LINKPOOL

LinkPool Staking Contracts v2

Smart Contract Security Review

Version: 2.0

January, 2023

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the updated LinkPool staking smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, 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 LinkPool Staking 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 LinkPool Staking smart contracts.

Overview

The LinkPool Staking protocol is composed of a set of smart contracts used to manage users' assets for staking. The protocol mainly supports LINK staking when they are available; it also optimises the users' liquidity by connecting them to DeFi strategies. Related rewards will be distributed to the users according to their staked amounts.

To stake with LinkPool, a certain amount of allowance is required. This allowance can be acquired or borrowed through the protocol.

Security Assessment Summary

This review was conducted on the files hosted on the LinkPool Staking Contracts repository and were assessed at commit c5f4de7 which was then updated to commit 1721bb3.

A subsequent round of testing targeted commit ce76565 and focused solely on verifying whether the previously identified issues had been resolved.

Specifically, the files in scope are as follows:

- `MerkleDistributor.sol`
- `BorrowingPool.sol`
- `LendingPool.sol`
- `PoolOwners.sol`
- `PoolRouter.sol`
- `RewardsPool.sol`
- `StakingPool.sol`
- `RewardsPoolController.sol`
- `StakingRewardsPool.sol`
- `Strategy.sol`
- `LinkPoolNFT.sol`
- `StakingAllowance.sol`
- `WrappedSDToken.sol`
- `ERC677.sol`
- `VirtualERC20.sol`
- `VirtualERC677.sol`

Note: the OpenZeppelin, PRBMath, and Solidity Bytes Utils libraries and dependencies were excluded from the scope of this assessment.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities 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 all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

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

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

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 11 issues during this assessment. Categorised by their severity:

- High: 1 issue.
- Medium: 2 issues.
- Low: 3 issues.
- Informational: 5 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the LinkPool Staking Contracts smart contracts. 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
LPS-01	Staker Potentially Missing Out on Rewards or Losses	High	Open
LPS-02	Strategy Removal Always Fails When Deposits Exist	Medium	Open
LPS-03	Non-standard ERC20 Tokens Are Not Supported and Might Be Locked	Medium	Resolved
LPS-04	Staked Amount Can Exceed `maxDeposits()`	Low	Resolved
LPS-05	`safeApprove` Prevents Multiple Pools of a Same Token-Pool Pair	Low	Resolved
LPS-06	Remove Token With Non-zero Balance	Low	Closed
LPS-07	Renouncing Ownership May Invalidate Functionalities	Informational	Closed
LPS-08	Lack of Token Amount Management	Informational	Resolved
LPS-09	Incremental Gas Cost of `addStrategy()`	Informational	Closed
LPS-10	`onTokenTransfer()` Trust Assumption	Informational	Closed
LPS-11	Miscellaneous General Comments	Informational	Resolved

LPS-01	Staker Potentially Missing Out on Rewards or Losses		
Asset	`StakingPool.sol`		
Status	Open		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

The `StakingPool` contract receives staked tokens from stakers (through `PoolRouter` or `LendingPool`) and distributes the tokens to strategies. When the strategies earn profits (or suffer losses), the stakes are affected. This means the stakers receive profits or losses from the strategies depending on how much tokens they staked (called `shares` in `StakingRewardsPool.sol`).

The profits and losses will be reflected in the stakers' shares only when function `updateStrategyRewards()` is called to update strategies that have earned profits or losses. If this function is not called before calling function `withdraw()` (for stake withdrawal), the stakers' withdrawn amounts will not reflect the profits or losses.

This is because function `StakingRewardsPool.balanceOf()` takes into account `totalShares` and `totalStaked`. A user's balance is calculated in `StakingRewardsPool.getStakeByShares()` as:

StakingRewardsPool.getStakeByShares()

```
78 return (_amount * _totalStaked()) / totalShares;
```

where `_totalStaked()` is `totalStaked` or the total staked amount of all stakers adjusted with `totalRewards` (on line [287] of `StakingPool.sol`). This essential adjustment only occurs when `updateStrategyRewards()` is called, and the related strategies submit their changed deposits (on line [271]).

Consider the following scenario:

Alice stakes 100 tokens to a `StakingPool` which then distributes the staked tokens to `Strategy S`. Alice's stake is 10% of all stakes in the pool. After a while, `Strategy S` earns 10 tokens. For simplicity, all rewards are transferred to the users proportionally to their staked token percentage. In this case, Alice is supposed to receive 101 tokens upon stake withdrawal. However, if Alice (or another user) calls function `withdraw()` before calling function `updateStrategyRewards()`, then Alice will receive only 100 tokens, which is the exact same amount as her initial stake.

In a similar fashion, if `Strategy S` loses 10 tokens, then Alice is supposed to withdraw 99 tokens. However, since the `StakingPool` has not updated the strategy rewards, Alice receives 100 tokens after calling function `withdraw()`, assuming that the total balance of the `StakingPool` is sufficient for Alice's withdrawal and function `updateStrategyRewards()` has not been called. In this case, the other users will suffer more losses (Alice's and their own portion of losses).

Furthermore, a user may take profits and avoid losses by observing and taking the appropriate action accordingly.

For example, Carol observes that `Strategy S` with strategy index of 0 has earned 10 tokens in a `StakingPool`. She then stakes as many tokens as possible to `StakingPool`, such that her share becomes significantly higher than other stakers. She then calls function `updateStrategyRewards([0])` and function `withdraw()` sequentially to withdraw the staked tokens and receive most of the profits.

The user's ability to update only the strategies that gain profits enable them to maximise the profits and avoid losses. Furthermore, the contract does not limit how many tokens a user can stake such that the shares portion can greatly

change instantly.

Note that calling function `StakingPool.withdraw()` must be done from `PoolRouter`. However, this does not change the behaviour of the `StakingPool` contract.

Recommendations

The testing team recommends calling function `updateStrategyRewards()` on all strategies before calling function `withdraw()` (for stake withdrawal).

To avoid a user snatching profits without staking for a considerable amount of time, the testing team recommends adding a snapshot to distribute profits and losses more evenly.

Resolution

The development team has added the contract `SlashingKeeper`. This contract will update the strategy rewards if any losses have been incurred. However, there is no changes in the case of profit, and the users can still do not get their rewards if the function `updateStrategyRewards()` is not called when necessary.

LPS-02	Strategy Removal Always Fails When Deposits Exist		
Asset	`StakingPool.sol`		
Status	**Open**		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

When a strategy is removed from a `StakingPool` (through function `removeStrategy()`), the contract checks whether there are any deposits made to the strategy. If there are, the contract will withdraw the deposits on the removed strategy.

An incorrect implementation on line [**174**] would cause transaction revert on line [**176**] with *Total deposits must remain >= minimum*. This is because the withdrawn amount ignores the minimum deposits that must remain in the strategy. As a result of this error, the contract will always fail to remove the strategy when there are deposits on the strategy (i.e., when the condition on line [**175**] holds).

Note that the tests were conducted using `StrategyMock` contract. The revert message can be different depending on the implementation of the `Strategy` contract.

Recommendations

The testing team recommends replacing `strategy.totalDeposits()` with `strategy.canWithdraw()` on line [**174**].

LPS-03	Non-standard ERC20 Tokens Are Not Supported and Might Be Locked		
Asset	`LendingPool.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

Non-ERC20 compliant tokens might not be supported by the `LendingPool` contract. Specifically, this is true when the `transfer()` function does not correctly implement to the `IERC20` interface.

One prominent example of such tokens is the stable-coin `USDT`.

When a user tries to withdraw such token using the `withdraw()` function, the transaction reverts because the call to `IERC20(token).transfer()` in line [**217**] does not match the expected return value of `transfer()` on the target contract. There is no other way to reclaim these tokens, so they would be locked in one of the pool contracts forever.

Similarly, when trying to call `addPool()` with such tokens, the transaction reverts. This is because the call to `IERC20(token).approve()` does not match the expected return value of `approve()` on the target contract. Hence, the owner will not be able to add this type of tokens to the `LendingPool`.

Recommendations

Appropriate handling of non-standard ERC20 contracts is necessary if these tokens are to be supported. A common way to handle this is by using a vetted library such as OpenZeppelin's SafeERC20. Therefore, consider replacing `IERC20(token).transfer()` by `IERC20(token).safeTransfer()`

Furthermore, consider replacing `IERC20(token).approve()` by `IERC20(token).safeApprove()` and add a check whether previous allowance exists before calling `safeApprove()` since the latter requires that the current approval is zero before setting the new one.

Resolution

The development team has changed the architecture of the project and has removed the contract `LendingPool` in the commit ce76565. Therefore, this issue does not exist anymore.

LPS-04	Staked Amount Can Exceed `maxDeposits()`		
Asset	`StakingPool.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

The `maxDeposits()` function computes the maximum amount of tokens that can be deposited into strategies. It also takes into account whether `liquidityBuffer` is set (i.e., `liquidityBuffer > 0`).

Based on what the return value and developer comments, this function is used to identify how many tokens can be staked into `StakingPool`. However, the related `stake()` function does not call `maxDeposits()` when accepting staking requests. As a result, the staked amount can exceed the maximum amount of tokens that can be deposited to strategies. This means there can be higher liquidity buffers than expected, which would not earn yield from strategies.

Recommendations

The testing team recommends checking that `totalStaked + _amount <= maxDeposits()` must hold on function `stake()`.

Resolution

The development team has resolved this issue in commit ce76565.

The function `PoolRouter._stake()` now includes an additional requirement using the function `PoolRouter.canDeposit()` which takes in consideration the maximum amount that can be deposited in the `StakingPool`.

LPS-05	`safeApprove` Prevents Multiple Pools of a Same Token-Pool Pair		
Asset	`PoolRouter.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `addPool()` function allows the owner to add new pools to the `PoolRouter` contract. Based on the implementation, the `PoolRouter` allows for multiple instances of the same `Token-StakingPool` pair, with different indices assigned to them. That is, to support cases where one index requires allowance (`_allowanceRequired == True`) and another index does not (`_allowanceRequired == False`).

However, the `safeApprove()` function will revert a transaction that adds a pool of existing `Token-StakingPool` pair. This is because the function `safeApprove()` does not allow for allowance override. In order for `safeApprove()` to be successful, the previous allowance should be zero (see OpenZeppelin's `SafeERC20.sol` library for more details).

Note that the testing team utilises OpenZeppelin Library version 4.7.0 for testing. It is possible that the behaviour of `safeApprove()` may differ in other versions of the library.

Recommendations

Make sure this behaviour is intended. The testing team recommends checking whether previous allowances exist before calling `safeApprove()` on line [**264**] to allow for existing `Token-StakingPool` pairs to be added through `addPool()` .

Resolution

The recommendation has been implemented in commit ce76565.

LPS-06	Remove Token With Non-zero Balance		
Asset	`RewardsPoolController.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The owner can add and remove tokens to the contract `RewardsPoolController`.

The `removeToken()` function does not check if the token balance of the contract is zero or not. If the balance of a token is not zero and this token is removed, the owner should add it again to the contract to withdraw the remaining amount of tokens.

Recommendations

Consider adding a require statement in `removeToken()` that checks for the token contract balance.

Resolution

This issue has been acknowledged by LinkPool team.

LPS-07	Renouncing Ownership May Invalidate Functionalities
Asset	`MerkleDistributor.sol, StakingPool.sol, PoolRouter.sol, BorrowingPool.sol, LendingPool.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

Contract `MerkleDistributor`'s main functions, namely `addDistribution()` and `addDistributions()` require the caller to be the owner of the contract. This contract derives from `Ownable` and therefore, has a function to renounce ownership through function `renounceOwnership()`. If this function is executed, either by accident or by malicious intent, the contract will become unusable.

As a remedy if this occurs intentionally, the former owner can deploy a new `MerkleDistributor` contract to replace the current one. However, the information propagation about the change may take time and effort.

Other contracts such as `StakingPool` may have a more devastating impact if the owner is renounced.

Furthermore, by inheriting OpenZeppelin's `Ownable`, ownership transfers in `transferOwnership()` are unilateral, meaning that ownership can be accidentally transferred to an uncontrolled address.

Recommendations

The testing team recommends overriding the `renounceOwnership()` function to prevent it from invalidating core contract functionalities.

The testing team also recommends changing the ownership transfer to a propose/accept model.

One convenient way to do this is to replace Openzeppelin's `Ownable` with Chainlink's `ConfirmedOwnerWithProposal`.

Resolution

This issue has been acknowledged by LinkPool team.

LPS-08	Lack of Token Amount Management
Asset	`MerkleDistributor.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `MerkleDistributor` contract manages token distributions and claim requests. If a claim request is successfully verified, the requested token amount will be transferred to the claimer. This indicates that contract `MerkleDistributor` should have enough tokens to fulfill the claim request. In a normal operation, the tokens should be transferred to `MerkleDistributor` before the first claim request of a distribution is made.

The `MerkleDistributor` contract does not have a mechanism to manage the token balances and therefore the claim may fail if not enough tokens are available.

Recommendations

The testing team recommends updating the `MerkleDistributor.sol` contract to introduce token balance checks, allowing it to ensure sufficient balances before distribution.

Additionally, it is also recommended to allow the contract owner to *"skim"* excess tokens after the distribution is complete.

Resolution

The development team has fixed this issue in the commit ce76565 by adding the transfer of the necessary amount of token to the contract in the `addDistribution()` function. Added the that, the development team has added the function `withdrawUnclaimedTokens()` to withdraw the unclaimed tokens.

LPS-09	Incremental Gas Cost of `addStrategy()`
Asset	`StakingPool.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `addStrategy()` function adds a new strategy to the pool. Before adding a new strategy, the contract checks whether the strategy exists by calling a private function `_strategyExists()`. This private function searches through the list of strategies to see if the strategy already exists. This method increases the gas cost of `addStrategy()` as the number of strategies increases.

Our test indicates that calling `addStrategy()` when there is no strategy in the pool costs 94,043 gas. However, calling `addStrategy()` when there are 99 strategies in the pool costs 341,999 gas.

Recommendations

The testing team recommends improving the check for whether the strategy already exists, for example, by using a `mapping` to store the strategies. Alternatively, OpenZeppelinś `EnumerableSet` library could also be used.

Resolution

This issue has been acknowledged by LinkPool team.

LPS-10	`onTokenTransfer()` Trust Assumption
Asset	`PoolOwners.sol, LendingPool.sol, PoolRouter.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

As an ERC677 token receiver, the `PoolOwners` contract implements the `onTokenTransfer()` function which will be called by the ERC677 token contract when transferring tokens through `transferAndCall()`.

The `onTokenTransfer()` function contains several inputs, including `_value`. This variable is typically used by the receiving contract to determine how much tokens were transferred without rechecking the receiver's token balance change.

This assumes that the ERC677 contract behaves honestly. Since only selected token(s) are allowed to be transferred, it is assumed that the contract creator carefully checks the token.

A hardening strategy could be to have the receiving contract check the token balance change, rather than relying on the ERC677 contract to supply the value.

This issue affects other contracts such as `LendingPool` and `PoolRouter`.

Recommendations

Make sure this behaviour is intended. The testing team recommends ensuring that the `_value` matches the actual token balance change. The check however, will increase the transaction gas cost.

Resolution

This issue has been acknowledged by LinkPool team.

LPS-11	Miscellaneous General Comments
Asset	`contracts/*`
Status	**Resolved:** See **??**
Rating	Informational

Description

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

1. **No Deposit/Withdrawal Amount Check on `strategyDeposit()` and `strategyWithdrawal()`**

   Related Asset(s): StakingPool.sol

   Both functions, `strategyDeposit()` and `strategyWithdrawal()`, do not check whether the amounts are valid. The testing team assumes that the checks will be conducted on the `Strategy` contract. Make sure this is the case.

2. **Unreachable Code on Function `reorderStrategies()`**

   Related Asset(s): StakingPool.sol

   Function `reorderStrategies()` reorders the strategies based on the new order in the input. This function has a check on line [**199**] to ensure that all indices are valid. However, this code may not be useful because if one of the indices is invalid, the function will revert with *Index out of range* because it tries to access an array item that is not there.

   The code on line [**199**] can be safely removed.

3. **Transfer Emergency Wallet to Zero Address**

   Related Asset(s): StakingPool.sol

   Function `transferEmergencyWallet()` allows the current emergency wallet account to be transferred to a new account. There is no check whether the new account is a non-zero address. Although it is possible to use this function to nullify the emergency wallet account, it may also produce an undesirable outcome.

   Make sure this behaviour is intended.

4. **Zero Address on Event `RemovePool`**

   Related Asset(s): PoolRouter.sol

   When function `removePool()` is called, the event `RemovePool` is emitted with the `address(pool.stakingPool)` as the `pool` parameter. This value will always be a zero address because the `pool` data has been deleted on line [**277**].

   The testing team recommends replacing pool address with the removed pool index on event `RemovePool`.

5. **Typo**

   Related Asset(s): PoolRouter.sol

   On line [**275**]: *Only can remove a pool with no active stake* can be replaced with *Can only remove a pool with no active stake*.

6. **Event on `setRateConstants()`**

   Related Asset(s): LendingPool.sol

   Function `setRateConstants()` changes important variables of the `LendingPool` contract. Therefore, the testing team recommends emitting an event on function `setRateConstants()`.

7. **Variables can be declared as `immutable`.**
   Variables set in the `constructor` and never updated otherwise can be declared as `immutable`. This is more gas efficient than declaring a mutable variable. Here is the list of variables that can be declared as `immutable` :
   - **StakingPool** : `poolRouter`
   - **RewardsPool** : `controller` and `token`
   - **PoolRouter** : `allowanceToken`
   - **PoolOwners** : `token`
   - **LendingPool** : `allowanceToken` and `poolRouter`
   - **BorrowingPool** : `baseToken`, `poolIndex`, `lendingPool` and `stakingPool`
   - **WrappedSDToken** : `sdToken`
   - **LinkPoolNFT** : `lpMigration`
   - **StakingRewardsPool** : `token`
8. **Redundant check**
   Related Asset(s): PoolRouter.sol
   The `poolExists` modifier is checked three times in the `stake()` function:
   (a) In `stake()` function.
   (b) In `allowanceRequired()` that is called inside the internal function `_stake()` .
   (c) In `allowanceInUse()` that is called inside the internal function `_stake()` .

   Related Asset(s): MerkleDistributor.sol
   - Redundant check of the `distributionExists` modifier in the `claim()` function as this modifier is also used in `isClaimed()` function.
   - Function `addDistributions()` checks whether the caller is the owner of the contract multiple times due to repetitive calls to the `onlyOwner` modifier in `addDistribution()` .

Recommendations

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

The development team has acknowledged issues detailed above and addressed where deemed appropriate in commit ce76565. In particular:

1. Note has been acknowledged.
2. Note has been acknowledged.
3. The `transferEmergencyWallet()` has been removed from the `PoolRouter` contract as the `emergencyWallet` has been also removed from the `struct Pool` .
4. Event `RemovePool` now emits before deleting the pool and display the correct pool's address.
5. Typo has been fixed as recommended.
6. Function `RampUpCurve.setRateConstants()` now emits event `RateConstantsSet` .
7. The recommendation has been implemented.
8. Note has been acknowledged.

Appendix A Test Suite

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

```
test_init                                       PASSED   [1%]
test_update_rewards                             PASSED   [2%]
test_init                                       PASSED   [3%]
test_transfer_and_call                          PASSED   [4%]
test_init                                       PASSED   [5%]
test_init_configure                             PASSED   [6%]
test_on_token_transfer                          PASSED   [7%]
test_lend_allowance                             PASSED   [8%]
test_withdraw_allowance                         PASSED   [9%]
test_stake                                      PASSED   [10%]
test_withdraw                                   PASSED   [11%]
test_setRateConstants                           PASSED   [12%]
test_add_remove_pool                            PASSED   [13%]
test_remove_pool_stake                          PASSED   [14%]
test_add_pool_non_standard_ERC20_token          XFAIL    [15%]
test_init                                       PASSED   [16%]
test_mint                                       PASSED   [17%]
test_set_base_uri                               PASSED   [18%]
test_init                                       PASSED   [19%]
test_add_distribution                           PASSED   [20%]
test_add_distributions                          PASSED   [21%]
test_no_balance                                 PASSED   [22%]
test_renounce_ownership                         PASSED   [23%]
test_init                                       PASSED   [24%]
test_on_token_transfer                          PASSED   [25%]
test_stake_withdraw                             PASSED   [26%]
test_init                                       PASSED   [27%]
test_init_configure                             PASSED   [28%]
test_view_init                                  PASSED   [29%]
test_on_token_transfer                          PASSED   [30%]
test_on_token_transfer_no_allowance             PASSED   [31%]
test_stake                                      PASSED   [32%]
test_withdraw                                   PASSED   [34%]
test_withdraw_exceeds                           PASSED   [35%]
test_withdraw_profits                           PASSED   [36%]
test_stake_allowance                            PASSED   [37%]
test_withdraw_allowance                         PASSED   [38%]
test_withdraw_allowance_in_use                  PASSED   [39%]
test_add_pool                                   XFAIL    (safeApprovecauses
test_remove_pool                                PASSED   [41%]
test_remove_pool_active_stake                   PASSED   [42%]
test_set_allowance_required                     PASSED   [43%]
test_set_pool_status                            PASSED   [44%]
test_transfer_emergency_wallet                  PASSED   [45%]
test_init                                       PASSED   [46%]
test_balance_of                                 PASSED   [47%]
test_withdraw                                   PASSED   [48%]
test_withdraw_updated                           PASSED   [49%]
test_on_token_transfer                          PASSED   [50%]
test_init                                       PASSED   [51%]
test_init_config                                PASSED   [52%]
test_staked                                     PASSED   [53%]
test_rewards_address                            PASSED   [54%]
test_distribute_token                           PASSED   [55%]
test_distribute_tokens                          PASSED   [56%]
test_distribute_tokens_stable                   PASSED   [57%]
test_add_remove_token                           PASSED   [58%]
test_remove_token_with_non_zero_balance         XFAIL    [59%]
test_constructor                                PASSED   [60%]
test_init                                       PASSED   [61%]
test_mint_burn                                  PASSED   [62%]
```

```
test_mint_to_contract                                PASSED   [63%]
test_init                                            PASSED   [64%]
test_stake_fail                                      PASSED   [65%]
test_stake_withdraw                                  PASSED   [67%]
test_strategy_deposit_withdraw                       PASSED   [68%]
test_init_configure                                  PASSED   [69%]
test_add_strategy_bulk                               SKIPPED  [70%]
test_add_remove_reorder_strategies                   PASSED   [71%]
test_reorder_strategies_duplicated                   PASSED   [72%]
test_remove_strategy_profit                          XFAIL    (Abug
test_add_update_fee                                  PASSED   [74%]
test_set_wsdtoken                                    PASSED   [75%]
test_set_liquidity_buffer                            PASSED   [76%]
test_stake_max                                       PASSED   [77%]
test_balance_of_profit                               PASSED   [78%]
test_update_strategy_rewards_profit                  PASSED   [79%]
test_update_strategy_rewards_loss                    PASSED   [80%]
test_deposit_liquidity_no_stake                      PASSED   [81%]
test_deposit_liquidity_full_stake                    PASSED   [82%]
test_stake_withdraw_profit                           PASSED   [83%]
test_stake_withdraw_profit_after_update_strategy     PASSED   [84%]
test_stake_withdraw_loss                             PASSED   [85%]
test_stake_withdraw_loss_after_update_strategy       PASSED   [86%]
test_stake_profit_stake_withdraw                     PASSED   [87%]
test_ownership                                       PASSED   [88%]
test_init                                            PASSED   [89%]
test_deposit                                         PASSED   [90%]
test_set_deposits                                    PASSED   [91%]
test_init                                            PASSED   [92%]
test_transfer                                        SKIPPED  [93%]
test_transfer_from                                   SKIPPED  [94%]
test_init                                            PASSED   [95%]
test_transfer_and_call                               PASSED   [96%]
test_init                                            PASSED   [97%]
test_on_token_transfer                               PASSED   [98%]
test_wrap_unwrap                                     PASSED   [100%]
```

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.

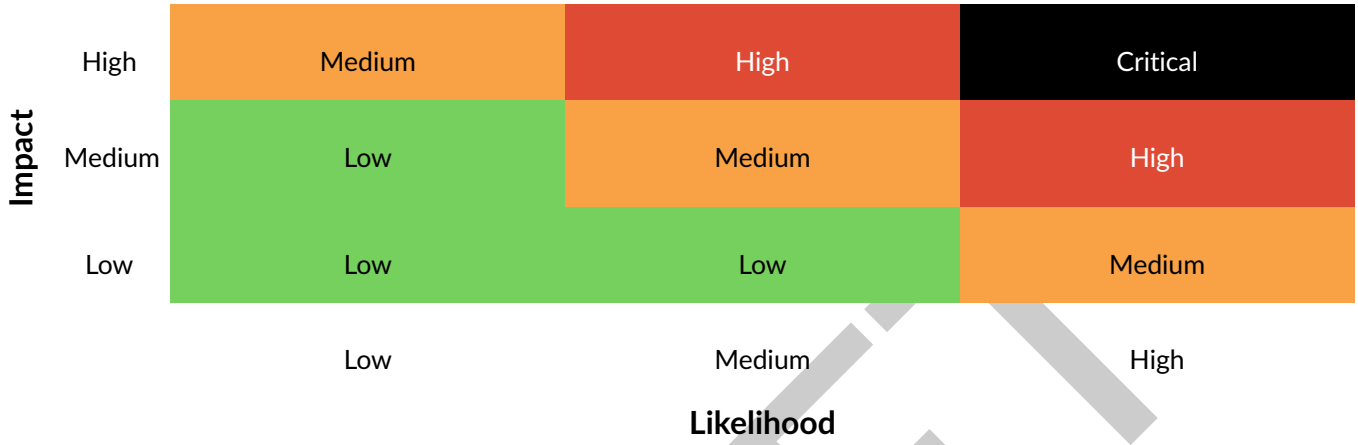

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

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].