Treehouse tAVAX Audit Report

Aug 22, 2025

Table of Contents

Summary **2**

Overview **3**

Issues **4**

[WP-M1] Current implementation cannot claim rewards from Aave's `Merit Programs` for asAVAX 4

[WP-L2] Approving the sAVAX contract to transfer tokens from this contract to itself during `requestUnlock` is unnecessary 5

[WP-L3] `NavErc20.nav()` Unexpectedly Reverts When `_nav` is Small 7

[WP-N4] The NatSpec documentation for `ASAVAXRateProvider` appears to be mistakenly copied from WAVAX documentation. 9

[WP-L5] Unnecessary precision loss and gas waste in `TAvaxExchangeRateProvider.getPooledAvaxByShares()` 10

[WP-M6] ChainlinkRateProvider fails to account for token decimals and updatedAt (whether the price data is stale) 12

Appendix **16**

Disclaimer **17**

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/tAVAX-protocol**
Commit	**4b9a9d0881ded6b966172c5f587041bd50a349b8**
Language	**Solidity**

Audit Summary

Delivery Date	**Aug 22, 2025**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**6**

[WP-M1] Current implementation cannot claim rewards from Aave's `Merit Programs` for asAVAX

Medium

Issue Description

Aave is currently offering rewards for asAVAX through its `Merit Programs` . The reward APR is 0.32% at the time of writing.

To receive `asAVAX` rewards, the contract needs to implement `Merit Programs` reward claiming in the action.

The current implementation only claims internal Aave rewards from the `RewardsController` but not the rewards from the Merit Programs.

> Merit Programs Third-party programs that provide extra APR rewards for specific lending activities. These require separate claiming through external platforms.

> Aave Governance Rewards Incentives set by Aave governance through the RewardsController, typically distributed in AAVE tokens or other approved reward tokens.

Status

ⓘ Acknowledged

[WP-L2] Approving the sAVAX contract to transfer tokens from this contract to itself during `requestUnlock` is unnecessary

Low

Issue Description

https://github.com/treehouse-gaia/tAVAX-protocol/blob/1ad0b0d012dd2a543bd7f67717d055a8fd3df2c0/contracts/strategy/actions/savax/SavaxUnlock.sol#L41-L54

```solidity
41  function _savaxUnlock(uint _amount) internal returns (uint amountUnlock, bytes
    memory logData) {
42      ISAVAX savax = ISAVAX(sAVAX);
43      amountUnlock = _amount;
44
45      // Check if we have enough sAVAX balance
46      uint balance = savax.balanceOf(address(this));
47      require(balance >= amountUnlock, 'Insufficient sAVAX balance');
48
49      // Approve sAVAX contract to transfer tokens from this contract to itself
    during requestUnlock
50      TokenUtils.approveToken(sAVAX, sAVAX, amountUnlock);
51      savax.requestUnlock(amountUnlock);
52
53      logData = abi.encode(sAVAX, amountUnlock);
54    }
```

```solidity
222 /**
223     * @notice Start unlocking cooldown period for `shareAmount` AVAX
224     * @param shareAmount Amount of shares to unlock
225     */
226 function requestUnlock(uint shareAmount) external nonReentrant whenNotPaused {
227     require(shareAmount > 0, "Invalid unlock amount");
228     require(shareAmount <= shares[msg.sender], "Unlock amount too large");
229
230     userSharesInCustody[msg.sender] =
    userSharesInCustody[msg.sender].add(shareAmount);
231     _transfer(msg.sender, address(this), shareAmount);
```

```

    userUnlockRequests[msg.sender].push(UnlockRequest(
        block.timestamp,
        shareAmount
    ));

    emit UnlockRequested(msg.sender, shareAmount);
}
```

Status

✓ **Fixed**

[WP-L3] `NavErc20.nav()` Unexpectedly Reverts When `_nav` is Small

Low

Issue Description

When `_nav` in `NavErc20.nav()` is so small that $\lfloor \frac{\_nav \times sAVAX.totalShares}{sAVAX.totalPooledAvax} \rfloor$ equals 0, `NavErc20.nav()` will unexpectedly revert.

For example, if the `_target` 's `_nav` is initially `0` , and someone transfers 1 wei of native token AVAX to `_target` before `NavErc20.nav()` executes, `NavErc20.nav()` L56 `sAVAX.getSharesByPooledAvax(_nav)` will unexpectedly revert due to the implementation in `sAVAX` L206.

The threshold of `_nav` that triggers the `Error("Invalid share count")` depends on how small the ratio $\frac{sAVAX.totalShares}{sAVAX.totalPooledAvax}$ is.

Currently, with `sAVAX.totalShares` at 12900365882030527603020882 and `sAVAX.totalPooledAvax` at 15753963803434694283510120, this issue only occurs when `_nav` equals 1.

`NavErc20WithDebt` has a similar issue.

```
  /**
   * @notice Calculate total NAV of native AVAX + ERC20 tokens for a target address
   * @param _target Address to calculate NAV for
   * @param _tokens Array of ERC20 token addresses
   * @return _nav Total NAV in sAVAX terms
   * @dev Returns raw sAVAX balance if no other assets, otherwise converts AVAX to sAVAX shares + raw sAVAX
   * @dev Silently ignores tokens with zero rates from the registry
   */
  function nav(address _target, address[] memory _tokens) external view returns (uint _nav) {
    _nav += _target.balance;

    uint wip;
    uint sAVAXBalance;
    for (uint i; i < _tokens.length; ++i) {
```

```
    wip = IERC20(_tokens[i]).balanceOf(_target);

    if (wip > 0) {
     if (_tokens[i] == address(sAVAX)) {
      sAVAXBalance = wip;
     } else {
      uint rate = RATE_PROVIDER_REGISTRY.getRateInAvax(_tokens[i]);
      if (rate > 0) {
       _nav += Math.mulDiv(rate, wip, 1e18);
      }
      // Note: Zero-rate tokens are silently ignored
     }
    }
   }
   if (_nav == 0) {
    _nav = sAVAXBalance;
   } else {
    _nav = sAVAX.getSharesByPooledAvax(_nav) + sAVAXBalance;
   }
  }
```

```
   /**
    * @return The amount of shares that corresponds to `avaxAmount` protocol-controlled AVAX.
    */
   function getSharesByPooledAvax(uint avaxAmount) public view returns (uint) {
       if (totalPooledAvax == 0) {
           return 0;
       }

       uint shares = avaxAmount.mul(totalShares).div(totalPooledAvax);
       require(shares > 0, "Invalid share count");

       return shares;
   }
```

Status

✓ Fixed

[WP-N4] The NatSpec documentation for `ASAVAXRateProvider` appears to be mistakenly copied from WAVAX documentation.

Issue Description

The implementation is not a ”Wrapped avax Rate Provider”, and there is no ”hardcoded value of 1e18”.

Also, `sAVAX` on L32 might be a copy-paste error, consider changing `sAVAX` to `asAVAX` .

https://github.com/treehouse-gaia/tAVAX-protocol/blob/1ad0b0d012dd2a543bd7f67717d055a8fd3df2c0/contracts/rate-providers/ASAVAXRateProvider.sol

```
     @@ 1,17 @@
18   import { ISAVAX } from '../interfaces/savax/ISAVAX.sol';
19
20   /**
21    * @title Wrapped avax Rate Provider
22    * @notice Returns the hardcoded value of 1e18
23    */
24   contract ASAVAXRateProvider is IRateProvider {
25     ISAVAX public immutable sAVAX;
26
27     constructor(ISAVAX _sAVAX) {
28       sAVAX = _sAVAX;
29     }
30
31     /**
32      * @return the value of sAVAX in terms of AVAX
33      */
34     function getRate() external view override returns (uint256) {
35       return sAVAX.getPooledAvaxByShares(1e18);
36     }
37   }
```

Status

✓ Fixed

[WP-L5] Unnecessary precision loss and gas waste in `TAvaxExchangeRateProvider.getPooledAvaxByShares()`

Low

Issue Description

Current implementation:

$$\left\lfloor \frac{tAVAX.convertToAssets(10^{18}) \times sAVAX.getPooledAvaxByShares(10^{18})}{10^{18}} \right\rfloor$$
$$= \left\lfloor \frac{tAVAX.convertToAssets(10^{18}) \times \left\lfloor \frac{10^{18} \times sAVAX.totalPooledAvax}{sAVAX.totalShares} \right\rfloor}{10^{18}} \right\rfloor$$

Simplified version:

$$\left\lfloor \frac{tAVAX.convertToAssets(10^{18}) \times sAVAX.totalPooledAvax}{sAVAX.totalShares} \right\rfloor$$

```
  /**
   * @return the value of tAVAX in terms of AVAX
   */
  function getRate() external view returns (uint256) {
    return (tAVAX.convertToAssets(1e18) * sAVAX.getPooledAvaxByShares(1e18)) /
1e18;
  }
```

```
    /**
     * @return The amount of AVAX that corresponds to `shareAmount` token shares.
     */
    function getPooledAvaxByShares(uint shareAmount) public view returns (uint) {
        if (totalShares == 0) {
            return 0;
        }

        return shareAmount.mul(totalPooledAvax).div(totalShares);
    }
```

Recommendation

Consider changing to something like:

```
  /**
   * @return the value of tAVAX in terms of AVAX
   */
  function getRate() external view returns (uint256) {
    return sAVAX.getPooledAvaxByShares(tAVAX.convertToAssets(1e18));
  }
```

Status

✓ Fixed

[WP-M6] ChainlinkRateProvider fails to account for token decimals and updatedAt (whether the price data is stale)

Medium

Issue Description

Based on the usage of `IRateProvider.getRate()` , the expected return value should be ”how many wei AVAX per 1 wei asset” multiplied by 10^{18}, i.e., $\frac{10^{18} \times avaxAmountInWei}{assetAmountInWei}$

The Chainlink price feed’s `answer` represents ”how many whole B tokens (i.e., $10^{bTokenDecimals}$ wei B tokens) one whole A token (i.e., $10^{aTokenDecimals}$ wei A token) is worth” multiplied by $10^{priceFeed.decimals}$, i.e.,

$$answer = \frac{10^{priceFeed.decimals} \times \frac{bTokenAmountInWei}{10^{bTokenDecimals}}}{\frac{aTokenAmountInWei}{10^{aTokenDecimals}}} = \frac{10^{priceFeed.decimals} \times bTokenAmountInWei \times 10^{aTokenDecima}}{aTokenAmountInWei \times 10^{bTokenDecimals}}$$

After transformation:

$$\frac{answer \times 10^{18} \times 10^{bTokenDecimals}}{10^{priceFeed.decimals} \times 10^{aTokenDecimals}} = \frac{10^{18} \times bTokenAmountInWei}{aTokenAmountInWei}$$

If B token is AVAX and A token is the asset token:

$$\frac{answer \times 10^{18} \times 10^{avaxDecimals}}{10^{priceFeed.decimals} \times 10^{assetDecimals}} = \frac{10^{18} \times avaxAmountInWei}{assetAmountInWei}$$

Conclusion

`ChainlinkRateProvider.getRate()`

- Expected result:

$$\frac{answer \times 10^{18} \times 10^{avaxDecimals}}{10^{priceFeed.decimals} \times 10^{assetDecimals}} = \frac{answer \times 10^{18} \times 10^{18}}{10^{priceFeed.decimals} \times 10^{assetDecimals}}$$

- Current implementation:

$$answer \times \_scalingFactor = answer \times \frac{10^{18}}{10^{priceFeed.decimals}}$$

  - Compared to the expected result, it's missing the $\frac{10^{18}}{10^{assetDecimals}}$ component

The current implementation will only work when `assetDecimals` happens to be 18.

Furthermore, `ChainlinkRateProvider.getRate()` directly uses the `answer` from `pricefeed.latestRoundData()` without validating the `updatedAt` return value, which could lead to using stale or inaccurate price data.

```
@@ 1,13 @@

pragma solidity ^0.8.0;

import '@chainlink/contracts/src/v0.8/shared/interfaces/AggregatorV3Interface.sol';
import '@chainlink/contracts/src/v0.8/interfaces/FeedRegistryInterface.sol';
import './IRateProvider.sol';

/**
 * @title Chainlink Rate Provider
 * @notice Returns a Chainlink price feed's quote for the provided currency pair
 * @dev This rate provider is a simplification of ChainlinkReistryRateProvider
which is fixed to a particular pricefeed.
 *      This is expected to be used in environments where the Chainlink registry
is not available.
 */
contract ChainlinkRateProvider is IRateProvider {
  AggregatorV3Interface public immutable pricefeed;

  // Rate providers are expected to respond with a fixed-point value with 18
decimals
  // We then need to scale the price feed's output to match this.
  uint256 internal immutable _scalingFactor;

  /**
   * @param feed - The Chainlink price feed contract
   */
  constructor(AggregatorV3Interface feed) {
    pricefeed = feed;
    _scalingFactor = 10 ** (18 - feed.decimals());
```

```solidity
  }

  /**
   * @return the value of the quote currency in terms of the base currency
   */
  function getRate() external view override returns (uint256) {
    (, int256 price, , , ) = pricefeed.latestRoundData();
    require(price > 0, 'Invalid price rate response');
    return uint256(price) * _scalingFactor;
  }
}
```

```solidity
  /**
   * @notice Returns the rate of an asset in avax terms
   * @param _asset token address, must be the base currency (not quote)
   * @return _rateInAvax the exchange rate in 1e18
   */
  function getRateInAvax(address _asset) external view returns (uint _rateInAvax) {
    if (_asset == WAVAX) return 1e18;

    if (rateProviders[_asset] == address(0)) revert RateProviderNotFound();
    _rateInAvax = IRateProvider(rateProviders[_asset]).getRate();
  }
```

```solidity
  /**
   * @notice Calculate total NAV of native AVAX + ERC20 tokens for a target address
   * @param _target Address to calculate NAV for
   * @param _tokens Array of ERC20 token addresses
   * @return _nav Total NAV in sAVAX terms
   * @dev Returns raw sAVAX balance if no other assets, otherwise converts AVAX to sAVAX shares + raw sAVAX
   * @dev Silently ignores tokens with zero rates from the registry
   */
  function nav(address _target, address[] memory _tokens) external view returns (uint _nav) {
    _nav += _target.balance;

    uint wip;
    uint sAVAXBalance;
    for (uint i; i < _tokens.length; ++i) {
      wip = IERC20(_tokens[i]).balanceOf(_target);

      if (wip > 0) {
        if (_tokens[i] == address(sAVAX)) {
          sAVAXBalance = wip;
        } else {
          uint rate = RATE_PROVIDER_REGISTRY.getRateInAvax(_tokens[i]);
          if (rate > 0) {
            _nav += Math.mulDiv(rate, wip, 1e18);
          }
          // Note: Zero-rate tokens are silently ignored
        }
      }
    }
    if (_nav == 0) {
      _nav = sAVAXBalance;
    } else {
      _nav = sAVAX.getSharesByPooledAvax(_nav) + sAVAXBalance;
    }
  }
```

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