Treehouse Fastlane Audit Report

Dec 2, 2024

WATCHPUG

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-protocol**
Commit	**f72c946322cbf56ad73a89f862262a156b8111cf**
Language	**Solidity**

Audit Summary

Delivery Date	**Dec 2, 2024**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**9**

[WP-M1] Inverted condition for zero address check

Medium

Issue Description

In the `setFeeContract` function, the check for a zero address is incorrectly implemented.

The condition `_newContract != IFastlaneFee(address(0)` should be `_newContract == IFastlaneFee(address(0))` .

```
function setFeeContract(IFastlaneFee _newContract) external onlyOwner {
  if (_newContract != IFastlaneFee(address(0))) revert ZeroAddress();
  emit FeeContractUpdated(_newContract, feeContract);
  feeContract = _newContract;
}
```

Status

✓ Fixed

[WP-M2] The unit of `getRedeemableAmount` should be in assets, but `totalRedeeming` 's `_earmark` is in shares

Medium

Issue Description

The function `getRedeemableAmount()` attempts to calculate `_totalRedeemable` by subtracting two values with inconsistent units.

The issue is that `_earmark` is denominated in shares while `_underlyingInVault` is in assets.

```
function getRedeemableAmount() public view returns (uint _totalRedeemable) {
    uint _underlyingInVault = IERC20(UNDERLYING).balanceOf(address(VAULT));
    uint _earmark = ITreehouseRedemption(REDEMPTION_CONTRACT).totalRedeeming();

    _totalRedeemable = _earmark > _underlyingInVault ? 0 : _underlyingInVault - _earmark;
}
```

```
function redeem(uint96 _shares) external nonReentrant whenNotPaused {
    uint128 _assets = IERC4626(TASSET).previewRedeem(_shares).toUint128();
    if (_assets < minRedeemInEth) revert MinimumNotMet();

    IERC20(TASSET).safeTransferFrom(msg.sender, address(this), _shares);

    redemptionInfo[msg.sender].push(
      RedemptionInfo({
        startTime: block.timestamp.toUint64(),
        assets: _assets,
        shares: _shares,
        baseRate: _getBaseRate().toUint128()
      })
    );

    unchecked {
      redeeming[msg.sender] += _shares;
      totalRedeeming += _shares;
    }

    emit Redeemed(msg.sender, _shares, _assets);
}
```

https://github.com/treehouse-gaia/tETH-protocol/blob/40812a5bb857fa027c962bbfa55ae214356c039f/contracts/TreehouseFastlane.sol#L72-L89

```
72  function redeemAndFinalize(uint96 _shares) external nonReentrant whenNotPaused {
73      uint _assets = IERC4626(TASSET).previewRedeem(_shares);
74      if (_assets < minRedeemInUnderlying) revert MinimumNotMet();
75      if (getRedeemableAmount() < _assets) revert InsufficientFundsInVault();
76
77      IERC20(TASSET).safeTransferFrom(msg.sender, address(this), _shares);
78      _assets = IERC4626(TASSET).redeem(_shares, address(this), address(this));
79      uint _fee = feeContract.applyFee(_assets);
80
81      IInternalAccountingUnit(IAU).burn(_assets);
82
83      REDEMPTION_CONTROLLER.redeem(_assets - _fee, msg.sender);
84      REDEMPTION_CONTROLLER.redeem(_fee, treasury);
85      // UNDERLYING.safeTransferFrom(address(VAULT), msg.sender, _assets - _fee);
86      // UNDERLYING.safeTransferFrom(address(VAULT), treasury, _fee);
87
88      emit Redeemed(msg.sender, _shares, _assets, _fee);
89  }
```

Status

✓ **Fixed**

[WP-L3] The implementation (equivalent to require _newFee <= 501) is inconsistent with the NatSpec documentation's stated expectation of "fees (max 5%)"

Low

Issue Description

501 (5.01%) is now allowed.

The expected max fee should be `500` instead of `501` .

```
/**
 * @notice Fastlane fee contract
 */
contract FastlaneFee is IFastlaneFee, Ownable2Step {
  uint constant PRECISION = 1e4;
  uint public fee = 200; // 2% in bips
@@ 18,35 @@
  /**
   * Set fees (max 5%)
   * @dev onlyOwner
   * @param _newFee new fee in bips
   */
  function setFee(uint _newFee) external onlyOwner {
    if (_newFee > 501) revert MaxFeeExceeded(); // max out at 5%;
    emit FeeUpdated(_newFee, fee);
    fee = _newFee;
  }
}
```

Recommendation

Consider changing to:

```
  /**
   * Set fees (max 5%)
   * @dev onlyOwner
```

```
 * @param _newFee new fee in bips
 */
function setFee(uint _newFee) external onlyOwner {
  if (_newFee > 500) revert MaxFeeExceeded(); // max out at 5%;
  emit FeeUpdated(_newFee, fee);
  fee = _newFee;
}
```

Status

✓ **Fixed**

[WP-L4] Modifying `waitingPeriod` affects all pending redemption requests, not just the redemption requests after the modification.

Low

Issue Description

An alternative design would be to store the maturity time in the redeeming struct so that `setWaitingPeriod` only affects future redemption requests.

```solidity
  /**
   * @notice Finalize tAsset redemption
   * @param _redeemIndex index to finalize
   */
  function finalizeRedeem(
    uint _redeemIndex
  ) external nonReentrant whenNotPaused validateRedeem(msg.sender, _redeemIndex) {
    RedemptionInfo storage _redeem = redemptionInfo[msg.sender][_redeemIndex];

    if (block.timestamp < _redeem.startTime + waitingPeriod) revert InWaitingPeriod();
    uint _assets = IERC4626(TASSET).redeem(_redeem.shares, address(this), address(this));
    redeeming[msg.sender] -= _redeem.shares;
    totalRedeeming -= _redeem.shares;

    address _underlying = VAULT.getUnderlying();

    uint _returnAmount = _getReturnAmount(_redeem.assets, _redeem.baseRate, _assets, _getBaseRate());
    uint _fee = (_returnAmount * redemptionFee) / PRECISION;
    _returnAmount = _returnAmount - _fee;

    if (_returnAmount > _redeem.assets) revert RedemptionError();

    if (IERC20(_underlying).balanceOf(address(VAULT)) < _returnAmount) revert InsufficientFundsInVault();
    IInternalAccountingUnit(IAU).burn(_returnAmount);
    REDEMPTION_CONTROLLER.redeem(_returnAmount, msg.sender);

    // reused assignment - transfer leftover asset back into 4626
    _assets = IERC20(IAU).balanceOf(address(this));

    if (_assets > 0) {
      IERC20(IAU).safeTransfer(TASSET, _assets);
    }

    emit RedeemFinalized(msg.sender, _returnAmount, _fee);

    // last because deletes are in-place
    _deleteRedeemEntry(_redeemIndex);
  }

  /**
   * @notice Set the waiting period for finalizing redeems
   * @param _newWaitingPeriod new waiting period in seconds
   */
  function setWaitingPeriod(uint32 _newWaitingPeriod) external onlyOwner {
    emit WaitingPeriodUpdated(_newWaitingPeriod, waitingPeriod);
    waitingPeriod = _newWaitingPeriod;
  }
```

Status

Acknowledged

[WP-G5] Redundant `whenNotPaused` modifier in `redeemAndFinalize()`

Gas

Issue Description

The `REDEMPTION_CONTROLLER.redeem()` at TreehouseFastlane.solL83-84 already includes a `whenNotPaused` modifier, and `TreehouseFastlane` shares the same `paused` status with `REDEMPTION_CONTROLLER` .

```
68    /**
69     * @notice Atomically redeem tAsset
70     * @param _shares amount of tAsset to redeem
71     */
72    function redeemAndFinalize(uint96 _shares) external nonReentrant whenNotPaused {
73      uint _assets = IERC4626(TASSET).previewRedeem(_shares);
74      if (_assets < minRedeemInUnderlying) revert MinimumNotMet();
75      if (getRedeemableAmount() < _assets) revert InsufficientFundsInVault();
76
77      IERC20(TASSET).safeTransferFrom(msg.sender, address(this), _shares);
78      _assets = IERC4626(TASSET).redeem(_shares, address(this), address(this));
79      uint _fee = feeContract.applyFee(_assets);
80
81      IInternalAccountingUnit(IAU).burn(_assets);
82
83      REDEMPTION_CONTROLLER.redeem(_assets - _fee, msg.sender);
84      REDEMPTION_CONTROLLER.redeem(_fee, treasury);
85      // UNDERLYING.safeTransferFrom(address(VAULT), msg.sender, _assets - _fee);
86      // UNDERLYING.safeTransferFrom(address(VAULT), treasury, _fee);
87
88      emit Redeemed(msg.sender, _shares, _assets, _fee);
89    }
```

```
131   /**
132    * Inherits pause state from RedemptionController
133    */
134   function paused() public view override returns (bool) {
135     return REDEMPTION_CONTROLLER.paused();
}
```

```
/**
 * Redeem underlying from vault to `_recipient`
 * @dev only redemption contracts
 * @param _amount amount to redeem
 * @param _recipient recipient
 */
function redeem(uint _amount, address _recipient) external whenNotPaused {
  if (_redemptionContracts.contains(msg.sender) == false) revert Unauthorized();
  IERC20(UNDERLYING).safeTransferFrom(address(VAULT), _recipient, _amount);
}
```

Recommendation

Recommend removing the `Pausable` parent contract inheritance from `TreehouseFastlane` .

Status

ⓘ Acknowledged

[WP-I6] `finalizeRedeem` can be frontrun when the size of the redeem is large enough.

Informational

Issue Description

`finalizeRedeem` returns both the fee and share price difference back to the 4626; a huge redeem can cause a surge in share price.

If the sudden increase in share price ratio exceeds the Fastlane fee, attackers can profit by front-running the `finalizeRedeem` transaction and then back-running the Fastlane Redeem.

Given the fact that Fastlane has a significant fee, it’s quite hard in practice for the frontrun to be profitable.

```
function finalizeRedeem(
    uint _redeemIndex
  ) external nonReentrant whenNotPaused validateRedeem(msg.sender, _redeemIndex) {
    RedemptionInfo storage _redeem = redemptionInfo[msg.sender][_redeemIndex];

    if (block.timestamp < _redeem.startTime + waitingPeriod) revert
InWaitingPeriod();
    uint _assets = IERC4626(TASSET).redeem(_redeem.shares, address(this),
address(this));
    redeeming[msg.sender] -= _redeem.shares;
    totalRedeeming -= _redeem.shares;

    address _underlying = VAULT.getUnderlying();

    uint _returnAmount = _getReturnAmount(_redeem.assets, _redeem.baseRate,
_assets, _getBaseRate());
    uint _fee = (_returnAmount * redemptionFee) / PRECISION;
    _returnAmount = _returnAmount - _fee;

    if (_returnAmount > _redeem.assets) revert RedemptionError();

    if (IERC20(_underlying).balanceOf(address(VAULT)) < _returnAmount) revert
InsufficientFundsInVault();
    IInternalAccountingUnit(IAU).burn(_returnAmount);
    REDEMPTION_CONTROLLER.redeem(_returnAmount, msg.sender);

    // reused assignment - transfer leftover asset back into 4626
    _assets = IERC20(IAU).balanceOf(address(this));

    if (_assets > 0) {
      IERC20(IAU).safeTransfer(TASSET, _assets);
    }

    emit RedeemFinalized(msg.sender, _returnAmount, _fee);

    // last because deletes are in-place
    _deleteRedeemEntry(_redeemIndex);
  }
```

```
    uint public fee = 200; // 2% in bips
```

Status

Acknowledged

[WP-I7] `minRedeemInEth` can be misleading/prone to error when TASSET's `decimals` is not 18.

Informational

Issue Description

For example, let's say the asset is USDC and the TASSET is tUSDC with a `decimals` of 8; then `minRedeemInEth` would be 1e10 times bigger than the face value.

```
function redeem(uint96 _shares) external nonReentrant whenNotPaused {
  uint128 _assets = IERC4626(TASSET).previewRedeem(_shares).toUint128();
  if (_assets < minRedeemInEth) revert MinimumNotMet();

  IERC20(TASSET).safeTransferFrom(msg.sender, address(this), _shares);

  redemptionInfo[msg.sender].push(
    RedemptionInfo({
      startTime: block.timestamp.toUint64(),
      assets: _assets,
      shares: _shares,
      baseRate: _getBaseRate().toUint128()
    })
  );

  unchecked {
    redeeming[msg.sender] += _shares;
    totalRedeeming += _shares;
  }

  emit Redeemed(msg.sender, _shares, _assets);
}
```

```
/**
 * @notice Set the minumum redemption size
 * @param _newMinRedeemInUnderlying new minimum in 1e18
 */
function setMinRedeem(uint96 _newMinRedeemInUnderlying) external onlyOwner {
  emit MinRedeemUpdated(_newMinRedeemInUnderlying, minRedeemInUnderlying);
```

```
        minRedeemInUnderlying = _newMinRedeemInUnderlying;
    }

```

Status

✓ **Fixed**

[WP-N8] `Unauthorized` and `RedemptionError` are defined but never used

Issue Description

TreehouseRedemptionV2.sol

```
19    error Unauthorized();
```

```
20    error RedemptionError();
```

Status

✓ Fixed

[WP-N9] Unconventional event parameter naming style

Issue Description

Event parameter names typically don't start with `_` .

When accessing event data, using parameter names that start with `_` becomes necessary, which is not ideal.

TreehouseFastlane.sol

```
interface ITreehouseFastlane {
  error MinimumNotMet();
  error InsufficientFundsInVault();
  error RedemptionError();
  error ZeroAddress();

  event Redeemed(address indexed _user, uint _shares, uint _assets, uint _fee);
  event MinRedeemUpdated(uint128 _new, uint128 _old);
  event FeeContractUpdated(IFastlaneFee _new, IFastlaneFee _old);
}
```

Status

✓ **Fixed**

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.