LINKPOOL

LinkPool Staking Contracts v4

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 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.

A subsequent round of testing targeted commit 721beb7 and focused solely on verifying whether the previously identified issues had been resolved.

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 20 issues during this assessment. Categorised by their severity:

- High: 1 issue.

- Medium: 1 issue.
- Low: 17 issues.
- Informational: 1 issue.

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	Excessive Fees Can Reduce Principal or Exceed The Reward	High	Resolved
LP4-02	Updating Multiple Strategies At Once May Reduce Receivers' Fees	Medium	Open
LP4-03	Lack of `_token` Validation in `addPool()` May Render Staking Pools Unusable	Low	Open
LP4-04	Integer Overflow in `_liquidityBufferAmount()`	Low	Open
LP4-05	Lack of Overflow and Index Out of Bounds Checks in `updateStrategyRewards()`	Low	Open
LP4-06	Removal of a Staking Pool Reorders Pool's Index in `PoolRouter`	Low	Open
LP4-07	Strategies Can Override Settings to Bypass Liquidity Buffer Limit	Low	Closed
LP4-08	Tokens With Non-standard Decimals Can Incur Extensive Fees	Low	Resolved
LP4-09	Reward Distribution Can Revert When No Allowance Tokens Are Staked	Low	Resolved
LP4-10	Implementation of Algebraic Expression Does Not Match Specification	Low	Resolved
LP4-11	Reward Tokens Can Be Subject to Precision Loss	Low	Closed
LP4-12	Possible Integer Overflow in `getMaxDeposits()`	Low	Open
LP4-13	Lack of `FlatFee` Validation in Constructor	Low	Closed
LP4-14	Lack of Validation of `minDepositThreshold` in `__VaultControllerStrategy_init()`	Low	Resolved
LP4-15	Operator Cannot Be Set	Low	Resolved
LP4-16	Lack of Fees Validation in `StakingPool` and `VaultControllerStrategy`	Low	Open
LP4-17	Removed Pools Maintain Token Spending Allowance	Low	Resolved
LP4-18	Multiple Transactions to Remove Strategies Could Interfere With Each Other	Low	Closed
LP4-19	Gas Inefficient Token Handling	Low	Closed
LP4-20	Miscellaneous General Comments	Informational	Resolved

LP4-01	Excessive Fees Can Reduce Principal or Exceed The Reward		
Asset	`core/StakingPool.sol`, `core/PoolRouter.sol`		
Status	**Resolved:** See Resolution		
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`, 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.

Resolution

The development team has fixed this issue in the commit 721beb7 by removing in the function `updateStrategyRewards()` the part related to the `DelegatorPool`'s fee. The `DelegatorPool`'s fee could now be set during initialization and could be updated later using `StakingPool.updateFee()`. Added to that, the development team has added a `require()` statement inside both `StakingPool.updateFee()` and `StakingPool.addFee()` to limit the `StakingPool` total fee to 50%.

LP4-02	Updating Multiple Strategies At Once May Reduce Receivers' Fees		
Asset	`core/StakingPool.sol`		
Status	Open		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

Updating multiple strategies at once via `updateStrategyRewards()` could prevent `StakingPool` fees' receivers from receiving their rewards.

When profit from the `StakingPool` first strategy is smaller than the loss from the second strategy, the `StakingPool` fee receivers would not get fees on rewards generated by the first strategy.

This is due to `updateStrategyRewards()` keeping track of cumulative pool rewards in `totalRewards` variable (line [**348**]) and distributing fees only if its value is larger than `0` (line [**364**]):

```
348:  int rewards = strategy.depositChange();
          // (...)
          totalRewards += rewards;

          // (...)

364:  if (totalRewards > 0) {
          receivers[receivers.length - 1] = new address[](fees.length);
          feeAmounts[feeAmounts.length - 1] = new uint256[](fees.length);
          totalFeeCount += fees.length;

          for (uint256 i = 0; i < fees.length; i++) {
              receivers[receivers.length - 1][i] = fees[i].receiver;
              feeAmounts[feeAmounts.length - 1][i] = (uint256(totalRewards) * fees[i].basisPoints) / 10000;
              totalFeeAmounts += feeAmounts[feeAmounts.length - 1][i];
          }
      }
```

If cumulative value of all strategies' deposit changes is negative, the fees will not be distributed, regardless if a single strategy actually generated profit.

Recommendations

Do not calculate fees based on a cumulative value of all rewards, as it may omit fees for individual strategies that actually generated profit.

Consider changing implementation of fees calculation and distribution, depending on intended economics.

LP4-03	Lack of `_token` Validation in `addPool()` May Render Staking Pools Unusable		
Asset	`core/PoolRouter.sol`		
Status	**Open**		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

When adding a new staking pool through `PoolRouter.addPool(_token, _stakingPool, _status, _reservedModeActive)`, there are no checks to verify if provided token `_token` matches the one supported by the added staking pool `_stakingPool`. As a result, users may expect an incorrect token to be withdrawn from their wallet when attempting to stake at the given staking pool. Subsequently, any staking attempts for the given staking pool via the `PoolRouter` would always fail.

There are also no checks to verify if a pool already exists in the `PoolRouter`, which will lead to duplicate entries and wasted storage space.

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

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

If the first condition is not checked, staking tokens through `_stake()` will fail. Consider the following scenario:

The owner pairs `Token B` with `StakingPool A`, while `StakingPool A`'s accepted token is `Token A`. When a user calls `PoolRouter.stake()`, `Token B` will be transferred from the user to `PoolRouter` and approved for transfer to `StakingPool A`. Next, when calling `PoolRouter.stake()` the transaction will fail as the `StakingPool A` will attempt to transfer `Token A`, which `PoolRouter` does not have nor has approved for transfers.

If the second check is not performed, the function will store the same `StakingPool` under new index in the `PoolRouter`, creating duplicate entry and wasting storage space. This will also overwrite the `StakingPool`'s `poolIndex` variable with latest index from `PoolRouter`.

Recommendations

1. Do not rely on `_token` value passed as a parameter to `addPool()` function, instead obtain staking token's address directly from `_stakingPool` parameter by calling `_stakingPool.token()`.
2. Prior adding a new staking pool to `PoolRouter`, verify if it has already been added previously. This could be done by calling `getPool(_stakingPool.token, _stakingPool.poolIndex)`.

Resolution

The first recommendation has been implemented in commit 721beb7 - the `addPool()` function does not take `_token` parameter anymore and retrieves staking token's address directly from provided `_stakingPool`:

```
function addPool(
    address _stakingPool,
    PoolStatus _status,
    bool _reservedModeActive
) external onlyOwner {
    address token = IStakingPool(_stakingPool).token();
    // ...
```

The second recommendation has not been implemented - duplicate staking pools can still be added to the router, resulting in an index of a staking pool in `PoolRouter` contract to be changed in staking pool's variable `StakingPool.poolIndex`.

LP4-04	Integer Overflow in `_liquidityBufferAmount()`		
Asset	`core/StakingPool.sol`		
Status	Open		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

An integer overflow in `_liquidityBufferAmount()` exists when `_maxDeposits` and `liquidityBuffer` are within specific values.

Consider the following function from line [455]:

```
function _liquidityBufferAmount(uint256 _maxDeposits) internal view returns (uint256) {
    if (_maxDeposits == type(uint256).max) {
        return 0;
    } else if (liquidityBuffer < 1 ether) {
        return (_maxDeposits * liquidityBuffer) / 10000;
    } else {
        return liquidityBuffer;
    }
}
```

Depending on values of `_maxDeposits` and `liquidityBuffer`, it is possible for the `(_maxDeposits * liquidityBuffer) / 10000` calculation result to overflow `type(uint256).max` and cause a revert.

Recommendations

Implement an additional check to ensure that the result of `(_maxDeposits * liquidityBuffer) / 10000` calculation is within `type(uint256).max`.

This could be done as follows:

```
function _liquidityBufferAmount(uint256 _maxDeposits) internal view returns (uint256) {
       if (_maxDeposits == type(uint256).max) {
           return 0;
       } else if (liquidityBuffer < 1 ether) {
           if (type(uint256).max * 10000) / liquidityBuffer  > _maxDeposits {
             return type(uint256).max - max;
           }

           else {
           return (_maxDeposits * liquidityBuffer) / 10000;
           }
       } else {
           return liquidityBuffer;
       }
```

LP4-05	Lack of Overflow and Index Out of Bounds Checks in `updateStrategyRewards()`		
Asset	`core/StakingPool.sol`		
Status	Open		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

There is a potential for overflow when calculating the `sharesToMint` variable:

```
uint256 sharesToMint = (totalFeeAmounts * totalShares) / (totalStaked - totalFeeAmounts);
```

If the `totalFeeAmounts` and `totalShares` are very large, the division operation could result in an overflow, where the value of `sharesToMint` exceeds the maximum value that can be stored in a `uint256`. Also, an edge case exists where the `totalStaked - totalFeeAmounts` operation can be negative, resulting in an error and revert of the transaction. Additionally, the function does not check whether the index passed to the function is within the bounds of the strategies array (line [344]), which could lead to an out of bound access and revert:

```
function updateStrategyRewards(uint256[] memory _strategyIdxs) public {
// (...)
for (uint256 i = 0; i < _strategyIdxs.length; i++) {
    IStrategy strategy = IStrategy(strategies[_strategyIdxs[i]]);
```

Recommendations

Add overflow protection in the calculation of `sharesToMint`, for example by using the `SafeMath` library or manually checking for overflow conditions before performing the division and subtraction operations.

Implement checks to ensure the index passed to the function is within the bounds of the strategies array.

LP4-06	Removal of a Staking Pool Reorders Pool's Index in `PoolRouter`		
Asset	`core/PoolRouter.sol`		
Status	**Open**		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

Function `removePool()` reorders pools' index which may confuse stakers.

Function `removePool()` removes a pool from `PoolRouter`. The removed pool is identified by two inputs, namely `_token` and `_index`. The function then reorders the pools' index by replacing the removed pool with the last pool in `pools` array. This operation may confuse stakers as the index of the pool they are staking to will change.

Consider the following scenario. A staker stakes `Token A` to `StakingPool A` with `_index` of 5 which is the last index in the `pools` array. The owner then removes a `pool` with `_index` of 3. This operation remaps `StakingPool A`'s `_index` from 5 to 3. If later the owner adds a new pool `StakingPool Z` that accepts `Token A` as a staking token to `PoolRouter`, then the new pool `StakingPool Z` will have an index of 5.

Later if the staker intends to stake to `StakingPool A` and does not realise the index change, the staker will stake `Token A` to `StakingPool Z` instead of `StakingPool A`. If the staker wishes to withdraw from `StakingPool A`, the staker will have to withdraw from `StakingPool Z` instead.

Recommendations

Consider refraining from reordering the `pools`' index when removing a pool from `PoolRouter`. If this occurs, make sure the UI keeps track of the index change and informs the stakers.

LP4-07	Strategies Can Override Settings to Bypass Liquidity Buffer Limit		
Asset	`core/StakingPool.sol`, `core/base/Strategy.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

The staking pool contract does not enforce its `liquidityBuffer` setting, which may lead to reduction of pool liquidity and cause unnecessary gas expenditure for small withdrawals.

Staking pools have `liquidityBuffer` setting, which purpose is to increase the max staking limit of the pool by always keeping a percentage of the staked tokens as liquid within the pool. However, the staking pool contract does not enforce this value in its implementation.

When deciding how many tokens are available to deposit into strategies `StakingPool.depositLiquidity` simply calls `token.balanceOf(address(this))` on line [**378**] and then distributes this value amongst the strategy contracts, without taking `liquidityBuffer` into consideration. The only potential distribution limits are enforced by the strategy contract itself via `Strategy.canDeposit()`.

Recommendations

Enfore `liquidityBuffer` setting in `StakingPool` contract.

One possible solution could be to change line [**378**] of `StakingPool.depositLiquidity` to:

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

It is also recommended to enforce similar checks on `liquidityBuffer` within the `_withdrawLiquidity()` function.

Resolution

This issue has been acknowledged by the development team.

LP4-08	Tokens With Non-standard Decimals Can Incur Extensive Fees		
Asset	`core/StakingPool.sol`, `core/PoolRouter.sol`, `core/DelegatorPool.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

Due to fixed precision multiplier of `1e18` in `PoolRouter.poolUtilisation()`, the `DelegatorPool` fees can easily reach up to 95% for tokens with non standard decimals.

Consider a scenario where a staking token in the `StakingPool` has `decimals = 6`, `StakingPool.totalSupply()` and `StakingPool.getMaxDeposits()` will both be in the scale of `1e6`.

Subsequently, `PoolRouter.poolUtilisation()` on line [141] will return higher value because `totalSupply` is multiplied by a fixed precision, larger than the token's:

```
return (maxDeposits > totalSupply) ? (1e18 * totalSupply) / maxDeposits : 1 ether;
```

Recommendations

Do not used fixed precision multipliers and ensure the implementation handles dealing with non-standard decimal tokens.

This can be done by calling `token.decimals()` and using this value in relevant calculations.

Resolution

The function `PoolRouter.poolUtilisation()` has been removed in commit 721beb7. Current fee calculations do not take into account pool utilisation or percentage of allowance token staked.

LP4-09	Reward Distribution Can Revert When No Allowance Tokens Are Staked		
Asset	core/DelegatorPool.sol, core/PoolRouter.sol, core/RewardsPool.sol, core/StakingPool.sol		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Reward distribution reverts if no allowance tokens are staked. *Note, this is contrary to what has been communicated by the development team - it should be possible for staking pools to operate with no requirements for allowance tokens at all.*

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.

Resolution

The development team has addressed this issue in the commit 721beb7 by removing in the function `updateStrategyRewards()` the part related to the `DelegatorPool`'s fee. The `DelegatorPool` is considered now as `StakingPool` fee receiver for reserved mode pools. As a result, pools which are not in reserved mode will not revert when calling `StakingPool.updateStrategyRewards()` if nothing is staked in the `DelegatorPool`.

LP4-10	Implementation of Algebraic Expression Does Not Match Specification		
Asset	core/feeCurves/RampUpCurve.sol		
Status	**Resolved:** See Resolution		
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)`.

Resolution

The contract `RampUpCurve.sol` has been removed in commit 721beb7.

LP4-11	Reward Tokens Can Be Subject to Precision Loss		
Asset	`core/RewardsPool.sol`		
Status	**Closed:** Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Because of a fixed precision multiplier, in extreme cases, calculated rewards 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, 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.

Resolution

This issue has been acknowledged by the development team.

LP4-12	Possible Integer Overflow in `getMaxDeposits()`		
Asset	`core/StakingPool.sol`		
Status	Open		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `getMaxDeposits()` function calls multiple contracts and adds their return values to a `max` variable. It is possible the values could overflow the maximum limit and cause a revert.

Additionally, approach of looping through a potentially large array will incur ever increasing gas cost and may get to the point where the function will revert due to "out of gas" errors.

```
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, and as such would cause a revert. As this function is invoked in multiple instances throughout the project, there would be a significant loss of functionality if this were to occur.

Note, this issue is partially 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

To address `max` overflow, 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).

To address the gas cost issue, instead of looping through the large array of elements, keep track of `max` variable and every time a new strategy is added to the `strategies` array, increment the locally tracked `max` value with `strategy.getMaxDeposits()`. This way when `getMaxDeposits()` is called, simply `max` value is returned without iterating through the entire array.

Note, with the above solution, if `maxDeposits` value was updated after adding the strategy to the staking pool, the `max` value in the staking pool would also need to be updated accordingly.

Resolution

The recommendation has been implemented in commit 721beb7. The function `StakingPool._maxDepositsWithoutBuffer()` now has a condition that will return `type(uint256).max` if `strategy.getMaxDeposits() >= type(uint256).max - max` is true.

Gas efficiency recommendation has not yet been implemented - the `max` value is still obtained by looping through the entire array.

LP4-13	Lack of `FlatFee` Validation in Constructor		
Asset	`core/feeCurves/FlatFee.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

It is possible to set `feeBasisPoint` to be larger than 95% on the `FlatFee` contract construction.

THe contract owner can modify the flat fee via `setFeeBasisPoints()` and there is a check to prevent the fee going over 95%. However, the same check is not implemented on setting the fee when the contract is first deployed:

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

In contrast, `setFeeBasisPoints()` enforces the following:

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

Recommendations

Implement the same checks in both `setFeeBasisPoints()` and the `constructor()` to ensure `feeBasisPoints` is within a valid range.

Resolution

The contract `FlatFee.sol` has been removed in commit 721beb7.

LP4-14	Lack of Validation of `minDepositThreshold` in `__VaultControllerStrategy_init()`		
Asset	`linkStaking/base/VaultControllerStrategy.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

There are no checks implemented when the contract is first initialised to ensure `minDepositThreshold` is greater than `vaultMinDeposits`.

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`. This is not enforced on initialisation.

Recommendations

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

Resolution

The recommendation has been implemented in commit 721beb7.

LP4-15	Operator Cannot Be Set		
Asset	`linkStaking/OperatorVault.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

It is not possible to set 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 set

Recommendations

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

Resolution

The recommendation has been implemented in commit 721beb7. The contract `OperatorVCS` now has a function `setOperator()` to set the `operator` of the `OperatorVault`.

LP4-16	Lack of Fees Validation in `StakingPool` and `VaultControllerStrategy`		
Asset	`core/StakingPool.sol`, `linkStaking/base/VaultControllerStrategy.sol`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

It is possible to set total fee to be over 50% on `StakingPool` and `VaultControllerStrategy` contracts initialization.

In `StakingPool` and `VaultControllerStrategy` contracts, there is a check in `addFee()` and `updateFee()` functions to prevent the total fee going over 50%. However, the same check is not implemented on the fees when the contract is first initialised.

Recommendations

Implement the same checks in `VaultControllerStrategy.__VaultControllerStrategy_init()` and `StakingPool.initialize()` as in the `addFee()` and `updateFee()` functions:

```
require(_totalFeesBasisPoints() <= 5000, "Total fees must be <= 50%");
```

LP4-17	Removed Pools Maintain Token Spending Allowance		
Asset	`core/PoolRouter.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

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.removePool()`.

Resolution

The recommendation has been implemented in commit 721beb7.

LP4-18	Multiple Transactions to Remove Strategies Could Interfere With Each Other		
Asset	`core/StakingPool.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

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 partially 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.

Resolution

This issue has been acknowledged by the development team.

LP4-19	Gas Inefficient Token Handling		
Asset	`core/base/RewardsPoolController.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `RewardsPoolController` contract loops through all of its supported tokens and calls every single one for the balance and withdrawable rewards. When looping through multiple elements without knowing in advance how many elements to loop through, the number of iterations can grow beyond the block gas limit.

Also, if there is any issue with one of the balance or rewards checks causing a revert, the entire transaction will fail. `RewardsPoolController` contains the following loops on line [**73**] and line [**220**]:

```
function tokenBalances() external view returns (address[] memory, uint256[] memory) {
    uint256[] memory balances = new uint256[](tokens.length);

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

    return (tokens, balances);
}

// (...)

function withdrawableRewards(address _account) external view returns (uint256[] memory) {
    uint256[] memory withdrawable = new uint256[](tokens.length);

    for (uint256 i = 0; i < tokens.length; i++) {
        withdrawable[i] = tokenPools[tokens[i]].withdrawableRewards(_account);
    }

    return withdrawable;
}
```

Additionally, `removeToken()` and `_updateRewards()` on line [**260**] and line [**309**] follow the same logic of iterating through all elements of `tokens` array:

```
function removeToken(address _token) external onlyOwner {
    require(isTokenSupported(_token), "Token is not supported");

    IRewardsPool rewardsPool = tokenPools[_token];
    delete (tokenPools[_token]);
    for (uint256 i = 0; i < tokens.length; i++) {
        if (tokens[i] == _token) {
            tokens[i] = tokens[tokens.length - 1];
            tokens.pop();
            break;
        }
    }

    emit RemoveToken(_token, address(rewardsPool));
}

// (...)

function _updateRewards(address _account) internal {
    for (uint256 i = 0; i < tokens.length; i++) {
        tokenPools[tokens[i]].updateReward(_account);
    }
}
```

If the queries are performed on a rewards pool with large number of supported tokens, it may result in high gas costs, potentially exceeding the block gas limit or yielding significant gas costs.

Note, the `tokenBalances()` and `withdrawableRewards()` functions did not appear to be called internally within the code and might be primarily intended for use in external user interfaces. However, this does not prevent them from being called within a transaction.

Recommendations

For `tokenBalances()` and `withdrawableRewards()` consider only returning array of tokens via `supportedTokens()`, instead of querying all token balances and rewards from the `RewardsPoolController` contract. External user interfaces can then individually query balances, rewards, and any other information about the tokens themselves.

Alternatively, to avoid iterating through array elements, particularly for `removeToken()` and `_updateRewards()`, consider implementing `tokens` as a mapping instead of an array. For `_updateRewards()`, an additional mapping of accounts to tokens would need to be created to make locating tokens linked to a given account a lot quicker, rather than iterating through an entire array of all tokens.

Check balances within a `try` block to prevent a revert from one token contract blocking the entire function call and reverting the transaction.

Resolution

This issue has been acknowledged by the development team.

LP4-20	Miscellaneous General Comments
Asset	`contracts/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

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

1. **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.

Resolution

The development team has acknowledged issues detailed above and addressed where deemed appropriate in commit 721beb7. In particular:

1. Note has been acknowledged.
2. The contract `RampUpCurve.sol` has been removed.
3. (a) Note has been acknowledged.
   (b) The contract `FlatFee.sol` has been removed.
4. The necessary check has been added to the function `PoolRouter.stakeETH()`.
5. Note has been acknowledged.
6. The contract `FlatFee.sol` has been removed.
7. The unreachable code has been removed from the function `DelegatorPool.withdrawAllowance()`.
8. A getter function `getFees()` has been added to the contract `VaultControllerStrategy.sol`.
9. The modifier `poolExists` has been added to the function `PoolRouter.setReservedMode()`.
10. Note has been acknowledged.
11. The Typo has been fixed.
12. Note has been acknowledged.
13. The necessary requirement statement has been added to `addFee()` and `updateFee()` to limit the value of the fees.

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   [5%]
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   [9%]
test_add_role                                           PASSED   [10%]
test_grant_role                                         PASSED   [11%]
test_revoke_role                                        PASSED   [11%]
test_renounce_role                                      PASSED   [12%]
test_add_role_functions                                 PASSED   [13%]
test_remove_role_functions                              PASSED   [14%]
test_call_function                                      PASSED   [15%]
test_has_function                                       PASSED   [16%]
test_get_roles                                          PASSED   [17%]
test_init                                               PASSED   [17%]
test_mint                                               PASSED   [18%]
test_set_base_uri                                       PASSED   [19%]
test_init                                               PASSED   [20%]
test_add_vault                                          PASSED   [21%]
test_deposit                                            PASSED   [22%]
test_deposit_buffered_tokens                            PASSED   [23%]
test_deposit_change                                     PASSED   [23%]
test_update_deposits_profit                             PASSED   [24%]
test_update_deposits_loss                               PASSED   [25%]
test_perform_upkeep                                     PASSED   [26%]
test_add_fee                                            PASSED   [27%]
test_update_fee                                         PASSED   [28%]
test_set_min_deposit_threshold                          PASSED   [29%]
test_upgrade_vaults                                     PASSED   [29%]
test_pending_fees                                       PASSED   [30%]
test_init                                               PASSED   [31%]
test_raise_alert                                        PASSED   [32%]
test_set_operator                                       PASSED   [33%]
test_deposit                                            PASSED   [34%]
test_init                                               PASSED   [35%]
test_init_configure                                     PASSED   [35%]
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   [41%]
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   [45%]
test_set_pool_status                                    PASSED   [46%]
test_set_reserved_space_multiplier                      PASSED   [47%]
test_stake_eth                                          PASSED   [47%]
test_withdraw_eth                                       PASSED   [48%]
test_stake_withdraw_staked_amount_equal_max             PASSED   [49%]
test_add_pool_issue                                     XFAIL    (Missingchecks
test_init                                               PASSED   [51%]
test_update_reward                                      PASSED   [52%]
```

```
test_withdraw                                           PASSED   [52%]
test_withdraw_updated                                   PASSED   [53%]
test_on_token_transfer                                  PASSED   [54%]
test_init                                               PASSED   [55%]
test_init_config                                        PASSED   [56%]
test_staked                                             PASSED   [57%]
test_rewards_address                                    PASSED   [58%]
test_distribute_token                                   PASSED   [58%]
test_distribute_tokens                                  PASSED   [59%]
test_distribute_tokens_stable                           PASSED   [60%]
test_add_remove_token                                   PASSED   [61%]
test_constructor                                        PASSED   [62%]
test_distribute_rewards                                 PASSED   [63%]
test_update_reward                                      PASSED   [64%]
test_withdraw                                           PASSED   [64%]
test_constructor                                        PASSED   [65%]
test_perform_up_keep_loss                               PASSED   [66%]
test_perform_up_keep_profit                             PASSED   [67%]
test_constructor                                        PASSED   [68%]
test_mint                                               PASSED   [69%]
test_mint_to_contract                                   PASSED   [70%]
test_burn                                               PASSED   [70%]
test_burn_from                                          PASSED   [71%]
test_init                                               PASSED   [72%]
test_stake_fail                                         PASSED   [73%]
test_stake_withdraw                                     PASSED   [74%]
test_strategy_deposit_withdraw                          PASSED   [75%]
test_init_configure                                     PASSED   [76%]
test_add_strategy_bulk                                  SKIPPED  [76%]
test_add_strategy_zero                                  PASSED   [77%]
test_add_remove_reorder_strategies                      PASSED   [78%]
test_reorder_strategies_duplicated                      PASSED   [79%]
test_remove_strategy_profit                             XFAIL    (SeeLP...)
test_add_update_fee                                     PASSED   [81%]
test_set_liquidity_buffer                               PASSED   [82%]
test_stake_max                                          PASSED   [82%]
test_balance_of_profit                                  PASSED   [83%]
test_update_strategy_rewards_profit                     PASSED   [84%]
test_update_strategy_rewards_loss                       PASSED   [85%]
test_deposit_liquidity_no_stake                         PASSED   [86%]
test_deposit_liquidity_full_stake                       PASSED   [87%]
test_stake_withdraw_profit                              PASSED   [88%]
test_stake_withdraw_profit_after_update_strategy        PASSED   [88%]
test_stake_withdraw_loss                                PASSED   [89%]
test_stake_withdraw_loss_after_update_strategy          PASSED   [90%]
test_ownership                                          PASSED   [91%]
test_set_pool_index_router                              PASSED   [92%]
test_set_pool_index                                     PASSED   [93%]
test_get_deposits                                       PASSED   [94%]
test_stake_two_strategies                               XFAIL    (Updating...)[
test_init                                               PASSED   [95%]
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.

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

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

References

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

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