()` .

In `_rebaseWeights()` , the following condition is used:

```
if ((isBid && _currentTick > _lstParams[i].upperTick) || (!isBid && _currentTick <
_lstParams[i].lowerTick))
```

This condition checks whether the price is outside the tick range, which is the intended behavior for providing knockout liquidity. However, in `_provideLiquidity()` , we have:

```
if (!(_isBid && _currentTick > lstParam.upperTick) || (!_isBid && _currentTick < lstParam.lowerTick))
```

This logic is not correctly structured to achieve the opposite of the `_rebaseWeights()` logic, resulting in incorrect liquidity provision behavior. If the LP provision is an ask, knockout liquidity will be provided when the current price is greater than the lower tick. This will cause the `userCmd()` to revert because the placement of knockout liquidity is incorrect relative to the current price.

Recommendation: Refactor the conditional logic in `_provideLiquidity()` to align with the logic in `_rebaseWeights()` , ensuring that liquidity is correctly provided only when the price is outside the tick range.

To correctly represent the inverse logic, the condition in `_provideLiquidity()` should be refactored to:

```
if (!(_isBid && _currentTick > lstParam.upperTick) && !(!_isBid && _currentTick < lstParam.lowerTick))
```

This change ensures that liquidity is only provided when the price is outside the tick range, accurately reflecting the intended behavior and preventing any liquidity from being incorrectly allocated within the range.

Additionally, add tests for scenarios where `assetIdx` is not 0.

TEM-2 Improper Use of Oracle May Lead to Incorrect Outcomes

● Medium ⓘ Fixed

Update

The client refactored the `UsdcETHOracle` into a new contract, `BridgeOracle`, which is now used by both `BaseAmbientStrategy` and `SymetricAmbientStrategy`. The `BridgeOracle` utilizes `latestRoundData()` and includes checks to ensure the quality of the response data in `6ea5aa14dc269642654260c7b95f42f7bc8f1e34`. Note that the strategy implementation will use the deprecated function without staleness checks if the strategy oracle was set directly to third-party oracles.

File(s) affected: `BaseAmbientStrategy.sol`, `SymetricAmbientStrategy.sol`, `RswEthStrategy.sol`, `WstEthStrategy.sol`

Description: All four strategy contracts use the deprecated `latestAnswer()` function to fetch price data from oracles. While `latestAnswer()` is overridden in `RswEthStrategy` and `WstEthStrategy` to directly access price functions (`rswETHToETHRate()` and `stEthPerToken()`), this is not the case for `BaseAmbientStrategy` and `SymetricAmbientStrategy`, which still rely on `latestAnswer()`.

Additionally, there are several unsafe practices in the protocol when handling oracle data:

1. Unsafe Casting: At several locations in the protocol, the operation `uint256(IOracle(oracle).latestAnswer())` is performed, which casts the return value of `latestAnswer()` (an `int256`) to `uint256`. This cast can lead to overflows if `latestAnswer()` returns a negative value, potentially causing incorrect return values and unexpected behavior.
2. Lack of Zero Value Checks: The code does not check for the return value of `0`, which may indicate an error in fetching data from the oracle.
3. No Freshness Checks: In all four strategies, the protocol does not check if the price data returned from the oracle is fresh, which may lead to the use of stale or outdated price information.

Recommendation:

1. Replace `latestAnswer()` with `latestRoundData()`: Use `latestRoundData()` to retrieve more robust and detailed price information, including round IDs and update times, ensuring more accurate and reliable data.
2. Implement Freshness and Value Checks: Introduce checks for price freshness and correctness when accessing oracle data:

```
(
    uint80 roundId,
    int256 answer,
    uint256, startedAt,
    uint256 updateTime,
    uint80 answeredInRound
) = oracle.latestRoundData();
if (answer == 0) revert();
if (block.timestamp - updateTime > MAX_TIME_DIFF) revert();
```

3. Avoid Unsafe Casting: Ensure safe casting from `int256` to `uint256` by first validating that the returned value is non-negative.

TEM-3 Miscalculation of Profit and Fees when Depositing

• Low ⓘ Fixed

Update

The client fixed the issue in `463218be9476d7916b7c9fbe7c6ec07f2aaffa5e` following the recommendation.

File(s) affected: `WstETHStrategy.sol`, `RswEthStrategy.sol`

Description: The `_updateStratPnLAndFees()` function is called after the `invested` amount is updated but before the deposit amount is transferred when the underlying `assetIdx` is not a native currency. This sequence can result in incorrect profit (`stratPnL`) and fee (`feeToClaim`) calculations for ERC20 deposits, as the deposited amount is only included in `totalAssets()` after the transfer. This timing discrepancy can artificially lower the reported profit, leading to lower fees being claimed than should be accurately calculated.

We note however that the current implementation is due to the assumption that the deposited currency will always be ETH for the affected contracts. The risk of this issue primarily applies to related or derived contracts that deviate from this assumption.

Recommendation: Call `_updateStratPnLAndFees()` after the deposit amount is included in `totalAssets()` to ensure that profit and fees are calculated correctly, reflecting the actual state of assets.

TEM-4 Rounding Direction Can Lead to Loss

• Low ⓘ Fixed

Update

The client fixed the issue in `ef5b7a58ac7f5edc4b84b2f2cdb1bfe1f249abde` by rounding away from zero when calculating the shares to burn.

File(s) affected: `BaseAmbientStrategy.sol`, `SymetricAmbientStrategy.sol`, `WstETHStrategy.sol`, `RswEthStrategy.sol`

Description: The `mulDiv` function is used within `_convertToShares` to calculate `floor(x * y / denominator)` with full precision. This rounding behavior affects several key functions:

- `previewDeposit()`: This function rounds down the number of shares a user receives when depositing assets, favoring the protocol.
- `convertToShares()`: This function rounds down the conversion of amounts to shares, favoring the protocol.
- `previewWithdraw()`: This function rounds down the number of shares required to withdraw assets, which works against the protocol. Users may end up burning fewer shares than necessary to redeem the full amount of assets, allowing potential precision attacks where users can withdraw slightly more than intended over time, leading to gradual losses for the protocol.

Recommendation: Modify the `previewWithdraw()` function to round up instead of rounding down when calculating the number of shares required for withdrawal. This change would ensure that users burn the correct number of shares to redeem their assets and prevent potential precision-related concerns.

TEM-5 First Depositor Vulnerable to Inflation Attack

• Low ⓘ Mitigated

Update

The client mitigates the issue in commit `55c0e8b65c3b6013a71642a9be2afc05436e95af` by implementing the minting of dead shares during the first deposit. This solution mitigates the risk of an inflation attack by reducing the profit margin for potential attackers. However, it introduces a downside: the first depositor must forfeit a small portion of their deposit to cover the dead shares. Additionally, any profits earned by these dead shares are effectively lost, as they are sent to an inaccessible address.

Note that, although the code documentation indicates that the burnt shares are sent to `address (0)`, the burnt shares are sent to the vault.

File(s) affected: `BaseAmbientStrategy.sol`, `SymetricAmbientStrategy.sol`, `WstETHStrategy.sol`, `RswEthStrategy.sol`

Description: The share calculation for deposits into the strategy vault is vulnerable to manipulation through front-running and sandwich attacks, particularly targeting the first depositor. The formula used for share calculation (`amount.mulDiv(supply, _totalAsset)`) can be exploited in an inflation attack. Attackers can manipulate the rounding direction of shares to their advantage by increasing the total assets in the vault by front-running the first depositor and donating assets to the strategy. This manipulation results in the depositor receiving an unfairly low number of shares or, in extreme cases, no shares at all for their deposit.

Recommendation: To mitigate this attack, implement a slippage protection mechanism that allows users to specify a minimum acceptable number of shares for their deposits. This ensures that if the calculated shares fall below a user-defined threshold, the transaction reverts, protecting depositors from receiving too few shares.

Alternatively, consider other inflation attack mitigation techniques, such as tracking total assets internally or creating dead shares to prevent share inflation exploits.

TEM-6 Rebalancing May Be Blocked by Excess Token Balance

• Low ⓘ Fixed

Update

The client fixed the issue in `463218be9476d7916b7c9fbe7c6ec07f2aaffa5e` following the recommendation.

File(s) affected: `RswEthStrategy.sol`

Description: In `_claimFromSwell()`, when creating withdrawal requests towards the end of the function, a loop is made through `numberOfSplit`. However, this number may be greater than the length of the array `amounts`. Should this be the case, executing the loop may result in an index-out-of-bounds error.

Since `_claimFromSwell()` is called directly by `rebalance()`, it may happen that if the corresponding token balance of the contract is too high, this would block all attempts at rebalancing. The remedy in such a case would be to transfer out tokens from the contract via `execute()` which would require the use of `GOVERNANCE_ROLE` or manually claim through `claimFromSwell()` through `GUARDIAN`.

Recommendation: Ensure the length of the loop is `actualNumberOfSplit`.

TEM-7 Incorrect Emission of Withdrawal Amount in `_claimFromLido()`

• Low ⓘ Fixed

Update

The client fixed the issue in `463218be9476d7916b7c9fbe7c6ec07f2aaffa5e` following the recommendation.

File(s) affected: `WstETHStrategy.sol`

Description: In the `_claimFromLido()` function, the `RequestWithdrawal` event emits the total `amountUnwrappedLST` for each request ID, rather than the specific amount associated with each individual request. This can lead to confusion when tracking the specific amounts tied to each withdrawal request.

Recommendation: Modify the `RequestWithdrawal` event emission to log `amounts[i]` instead of the total `amountUnwrappedLST` for each `requestId`. This change will ensure that the event accurately reflects the amount associated with each individual request.

TEM-8 Redundant Ambient Finance LP Burn User Command

● Informational Fixed

Update
The client removed the redundant command in `b3203a6f2ee9e882269da3075cb71f4e022f48bf` when collecting fees from positions.

File(s) affected: `BaseAmbientStrategy.sol`, `SymetricAmbientStrategy.sol`

Description: When interacting with Ambient Finance, harvest user command type `5` is designed and gas-optimized for harvesting accumulated fees from concentrated liquidity positions. The burn user command (`2`) within the `_collectFees()` function is redundant because the fee harvesting command (`5`) already includes the burn operation at zero liquidity.

Recommendation: Remove the redundant burn command (`2`) from the `_collectFees()` function.

TEM-9 Missing Input Validation

● Informational Mitigated

Update
Marked as "Fixed" by the client.
Addressed in: `17c7bafd868219613ada7680332602c28895b2db`. However, we deem the issue mitigated due to the remaining unresolved sub-issues.

File(s) affected: `BaseAmbientStrategy.sol`, `SymetricAmbientStrategy.sol`, `WstEthStrategy.sol`, `RswEthStrategy.sol`, `libraries/VaultLibrary.sol`,

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. The following functions do not have a proper validation of input parameters:

1. `BaseAmbientStrategy`:
   1. Fixed `constructor()`: `sParams.assetIdx` should be either `0` or `1`.
   2. `constructor()`: `crocsQuery` and `crocsSwapDex` should not be zero addresses.
   3. Fixed `constructor()`: the `tokenAddresses[0]` should not be the same as `tokenAddresses[1]`.
2. `SymetricAmbientStrategy`:
   1. `constructor()`: `crocsQuery` and `crocsSwapDex` should not be zero addresses.
   2. Fixed `constructor()`: the `tokenAddresses[0]` should not be the same as `tokenAddresses[1]`.
3. `WstEthStrategy`:
   1. `constructor()`: `crocsQuery` and `crocsSwapDex` should not be zero addresses.
   2. Fixed `constructor()`: the `tokenAddresses[0]` should not be the same as `tokenAddresses[1]`.
   3. Fixed `claimFromLido()`: the `amount` should be greater than `MIN_STETH_WITHDRAWAL_AMOUNT`.
   4. Fixed `_investBalance()`: the `investedAmount` should be greater than zero before proceeding with liquidity provision.
   5. Fixed `_withdraw()`: the computed `shares` should be greater than zero.
   6. Fixed `deposit()`: the `amount` should be greater than zero before proceeding with liquidity provision.
4. `RswEthStrategy`:
   1. `constructor()`: `crocsQuery` and `crocsSwapDex` are not checked for zero addresses.
   2. Fixed `constructor()`: the `tokenAddresses[0]` should not be the same as `tokenAddresses[1]`.
   3. Fixed `_investBalance()`: the `investedAmount` should be greater than zero before proceeding with liquidity provision.
   4. Fixed `_withdraw()`: the computed `shares` should be greater than zero.
   5. Fixed `deposit()`: the `amount` should be greater than zero before proceeding with liquidity provision.
5. `VaultLibrary.sol`:
   1. Fixed `_checkTicks()`: the `_upperTick` and `_lowerTick` of the `_checkTicks()` should not be equal.
   2. Fixed `checkLiqParams()`: when validating the liquidity ranges, there should not be any duplicate ranges (`_lowerTick`, `_upperTick`).

Recommendation: Consider adding according input checks.

TEM-10 Loop / Gas Concerns

● Informational Acknowledged

Update
The client acknowledged the issue and provided the following explanation:

```
For now the number of tick ranges is managed by our Governance account.
  We will keep it in mind for our operation.
```

File(s) affected: `BaseAmbientStrategy.sol`, `SymetricAmbientStrategy.sol`, `RswEthStrategy.sol`, `WstEthStrategy.sol`

Description: Several protocol-critical functions such as collecting fees, which are executing when depositing, withdrawing or redeeming, iterate over the number of tick ranges. However, the number of tick ranges is not constrained, allowing them to cause out-of-gas issues when trying to process them.

Recommendation: We recommend constraining the number of tick ranges (i.e. in the constructor and in `_rebalance()`) to prevent operations from running out-of-gas and thereby denying service to the protocol.

TEM-11
Inconsistent Liquidity Allocation Due to Dynamic Recalculation of LP Weights

● Informational ⓘ Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

```
We acknowledge that, but fixing this issue might lead to a big modification to all of those functions.
we decide not to fix it at this point, consider this is informational
```

File(s) affected: `WstETHStrategy.sol`, `RswEthStrategy.sol`

Description: The protocol ensures that the knockout LP weights add up to `BASE` (representing 100%) during setup, establishing a specific distribution for liquidity allocation (e.g., 15%, 75%, 10%, 5%). However, during actual liquidity provision, the strategy dynamically recalculates the total LP weights based on the current tick and whether certain tick ranges are out of range. This dynamic recalculation can lead to significant deviations from the original allocation.

For example, if only the positions corresponding to 15%, 10%, and 5% are out of range, the recalculated weights could shift to 50%, 33%, and 17%, respectively. As a result, the actual liquidity allocation may differ substantially from the intended distribution. This misalignment between the initial weight validation and the dynamic recalculation during liquidity provision can lead to an imbalanced portfolio, where certain positions receive more or less liquidity than initially planned.

Recommendation: Clarify whether this behavior is intended. If this is not the intended behavior, review and refine the dynamic recalculation logic used during liquidity provision to ensure it closely aligns with the intended portfolio allocation strategy.

TEM-12
Anyone Can Re-Invest `investedPercentage` Knockout Liquidity

● Undetermined ⓘ Fixed

Update

The client updated the deposit action in `34b892184f993030108314217f46356b411ca3fc`. During a user's deposit, only the `investedPercentage` of the deposit `amount` is invested.

File(s) affected: `WstETHStrategy.sol`, `RswEthStrategy.sol`

Description: When depositing into the strategy, the contract is designed to invest a specified `investedPercentage` of the current strategy balance into knockout positions. Although there is a separate function for the `GUARDIAN_ROLE` to invest the current contract balance by governance, users can achieve the same result by depositing a small amount to trigger the investment of the `investedPercentage` of the entire contract balance. For instance, it may be desired for some extra liquidity to remain idle after `claimQueued()`, whereas users may prevent this through their calls of `deposit()`.

Recommendation: Review the `investedPercentage` mechanism to prevent users from forcing the protocol into unintended liquidity provision scenarios. Consider whether users should be allowed to invest contract deposits beyond their own deposit amounts.

Auditor Suggestions

S1 Using `transfer()` for ETH May Fail in The Future

Fixed

Update
The client fixed in `7ccb1ab1a8359fba56b21b3839742eae9902502f` by changing the usages of `transfer()` to `call()`.

File(s) affected: `BaseAmbientStrategy.sol`, `SymetricAmbientStrategy.sol`, `RswEthStrategy.sol`, `WstEthStrategy.sol`

Description: The functions `address.transfer()` and `address.send()` assume a fixed amount of gas to execute the call. The use of these functions protects against reentrancy attacks. The default amount of gas, however, may change in the future. In the worst case, it could lead to failed transfers.

Recommendation: We want you to be aware of this possibility. Whereas we do not recommend taking any action now, if you need more flexibility regarding the forwarded gas, you can use `call()` to perform transfers.

S2 Unlocked Pragma

Acknowledged

Update
The client has provided `3007d3d85a8037b9ce31fe6fd4e52ccc6491c62d`. However, the pragma remains unlocked with the current setting of `>= 0.8.23`.

File(s) affected: `BaseAmbientStrategy.sol`, `RswEthStrategy.sol`, `libraries/VaultLibrary.sol`, `libraries/LiquidityAmountsNative.sol`, `libraries/LiquidityAmounts.sol`, `libraries/FixedPoint.sol`, `libraries/TickMath.sol`, `libraries/SafeCast.sol`, `libraries/PoolSpecs.sol`, `WstEthStrategy.sol`, `utils/UsdcETHOracle.Sol`, `utils/Errors.sol`, `utils/Variables.Sol`, `utils/WstETHOracle.sol`, `utils/Constants.sol`, `utils/RswETHOracle.sol`, `SymetricAmbientStrategy.sol`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

S3 Remove Reliance on Protocol Actions for Fee Collection

Fixed

Update
The client added fee collection when updating fees in `fd9e385e6f9975a595ab9c2d74125d0243ffa09e`. However, note that if the protocol operator only intends to collect fees, they need to update the fee by setting it to the same value again.

File(s) affected: `WstETHStrategy.sol`, `RswEthStrategy.sol`, `BaseAmbientStrategy.sol`, `SymetricAmbientStrategy.sol`

Description: The protocol depends on the `rebalance()` function to claim protocol fees, as fees are only claimed during the rebalance process. However, if all liquidity positions are knocked out and there is no idle balance available, the protocol may not be able to claim fees. This reliance on rebalancing could limit the ability to collect fees efficiently when there are no rebalancing opportunities.

Similarly for `BaseAmbientStrategy` and `SymetricAmbientStrategy`, the protocol requires a protocol action (deposit, withdraw, redeem, or rebalance) to withdraw strategy management fees.

Recommendation: For LRT strategy vaults, consider adding a separate function that allows the `GUARDIAN_ROLE` to claim protocol fees independently of the `rebalance()` process. This would provide greater flexibility in managing fee collection and ensure the protocol can claim fees even when a rebalance is not required.

For other strategy vaults, consider adding a separate function for fee collections separate from protocol actions.

S4 Missing Slippage Setting

Fixed

Update
The client set the liquidity slippage during contract deployment in `e4f646b95035fbb7f1cd47018ccfe7eef2bd7204`.

File(s) affected: `BaseAmbientStrategy.sol`

Description: When setting liquidity in `BaseAmbientStrategy.constructor`, the `sParams.liqSlippage` is defined and validated, the state variable `liqSlippage` is not set during the contract deployment.

Recommendation: Set `liqSlippage` in `BaseAmbientStrategy.constructor()`.

S5 Unlicensed Contracts

Fixed

Update
The client added licenses in `f1462aeea5b972b8bfface092b1b674712396d42`.

File(s) affected: `BaseAmbientStrategy.sol`, `RswEthStrategy.sol`, `WstEthStrategy.sol`, `utils/UsdcETHOracle.Sol`, `utils/Errors.sol`, `utils/Variables.Sol`, `utils/WstETHOracle.sol`, `utils/Constants.sol`, `utils/RswETHOracle.sol`, `SymetricAmbientStrategy.sol`

Description: A license, as marked at the top of a contract after `// SPDX-License-Identifier:`, may help control who may copy and modify these contracts and to what extend. A missing license, i.e. as marked via `UNLICENSED`, forfeits any such control and allows for arbitrary modifications and could even be re-licensed with different terms, potentially preventing the original author to perform any changes.

Several contracts from this protocol are currently marked as `UNLICENSED` and could therefore be subject to above mentioned changes.

Recommendation: We recommend adding a license to the mentioned files before publishing them.

S6 Use Immutable for Unchanging Variables

Acknowledged

Update
The client acknowledged the issue and provided the following explanation:

```
We are using upgradable contract so immutable is not usable.
```

File(s) affected: `BaseAmbientStrategy.sol`, `RswEthStrategy.sol`, `SymetricAmbientStrategy.sol`, `WstEthStrategy.sol`

Description: Several variables within the strategy contracts are initialized once and never changed afterward. These variables, currently defined as state variables, could be marked as `immutable` to save gas.

The following variables are good candidates for being made immutable:
1. `BaseAmbientStrategy.cmdId`
2. `RswEthStrategy.unWrappedToken`
3. `RswEthStrategy.withdrawalQueue`
4. `SymetricAmbientStrategy.cmdId`
5. `WstEthStrategy.unWrappedToken`
6. `WstEthStrategy.withdrawalQueue`

Recommendation: Consider declaring these variables as `immutable` to reduce gas costs.

S7 Major Code Overlap Between Contracts May Lead to Missed Fixes

Fixed

Update
The client fixed the issue in `5d023ab3f64ffc69308200436c67d874c90a0c4e`, `513a5867401361418a3b64561f74d269e4d07a7f` and `17c7bafd868219613ada7680332602c28895b2db` by refactoring the codebase.

File(s) affected: `BaseAmbientStrategy.sol`, `SymetricAmbientStrategy.sol`, `RswEthStrategy.sol`, `WstEthStrategy.sol`

Description: Contracts `BaseAmbientStrategy` and `SymetricAmbientStrategy`, as well as `RswEthStrategy` and `WstEthStrategy` share a majority of the code nearly 1:1. This leads to increased maintenance as well as the potential to introduce bugs or to miss bug fixes in all relevant places when performing changes in the future.

Recommendation: We recommend outlining common code into shared base contracts to reduces maintenance and mitigate aforementioned risks when performing changes in the future.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.

- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Files

- `0f0...2f1 ./src/BaseAmbientStrategy.sol`
- `af0...7a6 ./src/RswEthStrategy.sol`
- `46b...852 ./src/WstEthStrategy.sol`
- `308...8da ./src/SymetricAmbientStrategy.sol`
- `0a4...fd7 ./src/interfaces/IrswETH.sol`
- `a82...2c8 ./src/interfaces/CrocsSwapDex.sol`
- `a61...d85 ./src/interfaces/IWithdrawalQueueERC721.sol`
- `1e4...220 ./src/interfaces/IrswEXIT.sol`
- `2d3...2ed ./src/interfaces/IWstETH.sol`
- `127...2d1 ./src/interfaces/IOracle.sol`
- `6fc...a3a ./src/interfaces/CrocsQuery.sol`
- `a8f...5bb ./src/lst/LstAmbientLiquidity.sol`
- `c4c...df5 ./src/lst/LstAmbientStorageLayout.sol`
- `c53...7d1 ./src/lst/LstAmbientOperator.sol`
- `187...4c8 ./src/lst/LstAmbientInterface.sol`
- `faa...fb0 ./src/lst/LstAmbientCommon.sol`
- `b0f...6c7 ./src/lst/LstAmbientUser.sol`
- `8a3...e0f ./src/lst/wsteth/WstEthAmbientOperator.sol`
- `3be...3dd ./src/lst/wsteth/WstEthAmbientCommon.sol`
- `339...340 ./src/lst/wsteth/WstEthAmbientUser.sol`
- `3d2...d09 ./src/lst/wsteth/WstEthAmbientStorageLayout.sol`
- `e7b...63f ./src/lst/rsweth/RswEthAmbientOperator.sol`
- `4c6...690 ./src/lst/rsweth/RswEthAmbientUser.sol`
- `059...3ce ./src/lst/rsweth/RswEthAmbientStorageLayout.sol`
- `dc6...a41 ./src/lst/rsweth/RswEthAmbientCommon.sol`
- `da3...b60 ./src/utils/BridgeOracle.sol`
- `032...7a8 ./src/utils/Errors.sol`
- `114...0c9 ./src/utils/Variables.sol`
- `bdc...f80 ./src/utils/DepositWithdrawPausable.sol`
- `ea2...6d2 ./src/utils/WstETHOracle.sol`
- `526...ede ./src/utils/Constants.sol`
- `e6f...713 ./src/utils/RswETHOracle.sol`
- `5af...eb1 ./src/ambient-lp/SymetricAmbientStorageLayout.sol`
- `7de...909 ./src/ambient-lp/AmbientUser.sol`
- `ef6...3df ./src/ambient-lp/AmbientInterface.sol`

- `0a2...f51 ./src/ambient-lp/AmbientLiquidity.sol`
- `091...728 ./src/ambient-lp/SymetricAmbientOperator.sol`
- `040...124 ./src/ambient-lp/AmbientStorageLayout.sol`
- `a87...dc5 ./src/ambient-lp/BaseAmbientOperator.sol`
- `28a...a3b ./src/ambient-lp/AmbientCommon.sol`
- `7d0...2a4 ./src/ambient-lp/AmbientOperator.sol`
- `40b...2e5 ./src/libraries/VaultLibrary.sol`
- `ba9...416 ./src/libraries/LiquidityMath.sol`
- `4e7...290 ./src/libraries/LiquidityAmountsNative.sol`
- `fe3...cf3 ./src/libraries/FixedPoint.sol`
- `f97...0c4 ./src/libraries/TickMath.sol`
- `112...058 ./src/libraries/CompoundMath.sol`
- `9dc...43b ./src/libraries/CurveMath.sol`
- `5ef...683 ./src/libraries/SafeCast.sol`
- `1b6...7fd ./src/libraries/PoolSpecs.sol`

Tests

- `baf...ee8 ./test/DepositSymTestToken1.t.sol`
- `3ba...afc ./test/DepositSymTestToken0.t.sol`
- `960...81e ./test/RswETHTestToken.t.sol`
- `07f...60d ./test/DepositTestToken1.t.sol`
- `f4f...acf ./test/WstETHClaimTestToken.t.sol`
- `359...6ca ./test/FeeCollection.t.sol`
- `92d...d14 ./test/WstETHTestToken.t.sol`
- `0ef...1a6 ./test/RswETHClaimTestToken.t.sol`
- `e98...c96 ./test/DepositTestToken0.t.sol`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:
- Slither v0.10.0

Steps taken to run the tools:
1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

We have executed Slither and filtered the issues that were reported and incorporated the valid ones in the report. Please note that only issues related to the scope of the audit are reported.

Test Suite Results

The tests were generated using `forge test`.

```
Ran 1 test for test/mock/TestRswEthAmbientOperator.sol:TestRswEthAmbientOperator
[PASS] test() (gas: 276)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 5.93ms (545.38µs CPU time)

Ran 1 test for test/mock/TestWstEthAmbientOperator.sol:TestWstEthAmbientOperator
[PASS] test() (gas: 276)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 5.69ms (572.63µs CPU time)
```

```
Ran 1 test for test/mock/TestRswEthStrategy.sol:TestRswEthStrategy
[PASS] test() (gas: 276)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 5.44ms (580.13µs CPU time)

Ran 1 test for test/mock/TestWstEthStrategy.sol:TestWstEthStrategy
[PASS] test() (gas: 276)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 421.67µs (50.83µs CPU time)

Ran 1 test for test/RswETHClaimTestToken.t.sol:RswClaimTestToken
[PASS] test_ClaimCmd() (gas: 90874)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 954.29ms (3.27ms CPU time)

Ran 1 test for test/WstETHClaimTestToken.t.sol:LSTClaimTestToken
[PASS] test_ClaimCmd() (gas: 90874)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 957.36ms (3.29ms CPU time)

Ran 2 tests for test/FeeCollection.t.sol:FeeCollectionTest
[PASS] testFuzz_FeeCollection() (gas: 2414779)
[PASS] testFuzz_SetFee() (gas: 2221673)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 992.65ms (108.57ms CPU time)

Ran 5 tests for test/DepositTestToken1.t.sol:DepositTestToken1
[PASS] testFuzz_Deposit1(uint256) (runs: 1000, µ: 992679, ~: 1046626)
[PASS] testFuzz_DepositAndRebalance1(uint256) (runs: 1000, µ: 1324736, ~: 1368437)
[PASS] testFuzz_deposit1MultipeUser(uint256,uint256) (runs: 1000, µ: 1642908, ~: 1642908)
[PASS] testFuzz_depositAndWithdraw(uint256) (runs: 1000, µ: 1346064, ~: 1439731)
[PASS] testSimple1() (gas: 997196)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 64.74s (148.42s CPU time)

Ran 7 tests for test/DepositSymTestToken1.t.sol:DepositSymTestToken1
[PASS] testFuzz_Deposit1(uint256) (runs: 1000, µ: 845244, ~: 853931)
[PASS] testFuzz_DepositAndUpdateWidth(uint256) (runs: 1000, µ: 1296062, ~: 1322824)
[PASS] testFuzz_DepositSymAndRebalance1(uint256) (runs: 1000, µ: 1295084, ~: 1314549)
[PASS] testFuzz_FeeCollection() (gas: 1978236)
[PASS] testFuzz_SetFee() (gas: 1770128)
[PASS] testFuzz_deposit1MultipeUser(uint256,uint256) (runs: 1000, µ: 1453345, ~: 1453345)
[PASS] testFuzz_depositAndWithdraw(uint256) (runs: 1000, µ: 1068998, ~: 1084166)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 154.70s (141.29s CPU time)

Ran 4 tests for test/DepositTestToken0.t.sol:DepositTestToken0
[PASS] testFuzz_Deposit0(uint256) (runs: 1000, µ: 742072, ~: 809203)
[PASS] testFuzz_deposit0AndWithdraw(uint256) (runs: 1000, µ: 975799, ~: 1071317)
[PASS] testFuzz_depositMultipeUser(uint256,uint256) (runs: 1000, µ: 1313832, ~: 1366176)
[PASS] testSimple0() (gas: 797408)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 169.30s (87.09s CPU time)

Ran 9 tests for test/RswETHTestToken.t.sol:RswTestToken
[PASS] testFuzz_DepositAndRebalanceRsw(uint256,uint256[3]) (runs: 1000, µ: 1399286, ~: 1399697)
[PASS] testFuzz_DepositRsw(uint256) (runs: 1000, µ: 959748, ~: 959748)
[PASS] testFuzz_WithdrawAllFromSwell(uint256) (runs: 1000, µ: 4982757, ~: 4982757)
[PASS] testFuzz_WithdrawFromSwell(uint256) (runs: 1000, µ: 5683860, ~: 5683860)
[PASS] testFuzz_claimFromSwell(uint256) (runs: 1000, µ: 6436302, ~: 6436302)
[PASS] testFuzz_depositRswAndRedeem(uint256) (runs: 1000, µ: 1151648, ~: 1140804)
[PASS] testFuzz_depositRswAndWithdraw(uint256) (runs: 1000, µ: 1471537, ~: 1471549)
[PASS] testFuzz_depositRswInvestAndWithdraw(uint256) (runs: 1000, µ: 991371, ~: 983398)
[PASS] testFuzz_depositRswMultipeUser(uint256,uint256) (runs: 1000, µ: 1192345, ~: 1192470)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 169.30s (412.77s CPU time)

Ran 9 tests for test/DepositSymTestToken0.t.sol:DepositSymTestToken0
[PASS] testFuzz_DepositSym0(uint256) (runs: 1000, µ: 621869, ~: 630276)
[PASS] testFuzz_DepositSymAndUpdateWidth0(uint256) (runs: 1000, µ: 1269191, ~: 1269191)
[PASS] testFuzz_FeeCollection() (gas: 2032353)
[PASS] testFuzz_SetFee() (gas: 1825369)
[PASS] testFuzz_depositSym0AndWithdraw(uint256) (runs: 1000, µ: 857829, ~: 860730)
[PASS] testFuzz_depositSymMultipeUser(uint256,uint256) (runs: 1000, µ: 3307015, ~: 3343694)
[PASS] test_Deposits() (gas: 772576)
[PASS] test_Deposits2() (gas: 772552)
[PASS] test_DepositsAndWithdraw() (gas: 948643)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 169.30s (144.85s CPU time)

Ran 9 tests for test/WstETHTestToken.t.sol:WstETHTestToken
```

```
[PASS] testFuzz_DepositAndRebalanceLST(uint256,uint256[3]) (runs: 1000, μ: 1502617, ~: 1503047)
[PASS] testFuzz_DepositLST(uint256) (runs: 1000, μ: 1099825, ~: 1099825)
[PASS] testFuzz_WithdrawAllFromLido(uint256) (runs: 1000, μ: 2852538, ~: 2852538)
[PASS] testFuzz_WithdrawFromLido(uint256) (runs: 1000, μ: 3927514, ~: 3927526)
[PASS] testFuzz_claimFromLido(uint256) (runs: 1000, μ: 3300934, ~: 3300947)
[PASS] testFuzz_depositLSTAndRedeem(uint256) (runs: 1000, μ: 1265140, ~: 1259961)
[PASS] testFuzz_depositLSTAndWithdraw(uint256) (runs: 1000, μ: 1492102, ~: 1492115)
[PASS] testFuzz_depositLSTInvestAndWithdraw(uint256) (runs: 1000, μ: 1100932, ~: 1098815)
[PASS] testFuzz_depositLSTMultipeUser(uint256,uint256) (runs: 1000, μ: 1355546, ~: 1355672)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 169.28s (356.74s CPU time)

Ran 13 test suites in 169.98s (899.55s CPU time): 51 tests passed, 0 failed, 0 skipped (51 total tests)
```

Code Coverage

The code coverage was generated using forge coverage. We recommend adding more tests to improve overall test coverage. We recommend asserting more specific properties during testing to enhance the thoroughness of the test suite.

Fix Review: Following the code refactor and changes, the code coverage has increased with additional tests.

File	% Lines	% Statements	% Branches	% Funcs
script/Base.s.sol	0.00% (**0**/3)	0.00% (**0**/3)	100.00% (**0**/0)	0.00% (**0**/2)
script/DeployRswEthStrategy.s.sol	0.00% (**0**/30)	0.00% (**0**/42)	100.00% (**0**/0)	0.00% (**0**/1)
script/DeploySymStrategy.s.sol	0.00% (**0**/11)	0.00% (**0**/17)	100.00% (**0**/0)	0.00% (**0**/1)
script/UpgradeRswEthStrategy.s.sol	0.00% (**0**/6)	0.00% (**0**/9)	0.00% (**0**/2)	0.00% (**0**/1)
src/BaseAmbientStrategy.sol	92.31% (**12**/13)	93.33% (**14**/15)	100.00% (**0**/0)	66.67% (**2**/3)
src/RswEthStrategy.sol	84.62% (**11**/13)	83.33% (**15**/18)	100.00% (**0**/0)	60.00% (**3**/5)
src/SymetricAmbientStrategy.sol	90.91% (**20**/22)	93.33% (**28**/30)	50.00% (**1**/2)	75.00% (**3**/4)
src/WstEthStrategy.sol	83.33% (**10**/12)	82.35% (**14**/17)	100.00% (**0**/0)	25.00% (**1**/4)
src/ambient-**lp/**AmbientCommon.sol	100.00% (**30**/30)	100.00% (**50**/50)	100.00% (**0**/0)	100.00% (**6**/6)
src/ambient-**lp/**AmbientInterface.sol	60.87% (**70**/115)	65.61% (**103**/157)	9.09% (**2**/22)	54.84% (**17**/31)
src/ambient-**lp/**AmbientLiquidity.sol	95.56% (**43**/45)	96.43% (**54**/56)	75.00% (**12**/16)	100.00% (**6**/6)
src/ambient-**lp/**AmbientOperator.sol	93.75% (**15**/16)	95.24% (**20**/21)	50.00% (**3**/6)	100.00% (**4**/4)
src/ambient-**lp/**AmbientUser.sol	95.80% (**137**/143)	93.24% (**193**/207)	65.00% (**26**/40)	94.12% (**16**/17)
src/ambient-**lp/**BaseAmbientOperator.sol	100.00% (**15**/15)	95.24% (**20**/21)	50.00% (**1**/2)	100.00% (**2**/2)

File	% Lines	% Statements	% Branches	% Funcs
src/ambient-**lp/**SymetricAmbientOperator.sol	92.31% (**24/**26)	93.02% (**40/**43)	50.00% (**2/**4)	75.00% (**3/**4)
src/libraries/CompoundMath.sol	0.00% (**0/**43)	0.00% (**0/**78)	0.00% (**0/**12)	0.00% (**0/**7)
src/libraries/CurveMath.sol	36.59% (**15/**41)	36.49% (**27/**74)	30.00% (**3/**10)	50.00% (**6/**12)
src/libraries/FixedPoint.sol	83.33% (**5/**6)	84.62% (**11/**13)	50.00% (**1/**2)	75.00% (**3/**4)
src/libraries/LiquidityAmountsNative.sol	90.91% (**20/**22)	85.71% (**24/**28)	64.29% (**9/**14)	100.00% (**7/**7)
src/libraries/LiquidityMath.sol	2.86% (**1/**35)	3.51% (**2/**57)	0.00% (**0/**22)	6.67% (**1/**15)
src/libraries/PoolSpecs.sol	12.50% (**2/**16)	11.54% (**3/**26)	50.00% (**1/**2)	16.67% (**1/**6)
src/libraries/SafeCast.sol	36.36% (**4/**11)	33.33% (**5/**15)	21.43% (**3/**14)	42.86% (**3/**7)
src/libraries/TickMath.sol	100.00% (**91/**91)	98.56% (**137/**139)	89.13% (**41/**46)	100.00% (**2/**2)
src/libraries/VaultLibrary.sol	94.64% (**53/**56)	85.42% (**82/**96)	52.94% (**18/**34)	93.33% (**14/**15)
src/lst/LstAmbientCommon.sol	97.06% (**33/**34)	98.00% (**49/**50)	100.00% (**0/**0)	85.71% (**6/**7)
src/lst/LstAmbientInterface.sol	64.55% (**71/**110)	63.51% (**94/**148)	18.75% (**3/**16)	50.00% (**13/**26)
src/lst/LstAmbientLiquidity.sol	94.00% (**47/**50)	92.42% (**61/**66)	65.38% (**17/**26)	100.00% (**8/**8)
src/lst/LstAmbientOperator.sol	68.00% (**34/**50)	68.66% (**46/**67)	50.00% (**6/**12)	75.00% (**6/**8)
src/lst/LstAmbientUser.sol	96.43% (**54/**56)	95.06% (**77/**81)	83.33% (**15/**18)	88.89% (**8/**9)
src/lst/rsweth/RswEthAmbientCommon.sol	100.00% (**6/**6)	100.00% (**11/**11)	100.00% (**0/**0)	100.00% (**1/**1)
src/lst/rsweth/RswEthAmbientOperator.sol	98.04% (**50/**51)	95.83% (**69/**72)	58.33% (**7/**12)	80.00% (**4/**5)
src/lst/rsweth/RswEthAmbientUser.sol	0.00% (**0/**1)	0.00% (**0/**1)	100.00% (**0/**0)	0.00% (**0/**1)
src/lst/wsteth/WstEthAmbientCommon.sol	100.00% (**4/**4)	100.00% (**7/**7)	100.00% (**0/**0)	100.00% (**1/**1)
src/lst/wsteth/WstEthAmbientOperator.sol	97.96% (**48/**49)	94.81% (**73/**77)	64.29% (**9/**14)	83.33% (**5/**6)
src/lst/wsteth/WstEthAmbientUser.sol	0.00% (**0/**1)	0.00% (**0/**1)	100.00% (**0/**0)	0.00% (**0/**1)

File	% Lines	% Statements	% Branches	% Funcs
src/utils/BridgeOracle.sol	59.09% (**26/**44)	58.82% (**40/**68)	41.67% (**5/**12)	22.22% (**2/**9)
src/utils/DepositWithdrawPausable.sol	100.00% (**8/**8)	100.00% (**10/**10)	100.00% (**4/**4)	100.00% (**4/**4)
src/utils/RswETHOracle.sol	66.67% (**2/**3)	60.00% (**3/**5)	100.00% (**0/**0)	50.00% (**1/**2)
src/utils/WstETHOracle.sol	87.50% (**14/**16)	77.42% (**24/**31)	50.00% (**2/**4)	75.00% (**3/**4)
Total	74.77% (**975/**1304)	72.96% (**1406/**1927)	51.90% (**191/**368)	62.79% (**162/**258)

Changelog

- 2024-09-05 - Initial report
- 2024-10-03 - Final report
- 2024-10-07 - Final report update

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Disclaimer

Quantstamp