Treehouse TRH Token Audit Report

Mar 20, 2025

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/treehouse-token/**
Commit	**b7622bff3b3a12939a44dd90716c3de9fe9bac75**
Language	**Solidity**

Audit Summary

Delivery Date	**Mar 20, 2025**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**5**

[WP-I1] Low precision of `bonusRate` can lead to users receiving less money due to precision loss.

Informational

Issue Description

The precision of `bonusRate` is only 1e6.

Let's say if:

- tranchAllocation = 73127e18
- bonusAllocation = 1000e18
- bonusRate = 1000e18 * 1e6 / 73127e18 = 13674

Assume there is only one user and `isInstantRelease == true` .

Then the bonus amount received by the user, bonusAmount = 73127e18 * 13674 / 1e6 = 999938598000000000000.

The loss is 1000e18 - 999938598000000000000 = 61402000000000000.

Consider using 1e18 precision.

https://github.com/treehouse-gaia/treehouse-token/blob/b7622bff3b3a12939a44dd90716c3de9fe9bac75/contracts/TreehouseAirdropWithBonusPool.sol#L13-L14

```
13  contract TreehouseAirdropWithBonusPool is ITreehouseAirdropWithBonusPool,
    Ownable2Step {
14    uint constant PRECISION = 1e6;
```

```
66    function newTranche(
67      bytes32 _merkleRoot,
68      uint152 _tranchAllocation,
69      uint152 _bonusAllocation,
70      uint32 _startTime,
71      uint32 _endTime,
```

```
72      bool _isInstantRelease
73    ) external onlyOwner returns (uint trancheId) {
74      TOKEN.transferFrom(msg.sender, address(this), _tranchAllocation);
75
76      if (_bonusAllocation > 0) {
77        TOKEN.transferFrom(msg.sender, address(BONUS_POOL), _bonusAllocation);
78      }
79
80      trancheId = tranche;
81      merkleRoots[trancheId] = _merkleRoot;
82
83      if (_startTime >= _endTime || _bonusAllocation > _tranchAllocation || _endTime
    > SWEEP_TIME)
84        revert InvalidTrancheRelease();
85      trancheReleases[tranche] = TracheRelease({
86        startTime: _startTime,
87        endTime: _endTime,
88        bonusAllocation: _bonusAllocation,
89        bonusRate: uint32((_bonusAllocation * PRECISION) / _tranchAllocation),
90        isInstantRelease: _isInstantRelease
91      });
92
93      tranche += 1;
94
95      emit TrancheAdded(trancheId, _merkleRoot, _tranchAllocation);
96    }
```

https://github.com/treehouse-gaia/treehouse-token/blob/b7622bff3b3a12939a44dd90716c3de9fe9bac75/contracts/TreehouseAirdropWithBonusPool.sol#L150-L169

```
150    /**
151     * @dev return claimable amount and bonus amount.
152     */
153    function claimableBalance(uint _tranche, uint _amount) public view returns (uint
    claimableAmount, uint bonusAmount) {
154      TracheRelease memory tr = trancheReleases[_tranche];
155
156      if (tr.isInstantRelease) {
157        claimableAmount = _amount;
158        bonusAmount = (_amount * tr.bonusRate) / PRECISION;
```

```
159       } else {
160         claimableAmount = _amount;
161
162         uint _fullBonusAmount = (_amount * tr.bonusRate) / PRECISION;
163         if (block.timestamp < tr.endTime) {
164           bonusAmount = (_fullBonusAmount * (block.timestamp - tr.startTime)) /
    (tr.endTime - tr.startTime);
165         } else {
166           bonusAmount = _fullBonusAmount;
167         }
168       }
169     }
```

Status

ⓘ Acknowledged

[WP-I2] `trancheReleases[tranche].endTime` can be set to `SWEEP_TIME`, making it possible for an early sweep.

Informational

Issue Description

There is no waiting period between `trancheReleases[tranche].endTime` and `SWEEP_TIME` for users to claim, which could cause users waiting for full `_fullBonusAmount` bonus from non-`isInstantRelease` `trancheReleases[_tranche]` to unexpectedly miss their Airdrop.

According to L83, the latest `trancheReleases[tranche].endTime` could be `SWEEP_TIME`.

According to L163-166, for non-`isInstantRelease` `trancheReleases[_tranche]`, users need to wait until `block.timestamp >= trancheReleases[_tranche].endTime` to receive the full `_fullBonusAmount` bonus.

Therefore, for `trancheReleases[_tranche]` where `trancheReleases[tranche].endTime` equals (or is close to) `SWEEP_TIME`, users need to wait until >= `SWEEP_TIME` (or close to `SWEEP_TIME`) to claim their full `_fullBonusAmount` bonus.

However, the owner can execute `sweep()` at >= `SWEEP_TIME` (transferring all remaining `TOKEN` from `TreehouseAirdropWithBonusPool` and `BONUS_POOL` contracts), according to L120. This could unexpectedly prevent the aforementioned users from calling `claim()`.

Consider ensuring sufficient time for users to `claim()` before `sweep()` is executed.

```
13  contract TreehouseAirdropWithBonusPool is ITreehouseAirdropWithBonusPool,
    Ownable2Step {
    @@ 14,61 @@
62
63    /**
64     * @dev setup a new tranche
65     */
66    function newTranche(
67      bytes32 _merkleRoot,
68      uint152 _tranchAllocation,
69      uint152 _bonusAllocation,
70      uint32 _startTime,
```

```
71      uint32 _endTime,
72      bool _isInstantRelease
73    ) external onlyOwner returns (uint trancheId) {
74      TOKEN.transferFrom(msg.sender, address(this), _tranchAllocation);
75
76      if (_bonusAllocation > 0) {
77        TOKEN.transferFrom(msg.sender, address(BONUS_POOL), _bonusAllocation);
78      }
79
80      trancheId = tranche;
81      merkleRoots[trancheId] = _merkleRoot;
82
83      if (_startTime >= _endTime || _bonusAllocation > _tranchAllocation || _endTime
    > SWEEP_TIME)
84        revert InvalidTrancheRelease();
85      trancheReleases[tranche] = TracheRelease({
86        startTime: _startTime,
87        endTime: _endTime,
88        bonusAllocation: _bonusAllocation,
89        bonusRate: uint32((_bonusAllocation * PRECISION) / _tranchAllocation),
90        isInstantRelease: _isInstantRelease
91      });
92
93      tranche += 1;
94
95      emit TrancheAdded(trancheId, _merkleRoot, _tranchAllocation);
96    }
97

    @@ 98,113 @@

114
115   /**
116    * @notice sweeps any remaining tokens after `SWEEP_TIME`
117    * @dev onlyOwner
118    */
119   function sweep() external onlyOwner {
120     if (block.timestamp < SWEEP_TIME) revert TooEarly();
121     BONUS_POOL.sweep();
122
123     uint _amount = TOKEN.balanceOf(address(this));
124     TOKEN.transfer(rewardReserve, _amount);
125
126     emit Sweep(rewardReserve, _amount);
```

```
127     }
128
129     /**
130      * @dev claim token
131      */
132     function claim(uint _tranche, uint _claimableAmount, bytes32[] memory
    _merkleProof) external {
133       if (_claimableAmount == 0) revert InvalidAmount();
134       _claim(msg.sender, _tranche, _claimableAmount, _merkleProof);
135       _disburse(msg.sender, _tranche, _claimableAmount);
136     }
137
    @@ 138,148 @@
149
150     /**
151      * @dev return claimable amount and bonus amount.
152      */
153     function claimableBalance(uint _tranche, uint _amount) public view returns (uint
    claimableAmount, uint bonusAmount) {
154       TracheRelease memory tr = trancheReleases[_tranche];
155
156       if (tr.isInstantRelease) {
157         claimableAmount = _amount;
158         bonusAmount = (_amount * tr.bonusRate) / PRECISION;
159       } else {
160         claimableAmount = _amount;
161
162         uint _fullBonusAmount = (_amount * tr.bonusRate) / PRECISION;
163         if (block.timestamp < tr.endTime) {
164           bonusAmount = (_fullBonusAmount * (block.timestamp - tr.startTime)) /
    (tr.endTime - tr.startTime);
165         } else {
166           bonusAmount = _fullBonusAmount;
167         }
168       }
169     }
170
    @@ 171,199 @@
200
201     /**
202      * @dev transfer/disbute token to user.
203      */
```

```
204     function _disburse(address _to, uint _tranche, uint _amount) private {
205       (uint claimableAmount, uint bonusAmount) = claimableBalance(_tranche,
      _amount);
206
207       if (bonusAmount > 0) {
208         BONUS_POOL.claimBonus(_to, bonusAmount);
209       }
210
211       TOKEN.transfer(_to, claimableAmount);
212
213       emit Claimed(_to, _tranche, claimableAmount, bonusAmount);
214     }
215   }
```

Status

ⓘ Acknowledged

[WP-N3] Unused code

Issue Description

The two custom errors `TooEarly` and `ZeroAddress` are never used.

```
11      error TooEarly();
12      error ZeroAddress();
```

Status

✓ Fixed

[WP-N4] Typos

Issue Description

TracheRelease -> TrancheRelease

```
32    struct TracheRelease {
33      uint32 startTime;
34      uint32 endTime;
35      uint152 bonusAllocation;
36      uint32 bonusRate;
37      bool isInstantRelease;
38    }
```

$_{t}ranchAllocation$ -> _trancheAllocation

```
63    /**
64     * @dev setup a new tranche
65     */
66    function newTranche(
67      bytes32 _merkleRoot,
68      uint152 _tranchAllocation,
69      uint152 _bonusAllocation,
70      uint32 _startTime,
71      uint32 _endTime,
72      bool _isInstantRelease
73    ) external onlyOwner returns (uint trancheId) {
74      TOKEN.transferFrom(msg.sender, address(this), _tranchAllocation);
75
76      if (_bonusAllocation > 0) {
77        TOKEN.transferFrom(msg.sender, address(BONUS_POOL), _bonusAllocation);
78      }
79
80      trancheId = tranche;
81      merkleRoots[trancheId] = _merkleRoot;
82
83      if (_startTime >= _endTime || _bonusAllocation > _tranchAllocation || _endTime
    > SWEEP_TIME)
84        revert InvalidTrancheRelease();
85      trancheReleases[tranche] = TracheRelease({
86        startTime: _startTime,
```

```
87          endTime: _endTime,
88          bonusAllocation: _bonusAllocation,
89          bonusRate: uint32((_bonusAllocation * PRECISION) / _tranchAllocation),
90          isInstantRelease: _isInstantRelease
91        });
92
93        tranche += 1;
94
95        emit TrancheAdded(trancheId, _merkleRoot, _tranchAllocation);
96      }
```

disbute -> distribute

```
201     /**
202      * @dev transfer/disbute token to user.
203      */
204     function _disburse(address _to, uint _tranche, uint _amount) private {
205       (uint claimableAmount, uint bonusAmount) = claimableBalance(_tranche,
    _amount);
206
207       if (bonusAmount > 0) {
208         BONUS_POOL.claimBonus(_to, bonusAmount);
209       }
210
211       TOKEN.transfer(_to, claimableAmount);
212
213       emit Claimed(_to, _tranche, claimableAmount, bonusAmount);
214     }
```

Status

✓ Fixed

[WP-G5] Minimizing unnecessary computations and variables can save gas

Gas

Issue Description

NftClaim.sol

```
50    /**
51     * Claim airdrop for multiple nfts
52     * @param tokenIds array of tokenIds
53     */
54    function claimMultiple(uint[] memory tokenIds) external {
55      uint totalAmount;
56      for (uint i; i < tokenIds.length; ++i) {
57        _validateAndRecordClaim(msg.sender, tokenIds[i]);
58        totalAmount += wadPerNft;
59      }
60
61      TOKEN.transfer(msg.sender, totalAmount);
62      numberClaimed += uint16(tokenIds.length);
63    }
```

Recommendation

Consider changing to:

```
50    /**
51     * Claim airdrop for multiple nfts
52     * @param tokenIds array of tokenIds
53     */
54    function claimMultiple(uint[] memory tokenIds) external {
55      for (uint i; i < tokenIds.length; ++i) {
56        _validateAndRecordClaim(msg.sender, tokenIds[i]);
57      }
58
59      TOKEN.transfer(msg.sender, tokenIds.length * wadPerNft);
60      numberClaimed += uint16(tokenIds.length);
```

```
61     }
```

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.