Liquid Collective PRD Security Review

Auditors

Optimum, Lead Security Researcher

Saw-mon and Natalie, Lead Security Researcher

Xiaoming90, Security Researcher

Ellahi, Junior Security Researcher

Report prepared by: Noah Marconi

October 3, 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 the secure liquid staking standard: a protocol designed to meet the needs of institutions, built and run by a collective of leading web3 teams. Liquid Collective will be governed in a decentralized manner by a broad and dispersed community of industry participants.

Disclaimer: This review does not guarantee against a hack. It is a snapshot in time of commit 4618fe...224454 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 5 days in total, Liquid Collective engaged with Spearbit to review the liquid collective protocol. In this period of time a total of **12** issues were found.

Summary

Project Name	Liquid Collective
Repository	liquid collective protocol
Commit	4618fe...224454
Type of Project	Liquid Staking, DeFi
Audit Timeline	Sept. 6th - Sept. 12th
Two week fix period	Sept. 12th - Sept. 26th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	5	4	1
Gas Optimizations	2	2	0
Informational	5	4	1
Total	**12**	**10**	**2**

5 Findings

5.1 Low Risk

5.1.1 Schedule amounts cannot be revoked or released

Severity: Low Risk

Context: TLC_globalUnlockScheduleMigration.sol#L62-L72, ERC20VestableVotesUpgradeable.1.sol#L432-L434, ERC20VestableVotesUpgradeable.1.sol#L303-L306

Description: The migration for schedule ids `9` to `12` has the following parameters:

```
// 9 -> 12
migrations[3] = VestingScheduleMigration({
    scheduleCount: 4,
    newStart: 0,
    newEnd: 1656626400,
    newLockDuration: 72403200,
    setCliff: true,
    setDuration: true,
    setPeriodDuration: true,
    ignoreGlobalUnlock: false
});
```

The current start is `7/1/2022 0:00:00` and the updated/migrated end value would be `6/30/2022 22:00:00`, this will cause `_computeVestedAmount(...)` to always return `0` where one is calculating the released amount due to capping the time by the end timestamp. And thus tokens would not be able to be released.

Also these tokens cannot be revoked since the set [start, end] where end $<$ start would be empty.

Recommendation: Perhaps we should make sure that the new end timestamp is at least `start + cliffDuration + delta` (for partial release) or `start + duration` (for full release).

Liquid Collective: Fixed in commit b66cc8.

Spearbit: Fixed in commit b66cc8 by ensuring that `end = start + duration`.

5.1.2 A revoked schedule might be able to be fully released before the 2 year global lock period

Severity: Low Risk

Context: ERC20VestableVotesUpgradeable.1.sol#L410-L412, ERC20VestableVotesUpgradeable.1.sol#L458

Description: The `unlockedAmount` calculated in `_computeGlobalUnlocked(...)` is based on the original `scheduledAmount`. If a creator revokes its revocable vesting schedule and change the end time to a new earlier date, this formula does not use the new effective amount (the total vested amount at the new end date). And so one might be able to release the vested tokens before 2 years after the lock period.

Recommendation: If it is required for a beneficiary to only be able to release the effective total vested amount (even after revoking) after `2` years from the lock end date, the logic in `_releaseVestingSchedule(_index)` needs to be updated to take into consideration the updated effective vested amount at the end of the schedule and not the original schedule amount.

Liquid Collective:

Client confirms this is the expected behavior.

Spearbit: Acknowledged.

5.1.3 Unlock date of certain vesting schedules does not meet the requirement

Severity: Low Risk

Context: TLC_globalUnlockScheduleMigration.sol#L359

Description: All vesting schedules should have the unlock date (start + lockDuration) set to `16/10/2024 0:00 GMT+0` post-migration.

The following is the list of vesting schedules whose unlock date does not meet the requirement post-migration:

Index	Unlock Date
19,21,23	16/10/2024 9:00 GMT+0
36-60	16/10/2024 22:00 GMT+0

Recommendation: Update the affected vesting schedules to ensure that the unlock date is set to `16/10/2024 0:00 GMT+0` after the migration.

Liquid Collective: Fixed in commit 340d9f.

Spearbit: Fixed in commit 340d9f by implementing the auditor's recommendation to update the unlock date of the affected vesting schedules to `16/10/2024 0:00 GMT+0` post-migration.

5.1.4 `ERC20VestableVotesUpgradeableV1._computeVestingReleasableAmount`: Users with `VestingSchedule.releasedAmount > globalUnlocked` will be temporarily denied of service

Severity: Low Risk

Context: ERC20VestableVotesUpgradeable.1.sol#L413

Description: The current version of the code introduces a new concept; global unlocking. The idea is that wherever `IgnoreGlobalUnlockSchedule` is set to `false`, the releasable amount will be the minimum value between the original vesting schedule releasable amount and the global unlocking releasable amount (the constant rate of `VestingSchedule.amount / 24` for each month starting at the end of the locking period). The implementation ,however, consists of an accounting error caused by a wrong implicit assumption that during the execution of `_computeVestingReleasableAmount` `globalUnlocked` should not be less than `releasedAmount`. In reality, however, this state is possible for users that had already claimed vested tokens. In that case `globalUnlocked - releasedAmount` will revert for an underflow causing a delay in the vesting schedule which in the worst case may last for two years.

Originally this issue was meant to be classified as medium risk but since the team stated that with the current deployment, no tokens will be released whatsoever until the upcoming upgrade of the TLC contract, we decided to classify this issue as low risk instead.

Recommendation: Assuming that the intended functionality is that the 1/24 of the total vesting schedule amount should be releasable every month starting from the end of the locking period, regardless of previous claims, then you may want to consider the following proposal for the `_computeVestingReleasableAmount` function. Note that `_vestingSchedule.maxLeftToBeClaimed` should be initialized to `_vestingSchedule.amount - _vestingSchedule.releasedAmount` as part of the migration script.

```
function _computeVestingReleasableAmount(
    VestingSchedulesV2.VestingSchedule storage _vestingSchedule,
    uint256 _time,
    uint256 _index
) internal view returns (uint256) {
    uint256 releasedAmount = _vestingSchedule.releasedAmount;
    uint256 vestedAmount =
        _computeVestedAmount(_vestingSchedule, _time > _vestingSchedule.end ? _vestingSchedule.end :
↪  _time);
    if (vestedAmount > releasedAmount) {
        if (!IgnoreGlobalUnlockSchedule.get(_index)) {
            uint256 globalUnlocked = _computeGlobalUnlocked(
                _vestingSchedule.amount, _time - (_vestingSchedule.start +
↪  _vestingSchedule.lockDuration)
            );
            if(_vestingSchedule.amountAcc < _vestingSchedule.maxLeftToBeClaimed){
                globalUnlocked = LibUint256.min(globalUnlocked, _vestingSchedule.maxLeftToBeClaimed);
                uint256 amount = LibUint256.min(vestedAmount - releasedAmount, globalUnlocked -
↪  _vestingSchedule.amountAcc);
                _vestingSchedule.amountAcc += amount;
                return amount;
            }
            return LibUint256.min(vestedAmount - releasedAmount, globalUnlocked - releasedAmount);
        }
        unchecked {
            return vestedAmount - releasedAmount;
        }
    }

    return 0;
}
```

Please make sure to test this function thoroughly before deployment. In case the described issue is part of the intended functionality, consider adding a custom error for the described underflow and revert.

Liquid Collective: Fixed in commit 9d7cdb.

Spearbit: Fixed in commit 9d7cdb by implementing the auditor's recommendation to add a custom error for the described underflow.

5.1.5 `TlcMigration.migrate`: Missing input validation

Severity: Low Risk

Context: TLC_globalUnlockScheduleMigration.sol#L365-L386

Description: The upcoming change in some of the vesting schedules is going to be executed via the `migrate` function which at the current version of the code is missing necessary validation checks to make sure no erroneous values are inserted.

Recommendation: Consider adding the following post-effects validation checks:

1. Make sure that `VestingSchedule.cliffDuration` can not be longer than the total duration (`VestingSchedule.duration`).
2. `VestingSchedule.end` should not be less than `VestingSchedule.start + VestingSchedule.cliffDuration + delta` (for partial release) or `VestingSchedule.start + VestingSchedule.duration` (for full release).
3. Make sure that all vesting schedules have the unlock date `VestingSchedule.start + VestingSchedule.lockDuration` set to `16/10/2024 0:00 GMT+0`.

Liquid Collective: Fixed in commit 340d9f and commit 4f63c4.

Spearbit: Fixed in commit 340d9f and commit 4f63c4 by implementing the auditor's recommendations.

5.2 Gas Optimization

5.2.1 Optimise the release amount calculation

Severity: Gas Optimization

Context: ERC20VestableVotesUpgradeable.1.sol#L413

Description: In the presence of a global lock schedule one calculates the release amount as:

```
LibUint256.min(vestedAmount - releasedAmount, globalUnlocked - releasedAmount)
```

Recommendation: We can avoid one subtraction by moving the `releasedAmount` out of the `min` function:

```
LibUint256.min(vestedAmount, globalUnlocked) - releasedAmount
```

Note also that `LibUint256.min(vestedAmount, globalUnlocked)` represent the total amount of vested tokens that could have been released at the current timestamp.

Liquid Collective: Fixed in commit 0e8d82.

Spearbit: Fixed in commit 0e8d82.

5.2.2 Use `msg.sender` whenever possible

Severity: Gas Optimization

Context: ERC20VestableVotesUpgradeable.1.sol#L319, ERC20VestableVotesUpgradeable.1.sol#L356, ERC20VestableVotesUpgradeable.1.sol#L383

Description: In this context the parameters `vestingSchedule.{creator, beneficiary}` have already been checked to be equal to `msg.sender`:

```
if (msg.sender != vestingSchedule.X) {
    revert LibErrors.Unauthorized(msg.sender);
}
```

Recommendation: It would be cheaper to avoid reading from storage `vestingSchedule.{creator, beneficiary}` and use `msg.sender` in this context.

It might also make sense to change all occurrences of `msg.sender` to `_msgSender()`.

Liquid Collective: Fixed in commit e852d5.

Spearbit: Fixed in commit e852d5.

5.3 Informational

5.3.1 Test function `testMigrate` uses outdated values for assertion

Severity: Informational

Context: TLC_globalUnlockScheduleMigration.t.sol#L60-L108

Description: In commit `fbcc4ddd6da325d60eda113c2b0e910aa8492b88`, the `newLockDuration` values were updated in `TLC_globalUnlockScheduleMigration.sol`. However, the `testMigrate` function was not updated accordingly and still compares `schedule.lockDuration` to the outdated `newLockDuration` values, resulting in failing assertions.

Recommendation: Update the values being asserted to `schedule.lockDuration` in the `testMigrate` function to match the updated `newLockDuration` values, ensuring the test is up to date and the assertions pass.

Liquid Collective: Fixed in PR 232.

Spearbit: Acknowledged.

5.3.2 Rounding Error in Unlocked Token Amount Calculation at `ERC20VestableVotesUpgradeable.1.sol#L458`

Severity: Informational

Context: ERC20VestableVotesUpgradeable.1.sol#L458

Description: There is a rounding error in calculating the unlocked amount, which may lead to minor discrepancies in the tokens available for release.

Recommendation: To avoid this rounding error, you can change the formula from:

```
uint256 unlockedAmount = (scheduledAmount / 24) * (timeSinceLocalLockEnd / (365 days / 12));
```

to:

```
uint256 unlockedAmount = (scheduledAmount * timeSinceLocalLockEnd) / (24 * (365 days / 12));
```

Liquid Collective: The rounding error is wanted, we don't want to unlock the token linearly but in blocks of 1/24th every month.

Spearbit: Acknowledged

5.3.3 It might take longer than `2` years to release all the vested schedule amount after the lock period ends

Severity: Informational

Context: ERC20VestableVotesUpgradeable.1.sol#L413

Description: It is possible that in the presence of the global lock, releasing the total vested value might take longer than `2` years if the `lockDuration + 2 years` is comparatively small when compared to `duration` (or `start - end`). We just know that after `2` years all the scheduled amount can be released but only a portion of it might have been vested.

Recommendation: If this an expected behaviour it might be useful to at least add a comment mentioning this issue.

Liquid Collective: A comment has been added in commit 3dc76b.

Spearbit: A comment has been added in commit 3dc76b.

5.3.4 `_computeVestingReleasableAmount`'s `_time` input parameter can be removed/inlined

Severity: Informational

Context: ERC20VestableVotesUpgradeable.1.sol#L122-L127, ERC20VestableVotesUpgradeable.1.sol#L341-L348, ERC20VestableVotesUpgradeable.1.sol#L400-L404

Description: At both call sites to `_computeVestingReleasableAmount(...)`, `time` is `_getCurrentTime()`.

Recommendation: One can remove the `_time` input parameter from `_computeVestingReleasableAmount` and instead inline `_getCurrentTime()` in its implementation (unless the devs plan to use this internal function with times other than `_getCurrentTime()`).

Moreover, we can refactor the `time` validity just before the calls to `_computeVestingReleasableAmount(...)` and move them into that function by passing a flag to `_computeVestingReleasableAmount(...)`:

```
function _computeVestingReleasableAmount(
    VestingSchedulesV2.VestingSchedule memory _vestingSchedule,
    bool revokeInvalidTime,
    uint256 _index
) internal view returns (uint256) {
   uint256 time = _getCurrentTime();
    if (time < (vestingSchedule.start + vestingSchedule.lockDuration)) {
       if(revokeInvalidTime) {
          // before lock no tokens can be vested
          revert VestingScheduleIsLocked();
       } else {
          return 0;
       }
    }
   ...
}
```

Liquid Collective: Fixed in commit 42058b.

Spearbit: Fixed in commit 42058b.

5.3.5 Comments and NatSpec

Severity: Informational

Context: ERC20VestableVotesUpgradeable.1.sol#L343, ERC20VestableVotesUpgradeable.1.sol#L396-L404, ERC20VestableVotesUpgradeable.1.sol#L441

Description/Recommendation:

ERC20VestableVotesUpgradeable.1.sol#L343, Might be more accurate to say:

```
- // before lock no tokens can be vested
+ during the locked period no vested tokens can be released by the beneficiary
```

ERC20VestableVotesUpgradeable.1.sol#L396-L404, missing natspec: `@param _index`

ERC20VestableVotesUpgradeable.1.sol#L441, typo `completly` should be `completely`

Liquid Collective: Fixed in commit 1470cd.

Spearbit: Fixed in commit 1470cd.