Liquid Collective Security Review

Auditors

Saw-Mon and Natalie, Lead Security Researcher

Xiaoming90, Security Researcher

Report prepared by: Pablo Misirov

July 28, 2023

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Liquid Collective is a multichain capable enterprise-grade liquid staking protocol, launching first on Ethereum. It allows institutional investors to stake and earn staking rewards while evidencing ownership of staked tokens in the form of a liquid staking token. Liquid Collective offers a solution that caters to the needs of institutions including:

- KYC / AML allowlisting process for all participants (including validators).
- Top performing node operators with multi-cloud, multi-region, and multi-client infrastructure.
- Governance by a broad and dispersed collective of industry participants.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Liquid Collective according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 3 days in total, Liquid Collective engaged with Spearbit to review
3 code changes since the last engagement. In this period of time a total of **4** issues were found.

Summary

Project Name	Liquid Collective
Target 1	PR 211
Target 2	PR 206
Target 3	PR 202
Commit	638e03...3dd3
Type of Project	Liquid Staking, DeFi
Audit Timeline	May 22 - May 24

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	0	0	0
Gas Optimizations	0	0	0
Informational	4	3	1
Total	**4**	**3**	**1**

5 Findings

5.1 Informational

5.1.1 Calculation of `CurrentValidatorExitsDemand` and `TotalValidatorExitsRequested` using unsolicited exits can happen at the end of `_setStoppedValidatorCounts(...)`

Severity: Informational

Context:

- OperatorsRegistry.1.sol#L541-L546
- OperatorsRegistry.1.sol#L569-L574

Description: Calculation of `CurrentValidatorExitsDemand` and `TotalValidatorExitsRequested` using unsolicited exits can happen at the end of `_setStoppedValidatorCounts(...)` to avoid extra operations like taking minimum per iteration of the loops.

Note that:

$$a_n = a_{n-1} - \min(a_{n-1}, b_n) \Rightarrow a_n = a_0 - \min(a_0, \sum_{i=1}^{n} b_n) = \max(0, a_0 - \sum_{i=1}^{n} b_n)$$

Recommendation: Calculate the sum of all unsolicited exits and then add that to `TotalValidatorExitsRequested` and use the above formula to calculate `CurrentValidatorExitsDemand` at after the `2` for loops.

Liquid Collective: Fixed in `86d45e72e83de8ead43b8b5f85fa583d64599330`.

Spearbit: Fixed.

5.1.2 Define a new internal function to update `TotalValidatorExitsRequested`

Severity: Informational

Context:

- OperatorsRegistry.1.sol#L584-L585
- OperatorsRegistry.1.sol#L848-L849

Description/Recommendation: It would be best to refactor the logic of updating `TotalValidatorExitsRequested` and emitting the relevant event by introducing the new internal function:

```
function _setTotalValidatorExitsRequested(uint256 _currentValue, uint256 _newValue) internal {
   TotalValidatorExitsRequested.set(_newValue);
   emit SetTotalValidatorExitsRequested(_currentValue, _newValue);
}
```

Liquid Collective: Fixed in `18fa86eb117431bb2526d0708a359226fde6678a`.

Spearbit: Fixed.

5.1.3 use `_setCurrentValidatorExitsDemand`

Severity: Informational

Context:

- OperatorsRegistry.1.sol#L467
- OperatorsRegistry.1.sol#L589-L592

Description: If an update is needed for `CurrentValidatorExitsDemand` in `_setStoppedValidatorCounts(...)`, the internal function `_setCurrentValidatorExitsDemand` is not used.

Recommendation: Make sure `_setCurrentValidatorExitsDemand` is used whenever an update for `CurrentValidatorExitsDemand` is required

Liquid Collective: Fixed in `b39846d23642f833246d0d335c4ad2930ecb515e`.

Spearbit: Fixed.

5.1.4 Changes to the emission of `RequestedValidatorExits` event during catch-up

Severity: Informational

Context:

- OperatorsRegistry.1.sol#L488
- OperatorsRegistry.1.sol#L546

Description: The event log will be different between the old and new implementations.

In the old implementation, the latest `RequestedValidatorExits` event in the logs will always contain the most up-to-date count of requested exits (`count`) of an operator after a "catch-up" attempt. This is because a new `RequestedValidatorExits` event with the up-to-date `currentStoppedCount` is emitted at the end of the async `requestValidatorExits` function call.

However, in the new implementation, the latest `RequestedValidatorExits` event in the logs contains the outdated or previous `count` of an operator after a "catch-up" attempt since a new `RequestedValidatorExits` event is not emitted at the end of the Oracle reporting transaction.

If any off-chain component depends on the latest `RequestedValidatorExits` event in the logs to determine the count of requested exits (`count`), it might potentially cause the off-chain component to read and process outdated information. For instance, an operator's off-chain component might be reading the `count` within the latest `RequestedValidatorExits` event in the logs and comparing it against its internal counter to decide if more validators need to be exited.

The following shows the discrepancy between the events emitted between the old and new implementations.

Catch-up implementation in the previous design

1) Catch-up was carried out async when someone called the `requestValidatorExits > _pickNextValidatorsToExitFromActiveOperators` function
2) Within the `_pickNextValidatorsToExitFromActiveOperators` function. Assume an operator called $oper_a$ and its `currentRequestedExits` is less than the `currentStoppedCount`. It will attempt to "catch-up" by performing the following actions:
    1) Emit $UpdatedRequestedValidatorExitsUponStopped(oper_a, currentRequestedExits, currentStoppedCount)$ event.
    2) Let x be the no. of validator count to "catch-up" ($x = currentStoppedCount - currentRequestedExits$)
    3) $oper_a.picked$ will be incremented by x. Since $oper_a.picked$ has not been initialized yet, $oper_a.picked = x$
3) Assume that the $oper_a$ is neither the operator with the highest validation count nor the operator with the second highest. As such, $oper_a$ is not "picked" to exit its validators

4) Near the end of the `_pickNextValidatorsToExitFromActiveOperators` function, it will loop through all operators that have $operator.picked > 0$ and perform some actions. The following actions will be performed against $oper_a$ since $oper_a.picked > 0$:

   1) Emit *RequestedValidatorExits*($oper_a$, *currentStoppedCount*) event

   2) Set $oper_a$.*requestedExits* = *currentStoppedCount*.

5) After the transaction, two events were emitted for $oper_a$ to indicate a catch-up had been attempted.

   - *UpdatedRequestedValidatorExitsUponStopped*($oper_a$, *currentRequestedExits*, *currentStoppedCount*)

   - *RequestedValidatorExits*($oper_a$, *currentStoppedCount*)

Catch-up implementation in the new design

1. Catch-up was carried out within the `_setStoppedValidatorCounts` function during Oracle reporting.

2. Let `_stoppedValidatorCounts[idx]` be the `currentStoppedCount` AND `operators.requestedExits` be `currentRequestedExits`

3. Assume an operator called $oper_a$ and its `currentRequestedExits` is less than the `currentStoppedCount`. It will attempt to "catch-up" by performing the following actions:

   1. Emit *UpdatedRequestedValidatorExitsUponStopped*($oper_a$, *currentRequestedExits*, *currentStoppedCount*) event.

   2. Set $oper_a$.*requestedExits* = *currentStoppedCount*.

4. After the transaction, only one event was emitted for $oper_a$ to indicate a catch-up had been attempted.

   - *UpdatedRequestedValidatorExitsUponStopped*($oper_a$, *currentRequestedExits*, *currentStoppedCount*)

In addition, as per the comment below, it was understood that unsolicited exits are considered as if exit requests were performed for them. In this case, the latest `RequestedValidatorExits` event in the logs should reflect the most up-to-date count of exit requests for an operator including unsolicited exits at any time.

```
File: OperatorsRegistry.1.sol
573:                // we decrease the demand, considering unsollicited exits as if the exit requests
↪   were performed for them
574:                vars.currentValidatorExitsDemand -= LibUint256.min(unsollicitedExits,
↪   vars.currentValidatorExitsDemand);
```

Recommendation: Consider updating the implementation to ensure that the latest `RequestedValidatorExits` event in the logs contains the most up-to-date count of exit requests for an operator including unsolicited exits at any time.

Liquid Collective: We indeed considered that emitting `RequestedValidatorExits` on catch-up could confuse Node Operators because we are not requesting them to exit.

So what we can consider is

- `RequestedValidatorExits` is always emitted to signal a NO to perform the action to exit 1 or more validator key.

- `UpdatedRequestedValidatorExitsUponStopped` is emitted so indexers/off-chain system can update requested values but does not signal NOs.

Spearbit: Acknowledged.