Tempest Finance Update Audit Report

Version 2.0

Audited by:

xiaoming9090

SpicyMeatball

peakbolt

October 8, 2024

Contents

1 Introduction

1.1 About Renascence

Renascence Labs was established by a team of experts including HollaDieWaldfee, MiloTruck, alexxander and bytes032.

Our founders have a distinguished history of achieving top honors in competitive audit contests, enhancing the security of leading protocols such as Reserve Protocol, Arbitrum, MaiaDAO, Chainlink, Dodo, Lens Protocol, Wenwin, PartyDAO, Lukso, Perennial Finance, Mute and Taurus.

We strive to deliver tailored solutions by thoroughly understanding each client's unique challenges and requirements. Our approach goes beyond addressing immediate security concerns; we are dedicated to fostering the enduring success and growth of our partners.

More of our work can be found here.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an 'as-is' and 'as-available' basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

1.3.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality
- Low - Funds are **not** at risk

1.3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

2 Executive Summary

2.1 About Tempest Finance Update

Tempest Finance is an innovative ALM built on top of Ambient Finance, designed to simplify liquidity provision through two main strategies: the Symmetric Vault and the Arbitrage Vault.

Symmetric Vaults Tempest's Symmetric Vaults optimize user returns while mitigating impermanent loss. These vaults utilize a base symmetric order combined with range limit orders to address portfolio imbalances caused by price movements, avoiding the costly swap fees typical in active rebalancing strategies. This approach offers two key benefits:

Avoidance of Swap Fees: By not using swaps for rebalancing, the vault avoids paying transaction fees.

Passive Rebalancing with Price Movements: The vault rebalances as prices fluctuate within a predefined range (Limit Order Range), earning fees while rebalancing. Although it still faces some permanent loss, the fee income helps offset these losses. The strategy benefits from the fact that prices often do not move in straight lines, effectively betting that the fee income from the price volatility outweighs any permanent loss experienced.

Arbitrage Recapture Vault The Arbitrage Recapture Vault is the first of its kind to internalize MEV (Maximal Extractable Value) created when LSTs (Liquid Staking Tokens) depeg, which is typically the most volatile period for LST/LRT liquidity. It leverages knockout orders, which capture the delta created as LSTs deviate from their peg. This effectively forms a flexible buy wall that tracks the peg, capturing arbitrage opportunities that would otherwise be captured by external actors.

The Tempest Advantage Tempest Finance is redefining LST/LRT liquidity by recapturing profits for LPs and empowering Liquid Staking Protocols to offer new liquidity solutions that were previously only achievable by active arbitrageurs. All of this is achieved while maintaining the same swap functionality that traditional concentrated liquidity provides.

Tempest Finance is making LST/LRT liquidity great again, ensuring that liquidity providers can maximize their returns in a sustainable and efficient manner.

2.2 Overview

Project	Tempest Finance Update
Repository	tempest$_smart_contract$
Commit Hash	bba249659e82...
Mitigation Hash	3767bb08e1a1...
Date	29 September 2024 - 03 October 2024

2.3 Issues Found

Severity	**Count**
High Risk	0

Medium Risk	1
Low Risk	3
Informational	3
Total Issues	**7**

3 Findings Summary

ID	Description	Status
M-1	Idle unwrapped `stETH` in `WstEthStrategy` is not handled properly	Resolved
L-1	The spending allowance set in `_claimFromSwell()` might be higher than required	Resolved
L-2	Incorrect ‘`ethVal` used for providing liquidity in `deposits()`	Resolved
L-3	Temporary block first deposit in LST strategies	Resolved
I-1	Add storage gaps to reserve space for future state variables	Resolved
I-2	Using `Expand` instead `Ceil`	Resolved
I-3	`withdraw()`’s return values refactoring	Resolved

4 Findings

Medium Risk

[M-1] Idle unwrapped `stETH` in `WstEthStrategy` is not handled properly

Context: WstEthAmbientOperator.sol#L29-L33

Description: During a large rebalance/withdrawal from `WstEthStrategy`, there could be idle stETH that were unwrapped but not withdrawn from Lido due to the `MAX_NUMBER_OF_WITHDRAWAL_SPLIT`, which will limits the claim amount.

Due to that reason, `unstake()` is added to allow operator to explicitly withdraw idle stETH from Lido.

However, the `amount` is applied inconsistently, as it used to represent the quantity of both wstETH and stETH, in `getStETHByWstETH()` and `_claimFromLido()` respectively.

```
function unstake(uint256 amount) external returns (uint256[] memory amounts,
uint256[] memory requestIds) {
  WLstETH wlstETH = WLstETH(tokenAddresses[1 - assetIdx]);
  if (wlstETH.getStETHByWstETH(amount) <
  withdrawalQueue.MIN_STETH_WITHDRAWAL_AMOUNT()) revert(SMALL_AMOUNT);
  (amounts, requestIds) = _claimFromLido(amount);
}
```

Another issue is that the idle stETH are not accounted in `_totalAssets()`, causing a shortfall in assets value when this scenario occurs. That will cause vault users to incur a loss when redeeming their shares.

Recommendation: This issue can be fixed by using the stETH amount for the `wlstETH.getStETHByWstETH(amount)` check, and accounting for the idle stETH value to `_totalAssets()`.

Client: Fixed in `https://github.com/Tempest-Finance/tempest_smart_contract/pull/150/files`

Renascence: Resolved

Low Risk

[L-1] The spending allowance set in `_claimFromSwell()` might be higher than required

Context: RswEthAmbientOperator.sol#L50

Description: `_claimFromSwell()` will set the spending allowance for `withdrawalQueue` as `amount` to allow Swell to transfer out the rswETH for withdrawal.

However, the actual withdraw amount could be lower than `amount` due to `MAX_NUMBER_OF_WITHDRAWAL_SPLIT`. That means the allowance should be set to the `_totalWithdrawAmount` after performing the splits. Similiar to what was done for ‘wstEthAmbientOperator (see link).

```
function _claimFromSwell(uint256 amount) internal returns (uint256[] memory amounts,
uint256[] memory requestIds) {
  IOracleAdapter _oracle = IOracleAdapter(oracle);

  uint256 maxWithdrawAmount = withdrawalQueue.withdrawRequestMaximum();
  uint256 minWithdrawAmount = withdrawalQueue.withdrawRequestMinimum();
  if (amount < minWithdrawAmount) {
    amounts = new uint256[](0);
    requestIds = new uint256[](0);
    return (amounts, requestIds);
  }

  //@audit tthe allowance required could be lower than amount due to
  MAX_NUMBER_OF_WITHDRAWAL_SPLIT
  IERC20(unWrappedToken).safeIncreaseAllowance(address(withdrawalQueue), amount);

  // Swell `createWithdrawRequest` has the minimum and maximum withdraw amount
  // So, we split the original amount if needed
  uint256 numberOfSplit = (amount - 1) / maxWithdrawAmount + 1;
  uint256 actualNumberOfSplit = numberOfSplit.min(MAX_NUMBER_OF_WITHDRAWAL_SPLIT);

  amounts = new uint256[](actualNumberOfSplit);
  uint256 _totalWithdrawAmount = 0;
  for (uint256 i = 0; i < numberOfSplit; ++i) {
    uint256 withdrawAmount = maxWithdrawAmount;
    if (i == numberOfSplit - 1) {
      withdrawAmount = amount - _totalWithdrawAmount;
      _totalWithdrawAmount = amount;
      // Apply a hack to ensure the last piece of amount also greater than the
      minWithdrawAmount
      if (withdrawAmount < minWithdrawAmount) {
        amounts[i - 1] = amounts[i - 1] - minWithdrawAmount + withdrawAmount;
        amounts[i] = minWithdrawAmount;
      } else {
        amounts[i] = withdrawAmount;
      }
    } else {
      amounts[i] = withdrawAmount;
      _totalWithdrawAmount += withdrawAmount;
    }
    // To avoid the number of split is too high, we can skip the amount left for the
    next rebalance
    if (i == actualNumberOfSplit - 1) break;
  }
```

Recommendation: The issue can be fixed as follows,

```
function _claimFromSwell(uint256 amount) internal returns (uint256[] memory amounts,
uint256[] memory requestIds) {
     ...
-    IERC20(unWrappedToken).safeIncreaseAllowance(address(withdrawalQueue), amount);
          ...
+    IERC20(unWrappedToken).safeIncreaseAllowance(address(withdrawalQueue),
_totalWithdrawAmount);
 }
```

Client: Fixed in https://github.com/Tempest-Finance/tempest_smart_contract/pull/151

Renascence: The issue has been resolved as per recommendation.

[L-2] Incorrect “ethValused for providing liquidity indeposits()‘

Context: AmbientUser.sol#L156

Description: In `deposits()`, the `_provideLiquidity()` is incorrectly called with a wrong `ethVal` value.

For ETH native token, `ethVal` should use `amounts[0]` instead of `calcResult.amount0ToDeposit`, which has been subtracted by padding in `_calculateDepositValues()`. That means the provided ETH for minting of liquidity will be slightly lower due to the padding reduction.

Recommendation: This could be fixed as below,

```
    if (calcResult.liq != 0) {
      // Provide liquidity based on the calculated liquidity and current ticks
      _provideLiquidity(
        ProvideLiqParams({
          upperTick: _lpParams[0].upperTick,
          lowerTick: _lpParams[0].lowerTick,
          liq: calcResult.liq,
-          ethVal: _tokenAddresses[0] == address(0) ? calcResult.amount0ToDeposit : 0,
+          ethVal: _tokenAddresses[0] == address(0) ? amounts[0] : 0,
          checkSlippage: checkSlippage,
          sqrtOraclePrice: oraclePrice.sqrtPrice
        })
      );
    }
```

Client: Fixed in https://github.com/Tempest-Finance/tempest_smart_contract/pull/149 Added check before the transfer in https://github.com/Tempest-Finance/tempest_smart_contract/pull/153

Renascence: The issue has been resolved as per recommendation.

[L-3] Temporary block first deposit in LST strategies

Context: LstAmbientUser.sol#L84 LstAmbientLiquidity.sol#L309 VaultLibrary.sol#L134

Description: When the deposit function is called, the strategy triggers the _updateStratPnLAnd-Fees function:

```
    function _updateStratPnLAndFees(uint256 totalAssets, uint256 depositAmount)
    internal {
      uint256 _invested = invested;
      if (totalAssets > _invested) {
        uint256 assetForFees = ((totalAssets - _invested) * fee) / BASE;
        if (assetForFees != 0) {
          _mint(
            feeRecipient,
»          VaultLibrary._convertToShares(assetForFees, totalAssets - assetForFees -
depositAmount, totalSupply())
          );
        }
      }
      invested = totalAssets;
    }
```

In this logic, if there are profits, a portion is minted as fees for the fee recipient. However, if this is the first deposit (i.e., totalSupply == 0), the VaultLibrary._convertToShares function may revert because assetForFees < DEAD_SHARES:

```
    function _convertToShares(uint256 amount, uint256 totalAssets, uint256 totalSupply)
    internal pure returns (uint256) {
      if (totalSupply == 0) {
»      if (amount <= DEAD_SHARES) revert(SMALL_AMOUNT);
        return amount - DEAD_SHARES;
      }
      return amount.mulDiv(totalSupply, totalAssets);
    }
```

An attacker could exploit this by sending a small amount of assets to the strategy contract, causing the first deposit to mint fees and trigger a revert.

Recommendation: Send a small amount of assets to the strategy to unblock it if this attack happens.

Client: https://github.com/Tempest-Finance/tempest_smart_contract/pull/159/files

Renascence: Fixed. Fees are not collected if totalSupply = 0.

Informational

[I-1] Add storage gaps to reserve space for future state variables

Context:

- AmbientStorageLayout.sol#L27
- SymetricAmbientStorageLayout.sol#L9
- LstAmbientStorageLayout.sol#L29
- WstEthAmbientStorageLayout.sol#L16
- RswEthAmbientStorageLayout.sol#L17

Description: Consider adding storage gaps to reserve space for new state variables in the future without compromising storage compatibility with existing deployments.

Recommendation: It is recommended to declare storage gaps at the the end of each storage layout as follows:

```
abstract contract AmbientStorageLayout {
  uint8 internal assetIdx;
  uint8 internal _decimals;
  uint8 public padding;
  uint8 internal cmdId;
  uint16 public fee;
  uint16 public investedPercentage;
  uint16 public swapSlippage;
  uint16 public liqSlippage;
  address public oracle;
  address public feeRecipient;
  address public operatorPath;
  address public userPath;
  address[2] internal tokenAddresses;
  LpParam[] internal lpParams;
  CrocsSwapDex public crocsSwapDex;
  CrocsQuery public crocsQuery;
  uint256 public minimumDeposit;
+  uint256[50] __gap;
}
```

Client: Fixed in `https://github.com/Tempest-Finance/tempest_smart_contract/pull/147/files`

Renascence: The issue has been resolved as per recommendation.

[I-2] Using `Expand` instead `Ceil`

Context:

- VaultLibrary.sol#L151

Description: The OZ's ERC4626 library uses `Ceil` as shown here, and many of the codebases use `Ceil`. However, the codebase uses `Expand` instead.

Recommendation: Although `Ceil` and `Expand` behave similarly for positive numbers, it is safer to follow the battle-tested option in case there is some weird unknown edge case with using `Expand`.

Client: fixed here : `https://github.com/Tempest-Finance/tempest_smart_contract/pull/156`

Renascence: Fixed as per recommended.

[I-3] `withdraw()`s return values refactoring

Context:

- AmbientUser.sol#L212
- LstAmbientUser.sol#L156

Description: Observed that the return value refers to `// @return The amount of shares or assets burned for the withdrawal.`

However, the comment does not make sense, as assets cannot be burned. Only shares are burned during withdrawal.

```
File: AmbientUser.sol
174:   /// @return The amount of shares or assets burned for the withdrawal
175:   /// @dev This function collects fees, previews the withdrawal, calculates assets, and performs the withdrawal
176:   /// @dev This function is non-reentrant and ensures proper withdrawal state
177:   function withdraw(
178:     uint256 amount,
179:     bool isAssets,
180:     address receiver,
181:     address owner,
182:     uint256 minimumReceive,
183:     bool checkSlippage
184:   ) external returns (uint256) {
..SNIP..
212:     return isAssets ? shares : assets;
213:   }
```

Recommendation: It would be clearer to the caller if the returned values are as follows:

```
-  /// @return The amount of shares or assets burned for the withdrawal
+  /// @return The amount of shares burned for the withdrawal
+  /// @return The amount of assets received for the withdrawal
  /// @dev This function collects fees, previews the withdrawal, calculates assets,
  and performs the withdrawal
  /// @dev This function is non-reentrant and ensures proper withdrawal state
  function withdraw(
    uint256 amount,
    bool isAssets,
    address receiver,
    address owner,
    uint256 minimumReceive,
    bool checkSlippage
-  ) external returns (uint256) {
+  ) external returns (uint256 shares, uint256 assets) {
..SNIP..
- return isAssets ? shares : assets;
```

Client: fixed here https://github.com/Tempest-Finance/tempest_smart_contract/pull/158

Renascence: Fixed as per recommended.