Treehouse tETH Updates Audit Report

Dec 28, 2024

Table of Contents

Summary

This report has been prepared for Treehouse smart contract, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Overview

Project Summary

Project Name	**Treehouse**
Codebase	**https://github.com/treehouse-gaia/tETH**
Commit	**252b0a2ad4decc19d74c251c2f91b66d5ad0c44c**
Language	**Solidity**

Audit Summary

Delivery Date	**Dec 28, 2024**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**3**

[WP-M1] Double Impact of wstETH Price Drop During Redemption Waiting Period

Medium

Issue Description

During the redemption waiting period, if Lido stETH is subject to slashing penalties (causing the wstETH price to drop), users are unexpectedly forced to bear the impact of the wstETH price decline twice.

This is because in `finalizeRedeem`, `_getReturnAmount()` will be called to adjust the actual return amount based on the change in wstETH's rate since the `startTime` of the `redemption`.

```
76    /**
77     * @notice Redeem tAsset
78     * @param _shares amount of tAsset to redeem
79     */
80    function redeem(uint96 _shares) external nonReentrant whenNotPaused {
81      uint128 _assets = IERC4626(TASSET).previewRedeem(_shares).toUint128();
82      if (_assets < minRedeemInUnderlying) revert MinimumNotMet();
83
84      IERC20(TASSET).safeTransferFrom(msg.sender, address(this), _shares);
85
86      redemptionInfo[msg.sender].push(
87        RedemptionInfo({
88          startTime: block.timestamp.toUint64(),
89          assets: _assets,
90          shares: _shares,
91          baseRate: _getBaseRate().toUint128()
92        })
93      );
94
95      unchecked {
96        redeeming[msg.sender] += _shares;
97        totalRedeeming += _shares;
98      }
99
100     emit Redeemed(msg.sender, _shares, _assets);
101   }
```

```
102
103   /**
104    * @notice Finalize tAsset redemption
105    * @param _redeemIndex index to finalize
106    */
107   function finalizeRedeem(
108     uint _redeemIndex
109   ) external nonReentrant whenNotPaused validateRedeem(msg.sender, _redeemIndex) {
110     RedemptionInfo storage _redeem = redemptionInfo[msg.sender][_redeemIndex];
111
112     if (block.timestamp < _redeem.startTime + waitingPeriod) revert
    InWaitingPeriod();
113     uint _assets = IERC4626(TASSET).redeem(_redeem.shares, address(this),
    address(this));
114     redeeming[msg.sender] -= _redeem.shares;
115     totalRedeeming -= _redeem.shares;
116
117     address _underlying = VAULT.getUnderlying();
118
119     uint _returnAmount = _getReturnAmount(_redeem.assets, _redeem.baseRate,
    _assets, _getBaseRate());
120     uint _fee = (_returnAmount * redemptionFee) / PRECISION;
121     _returnAmount = _returnAmount - _fee;
122
123     if (_returnAmount > _redeem.assets) revert RedemptionError();
124
125     if (IERC20(_underlying).balanceOf(address(VAULT)) < _returnAmount) revert
    InsufficientFundsInVault();
126     IInternalAccountingUnit(IAU).burn(_returnAmount);
127     REDEMPTION_CONTROLLER.redeem(_returnAmount, msg.sender);
128
129     // reused assignment - transfer leftover asset back into 4626
130     _assets = IERC20(IAU).balanceOf(address(this));
131
132     if (_assets > 0) {
133       IERC20(IAU).safeTransfer(TASSET, _assets);
134     }
135
136     emit RedeemFinalized(msg.sender, _returnAmount, _fee);
137
138     // last because deletes are in-place
139     _deleteRedeemEntry(_redeemIndex);
140   }
```

```
@@ 212,219 @@
220   function _getReturnAmount(uint128 _b0, uint128 _c0, uint _bn, uint _cn) internal
      pure returns (uint) {
221     uint _minC;
222     uint _maxC;
223
224     if (_c0 > _cn) {
225       _minC = _cn;
226       _maxC = _c0;
227     } else {
228       _minC = _c0;
229       _maxC = _cn;
230     }
231
232     return (_minC * (_b0 > _bn ? _bn : _b0)) / _maxC;
233   }
234
235   /**
236    * @notice Get base rate of asset
237    */
238   function _getBaseRate() private view returns (uint) {
239     return IwstETH(payable(VAULT.getUnderlying())).stEthPerToken();
240   }
```

PoC

Initial State:

- Assume the initial exchange rate between wstETH and stETH is 1 wstETH = 1 stETH. Therefore, `_getBaseRate()` returns 1e18 (assuming both stETH and wstETH have 18 decimal places).
- User A holds 100 tAsset.
- User A decides to redeem 100 tAsset.
- `minRedeemInUnderlying` is satisfied.
- `redemptionFee` is set to 0 to simplify analysis and focus on the core issue.

Step 1: User Initiates Redemption (calls `redeem`)

1. User A calls `redeem(100)`.
2. `previewRedeem(100)` is called, assuming it returns 100e18 underlying assets (stETH). Thus, `_assets = 100e18`.
3. 100 tAsset is transferred from User A to the contract.
4. A new `RedemptionInfo` entry is added to `redemptionInfo[User A]`:

```
RedemptionInfo({
    startTime: current timestamp,
    assets: 100e18, // _assets
    shares: 100,
    baseRate: 1e18  // _getBaseRate(), assuming initial price is 1:1
})
```

Step 2: Waiting Period (wstETH Price Drops)

- During the waiting period, wstETH price drops, resulting in a new exchange rate of 1 wstETH = 0.9 stETH.
- At this point, `_getBaseRate()` returns 0.9e18.

Step 3: User Finalizes Redemption (calls `finalizeRedeem`)

1. User A calls `finalizeRedeem(0)` (assuming this is their only pending redemption).
2. TreehouseRedemptionV2.solL113 `IERC4626(TASSET).redeem(_redeem.shares, address(this), address(this))` is called, `_assets = 100e18`.
3. TreehouseRedemptionV2.solL119 `_getReturnAmount(100e18, 1e18, 100e18, 0.9e18)` is called:
    - `_b0 = 100e18`
    - `_c0 = 1e18`
    - `_bn = 100e18`
    - `_cn = 0.9e18`
    - Since `_c0 > _cn`, `_minC = 0.9e18`, `_maxC = 1e18`
    - `(_b0 > _bn ? _bn : _b0)` results in `100e18`
    - `_returnAmount = (0.9e18 * 100e18) / 1e18 = 90e18`
4. Final return amount `_returnAmount = 90e18 - 0 = 90e18`
5. TreehouseRedemptionV2.solL127 will transfer `90e18` UNDERLYING (wstETH) to the user.

https://github.com/treehouse-gaia/tETH/blob/252b0a2ad4decc19d74c251c2f91b66d5ad0c44c/

RedemptionController.sol

```
48    function redeem(uint _amount, address _recipient) external whenNotPaused {
49      if (_redemptionContracts.contains(msg.sender) == false) revert Unauthorized();
50      IERC20(UNDERLYING).safeTransferFrom(address(VAULT), _recipient, _amount);
51    }
```

The actual value of assets received by the user is `90 wstETH <> 90 * 0.9 === 81 stETH`

The expected amount should be `100 wstETH <> 90 stETH` .

Recommendation

We believe the TreehouseRedemptionV2 for tETH should not account for wstETH's yields or losses.

Since:

- `tETH.asset()` is `IAU_wstETH` whose `UNDERLYING` is wstETH
- TreehouseRedemptionV2's function for tETH is to withdraw wstETH from tETH

Therefore, we propose modifying finalizeRedeem to:

```
107   function finalizeRedeem(
108     uint _redeemIndex
109   ) external nonReentrant whenNotPaused validateRedeem(msg.sender, _redeemIndex) {
110     RedemptionInfo storage _redeem = redemptionInfo[msg.sender][_redeemIndex];
111
112     if (block.timestamp < _redeem.startTime + waitingPeriod) revert
    InWaitingPeriod();
113     uint _assets = IERC4626(TASSET).redeem(_redeem.shares, address(this),
    address(this));
114     redeeming[msg.sender] -= _redeem.shares;
115     totalRedeeming -= _redeem.shares;
116
117     address _underlying = VAULT.getUnderlying();
118
119     uint _returnAmount = _redeem.assets > _assets ? _assets : _redeem.assets;
120     uint _fee = (_returnAmount * redemptionFee) / PRECISION;
121     _returnAmount = _returnAmount - _fee;
122
```

```
123       if (_returnAmount > _redeem.assets) revert RedemptionError();
124
125       if (IERC20(_underlying).balanceOf(address(VAULT)) < _returnAmount) revert
      InsufficientFundsInVault();
126       IInternalAccountingUnit(IAU).burn(_returnAmount);
127       REDEMPTION_CONTROLLER.redeem(_returnAmount, msg.sender);
128
129       // reused assignment - transfer leftover asset back into 4626
130       _assets = IERC20(IAU).balanceOf(address(this));
131
132       if (_assets > 0) {
133         IERC20(IAU).safeTransfer(TASSET, _assets);
134       }
135
136       emit RedeemFinalized(msg.sender, _returnAmount, _fee);
137
138       // last because deletes are in-place
139       _deleteRedeemEntry(_redeemIndex);
140     }
```

This modification means users will bear the risk of tAsset share price decrease during the redemption period but won't share the gains from tAsset share price increases.

The yields or losses at the wstETH layer during this period are not considered.

Status

ⓘ Acknowledged

[WP-I2] `TreehouseRedemptionV2` only supports `TASSET` with `VAULT.getUnderlying()` being `wstETH`

Informational

Issue Description

`_getBaseRate()` assumes the `VAULT.getUnderlying()` must be `wstETH` .

```
235     /**
236      * @notice Get base rate of asset
237      */
238     function _getBaseRate() private view returns (uint) {
239       return IwstETH(payable(VAULT.getUnderlying())).stEthPerToken();
240     }
```

Implementing our recommendation in [WP-M1] will render this issue irrelevant as well.

Status

ⓘ Acknowledged

[WP-I3] The amount in the `RedeemFinalized` event of `finalizeRedeem()` is missing the `_assets - _redeem.assets` portion.

Informational

Issue Description

The TreehouseRedemptionV2.solL133 `IERC20(IAU).safeTransfer(TASSET, _assets)` transfer to `TASSET` causes a sudden increase in the TASSET share price, which includes two components:

- TreehouseRedemptionV2.solL119 `_assets - _redeem.assets`
- TreehouseRedemptionV2.solL120-121 `_fee`

However, the RedeemFinalized event at TreehouseRedemptionV2.solL136 only includes one of these components.

To improve observability, consider changing to `_assets` or separately including both `_assets - _fee` and `_fee` .

```
103     /**
104      * @notice Finalize tAsset redemption
105      * @param _redeemIndex index to finalize
106      */
107     function finalizeRedeem(
108       uint _redeemIndex
109     ) external nonReentrant whenNotPaused validateRedeem(msg.sender, _redeemIndex) {
110       RedemptionInfo storage _redeem = redemptionInfo[msg.sender][_redeemIndex];
111
112       if (block.timestamp < _redeem.startTime + waitingPeriod) revert
    InWaitingPeriod();
113       uint _assets = IERC4626(TASSET).redeem(_redeem.shares, address(this),
    address(this));
114       redeeming[msg.sender] -= _redeem.shares;
115       totalRedeeming -= _redeem.shares;
116
117       address _underlying = VAULT.getUnderlying();
118
119       uint _returnAmount = _getReturnAmount(_redeem.assets, _redeem.baseRate,
    _assets, _getBaseRate());
```

```
120     uint _fee = (_returnAmount * redemptionFee) / PRECISION;
121     _returnAmount = _returnAmount - _fee;
122
123     if (_returnAmount > _redeem.assets) revert RedemptionError();
124
125     if (IERC20(_underlying).balanceOf(address(VAULT)) < _returnAmount) revert
    InsufficientFundsInVault();
126     IInternalAccountingUnit(IAU).burn(_returnAmount);
127     REDEMPTION_CONTROLLER.redeem(_returnAmount, msg.sender);
128
129     // reused assignment - transfer leftover asset back into 4626
130     _assets = IERC20(IAU).balanceOf(address(this));
131
132     if (_assets > 0) {
133       IERC20(IAU).safeTransfer(TASSET, _assets);
134     }
135
136     emit RedeemFinalized(msg.sender, _returnAmount, _fee);
137
138     // last because deletes are in-place
139     _deleteRedeemEntry(_redeemIndex);
140   }
```

Status

ⓘ Acknowledged

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Smart Contract technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.