Treehouse Labs

Treehouse tETH OnChain/OffChain

Security Assessment Report

Version: 2.2

August, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Treehouse Labs smart contracts and supporting offchain Python scripts. The review focused on the security aspects of the Solidity implementation of the contracts and Python 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 and scripts 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.

The offchain review section focused on the Python bot tasked with managing DeFi positions for Treehouse to ensure they meet collateral requirements for Aave loans and can process user withdrawals in a timely manner.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the Treehouse Labs onchain and Treehouse Labs offchain repositories.

The scope of this time-boxed review was strictly limited to files at commit c00db74 for onchain contracts and c6a6a11 for offchain scripts.

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) for onchain contracts.

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 for the offchain Python scripts focused on identifying issues related to business logic implementation of the scripts including data validation, external input sanitation, vulnerable dependencies and Denial-of-Service.

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`
- Safety: `https://github.com/pyupio/safety/`

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 13 issues during this assessment. Categorised by their severity:

- Medium: 2 issues.
- Low: 4 issues.
- Informational: 7 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Treehouse Labs smart contracts and offchain scripts. 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
TREE-01	ERC4626 Vault Share Inflation	Medium	Resolved
TREE-02	Lido Hurdle Rate Calculation Timing Can Vary Admin Fee	Medium	Closed
TREE-03	Vulnerable Dependencies	Low	Resolved
TREE-04	No Checks Prior To Executing A Subprocess Call	Low	Closed
TREE-05	Accounting Losses Can Be Frontrun By Redemptions	Low	Resolved
TREE-06	Redemption Requests Can Exceed Block Gas Limit	Low	Closed
TREE-07	Code Quality Improvements	Informational	Closed
TREE-08	stETH Deposit Recording Inaccuracies	Informational	Closed
TREE-09	Accounting For Vault Gains Vulnerable To Frontrunning	Informational	Closed
TREE-10	Rate Providers Cannot Be Reset	Informational	Closed
TREE-11	Future Liquid Staking Token Additions	Informational	Closed
TREE-12	Arbitrary Asset Removal	Informational	Closed
TREE-13	Miscellaneous General Comments	Informational	Closed

TREE-01	ERC4626 Vault Share Inflation		
Asset	`TreehouseRouter.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

A malicious user may frontrun an accounting update when the protocol is empty to steal assets from another user.

The share inflation attack, common to ERC4626 implementations, is possible when the system posts a profit after having had a user deposit. The malicious user notices the profit accounting and frontruns the accounting transaction with a `redeem()` request of `original deposit - 1` size. Next, when another user deposits, if their deposit is half or less than the profit posted, they will lose their deposit as they are allocated no shares.

Given the need for the system to post a profit after one user depositing with no other current users of the system, the likelihood of this issue occurring has been rated as low.

Recommendations

Protection against users frontrunning accounting updates, such as using an offchain solution like Flashbots to avoid transactions being seen in the mempool, would prevent this issue from occurring.

The Treehouse team should also fine-tune the constants used in ERC4626, such as `_decimalOffset()`, to make this attack less profitable and even more unlikely to occur. A small sacrificial deposit could be placed with the creation of the protocol to enforce the ratio of assets to shares, as seen in discussion here.

Resolution

`TreehouseRouter.deposit()` will revert when no shares are minted.

This issue has been addressed in commit d34c3b3.

TREE-02	Lido Hurdle Rate Calculation Timing Can Vary Admin Fee		
Asset	`LidoAPR.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

If updates to the Lido Hurdle rate are delayed for multiple days, this can lead to an increase in the admin fee charged by Treehouse for exceeding the Lido hurdle rate.

This issue arises due to the compounding effect of multiple updates being processed at the same time and no time aspects being considered during the hurdle rate calculation.

The likelihood of this occurring is low as the Treehouse team have a dedicated offchain bot that processes accounting updates daily, therefore it would only be likely if very high gas fees were sustained for multiple days and caused automated actions to revert.

Recommendations

The simplest solution is to ensure the accounting is regularly carried out daily, with monitoring on the offchain bot to ensure transaction completion.

Alternatively, the team could devise a new accounting system that is consistent in generating fees regardless of how many update cycles have occurred.

Resolution

The issue was acknowledged by the project team with the following comment:

> *"We have sufficient checks in place to ensure accounting is carried out daily"*

TREE-03	Vulnerable Dependencies		
Asset	`tETH-offchain/*`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The following vulnerable dependencies have been identified to be in use:

- `certifi==2024.6.2` - Four (4) known vulnerabilities
- `idna==3.4` - One (1) known vulnerability
- `requests==2.31.0` - Two (2) known vulnerabilies
- `aiohttp==3.8.6` - Number of known bugs and missing features
- `eth-abi==4.2.1` - Two (2) known vulnerabilities
- `pycryptodome==3.19.0` - Number of known bugs and missing features

The vulnerabilities in the above dependencies range from information disclosure, denial-of-service conditions, to man-in-the-middle attacks. For more information, refer to the individual advisories issued for each of the packages.

Recommendations

Update identified dependencies to their latest available versions.

Resolution

This issue has been addressed in commit d8ba5d7.

TREE-04	No Checks Prior To Executing A Subprocess Call		
Asset	`tETH-offchain/src/env_handler/fork_blockchain.py`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

In `tETH-offchain/src/env_handler/fork_blockchain.py:38-42` a call is made to run `anvil_end.sh` and `anvil_start_fork.sh` standalone scripts, however there are no checks made prior to the execution to ensure the files exist and have not been modified.

If the files do not exist at their expected paths, execution of the `fork_blockchain.py` will simply fail unexpectedly.

If the files do exist, but have been modified by a malicious actor with relevant permissions, the execution will complete successfully without notifying the user that the scripts' behaviour may have been modified. This could be used by malicious actors to elevate privileges or establish persistence on the local system.

Recommendations

Implement additional checks prior to executing standalone scripts to ensure the referenced files exist and have not been modified. This can be achieved by simply verifying the file hash before execution.

Resolution

The issue was acknowledged by the project team with the following comment:

> *"Treehouse team has opted in for a process improvement to run a diff check with the known, good commit hash, to ensure any of the files being run have not been modified from their intended versions prior to execution."*

TREE-05	Accounting Losses Can Be Frontrun By Redemptions		
Asset	`TreehouseAccounting.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

Daily accounting updates can be blocked by a user frontrunning the request and initiating a withdrawal, resulting in the `doAccounting()` call reverting. Users can also utilise this strategy to avoid daily losses if they are sustained by the system.

This happens because `doAccounting()` relies on burning iETH present in the `tETH` contract to adjust for a protocol loss. If a user calls `TreehouseRedemption.redeem()` to start the redemption process, they return their tETH tokens and iETH is transferred from the `tETH` contract to the `TreehouseRedemption` contract and can reduce the balance of iETH in `tETH` below the amount needed for the accounting burn.

However, the likelihood of this occurring is low, as accounting is updated daily and therefore the recorded loss can at most match the `maxPnl()` figure. This means the issue can only occur when the redemption represents a large portion of the total iETH supply and the system posts a daily loss at the same time.

Recommendations

There are multiple possible solutions to this issue:

- Make use of an offchain solution such as Flashbots to prevent accounting updates from being visible in the mempool. This means users will not be able to frontrun such transactions. However if accounting transactions happen consistently in the 1-2am UTC window as suggested by the Treehouse team, it is likely you would still get some predictive frontrunning based off expected results an hour or so beforehand.
- Share losses with pending redemptions so that calling `redeem()` does not lock in the exchange rate used when a redemption is finalised.

Resolution

Shares are now redeemed on finalization, not start. Amount redeemed is changed to taking the min of starting and ending underlying assets returned. This ensures that losses are attributed to redeemers, but not profits. Excess iETH is then transferred back into the tETH vault.

This issue has been addressed in commit d34c3b3.

TREE-06	Redemption Requests Can Exceed Block Gas Limit		
Asset	`PnlAccountingHelper.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

If too many Lido redemptions are started, it can become impossible for the daily accounting call to be made, which would lead to the system accounting being wrong and profits or losses not being recorded or attributed correctly.

When triggering daily accounting via `doAccounting()`, it is important that the Treehouse team provides a full list of `requestIds` for all pending Lido redemptions. This list has no maximum length and so can cause the call to `doAccounting` to revert due to exceeding the block gas limit. While the team can elect to not provide a full `requestIds` list, this then results in incorrect accounting and could to lead to incorrectly reported losses.

This behaviour was only seen with a very long redemption list of over 100 pending redemptions and given that the redemptions can only be triggered by the Treehouse team, it is unlikely to occur in practice.

Recommendations

Add a limit to the maximum number of Lido withdrawal requests that can be pending at any one time or use offchain metrics to monitor list length.

Resolution

The issue was acknowledged by the project team with the following comment:

> *"If such a case were to eventuate, however improbable, we will push a new PnlAccountingHelper to address it."*

TREE-07	Code Quality Improvements
Asset	`tETH-offchain/*`
Status	**Closed:** See Resolution
Rating	Informational

Description

The following general code quality improvement opportunities have been identified:

- `env_handler/fork_blockchain.py:38` : `subprocess.run` used without explicitly defining the value for `check` . If `check` is set to `True` and the exit code was non-zero, it raises a `CalledProcessError` .
- `env_handler/fork_blockchain.py:40` : Consider using `with` for resource-allocating operations. Resource de-allocation automatically happens with use of `with` .
- `execution_handler/analyze_execution.py:55` : Raising too general an exception with `Exception` .
- `execution_handler/analyze_execution.py:69,110` : Unnecessary `else` after `return` . Remove the `else` .
- `execution_handler/execution.py:91` : Raising too general an exception with `Exception` .
- `execution_handler/execution.py:567` : Unused variable `txn_inputs` .
- `lido_withdraw_handler/query_lido_withdraw.py:117` : Do not use `len(request_ids)` without comparison to determine if a sequence is empty. Change it to `if len(weth_requests) == 0:` .
- `redemption_handler/earmarked.py:44` : Do not use `len(weth_requests)` without comparison to determine if a sequence is empty. Change it to `if len(request_ids) == 0:` .
- `simulate_action.py:35` : Raising too general exception `Exception` .
- `simulation_handler/offchain.py:39` : Raising too general an exception with `Exception` .
- `state_handler/generate_state.py:342` : Raising too general an exception with `Exception` .
- `state_handler/generate_state.py:447` : Consider iterating the dictionary directly instead of calling `.keys()` , i.e. using `assert k in pool_info` is slightly quicker.
- The functions `calc_aavev3_u_level()` , `calc_aavev3_borrow_rate()` and `calc_aavev3_supply_rate()` should handle the unlikely scenario where there is division by zero, similar to the other functions in `calc_func.py` .

Recommendations

Implement recommendations above as seen fit.

Resolution

The comments above have been acknowledged by the development team, and relevant changes actioned in d8ba5d7 where relevant.

TREE-08	stETH Deposit Recording Inaccuracies
Asset	`TreehouseRouter.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

Lido's mechanism for accounting with stETH advises using their share system due to the rebasing nature of stETH. In addition to this, currently stETH is valued equally to ETH when depositing. If future strategies add support for other LSTs, then it is advisable to make use of a price oracle to prevent abuse of the Treehouse system in the event of a black swan event occurring on one collateral token.

For example, if the protocol supports two collaterals stETH and rETH, and a bug in stETH's code leads to it no longer being valued at at 1 ETH, then some users may deposit stETH into Treehouse in an attempt to have a claim at the more valuable rETH backing it.

Recommendations

Be aware of accounting differences that may arise due to not using stETH's share system and plan accordingly for new collateral integrations, including checking formerly sound collateral assumptions.

Resolution

The issue was acknowledged by the project team.

TREE-09	Accounting For Vault Gains Vulnerable To Frontrunning
Asset	`PnlAccountingHelper.sol, Vault.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

If ETH funds are received directly by the Vault, it is possible for a user to frontrun their inclusion to the accounting system by depositing assets and, in turn, receive a proportion of these funds.

Direct ETH deposits to the vault do not mint iETH immediately and instead are accounted for when the system next runs `PnLAccountingHelper.doAccounting()`. This means that any user depositing assets between the direct ETH deposit and the call of `doAccounting()` will be allocated a share of these ETH funds despite not having been a depositor when the funds arrived.

This issue is rated as informational because there is no current mechanisms that make use of the Vault receiving ETH directly and so should not occur in practice.

Recommendations

If the direct ETH deposit to the Vault is intended to be used, the Treehouse team should ensure that is it called atomically with the `doAccounting()` update function to prevent theft of funds.

Making use of Flashbots to prevent the transaction being visible in the mempool would also prevent users from entering just before the direct deposit and leaving as soon as possible afterwards.

Resolution

The issue was acknowledged by the project team.

TREE-10	Rate Providers Cannot Be Reset
Asset	`Vault.sol, RateProviderRegistry.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

When an allowable asset is removed via `Vault.RemoveAllowableAsset()`, the rate provider of the asset is not reset.

This means that if the asset is added again in the future via `vault.addAllowableAsset()`, then the `checkHasRateProvider` check on line [**148**] is weakened as it will always pass.

If a rate provider is deprecated, this may cause problems such as a stale price feed or reverts.

Recommendations

When an allowable asset is removed, delete the associated asset rate provider or allow setting it to a special address denoting an unset provider.

Resolution

The issue was acknowledged by the project team with the following comment:

> *"RateProviders may be updated by calling RateProviderResistry.update(). RPs are meant to be an objective reference throughout the protocol and check is to enforce that any allowable assets added will have a rate provider."*

TREE-11	Future Liquid Staking Token Additions
Asset	`/*`
Status	**Closed:** See Resolution
Rating	Informational

Description

The Treehouse team have outlined that future updates will likely include support for different Liquid Staking Tokens (LSTs). Each LST is designed by a different team and so take different approaches to accounting and profit distribution.

These different approaches can make integrating multiple LSTs a challenging process. For example, Origin's OETH only receives staking rewards if the smart contract holding it opts in.

Recommendations

Careful research must be undertaken when expanding the current system to include other LSTs. It is recommended to check new integrations thoroughly with the token's developers to ensure all existing systems remain suitable.

Resolution

The issue was acknowledged by the project team.

TREE-12	Arbitrary Asset Removal
Asset	`Rescuable.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

As part of the intended design, contracts `Converter.sol`, `TreehouseRedemption.sol` and `TreehouseRouter.sol` all inherit `Rescuable.sol`, allowing the admin to withdraw any Ether or ERC20 tokens from these contracts.

Recommendations

While only intended as a fail-safe feature, the team should make end users aware of this functionality.

Ensure the `_rescuer` address is a multi-sig and is isolated correctly to reduce risk of malicious behaviour.

Resolution

The issue was acknowledged by the project team.

TREE-13	Miscellaneous General Comments
Asset	All contracts
Status	**Closed:** See Resolution
Rating	Informational

Description

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

1. **Execution Of Gnosis Trasactions Need To Be Refactored For Production**

   Related Asset(s): `execution.py`

   In the testing environment, for Gnosis transactions, `execution.execute_sequence()` dumps the orders to JSON rather than executing them immediately.

   When this function is implemented for production, it is necessary to refactor `send_order.py` so that it waits a sufficient amount of time after calling `execute_sequence()` before performing `analyze_state()` as the state will be incorrect if not all relevant transactions have been confirmed.

   Implement the suggestions above as seen fit.

2. **Ineffective Monitoring Of Lido Stake Rate**

   Related Asset(s): `analyze_state.py`

   The function `analyze_state.decide_rebal_target()` decides, based on certain metrics, whether rebalancing is necessary.

   One of the conditions, which sets rebalancing to true, occurs when the Lido stake rate falls below the `LVL_3` threshold of 1.05. The purpose of this condition is so that profit margins are monitored so that it does not fall below the configured threshold.

   ```
   if lido_stake_rate_alert == f"{LVL_3}_below_lower":
       need_rebal = True
       sta_logger.info("Lido stake rate is below lower bound.")
   ```

   However, this metric cannot be resolved by rebalancing as it is not part of the constraints of the rebalancing model. If this is the only condition which triggered rebalancing, it may result in unnecessary onchain transactions.

   If the intention is to provide a warning system to trigger potential manual intervention when the profit margin becomes too thin, consider logging a `warning` rather than `info` similar to when exit strategy is triggered.

   ```
   if exit_strat:
       need_rebal = True
       sta_logger.warning(
           "Got exit_strategy signal from price analysis, please adjust the config and rebalance."
       )
   ```

3. **Arithmetic Equations Readability Improvements**

   Related Asset(s): `tETH-offchain/utils/calc_func.py`, `tETH-offchain/utils/rebal_func.py`

   Consider improving readability of arithmetic equations by implementing parenthesis to explicitly indicate order of operations.

   Otherwise, Python will utilise PEMDAS (parenthesis, exponents, multiplication and division, addition and subtraction), in order from left to right for multiplication / division and addition / subtraction.

   The following expressions have been identified as candidates for readability improvements (note, the list below is not exhaustive):

- `tETH-offchain/utils/calc_func.py` - lines `27`, `57`, `113`, `153`, `205`, `229`.
- `tETH-offchain/utils/rebal_func.py` - lines `623`, `692`, `706`.

Introduce parenthesis in expressions to ensure improved readability.

4. **Hardcoded API Key**

   Related Asset(s): `tETH-offchain/config/thirdparty_config/strat_config_thirdparty.yml`

   API keys are hardcoded in `BLOCKSCAN_API_KEYS` parameter and stored in plaintext in the config file.

   Note, from the code comment it appears that the team is aware of it, so this finding is merely a reminder to address it before deployment in production.

   Do not store sensitive information in plaintext config files.

5. **Make Use of Lido Custom Aave V3 Market**

   Related Asset(s): `strategy/actions/aaveV3/helpers/MainnetAaveV3Addresses.sol`

   Aave have recently introduced a new market that is specially designed for leveraged lending ETH against wstETH. This market has safer parameters due to being fine-tuned for a single purpose and therefore is better suited for use by Treehouse with tETH.

   It is recommended to add the new Lido Aave V3 market as another strategy with the same functionality.

6. **Addresses Not Initialised On Construction**

   Related Asset(s): `Vault.sol`, `LidoAPR.sol`, `PnlAccountingHelper.sol`,

   Various contracts have addresses that are required for operation which are not set immediately on construction:

   - `Executor` in `LidoAPR.sol`
   - `Executor` in `PnlaAccountingHelper.sol`
   - `strategyStorage` in `Vault.sol`

   Ensure these roles are initialised by calling the correct setters immediately after construction.

7. **Update Logic Not Included In Constructor**

   Related Asset(s): `LidoAPR.sol`

   A zero check performed in `updateShareRate()` is missing from the constructor when initializing the value.

   Add the same zero rate check to the constructor for consistency.

8. **Asset Tracking Behaviours**

   Related Asset(s): `NavHelper.sol`, `PnlAccountingHelper.sol`

   Numerous different approaches are used to list assets related to vaults and strategies.

   For example, `PnlAccountingHelper.getNavOfStrategy()` hardcodes the tokens it considers relevant for calculating the net asset value to WETH and wstETH. Meanwhile, `NavHelper.getTokensNav()` will always record the value of a contract's ETH despite it not being specified in the list of tokens requested.

   While no direct impact has been found thus far, as the system intends to expand to more assets and strategies in future, it is advisable to adopt a uniform system for gathering relevant assets.

9. **Confusing Deposit Event Emission**

   Related Asset(s): `TreehouseRouter.sol`

   The router only rejects zero deposits when made using the stETH token, deposits using ETH, wETH or wstEth all succeed when the deposit amount is zero. This behaviour is inconsistent and can lead to confusing `Deposited` events being emitted that may break third party offchain tracking.

   Adopt a universal rejection of zero amount deposits.

10. **Magic numbers**

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

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

    Replace instances of hardcoded numbers or addresses with named constants.

11. **Asset Addresses Defined Multiple Times**

    Related Asset(s): ***strategy/libs/TokenUtils.sol, strategy/actions/lido/helpers/MainnetLidoAddresses.sol, TreehouseR***

    Several token addresses are defined in multiple contracts, this increases developer overhead when updating these addresses and can lead to bugs in the event where one contract is not updated.

    For contracts that are needed by multiple contracts, it is best these are defined in one contract and then inherited by all contracts using them.

12. **Equality To Boolean Checks**

    Related Asset(s): ***TreehouseRouter.sol, Vault.sol***

    In several places a boolean return condition is checked for equality to a boolean as follows:

    ```
    f(x) == false
    ```

    This is unnecessary as this can be replaced with the return of the left hand side of the assignment only.

    Remove redundant uses of equality to booleans in the files noted.

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 commit d8ba5d7 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 8 tests for test/tests-local/ActionRegistry.t.sol:ActionRegistryTest
[PASS] testAddNewContract() (gas: 40358)
[PASS] testAddNewContractFailsForExistingEntry() (gas: 39966)
[PASS] testApproveContractChange() (gas: 73647)
[PASS] testCancelContractChange() (gas: 52776)
[PASS] testNonExistentEntryOperations() (gas: 35945)
[PASS] testRevertToPreviousAddress() (gas: 76076)
[PASS] testRevertToPreviousAddressFailsWithNoPreviousAddress() (gas: 42163)
[PASS] testStartContractChange() (gas: 67017)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 4.49ms (726.75µs CPU time)

Ran 12 tests for test/tests-local/TreehouseAccounting.t.sol:TreehouseAccountingTest
[PASS] testFail_setFee_unauthorized() (gas: 12727)
[PASS] testFail_updateExecutor_unauthorized() (gas: 12804)
[PASS] testFail_updateTreasury_unauthorized() (gas: 12839)
[PASS] test_fullOwnershipTransfer() (gas: 34100)
[PASS] test_mark_burn() (gas: 57234)
[PASS] test_mark_mint() (gas: 148590)
[PASS] test_mark_unauthorized() (gas: 15688)
[PASS] test_setFee() (gas: 20630)
[PASS] test_setFee_tooHigh() (gas: 13323)
[PASS] test_setup() (gas: 33739)
[PASS] test_updateExecutor() (gas: 20761)
[PASS] test_updateTreasury() (gas: 20892)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 4.57ms (788.92µs CPU time)

Ran 7 tests for test/tests-local/tETH.t.sol:tETHTest
[PASS] test_decimals() (gas: 8685)
[PASS] test_deposit() (gas: 182180)
[PASS] test_mint() (gas: 182178)
[PASS] test_redeem() (gas: 177252)
[PASS] test_transfer() (gas: 148366)
[PASS] test_transferOwnership() (gas: 33546)
[PASS] test_withdraw() (gas: 180308)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 5.15ms (1.15ms CPU time)

Ran 10 tests for test/tests-local/Blacklistable.t.sol:BlacklistableTest
[PASS] testFail_updateBlacklister_notOwner() (gas: 12863)
[PASS] test_blacklist() (gas: 39964)
[PASS] test_blacklist_notBlacklister() (gas: 13527)
[PASS] test_isBlacklisted() (gas: 41834)
[PASS] test_transfer_blacklistedRecipient() (gas: 148128)
[PASS] test_transfer_blacklistedSender() (gas: 146003)
[PASS] test_unBlacklist() (gas: 31622)
[PASS] test_unBlacklist_notBlacklister() (gas: 13530)
[PASS] test_updateBlacklister() (gas: 20455)
[PASS] test_updateBlacklister_zeroAddress() (gas: 13609)
Suite result: ok. 10 passed; 0 failed; 0 skipped; finished in 5.69ms (689.58µs CPU time)

Ran 9 tests for test/tests-local/StrategyExecutor.t.sol:StrategyExecutorTest
[PASS] testFail_actionExecutor_executeActions_NoAccessControl() (gas: 33186)
[PASS] testFail_vaultPull_executeAction_NoAccessControl() (gas: 21516)
[PASS] test_executeNonWhitelistedAction() (gas: 599984)
[PASS] test_executeOnStrategy_VaultPull() (gas: 113079)
[PASS] test_executeOnStrategy_VaultPull_pause() (gas: 138454)
[PASS] test_executeOnStrategy_VaultPull_whitelistActions() (gas: 166422)
[PASS] test_executeOnStrategy_VaultPull_whitelistAssets() (gas: 200443)
[PASS] test_executeOnStrategy_VaultSend() (gas: 109963)
[PASS] test_setStrategyExecutor() (gas: 22311)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 6.05ms (1.77ms CPU time)
```

```
Ran 18 tests for test/tests-local/iETH.t.sol:iETHTest
[PASS] testFail_unauthorizedAddMinter() (gas: 17119)
[PASS] testFail_unauthorizedBurn() (gas: 12841)
[PASS] testFail_unauthorizedBurnFrom() (gas: 15099)
[PASS] testFail_unauthorizedMint() (gas: 13188)
[PASS] testFail_unauthorizedRemoveMinter() (gas: 17155)
[PASS] testFail_unauthorizedSetTimelock() (gas: 15013)
[PASS] testFail_unauthorizedTransfer() (gas: 17736)
[PASS] test_addMinter() (gas: 89870)
[PASS] test_burn() (gas: 34149)
[PASS] test_burnFrom() (gas: 43750)
[PASS] test_fullOwnershipTransfer() (gas: 34279)
[PASS] test_getMinters() (gas: 23235)
[PASS] test_mintTo() (gas: 52763)
[PASS] test_removeMinter() (gas: 75122)
[PASS] test_setTimelock() (gas: 20330)
[PASS] test_timelock() (gas: 12844)
[PASS] test_timelock_fullOwnershipTransfer() (gas: 36012)
[PASS] test_transfer() (gas: 63884)
Suite result: ok. 18 passed; 0 failed; 0 skipped; finished in 3.27ms (1.16ms CPU time)

Ran 10 tests for test/tests-local/TreehouseRouter.t.sol:TreehouseRouterTest
[PASS] test_constructor() (gas: 41743)
[PASS] test_deposit() (gas: 33893)
[PASS] test_depositETH() (gas: 229322)
[PASS] test_deposit_stETH() (gas: 421492)
[PASS] test_deposit_wETH() (gas: 451513)
[PASS] test_deposit_wstETH() (gas: 297689)
[PASS] test_mark_burn_frontrun_vuln() (gas: 420737)
[PASS] test_setDepositCap() (gas: 24933)
[PASS] test_setPause() (gas: 34284)
[PASS] test_share_inflation_vuln() (gas: 614594)
Suite result: ok. 10 passed; 0 failed; 0 skipped; finished in 3.90ms (2.75ms CPU time)

Ran 1 test for test/tests-fork/PnlAccountingHelper.t.sol:PnlAccountingHelperTest
[PASS] test_setup() (gas: 35225)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 14.24s (197.29µs CPU time)

Ran 1 test for test/tests-fork/Converter.t.sol:ConverterTest
[PASS] test_convert() (gas: 1551843)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 14.24s (4.48ms CPU time)

Ran 13 tests for test/tests-local/TreehouseRedemption.t.sol:TreehouseRedemptionTest
[PASS] test_constructor() (gas: 30785)
[PASS] test_finalizeRedeem_eth() (gas: 266448)
[PASS] test_finalizeRedeem_inWaitingPeriod() (gas: 154725)
[PASS] test_finalizeRedeem_insufficientFunds() (gas: 182747)
[PASS] test_finalizeRedeem_weth() (gas: 228273)
[PASS] test_fullOwnershipTransfer() (gas: 34345)
[PASS] test_getPendingRedeems() (gas: 192454)
[PASS] test_getRedeemInfo() (gas: 151866)
[PASS] test_getRedeemLength() (gas: 188619)
[PASS] test_redeem_belowMinimum() (gas: 60254)
[PASS] test_setMinRedeem() (gas: 20615)
[PASS] test_setPause() (gas: 29191)
[PASS] test_setWaitingPeriod() (gas: 20622)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 14.25s (2.16ms CPU time)

Ran 8 tests for test/tests-local/Vault.t.sol:VaultTest
[PASS] test_AddRemoveAllowableAsset() (gas: 62732)
[PASS] test_GetAllowableAssetCount() (gas: 78939)
[PASS] test_GetAllowableAssets() (gas: 90124)
[PASS] test_SetConverter() (gas: 20821)
[PASS] test_SetRedemption() (gas: 55163)
[PASS] test_SetStrategyStorage() (gas: 20798)
[PASS] test_VaultConvert() (gas: 656963)
[PASS] test_Withdraw() (gas: 225339)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 14.25s (7.06ms CPU time)

Ran 2 tests for test/tests-fork/TreehouseRouter.t.sol:TreehouseRouterTest
```

```
[PASS] test_deposit_fuzz(uint256[15],uint256[15]) (runs: 1000, μ: 1253070, ~: 1251465)
[PASS] test_full_cycle() (gas: 1246992)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 19.93s (5.70s CPU time)

Ran 12 test suites in 19.98s (76.95s CPU time): 99 tests passed, 0 failed, 0 skipped (99 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.

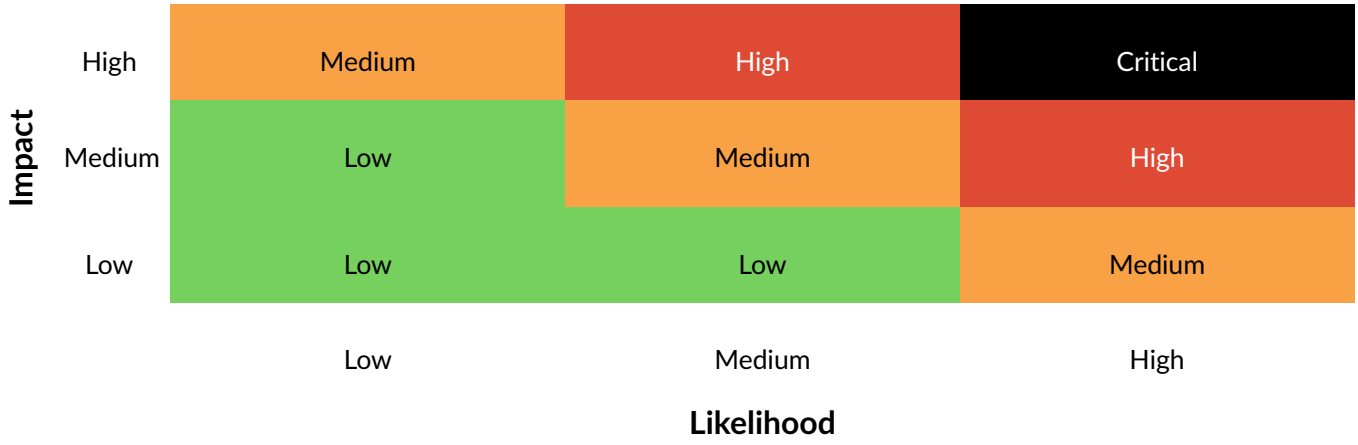

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

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].