Treehouse Labs

tETH Onchain/Offchain - Updates

Security Assessment Report

Version: 2.2

September, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of a set of Treehouse Labs smart contract and supporting offchain Typescript deployment script changes. The review focused solely on the security aspects of the Solidity implementation of the contract and Typescript code, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Treehouse Labs smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Treehouse Labs smart contracts.

Overview

Treehouse Protocol is a new restaking service launching with the liquid restaking token tETH. Built on top of other Liquid Staking Tokens such as Lido's stETH it aims to leverage opportunities such as Aave lending markets to outperform other LRTs and maximise staking yields delivered to end users.

This strategy is part of a wider Decentralised Offered Rates concept with an aim to converge onchain ETH interest rates. This review focused on the fundemental system underlying tETH and the initial strategy involving stETH and leveraged staking via Aave.

This engagement focuses exclusively on a set of changes introduced by Treehouse since Sigma Prime's previous review.

Security Assessment Summary

Scope

The review was conducted on the private Treehouse labs onchain repository. The scope of this time-boxed review was strictly limited to files at commit be100cb with particular focus on the diff between these files and commit 195a3ea, which was targeted in Sigma Prime's previous Treehouse tETH review.

This private repository at commit c1150db has been verified to be equivalent to the public repository at commit 459ccb1. From this point forward, all references will pertain to the public repository, though the testing and resolution commits refer to the private version these were originally documented on.

Note: third party libraries and dependencies, such as OpenZeppelin, were excluded from the scope of this assessment.

Approach

The manual review focused on identifying issues associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout).

Additionally, the manual review process focused on identifying vulnerabilities related to known Solidity anti-patterns and attack vectors, such as re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers.

For a more detailed, but non-exhaustive list of examined vectors, see [1, 2].

The manual review of the offchain Typescript deployment focused on identifying issues related to vulnerable dependancies, differences from anticipated setup and any issues relating to business logic.

To support this review, the testing team also utilised the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`
- Aderyn: `https://github.com/Cyfrin/aderyn`

Output for these automated tools is available upon request.

Coverage Limitations

Due to the time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 4 issues during this assessment. Categorised by their severity:

- Low: 1 issue.
- Informational: 3 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the changes in scope. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as "informational".

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
TREE2-01	Net Asset Calculations Vulnerable To stETH Depeg	Low	Resolved
TREE2-02	Strategies Cannot Forward Sent ETH	Informational	Resolved
TREE2-03	Redemption Fee Can Rise After Initiating Redemption	Informational	Closed
TREE2-04	Miscellaneous General Comments	Informational	Resolved

TREE2-01	Net Asset Calculations Vulnerable To stETH Depeg		
Asset	`NavHelper.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

When recording the system's Net Asset Value (NAV), `stETH` is treated as though it has a 1:1 peg with `ETH`. This can lead to inaccuracies in the accounting system should `stETH` depeg from this exchange rate.

In `getAaveV3Nav()`, the assumption can overstate net assets, which will lead to increased liquidation risks.

For `getLidoRedemptionNav()` and `getTokenNav()`, this pegging to `ETH` was a defensive assumption that `1 stETH = 1 ETH`, so that a depeg would not overvalue assets.

However, this defensive peg assumption could also negatively affect existing depositors in the event an accounting burn is triggered. This would lead to new depositors receiving a larger share of `tETH` than they deserve.

It is worth noting however that the likelihood of this is low as the market rate of `stETH` to `ETH` has never depegged further than `0.95 ETH` and the Treehouse team have made use of conservative borrowing parameters to decrease the risk of liquidation.

Recommendations

Make use of an external oracle system such as Chainlink to accurately capture the `stETH:ETH` ratio, rather than assuming a 1:1 peg.

Resolution

Net Asset Value calculations have been rewritten to adjust for the exchange rate of stETH to ETH using a Chainlink Oracle in commit c1150db.

TREE2-02	Strategies Cannot Forward Sent ETH
Asset	`Strategy.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

When a Strategy action is executed via `callExecute()`, no `msg.value` is passed to the `execute()` call. This means that if the action invoked relies on `ETH` sent with the call, it may revert or fulfil with zero `ETH` transferred. If the call succeeds, then any `ETH` sent to the original `callExecute()` function would be stuck in the `Strategy` contract.

This situation occurs because `callExecute()` performs an external call to `IStrategy(address(this)).execute(_target, _data);` in order to alter the calling context. As a result, fields such as `msg.sender` and `msg.value` are altered from those used in `callExecute()`. As `callExecute()` is marked as `payable`, it is possible to send `ETH` while calling the function, but not possible to use that `ETH`.

Note, this finding has been marked as informational as currently no actions make use of sending `ETH` and the `StrategyExecutor` which calls `callExecute()` does not support sending `ETH`.

Recommendations

If sending of `ETH` is not envisioned in future strategy actions, one possible solution would be to remove the `payable` modifier from `callExecute()` and `execute()`.

If `ETH` might be sent with strategy actions in the future, then altering `callExecute()` to forward all `msg.value` to `execute()` would be required.

Resolution

In commit ee11f45 the ability to forward the ETH attached to a call was added to both `StrategyExecutor.executeOnStrategy()` and `Strategy.execute()`.

TREE2-03	Redemption Fee Can Rise After Initiating Redemption
Asset	`TreehouseRedemption.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

Users who wish to redeem their deposit from the Treehouse Protocol can do so using `TreehouseRedemption`, which provides a two-step process to unlock assets from the vault strategy and then redeem them.

During the second stage, a redeeming user is charged a redemption fee that is set by the Treehouse Admin and can be as high as 100% of the withdrawal. Once the redemption has been initialised by calling `redeem()`, there is no mechanism to cancel it and the admin can arbitrarily alter the redemption fee charged.

This is because the redemption fee is only processed on calling `finalizeRedeem()`, which is the second stage of a redemption.

While the redemption fee could be abused by the admin to steal assets from exiting users, this is unlikely as the admin role is held by Treehouse team only and the majority of users would exit via the `tETH` liquidity pool rather than redemptions.

Note, this issue has been rated informational as the Treehouse team have communicated it is the intended design, as discussed in the resolution section.

Recommendations

One possible solution would be to alter the redemption process to record the redemption fee when a user initiates a redemption with `redeem()`, rather than when they finalise with `finalizeRedemption()`.

Also, to prevent the admin frontrunning calls to `redeem()` with a new redemption fee, the `redeem()` function should include a maximum acceptable fee that the user can specify. Then, if the redemption fee was higher than this, the call to `redeem()` would revert, preventing user funds from becoming locked in redemption.

Alternatively, allow a user to cancel a pending redemption. However, this would require substantial codebase changes and open a new avenue for griefing the vault asset allocation.

Resolution

The Treehouse team have noted that the purpose of the `redemptionFee` is to pass on borrow costs to the redeemer.

This is because redemptions are only intended for large withdrawals and these large withdrawals may have a meaningful impact on vault borrowing rates and reduced `wstETH` yield while being unwound.

For this reason, the issue will not be fixed as predicting what fee to charge redeeming end users is difficult due to dynamic costs involved.

TREE2-04	Miscellaneous General Comments
Asset	All contracts
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Ownership Not Transferred Immediately To Multisig**

   Related Asset(s): ignition/modules/ProtocolPhaseOne.ts

   After deployment, ownership of contracts should be transferred from the deployment externally owned account to the Treehouse multisig contract for security reasons, but this has not been implemented in the deployment script yet.

   Ensure the ownership transfer is added to the deployment script prior to use or perform rigourous checks after protocol deployment to ensure the ownership is manually transferred to the multisig account for all deployed contracts.

2. **Open TODO**

   Related Asset(s): ProtocolPhaseTwo.ts

   Phase Two deployment script has open `TODO`s for the `TREASURY` and `ACCOUNTING_EXECUTOR` addresses.

   ```
   const TREASURY = deployer //TODO
   const ACCOUNTING_EXECUTOR = deployer //TODO
   ```

   Assign relevant addresses to treasury and executor.

3. **Typos**

   Related Asset(s): /*

   Typos were noticed in the following areas:

   - line [44] of `VaultPull.sol` *"aspproved"* should read *"approved"*.

   Correct typos to improve code comment readability.

4. **Redundant Code**

   Related Asset(s): Vault.sol, TreehouseRedemption.sol

   Some checks in the code are redundant:

   - line [94] of `Vault.sol` has a condition that is already checked on line [87].
   - The check on line [122] of `TreehouseRedemption.sol` is redundant as constraints in `_getReturnAmount()` mean this error can never be triggered.

   Consider removing redundant code to reduce code complexity.

5. **Magic Numbers**

   Related Asset(s): NavHelper.sol, TreehouseRouter.sol

   Some contracts contain hardcoded numbers or addresses which can make updating the codebase take longer and risk introducing errors by update omission.

   Noted cases are on line [**109**] of `TreehouseRouter.sol` as well as line [**70**] and line [**161**] of `NavHelper.sol`.

   Replace instances of hardcoded numbers or addresses with named constants.

6. **Unclear Variable Names**

   Related Asset(s): TreehouseRedemption.sol, TAsset.sol

   Some variables have unclear names, particularly the arguments given in `TreehouseRedemption._GetReturnAmount()` and `_underlying`, a constructor argument in `TAsset` given it is refering to the InternalAccountUnit but `UNDERLYING` refers to `wstETH`.

   It is recommended to use clearer variable names to create less confusion in future development work.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The comments above have been acknowledged by the development team, and relevant changes actioned in commits 9fdf2e1 and 6e21ef2 where relevant.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `forge` framework was used to perform these tests and the output is given below.

```
Ran 12 tests for test/tests-local/TreehouseAccounting.t.sol:TreehouseAccountingTest
[PASS] testFail_setFee_unauthorized() (gas: 12683)
[PASS] testFail_updateExecutor_unauthorized() (gas: 12804)
[PASS] testFail_updateTreasury_unauthorized() (gas: 12817)
[PASS] test_fullOwnershipTransfer() (gas: 34162)
[PASS] test_mark_burn() (gas: 57215)
[PASS] test_mark_mint() (gas: 156467)
[PASS] test_mark_unauthorized() (gas: 15825)
[PASS] test_setFee() (gas: 20564)
[PASS] test_setFee_tooHigh() (gas: 13301)
[PASS] test_setup() (gas: 33910)
[PASS] test_updateExecutor() (gas: 20778)
[PASS] test_updateTreasury() (gas: 20887)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 4.12ms (826.78µs CPU time)

Ran 8 tests for test/tests-local/ActionRegistry.t.sol:ActionRegistryTest
[PASS] testAddNewContract() (gas: 40410)
[PASS] testAddNewContractFailsForExistingEntry() (gas: 39958)
[PASS] testApproveContractChange() (gas: 73699)
[PASS] testCancelContractChange() (gas: 52797)
[PASS] testNonExistentEntryOperations() (gas: 36001)
[PASS] testRevertToPreviousAddress() (gas: 76080)
[PASS] testRevertToPreviousAddressFailsWithNoPreviousAddress() (gas: 42199)
[PASS] testStartContractChange() (gas: 67065)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 4.58ms (589.33µs CPU time)

Ran 10 tests for test/tests-local/Blacklistable.t.sol:BlacklistableTest
[PASS] testFail_updateBlacklister_notOwner() (gas: 17757)
[PASS] test_blacklist() (gas: 45291)
[PASS] test_blacklist_notBlacklister() (gas: 18463)
[PASS] test_isBlacklisted() (gas: 47570)
[PASS] test_transfer_blacklistedRecipient() (gas: 156276)
[PASS] test_transfer_blacklistedSender() (gas: 154151)
[PASS] test_unBlacklist() (gas: 36500)
[PASS] test_unBlacklist_notBlacklister() (gas: 18422)
[PASS] test_updateBlacklister() (gas: 25746)
[PASS] test_updateBlacklister_zeroAddress() (gas: 18526)
Suite result: ok. 10 passed; 0 failed; 0 skipped; finished in 4.58ms (633.96µs CPU time)

Ran 7 tests for test/tests-local/tETH.t.sol:tETHTest
[PASS] test_decimals() (gas: 15772)
[PASS] test_deposit() (gas: 197380)
[PASS] test_mint() (gas: 197423)
[PASS] test_redeem() (gas: 185858)
[PASS] test_transfer() (gas: 158484)
[PASS] test_transferOwnership() (gas: 39169)
[PASS] test_withdraw() (gas: 189215)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 4.75ms (1.31ms CPU time)

Ran 9 tests for test/tests-local/StrategyExecutor.t.sol:StrategyExecutorTest
[PASS] testFail_actionExecutor_executeActions_NoAccessControl() (gas: 22575)
[PASS] testFail_vaultPull_executeAction_NoAccessControl() (gas: 10902)
[PASS] test_executeNonWhitelistedAction() (gas: 672430)
[PASS] test_executeOnStrategy_VaultPull() (gas: 116209)
[PASS] test_executeOnStrategy_VaultPull_pause() (gas: 141588)
[PASS] test_executeOnStrategy_VaultPull_whitelistActions() (gas: 169600)
[PASS] test_executeOnStrategy_VaultPull_whitelistAssets() (gas: 206647)
[PASS] test_executeOnStrategy_VaultSend() (gas: 117038)
[PASS] test_setStrategyExecutor() (gas: 22385)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 4.99ms (2.13ms CPU time)
```

```
Ran 18 tests for test/tests-local/IAU.t.sol:IAUTest
[PASS] testFail_unauthorizedAddMinter() (gas: 17141)
[PASS] testFail_unauthorizedBurn() (gas: 12818)
[PASS] testFail_unauthorizedBurnFrom() (gas: 15032)
[PASS] testFail_unauthorizedMint() (gas: 13055)
[PASS] testFail_unauthorizedRemoveMinter() (gas: 17155)
[PASS] testFail_unauthorizedSetTimelock() (gas: 14990)
[PASS] testFail_unauthorizedTransfer() (gas: 17714)
[PASS] test_addMinter() (gas: 90039)
[PASS] test_burn() (gas: 34104)
[PASS] test_burnFrom() (gas: 43696)
[PASS] test_fullOwnershipTransfer() (gas: 34320)
[PASS] test_getMinters() (gas: 23346)
[PASS] test_mintTo() (gas: 52740)
[PASS] test_removeMinter() (gas: 75210)
[PASS] test_setTimelock() (gas: 20288)
[PASS] test_timelock() (gas: 12825)
[PASS] test_timelock_fullOwnershipTransfer() (gas: 36090)
[PASS] test_transfer() (gas: 64042)
Suite result: ok. 18 passed; 0 failed; 0 skipped; finished in 5.00ms (785.03µs CPU time)

Ran 12 tests for test/tests-local/TreehouseRedemption.t.sol:TreehouseRedemptionTest
[PASS] test_constructor() (gas: 27665)
[PASS] test_finalizeRedeem() (gas: 265749)
[PASS] test_finalizeRedeem_inWaitingPeriod() (gas: 199560)
[PASS] test_finalizeRedeem_insufficientFunds() (gas: 222310)
[PASS] test_fullOwnershipTransfer() (gas: 34358)
[PASS] test_getPendingRedeems() (gas: 264844)
[PASS] test_getRedeemInfo() (gas: 197068)
[PASS] test_getRedeemLength() (gas: 260415)
[PASS] test_redeem_belowMinimum() (gas: 67767)
[PASS] test_setMinRedeem() (gas: 20676)
[PASS] test_setPause() (gas: 21788)
[PASS] test_setWaitingPeriod() (gas: 20622)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 4.98ms (1.77ms CPU time)

Ran 8 tests for test/tests-local/TreehouseRouter.t.sol:TreehouseRouterTest
[PASS] test_constructor() (gas: 38360)
[PASS] test_deposit() (gas: 34093)
[PASS] test_deposit_stETH() (gas: 614120)
[PASS] test_deposit_wstETH() (gas: 441064)
[PASS] test_mark_burn_frontrun() (gas: 483547)
[PASS] test_setDepositCap() (gas: 25109)
[PASS] test_setPause() (gas: 34284)
[PASS] test_share_inflation() (gas: 668772)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 6.17ms (2.34ms CPU time)

Ran 1 test for test/tests-fork/PnlAccountingHelper.t.sol:PnlAccountingHelperTest
[PASS] test_setup() (gas: 31743)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 806.77ms (101.48µs CPU time)

Ran 6 tests for test/tests-local/Vault.t.sol:VaultTest
[PASS] test_AddRemoveAllowableAsset() (gas: 62740)
[PASS] test_GetAllowableAssetCount() (gas: 78860)
[PASS] test_GetAllowableAssets() (gas: 90078)
[PASS] test_SetRedemption() (gas: 67296)
[PASS] test_SetStrategyStorage() (gas: 20850)
[PASS] test_Withdraw() (gas: 225140)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 812.98ms (2.07ms CPU time)

Ran 4 tests for test/tests-fork/TreehouseRouter.t.sol:TreehouseRouterTest
[PASS] test_depositETH() (gas: 645649)
[PASS] test_deposit_fuzz(uint256[15],uint256[15]) (runs: 1002, µ: 2057248, ~: 2055812)
[PASS] test_deposit_wETH() (gas: 893647)
[PASS] test_full_cycle() (gas: 1396768)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 6.65s (5.85s CPU time)

Ran 11 test suites in 6.65s (8.31s CPU time): 95 tests passed, 0 failed, 0 skipped (95 total tests)
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

	High	Medium	High	Critical
Impact	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: `https://blog.sigmaprime.io/solidity-security.html`. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: `http://www.dasp.co/`. [Accessed 2018].