LINKPOOL

LinkPool Staking Contracts v4

Smart Contract Security Review

Version: 1.0

January, 2023

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Linkpool 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 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 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.

Staking with LinkPool is free for pools that are not in the Reserved mode. However, if a pool is in the Reserved mode, the amount that can be staked is based on SDL staked.

`LinkStaking` is provided as a strategy for users staking `Link` with LinkPool. The strategies are used for managing multiple `Chainlink` Operator and Community staking vaults.

Security Assessment Summary

This review was conducted on the files hosted on the LinkPool Staking Contracts repository and were assessed at commit ce76565.

Specifically, the files in scope are as follows:

- `DelegatorPool.sol`
- `GovernanceController.sol`
- `PoolRouter.sol`
- `RewardsPool.sol`
- `RewardsPoolWSD.sol`
- `SlashingKeeper.sol`
- `StakingPool.sol`
- `RewardsPoolController.sol`
- `StakingRewardsPool.sol`
- `Strategy.sol`
- `Flat.sol`
- `RampUpCurve.sol`
- `LinkPoolNFT.sol`
- `LPLMigration.sol`
- `StakingAllowance.sol`
- `WrappedSDToken.sol`
- `ERC677.sol`
- `ERC677Upgradeable.sol`
- `CommunityVault.sol`
- `CommunityVCS.sol`
- `OperatorVault.sol`
- `OperatorVCS.sol`
- `Vault.sol`
- `VaultControllerStrategy.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 15 issues during this assessment. Categorised by their severity:

- High: 1 issue.
- Low: 9 issues.
- Informational: 5 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Linkpool 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
LP4-01	Potentially Excessive Fees Can Reduce Principal	High	Open
LP4-02	Strategies Can Override Settings to Bypass Liquidity Buffer Limit	Low	Open
LP4-03	Missing Validation on `addPool()` Can Lead to Undersired Behaviour	Low	Open
LP4-04	Reward Distribution Can Revert When No Allowance Tokens Are Staked	Low	Open
LP4-05	Inconsistent Algebraic Expression	Low	Open
LP4-06	Reward Tokens Can Be Subject to Precision Loss	Low	Open
LP4-07	Potential Integer Overflows in `getMaxDeposits()`	Low	Open
LP4-08	Lack of `FlatFee` Validation in Constructor	Low	Open
LP4-09	Lack of Validation of `minDepositThreshold` in `__VaultControllerStrategy_init()`	Low	Open
LP4-10	Operator Cannot Be Changed	Low	Open
LP4-11	Non-standard Decimal Tokens Can Lead to Extensive Fees	Informational	Open
LP4-12	Multiple Transactions to Remove Strategies Could Interfere With Each Other	Informational	Open
LP4-13	Removed Pools Maintain Token Spending Allowance	Informational	Open
LP4-14	Token Balance Queries Could Potentially Be Expensive and Error Prone	Informational	Open
LP4-15	Miscellaneous General Comments	Informational	Open

LP4-01	Potentially Excessive Fees Can Reduce Principal		
Asset	`StakingPool.sol`, `PoolRouter.sol`		
Status	**Open**		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

Function `updateStrategyRewards()` could pay excessive fees that would reduce the stakers' principal. In this case, fees would be higher than the reward.

Function `updateStrategyRewards()` pays a set of fees from `Strategies`, `StakingPool` and `DelegatorPool`. When the staked amount is equal to `StakingPool.getMaxDeposits()`, the fees by `DelegatorPool` can be up to 92% of the total reward. This is because the `PoolRouter.poolUtilisation()` will return `1e18` and as a result the `currentRate` of the `DelegatorPool` will be around 9200. Depending on the fee ratio to be paid to the Strategy receivers, the total fee could exceed 100%.

The total fee amount is then minted and paid to the respective parties as `shares` of the `StakingPool`'s stake. When the fee amount is higher than the reward, the new minted shares will be more significant than the `totalRewards` and hence the increase of the `totalShares` will be more important than the increase of the `totalStaked`, which in turn reduces the value of the stakers' shares in the `StakingPool`. This potentially reduces the stakers' principal amount.

Recommendations

Limit the total fee amount to a certain percentage of the reward. The `currentRate` can be as high as 92% which would be a significant amount if multiplied by the reward amount.

LP4-02	Strategies Can Override Settings to Bypass Liquidity Buffer Limit		
Asset	`StakingPool.sol`, `Strategy.sol`		
Status	Open		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

A staking pool has a setting, its liquidity buffer, which should enforce a minimum number of tokens that remain in the staking pool contract itself, thereby facilitating relatively small withdrawals without always requiring withdrawals from the strategy contracts. However, the staking pool contract does not enforce this buffer itself and a strategy contract can override it, potentially causing unnecessary gas expenditure for small withdrawals.

There is a variable, `StakingPool.liquidityBuffer`, which is described as *always keeping a % of the staked token as liquid within the pool*. However, this variable is not enforced in the actual implementation.

`StakingPool.depositLiquidity`, when deciding how many tokens are potentially available to deposit into strategies, simply calls `token.balanceOf(address(this))` on line [**378**] and then distributes this value amongst the strategy contracts, using only `strategy.canDeposit()` to limit the distribution.

Recommendations

Consider changing line [**378**] of `StakingPool.depositLiquidity` to something like:

```
uint256 buffer = (liquidityBuffer*totalStaked)/(liquidityBuffer+10000);
uint256 toDeposit = token.balanceOf(address(this));
if (toDeposit <= buffer) {
    toDeposit = 0;
} else {
    toDeposit -= buffer;
}
```

It might also be desirable to enforce a check on `liquidityBuffer` within the `_withdrawLiquidity()` function.

LP4-03	Missing Validation on `addPool()` Can Lead to Undersired Behaviour		
Asset	`PoolRouter.sol`, `StakingPool.sol`		
Status	**Open**		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

When adding a new pool through `PoolRouter.addPool()`, the function does not check for existing `StakingPool` that has been added, and for the associated `StakingPool`'s token, resulting in overwriting `StakingPool`, or a Denial-of-Service of the staking functionality.

Function `addPool()` takes `_token` and `_stakingPool` as inputs. The inputs are then stored in the pool's state variable. This operation does not check several conditions:

1. Whether the `_token` is accepted by the `_stakingPool`.
2. Whether `_stakingPool` has been added to `PoolRouter` before.

If the first condition is not met, staking tokens through `_stake()` will fail. Consider the following case. The owner pairs `Token A` with `StakingPool A`, while `StakingPool A`'s accepted token is `Token B`. When a user calls `PoolRouter.stake()`, `Token A` will be transferred from the user to `PoolRouter`, but while calling `StakingPool.stake()` on line [**397**] of function `PoolRouter._stake()`, `StakingPool A` expects `Token B`.

If the second condition is not met, the function will overwrite the `StakingPool`'s `poolIndex` on line [**242**] of `PoolRouter.sol`.

Recommendations

While adding a pool to `PoolRouter` through function `addPool()`, the function can take `_token` data directly from `StakingPool` by calling `StakingPool.token()`. Furthermore, prevent the function from overwriting existing pool data or otherwise `StakingPool.poolIndex` will be overwritten.

LP4-04	Reward Distribution Can Revert When No Allowance Tokens Are Staked		
Asset	`DelegatorPool.sol`, `PoolRouter.sol`, `RewardsPool.sol`, `StakingPool.sol`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

As communicated to the testing team by the development team, it should be possible for staking pools to operate with no requirements for allowance tokens at all. However, this does not seem to be the case: reward distribution reverts if no allowance tokens are staked.

If `poolRouter.addPool()` is called with its `_reservedModeActive` parameter set to `False`, then the pool in question should operate independently of allowance tokens. It is possible to stake in such a pool with no allowance tokens staked in `DelegatorPool`. However, when rewards are updated, the call reverts if no allowance tokens are staked.

This revert occurs in a call to `stakingPool.updateStrategyRewards()` because this function computes rewards to be sent to a set of receivers including `DelegatorPool` if `DelegatorPool.currentRate()` returns a value above zero. That function calls `feeCurve.currentRate()` which, for all the current versions of `feeCurve` (`RampUpCurve` and `Flat`), are likely to return a minimum constant value, and so be greater than zero.

This in turn triggers a set of calls from `DelegatorPool.onTokenTransfer()` to `RewardsPoolController.distributeToken()` to `RewardsPool.distributeRewards()`. This last function has line [77]:

```
require(controller.totalStaked() > 0, "Cannot distribute when nothing is staked");
```

As `controller` is `DelegatorPool`, it will revert if no allowance tokens are staked.

In summary, when `DelegatorPool` is set as `RewardsPool.controller`, any call to `RewardsPool.distributeRewards()` will revert if no allowance tokens are staked, and this is not the desired behaviour as stated by the development team.

Recommendations

Consider whether it is desirable for `DelegatorPool` to be set as `RewardsPool.controller` for a pool which does not require allowance tokens to be staked. If this is desirable, consider revising the logic in `DelegatorPool.currentRate()` to ensure it returns zero when no allowance tokens are staked.

LP4-05	Inconsistent Algebraic Expression		
Asset	`RampUpCurve.sol`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The comments in `RampUpCurve.sol` state that the equation used is `y = (A*x/B)^C + x/D + E`. However, the term `x/D` is excluded when `D` is equal to 1.

On line [**78**], there is an `if` test to add the term `x/D`:

```
if (rateConstantD > 1) {
    y = y + (x * 100).div(rateConstantD).toUint();
}
```

If `D` is 1, the term `x/D` would evaluate to `x` and so should still be included in the calculation to implement the formula as intended.

Recommendations

To ensure consistency, either the comment on line [**69**] should be modified or the statement on line [**78**] should be updated to `if (rateConstantD > 0)`.

LP4-06	Reward Tokens Can Be Subject to Precision Loss		
Asset	`RewardsPool.sol`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Because of a fixed precision multiplier, in extreme cases the rewards calculated might lose some precision.

On line [**115**], there is the following function:

```
function _updateRewardPerToken(uint256 _reward) internal {
    uint256 totalStaked = controller.totalStaked();
    require(totalStaked > 0, "Staked amount must be > 0");
    rewardPerToken += ((_reward * 1e18) / totalStaked);
}
```

Consider a scenario where the reward token is either high value, or has fewer than 18 decimals. In this case, the value of `_reward` would be relatively low. If the allowance token staked in `controller` is relatively low value, and is heavily staked in large numbers, there could be a loss of precision.

For example, consider a reward token with 6 decimals of precision, and a reward of 9876 tokens. If the staking token has 18 decimals and 10 billion tokens staked,

```
rewardPerToken += ((9_876e6 * 1e18) / 1e28);
```

This calculation would set `rewardPerToken` to zero. However, the reward would still be counted as having been distributed on line [**79**] in the function `RewardsPool.distributeRewards()`:

```
uint256 toDistribute = token.balanceOf(address(this)) - totalRewards;
totalRewards += toDistribute;
_updateRewardPerToken(toDistribute);
```

Recommendations

Consider adding a higher precision modifier for the `rewardPerToken` variable.

LP4-07	Potential Integer Overflows in `getMaxDeposits()`		
Asset	`StakingPool.sol`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

This function calls multiple contracts and adds their return values to a variable. It is possible the values could go over the maximum limit and cause a revert.

`getMaxDeposits()` contains the following loop:

```
uint256 max;

for (uint256 i = 0; i < strategies.length; i++) {
    IStrategy strategy = IStrategy(strategies[i]);
    max += strategy.getMaxDeposits();
}
```

Whilst the return value from a single call to `strategy.getMaxDeposits()` cannot exceed the maximum value of `uin256`, multiple calls added together might do so, and this would cause a revert in this function. As this function is invoked in multiple instances throughout the project, there would be a significant loss of functionality if this were to occur.

This issue is mitigated by the development team's control of the strategy contracts. However, it is possible that a dynamic calculation in multiple contracts' versions of `strategy.getMaxDeposits()` might grow unexpectedly over time and eventually encounter this problem.

Recommendations

One possible approach would be to cap the value of `max`. Within the loop, check that `strategy.getMaxDeposits() <= type(uint256).max - max`. If that condition is ever false, `type(uint256).max` can be returned as the value with no further calculation. This has the added advantage that strategy contracts can safely return `type(uint256).max` to give themselves an unlimited allowance (if that is ever desirable).

LP4-08	Lack of `FlatFee` Validation in Constructor		
Asset	`FlatFee.sol`		
Status	Open		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

When `owner` modifies the flat fee, there is a check to prevent the fee going over 95%. However, there is no check on the flat fee first set when the contract is deployed.

The `constructor()` of `FlatFee` simply reads:

```
constructor(uint256 _feeBasisPoints) {
    feeBasisPoints = _feeBasisPoints;
}
```

There are no checks on the value of `_feeBasisPoints`. It could even be above 100%.

In contrast, `setFeeBasisPoints()` enforces this check:

```
require(_feeBasisPoints >= 0 && _feeBasisPoints <= 9500, "Invalid flat fee");
```

Recommendations

Use the same code in both `setFeeBasisPoints()` and the `constructor()` to set `feeBasisPoints`.

LP4-09	Lack of Validation of `minDepositThreshold` in `__VaultControllerStrategy_init()`		
Asset	`VaultControllerStrategy.sol`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

According to the comment on line [260] and the require statement on line [265] in the `setMinDepositThreshold()` function, the `minDepositThreshold` should always be greater than `vaultMinDeposits`. However, there is no check on the `minDepositThreshold` first set when the contract is initialized.

Recommendations

Consider adding the require statement used in `setMinDepositThreshold()` to the `__VaultControllerStrategy_init()` to enforce the check of `minDepositThreshold`.

LP4-10	Operator Cannot Be Changed		
Asset	`OperatorVault.sol`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

It is not possible to change the `operator` of the `OperatorVault`.

The `setOperator()` function has the `onlyOwner` modifier as an access control for the function. However, the `owner` of the `OperatorVault` contract is the `OperatorVCS` as the vault is initialized from the `OperatorVCS`. Since there is no function in `OperatorVCS` that calls `OperatorVault.setOperator()`, the operator cannot be changed

Recommendations

Consider adding a function in `OperatorVCS` that calls the `OperatorVault.setOperator()` or set in the `OperatorVault.initialize()` the `owner` to an EOA.

LP4-11	Non-standard Decimal Tokens Can Lead to Extensive Fees
Asset	`StakingPool.sol`, `PoolRouter.sol`, `DelegatorPool.sol`
Status	**Open**
Rating	Informational

Description

Because of a fixed precision multiplier in `PoolRouter.poolUtilisation()`, the `DelegatorPool`' fees can easily reach 95%.

In fact, `StakingPool.totalSupply()` and `StakingPool.getMaxDeposits()` can be both in the scale of `1e6` if the staking token in the `StakingPool` has `decimals = 6`. In this case, the function `PoolRouter.poolUtilisation()` will return a high value because `StakingPool.totalSupply()` is multiplied `1e18` on line [**141**]. As a result, the `currentRate()` of the `DelegatorPool` will be up to 95%.

Recommendations

Consider adding the necessary changes when dealing with non-standard decimal tokens.

LP4-12	Multiple Transactions to Remove Strategies Could Interfere With Each Other
Asset	`StakingPool.sol`
Status	**Open**
Rating	Informational

Description

Because strategies are referenced by index number, multiple removals or reorderings could interact to remove the wrong strategy.

`removeStrategy()` takes a single argument, `_index`, to determine which strategy to remove. It then removes the strategy at that index and moves all the subsequent strategies down by index number.

If there were a problem submitting a transaction such that two attempts to remove a strategy in separate transactions were eventually submitted, this would result in the original strategy being removed followed by the strategy after it.

Similarly, a call to `reorderStrategies()` which is resolved before a call to `removeStrategy()` could change the indices, resulting in the wrong strategy being removed.

This issue is mitigated by the `onlyOwner` modifier's presence on both of these functions, meaning that a trusted user would presumably be calling them.

Recommendations

One possible approach would be to add a second parameter to `removeStrategy()` which contains the address of the strategy to be removed. This address could be checked against `strategies[_index]` to ensure that the correct strategy is being targetted.

LP4-13	Removed Pools Maintain Token Spending Allowance
Asset	`PoolRouter.sol`
Status	**Open**
Rating	Informational

Description

When a staking pool is removed from the pool router, its token spending allowance remains. In the long term, this legacy access to the router's tokens could have security implications.

When pools are added in `PoolRouter.addPool()`, they are approved by `PoolRouter` to spend `type(uint).max` tokens on line [239].

When pools are removed in `PoolRouter.removePool()`, this allowance is not revoked.

Recommendations

Consider revoking all token spending allowances when a pool is removed via `PoolRouter.removeStrategy()`.

LP4-14	Token Balance Queries Could Potentially Be Expensive and Error Prone
Asset	`RewardsPoolController.sol`
Status	Open
Rating	Informational

Description

Because this contract loops through all the supported tokens and calls each one for the balance, a rewards pool with many tokens could have a high gas cost to query. Also, if there is any issue with one (e.g. revert) balance check, the entire call will fail.

`tokenBalances()` contains the following loop:

```
for (uint256 i = 0; i < tokens.length; i++) {
    balances[i] = IERC20Upgradeable(tokens[i]).balanceOf(address(this));
}
```

External calls within a loop can unpredictably mount up in gas costs. If there are many tokens and if some have complex code, any call to this function within a transaction could have unexpectedly high costs.

This issue is mitigated by the `external` scope of the function. The function is not called anywhere within the code and might be mainly for use in user interfaces. However, this does not prevent it from being called within a transaction.

Recommendations

This issue can be managed by carefully curating the reward tokens to ensure they are all reliable and gas efficient. Alternatively, checking balances within a `try` block would prevent a revert from one token contract blocking the entire function call.

LP4-15	Miscellaneous General Comments
Asset	`contracts/*`
Status	Open
Rating	Informational

Description

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

1. **Code comments** – Readable code is easier to maintain and modify, leading to fewer security issues. It also is easier to review, resulting in cost savings for projects and increased likelihood of finding issues. For these reasons, it is recommended to add clarifying comments throughout the code.
2. **Specify units** – The previous point is particularly acute in `RampUpCurve.sol` which contains multiple parameters, variables and configured settings. The code would become significantly clearer if the units of all of these quantities were specified.
3. **Gas optimisations** –
   (a) Whenever looping through a list or count, a small gas saving can be obtained by incrementing using the form `++i` instead of `i++`.
   (b) In `FlatFee.sol` line [**24**], the check `_feeBasisPoints >= 0` is performed. However, `_feeBasisPoints` is a variable of type `uint256` so it will always have a value in this range, by definition.
4. **Zero value checks** –
   (a) In `PoolRouter.stakeETH()`, consider checking whether `msg.value > 0`
5. **Pools can be added with any status** – In `PoolRouter.addPool()`, the parameter `_status` will accept any pool status. It is questionable whether it is desirable to have the functionality to add a pool of any status other than `OPEN`.
6. **File name does not match contract name** – The file `Flat.sol` contains only one contract, which has the name `FlatFee`.
7. **Unreachable code** – In `DelegatorPool.withdrawAllowance()`, the line [**135-137**] are unreachable, because if `_amount = type(uint).max`, the transaction would revert because of the require statement in line [**130**]. Consider removing these lines.
8. **No getter function for the fees** – In `VaultControllerStrategy.sol`, the `fees` array is set as internal, however, there is no getter function to check the value of the corresponding fee value of a receiver. Consider adding a getter function for the fees.
9. **Missing check in `PoolRouter.setReservedMode()`** – The `PoolRouter.setReservedMode()` function does not check if the pool exists or not. Consider adding the `poolExists` modifier to this function.
10. **Lack of Index Tracking** – Pool information in `PoolRouter` is identified by the `token` and `index`. While the token addresses are emitted in events (such as `AddPool`), the `index` is not. This makes it difficult to get indexes of pools that have been added to PoolRouter. The testing team recommends emitting the `index` of the pool in events or providing a feature to list indexes based on token as input.

11. **Typo –**
    Related Asset(s): SlashingKeeper.sol
    On line [**11**]: *"inucurred"*, should be *"incurred"*.

12. **Redundant check –**
    Related Asset(s): PoolRouter.sol
    The `poolExists` modifier is checked two times in the `stake()` function:

    (a) In `stake()` function.

    (b) In `canDeposit()` that is called inside the internal function `_stake()` .

13. **Missing sanity checks on the fees –**
    Related Asset(s): StakingPool.sol, VaultControllerStrategy.sol
    There is not a bound check on the fees for the `addFee()`, and `updateFee()` functions. Consider adding the necessary require statement.

Recommendations

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

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   [0%]
test_deposit                                             PASSED   [1%]
test_deposit_buffered_tokens                             PASSED   [2%]
test_set_max_vault_deployments                           PASSED   [3%]
test_set_max_deposits                                    PASSED   [4%]
test_on_token_transfer_stake_allowance_without_vesting   PASSED   [4%]
test_on_token_transfer_stake_allowance_with_vesting      PASSED   [5%]
test_withdraw_allowance_without_vestingSchedule          PASSED   [6%]
test_withdraw_allowance_with_vestingSchedule             PASSED   [7%]
test_init                                                PASSED   [8%]
test_transfer_and_call                                   PASSED   [8%]
test_constructor                                         PASSED   [9%]
test_constructor_huge_fee                                PASSED   [10%]
test_currentRate                                         PASSED   [11%]
test_setFeeBasisPoints                                   PASSED   [12%]
test_add_role                                            PASSED   [12%]
test_grant_role                                          PASSED   [13%]
test_revoke_role                                         PASSED   [14%]
test_renounce_role                                       PASSED   [15%]
test_add_role_functions                                  PASSED   [16%]
test_remove_role_functions                               PASSED   [16%]
test_call_function                                       PASSED   [17%]
test_has_function                                        PASSED   [18%]
test_get_roles                                           PASSED   [19%]
test_init                                                PASSED   [20%]
test_mint                                                PASSED   [20%]
test_set_base_uri                                        PASSED   [21%]
test_init                                                PASSED   [22%]
test_add_vault                                           PASSED   [23%]
test_deposit                                             PASSED   [24%]
test_deposit_buffered_tokens                             PASSED   [24%]
test_deposit_change                                      PASSED   [25%]
test_update_deposits_profit                              PASSED   [26%]
test_update_deposits_loss                                PASSED   [27%]
test_perform_upkeep                                      PASSED   [28%]
test_add_fee                                             PASSED   [28%]
test_update_fee                                          PASSED   [29%]
test_set_min_deposit_threshold                           PASSED   [30%]
test_upgrade_vaults                                      PASSED   [31%]
test_init                                                PASSED   [32%]
test_raise_alert                                         PASSED   [32%]
test_operator_vault_owner                                XPASS    (Operat...)[
test_deposit                                             PASSED   [34%]
test_init                                                PASSED   [35%]
test_init_configure                                      PASSED   [36%]
test_view_init                                           PASSED   [36%]
test_on_token_transfer                                   PASSED   [37%]
test_on_token_transfer_no_allowance                      PASSED   [38%]
test_stake                                               PASSED   [39%]
test_can_deposit_by_allowance                            PASSED   [40%]
test_withdraw                                            PASSED   [40%]
test_withdraw_exceeds                                    PASSED   [41%]
test_withdraw_profits                                    PASSED   [42%]
test_add_remove_pool                                     PASSED   [43%]
test_remove_pool_active_stake                            PASSED   [44%]
test_set_reserved_mode_active                            PASSED   [44%]
test_set_pool_status                                     PASSED   [45%]
test_set_reserved_space_multiplier                       PASSED   [46%]
test_stake_eth                                           PASSED   [47%]
test_withdraw_eth                                        PASSED   [48%]
test_pool_utilisation                                    PASSED   [48%]
```

```
test_stake_withdraw_fees_too_high                    XFAIL    (S...)[
test_add_pool_issue                                  XFAIL    (Missingchecks
test_constructor                                     PASSED   [51%]
test_setRateConstants                                PASSED   [52%]
test_currentRate                                     PASSED   [52%]
test_init                                            PASSED   [53%]
test_update_reward                                   PASSED   [54%]
test_withdraw                                        PASSED   [55%]
test_withdraw_updated                                PASSED   [56%]
test_on_token_transfer                               PASSED   [56%]
test_init                                            PASSED   [57%]
test_init_config                                     PASSED   [58%]
test_staked                                          PASSED   [59%]
test_rewards_address                                 PASSED   [60%]
test_distribute_token                                PASSED   [60%]
test_distribute_tokens                               PASSED   [61%]
test_distribute_tokens_stable                        PASSED   [62%]
test_add_remove_token                                PASSED   [63%]
test_remove_token_with_non_zero_balance              XFAIL    [64%]
test_constructor                                     PASSED   [64%]
test_distribute_rewards                              PASSED   [65%]
test_update_reward                                   PASSED   [66%]
test_withdraw                                        PASSED   [67%]
test_stake_no_allowance_update_rewards               XFAIL    [68%]
test_constructor                                     PASSED   [68%]
test_perform_up_keep_loss                            PASSED   [69%]
test_perform_up_keep_profit                          PASSED   [70%]
test_constructor                                     PASSED   [71%]
test_mint                                            PASSED   [72%]
test_mint_to_contract                                PASSED   [72%]
test_burn                                            PASSED   [73%]
test_burn_from                                       PASSED   [74%]
test_init                                            PASSED   [75%]
test_stake_fail                                      PASSED   [76%]
test_stake_withdraw                                  PASSED   [76%]
test_strategy_deposit_withdraw                       PASSED   [77%]
test_init_configure                                  PASSED   [78%]
test_add_strategy_bulk                               SKIPPED  [79%]
test_add_strategy_zero                               PASSED   [80%]
test_add_remove_reorder_strategies                   PASSED   [80%]
test_reorder_strategies_duplicated                   PASSED   [81%]
test_remove_strategy_profit                          XFAIL    (SeeLP...)
test_add_update_fee                                  PASSED   [83%]
test_set_liquidity_buffer                            PASSED   [84%]
test_stake_max                                       PASSED   [84%]
test_balance_of_profit                               PASSED   [85%]
test_update_strategy_rewards_profit                  PASSED   [86%]
test_update_strategy_rewards_loss                    PASSED   [87%]
test_deposit_liquidity_no_stake                      PASSED   [88%]
test_deposit_liquidity_full_stake                    PASSED   [88%]
test_stake_withdraw_profit                           PASSED   [89%]
test_stake_withdraw_profit_after_update_strategy     PASSED   [90%]
test_stake_withdraw_loss                             PASSED   [91%]
test_stake_withdraw_loss_after_update_strategy       PASSED   [92%]
test_ownership                                       PASSED   [92%]
test_set_pool_index_router                           PASSED   [93%]
test_set_pool_index                                  PASSED   [94%]
test_get_deposits                                    PASSED   [95%]
test_init                                            PASSED   [96%]
test_deposit                                         PASSED   [96%]
test_set_deposits                                    PASSED   [97%]
test_init                                            PASSED   [98%]
test_on_token_transfer                               PASSED   [99%]
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.

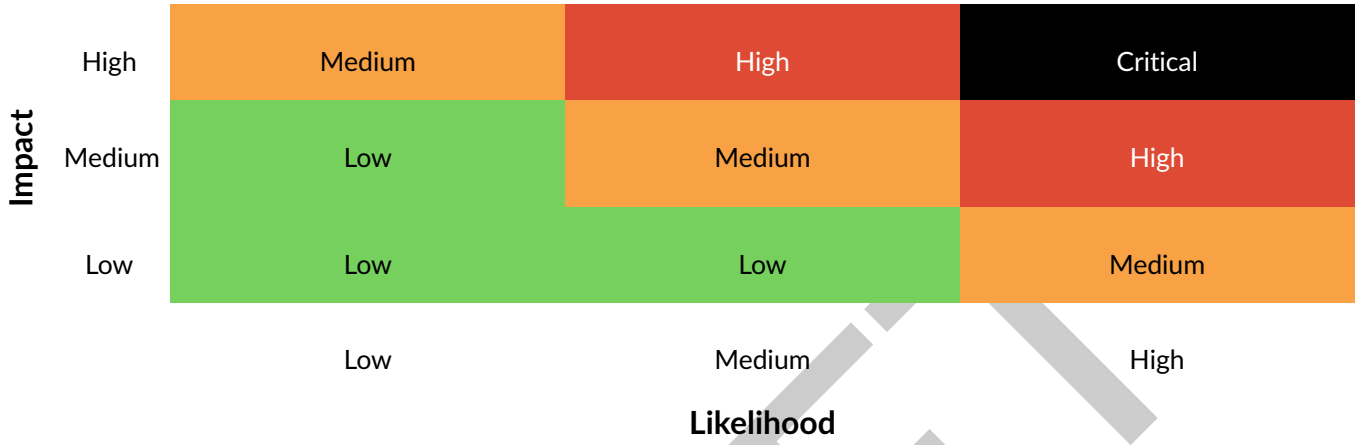

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

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].