Liquid Collective Security Review

Auditors

Optimum, Lead Security Researcher

Saw-Mon and Natalie, Lead Security Researcher

Xiaoming90, Security Researcher

Danyal Ellahi, Junior Security Researcher

Matt Eccentricexit, Junior Security Researcher

Report prepared by: Pablo Misirov

May 19, 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 10 days in total, Liquid Collective engaged with Spearbit to review the liquid collective protocol. In this period of time a total of **49** issues were found.

Summary

Project Name	Liquid Collective
Repository	liquid collective protocol
Feature	Withdrawals
Commit	3a23e607...dbae
Type of Project	Liquid Staking, DeFi
Audit Timeline	March 15 - March 28
Two week fix period	March 28 - April 11

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	2	2	0
High Risk	1	1	0
Medium Risk	3	3	0
Low Risk	14	9	5
Gas Optimizations	5	4	1
Informational	24	17	7
Total	**49**	**36**	**13**

5 Remediation Table

The following table contains all issues found during the audit together with its corresponding severity and fix PR

Number	Issue	Severity	PR
6.1.1	`_pickNextValidatorsToExitFromActiveOperators` uses the wrong index to query stopped validator count for operators	Critical	[SPEARBIT/03/23] OperatorsRegistry Fixes
6.1.2	Oracles reports votes are not stored in storage	Critical	[SPEARBIT/03/23] Oracle fixes
6.2.1	Users LsETH might be locked due to out-of-gas error during recursive calls	High	[SPEARBIT/03/23] RedeemManager Fixes
6.3.1	Allowed users can directly transfer their share to `RedeemManager`	Medium	[SPEARBIT/03/23] RedeemManager Fixes
6.3.2	Invariants are not enforced for stopped validator counts	Medium	[SPEARBIT/03/23] OperatorsRegistry Fixes
6.3.3	Potential out of gas exceptions	Medium	[SPEARBIT/03/23] OperatorsRegistry Fixes
6.4.1	The validator count to exit in `_requestExitsBasedOnRedeemDemandAfterRebalancings` assumes that the to-be selected validators are still active and have not been penalised	Low	Acknowledged
6.4.2	Burn `RedeemManager` share first before calling its `reportWithdraw` endpoint	Low	[SPEARBIT/03/23] River Fixes
6.4.3	`OracleManager` allows reporting for the same epoch multiple times, leading to unknown behavior	Low	[SPEARBIT/03/23] Oracle fixes
6.4.4	Missing event emit when user calls `deposit`	Low	[SPEARBIT/03/23] River Fixes
6.4.5	Reset the report data and increment the last epoch id before calling `River's setConsensusLayerData` when a quorum is made	Low	[SPEARBIT/03/23] Oracle fixes
6.4.6	Update `_BufferedExceedingEth` before calling `_sendRedeemManagerExceedingFunds`	Low	[SPEARBIT/03/23] RedeemManager Fixes
6.4.7	Any oracle member can censor almost quorum report variants by resetting its address	Low	[SPEARBIT/03/23] Oracle fixes
6.4.8	Incentive mechanism that encourages operators to respond quickly to exit requests might diminish under certain condition	Low	Acknowledged
6.4.9	`RedeemManager._claimRedeemRequests` transaction sender might be tricked to pay more eth in transaction fees	Low	Acknowledged

6.4.10	Claimable LsETH on the Withdraw Stack could exceed total LsETH requested on the Redeem Queue	Low	[SPEARBIT/03/23] RedeemManager Fixes
6.4.11	An oracle member can resubmit data for the same epoch multiple times if the `quorum` is set to `1`	Low	[SPEARBIT/03/23] Oracle fixes
6.4.12	Report's `validatorsCount's` historical non-decreseness does not get checked	Low	[SPEARBIT/03/23] River Fixes
6.4.13	The report's `slashingContainmentMode` and `bufferRebalancingMode` are decided by the oracle members which affects the exiting strategy of validators	Low	Acknowledged
6.4.14	Anyone can call `depositToConsensusLayer` and potentially push wrong data on-chain	Low	Acknowledged
6.5.1	Calculation of `currentMaxCommittableAmount` can be simplified	Gas Op	[SPEARBIT/03/23] River Fixes
6.5.2	Remove redundant array length check and variable to save gas	Gas Op	[SPEARBIT/03/23] River Fixes
6.5.3	Duplicated events emitted in `River` and `RedeemManager`	Gas Op	[SPEARBIT/03/23] RedeemManager Fixes
6.5.4	`totalRequestedExitsValue's` calculation can be simplified	Gas Op	[SPEARBIT/03/23] OperatorsRegistry Fixes
6.5.5	Report's `bufferRebalancingMode` and `slashingContainmentMode` are only used during the reporting transaction process	Gas Op	Acknowledged
6.6.1	Add more comments/documentation for `ConsensusLayerReport` and `StoredConsensusLayerReport` structs	Informational	[SPEARBIT/03/23] Docs, Styling and Naming
6.6.2	`postUnderlyingBalanceIncludingExits` and `preUnderlyingBalanceIncludingExits` can be removed from `setConsensusLayerData`	Informational	[SPEARBIT/03/23] RedeemManager Fixes
6.6.3	The formula or the parameter names for calculating `currentMaxDailyCommittableAmount` can be made more clear	Informational	[SPEARBIT/03/23] Docs, Styling & Naming
6.6.4	`preExitingBalance` is a rough estimate for signalling the number of validators to request to exit	Informational	[SPEARBIT/03/23] River Fixes
6.6.5	More documentation can be added regarding the `currentMaxDailyCommittableAmount` calculation	Informational	[SPEARBIT/03/23] Docs, Styling & Naming
6.6.6	`BalanceToRedeem` is only non-zero during a report processing transaction	Informational	Acknowledged
6.6.7	Improve clarity on `bufferRebalancingMode` variable	Informational	[SPEARBIT/03/23] River Fixes
6.6.8	Fix code style consistency issues	Informational	[SPEARBIT/03/23] Docs, Styling & Naming

6.6.9	Remove unused constants	Informational	[SPEARBIT/03/23] Oracle fixes
6.6.10	Document what `TotalRequestedExits` can potentially represent	Informational	spearbit audit 03/2023
6.6.11	Operators need to listen to `RequestedValidatorExits` and exit their validators accordingly	Informational	Acknowledged
6.6.12	Oracle members would need to listen to `ClearedReporting` and report their data if necessary	Informational	Acknowledged
6.6.13	The only way for an oracle member to change its report data for an epoch is to reset the reporting process by changing its address	Informational	Acknowledged
6.6.14	For a quorum making CL report the epoch restrictions are checked twice	Informational	Acknowledged
6.6.15	Clear report variants and report position data during the migration to the new contracts	Informational	[SPEARBIT/03/23] Oracle fixes
6.6.16	Remove unused functions from `Oracle`	Informational	[SPEARBIT/03/23] Oracle fixes
6.6.17	`RedeemManager._claimRedeemRequests` - Consider adding the `recipient` to the revert message in case of failure	Informational	[SPEARBIT/03/23] RedeemManager Fixes
6.6.18	`River.1.sol` should not have a permission to call `requestRedeem(uint256 lsETHAmount)`	Informational	[SPEARBIT/03/23] RedeemManager Fixes
6.6.19	Exit validator picking strategy does not consider slashed validator between reported epoch and current epoch	Informational	Acknowledged
6.6.20	Duplicated functions	Informational	[SPEARBIT/03/23] OperatorsRegistry Fixes
6.6.21	Funds might be pulled from CoverageFundV1 even when there has been no slashing incident	Informational	Acknowledged
6.6.22	Update inline documentation	Informational	[SPEARBIT/03/23] Docs, Styling & Naming
6.6.23	Document/mark unused (would-be-stale) storage parameters after migration	Informational	[SPEARBIT/03/23] Docs, Styling & Naming
6.6.24	`pullEth`, `pullELFees` and `pullExceedingEth` do not check for a non-zero amount before sending funds to `River`	Informational	[SPEARBIT/03/23] Recipient Fixes

6 Findings

6.1 Critical Risk

6.1.1 `_pickNextValidatorsToExitFromActiveOperators` uses the wrong index to query stopped validator count for operators

Severity: Critical Risk

Context: OperatorsRegistry.1.sol#L628

Description: In `_pickNextValidatorsToExitFromActiveOperators`, `OperatorsV2.CachedOperator[] memory operators` does not necessarily have the same order as the actual `OperatorsV2`'s operators, since the ones that don't have `_hasExitableKeys` will be skipped (the operator might not be active or all of its funded keys might have been requested to exit). And so when querying the stopped validator counts

```
for (uint256 idx = 0; idx < exitableOperatorCount;) {
    uint32 currentRequestedExits = operators[idx].requestedExits;
    uint32 currentStoppedCount = _getStoppedValidatorsCountFromRawArray(stoppedValidators, idx);
```

one should not use the `idx` in the cached operator's array, but the cached `index` of this array element, as the indexes of `stoppedValidators` correspond to the actual stored operator's array in storage.

Note that when emitting the `UpdatedRequestedValidatorExitsUponStopped` event, the correct index has been used.

Recommendation: The calculation for `currentStoppedCount` needs to be corrected to use `operators[idx].index`

```
uint32 currentStoppedCount = _getStoppedValidatorsCountFromRawArray(stoppedValidators,
↪   operators[idx].index);
```

Also, since this was not caught by test cases, it would be best to add some passing and failing test cases for `_pickNextValidatorsToExitFromActiveOperators`.

Liquid Collective: Fixed in `commit a109a1`.

Spearbit: Verified.

6.1.2 Oracles' reports votes are not stored in storage

Severity: Critical Risk

Context: Oracle.1.sol#L268

Description: The purpose of `Oracle.1.sol` is to facilitate the reporting and quorum of oracles. Oracles periodically add their reports and when consensus is reached the `setConsensusLayerData` function (which is a critical component of the system) is called. However, there is an issue with the current implementation as `ReportVariants` holds the reports made by oracles but `ReportVariants.get()` returns a memory array instead of a storage array, therefore resulting in an increase in votes that will not be stored at the end of the transaction and preventing `setConsensusLayerData` from being called. This is a regression bug that should have been detected by a comprehensive test suite.

Recommendation: Short term, modify `ReportVariants.get()` so that it returns a storage array instead of a memory array. By returning a storage array we can ensure that any increase in votes will be persisted to the blockchain and that `setConsensusLayerData` will be called when a quorum is reached. Long term, make sure to have a comprehensive test suite with high code coverage that is incorporated as part of the CI/CD process.

Liquid Collective: Fixed in `commit f73551` by following the auditor's recommendation.

Spearbit: Verified.

6.2 High Risk

6.2.1 User's LsETH might be locked due to out-of-gas error during recursive calls

Severity: High Risk

Context: RedeemManager.1.sol#L417

Description: Let $W_0, W_1, ... W_7$ represent the withdrawal events in the withdrawal stack.

Let R_0, R_1, R_2 represent the users' redeem requests in the redeem queue.

Assume that Alice is the owner of $R1$. When Alice called the `resolveRedeemRequests` function against $R1$, it will resolve to $W1$.

Next, Alice called the `_claimRedeemRequest` function with $R1$ and its corresponding $W1$.

The `_claimRedeemRequest` will first process $W1$. At the end of the function, it will check if $W1$ matches all the amounts of $R1$. If not, it will call the `_claimRedeemRequest` function recursively with the same request id ($R1$) but increment the withdrawal event id ($W_2 = W_1 + 1$).

The `_claimRedeemRequest` function recursively calls itself until all the amount of redeem request is "expended" or the next withdrawal event does not exist. In the above example, the `_claimRedeemRequest` will be called 7 times with $W_1 ... W_7$, until all the amount of $R1$ is "expended" ($R1.amount == 0$)

However, if the amount of a redeem request is large (e.g. 1000 LsETH), and this redeem request is satisfied by many small chunks of withdrawal events (e.g. one withdrawal event consists of less than 10 LsETH), then the recursion depth will be large. The function will keep calling itself recursively until an out-of-gas error happens. If this happens, there is no way to claim the redemption request, and the user's LsETH will be locked.

In the current implementation, users cannot break the claim into smaller chunks to overcome the gas limit. In the above example, if Alice attempts to break the claim into smaller chunks by first calling the `_claimRedeemRequest` function with $R1$ and its corresponding W_5, the `_isMatch` function within it will revert.

Recommendation: Consider allowing the users to break the claim into smaller chunks to overcome the gas limit.

Liquid Collective: Fixed in `commit e8ca5d`.

Spearbit: Verified.

6.3 Medium Risk

6.3.1 Allowed users can directly transfer their share to `RedeemManager`

Severity: Medium Risk

Context: RedeemManager.1.sol#L280-L281, SharesManager.1.sol#L105 , SharesManager.1.sol#L119

Description: An allowed user can directly transfer its shares to the `RedeemManager` without requesting a redeem. This would cause the withdrawal stack to grow, since the redeem demand (2) which is calculated based on the `RedeemManager`'s share of `LsETH` increases. `RedeemQueue` would be untouched in this case.

In case of an accidental mistake by a user, the locked shares can only be retrieved by a protocol update.

Recommendation: Make sure `ShareManager` would not allow users to directly transfer their shares to the `Redeem-Manager` unless the call to `transferFrom` is made by the `RedeemManager`.

Liquid Collective: Fixed.

Spearbit: This issue has been addressed by introducing a storage variable `RedeemDemand` that would keep track of redeem demand for the redeem manager contract:

- `commit dda655`
- `commit bc3afe`

It does not address the issue that shares can be directly transferred to the redeem manager by a user.

6.3.2 Invariants are not enforced for stopped validator counts

Severity: Medium Risk

Context: OperatorsRegistry.1.sol#L419-L440

Description: `_setStoppedValidatorCounts` does not enforce the following invariants:

- `stoppedValidatorCounts[0] <= DepositedValidatorCount.get()`
- `stoppedValidatorCounts[i]` needs to be a non-decreasing function when viewed on a timeline
- `stoppedValidatorCounts[i]` needs to be less than or equal to the `funded` number of validators for the corresponding operator.

Currently, the oracle members can report values that would break these invariants.

As a consequence, the oracle members can signal the operators to exit more or fewer validators by manipulating the `preExitingBalance` value. And `activeCount` for exiting validators picking algorithm can also be manipulated per operator.

Recommendation: Make sure the invariants mentioned above are enforced on-chain before oracle reports get used and/or stored.

Liquid Collective: Fixed in PR 195.

Spearbit: Verified.

6.3.3 Potential out of gas exceptions

Severity: Medium Risk

Context: OracleManager.1.sol#L432, OperatorsRegistry.1.sol#L529

Description: The purpose of `_requestExitsBasedOnRedeemDemandAfterRebalancings` is to release liquidity for withdrawals made in the `RedeemManager` contract. The function prioritizes liquidity sources, starting with `BalanceToRedeem` and then `BalanceToDeposit`, before asking validators to exit. However, if the validators are needed to release more liquidity, the function uses `pickNextValidatorsToExit` to determine which validators to ask to exit. This process can be quite gas-intensive, especially if the number of validators is large.

The gas consumption of this function depends on several factors, including `exitableOperatorCount`, `stoppedValidators.length`, and the rate of decrease of `_count`. These factors may increase over time, and the `msg.sender` does not have control over them. The function includes two nested loops that contribute to the overall gas consumption, and this can be problematic for certain inputs. For example, if the `operators` array has no duplications and the difference between values is exactly 1, such as `[n, n-1, n-2 ... n-k]` where `n` can be any number and `k` is a large number equals `exitableOperatorCount - 1` and `_count` is also large, the function can become extremely gas-intensive.

The main consequence of such a potential issue is that the function may not release enough liquidity to the `RedeemManager` contract, resulting in partial fulfillment of redemption requests.

Similarly, `_pickNextValidatorsToDepositFromActiveOperators` is also very gas intensive. If the number of desired validators and current operators (including fundable operators) are high enough, `depositToConsensusLayer` is no longer callable.

Recommendation: To address the potential gas consumption issue with `_requestExitsBasedOnRedeemDemandAfterRebalancings`, the following steps can be considered

1. Allow `_requestExitsBasedOnRedeemDemandAfterRebalancings` to be called independently and not just as part of `setConsensusLayerData`.
2. Currently, the `_count` parameter is dependent on several variables that cannot be determined by the caller, including `redeemManagerDemandInEth`, `availableBalanceToRedeem`, `exitingBalance`, and `preExitingBalance`. To improve efficiency, it would be better to allow the caller to specify a value for `_count` and use the minimum value between the caller's input and `validatorCountToExit` to determine the number of validators to exit.

3. It's important to run tests to measure and monitor the worst-case scenario, particularly around the `while (_count > 0) {` section. If necessary, consider replacing the decision algorithm with a more efficient approach to reduce gas consumption.

For `_pickNextValidatorsToDepositFromActiveOperators`:

1. Similar to the previous suggestion, allow it to run independently from `depositToConsensusLayer`.
2. Consider using a min heap to select operators with the lowest active validator count.

Liquid Collective: The process for picking validators to exit upon report quorum is broken into 2 or more transactions now by introducing the following functions:

- `demandValidatorExits`
- `requestValidatorExits`

in PR 195.

Recommendations not applied for `_pickNextValidatorsToDepositFromActiveOperators`.

Spearbit: Verified.

6.4 Low Risk

6.4.1 The validator count to exit in `_requestExitsBasedOnRedeemDemandAfterRebalancings` assumes that the to-be selected validators are still active and have not been penalised.

Severity: Low Risk

Context: River.1.sol#L568-L571

Description: The `validatorCountToExit` is calculated as follows

```
uint256 validatorCountToExit = LibUint256.ceil(
    redeemManagerDemandInEth - (availableBalanceToRedeem + exitingBalance + preExitingBalance),
    DEPOSIT_SIZE
);
```

This formula assumes that the to-be selected validators exit by the `pickNextValidatorsToExit` are:

1. Still active
2. Have not been queued to be exited and
3. Have not been penalized and their balance is at least `MAX_EFFECTIVE_BALANCE`

Recommendation: Just like stopped validator count, the oracles could report back more granular data including consensus layer balances per validator to make sure the exit strategy for the validators roughly guarantees the current redeem demand. This would still not be perfect, since there will always be a lag/delay between the reported data by oracles for a past epoch and the current balances on the validators.

Liquid Collective: This would be pretty hard to implement, we're ok to not have this accuracy and have a protocol able to respond in an async manner to possible deltas between demand and actual volume. Here requesting a 32 ETH exit, if only ~20 are entering the exiting balance buffer, we are going to request another exit on the next report.

Spearbit: Acknowledged.

6.4.2 Burn `RedeemManager`'s share first before calling its `reportWithdraw` endpoint

Severity: Low Risk

Context: River.1.sol#L509-L518

Description: The current implementation of `_reportWithdrawToRedeemManager` calls `RedeemManager`'s `reportWithdraw` and then burns the corresponding shares for the `RedeemManager`

```
// perform a report withdraw call to the redeem manager
redeemManager_.reportWithdraw{value: suppliedRedeemManagerDemandInEth}(suppliedRedeemManagerDemand);

// we burn the shares of the redeem manager associated with the amount of eth provided
_burnRawShares(address(RedeemManagerAddress.get()), suppliedRedeemManagerDemand);
```

Recommendation: To avoid potential future reentrancy from `RedeemManager` it would be best to burn the shares first and then call the `reportWithdraw` endpoint

```
// we burn the shares of the redeem manager associated with the amount of eth provided
_burnRawShares(address(RedeemManagerAddress.get()), suppliedRedeemManagerDemand);

// perform a report withdraw call to the redeem manager
redeemManager_.reportWithdraw{value: suppliedRedeemManagerDemandInEth}(suppliedRedeemManagerDemand);
```

Liquid Collective: Fixed in `commit e47cb8`.

Spearbit: Verified.

6.4.3 OracleManager allows reporting for the same epoch multiple times, leading to unknown behavior.

Severity: Low Risk

Context: OracleManager.1.sol#L463

Description: Currently, it is possible for the oracle to report on the same epoch multiple times, because `_isValidEpoch` checks that the report's `epoch >= LastConsensusLayerReport.get().epoch`.

This can lead the contract to unspecified behavior

- The code will revert if the report increases the balance, not with an explicit check but reverting due to a subtraction underflow, since `maxIncrease == 0` and
- Allowing other code paths to execute to completion.

Recommendation: Explicitly disallow this execution path by using a strictly increasing criterion

```diff
- epoch >= LastConsensusLayerReport.get()
+ epoch > LastConsensusLayerReport.get()
```

Liquid Collective: Fixed in PR 193.

Spearbit: Verified.

6.4.4 Missing event emit when user calls `deposit`

Severity: Low Risk

Context: UserDepositManager.1.sol#L50, River.1.sol#L435

Description: Whenever `BalanceToDeposit` is updated, the protocol should emit a `SetBalanceToDeposit`, but when a user calls `UserDepositManager.deposit`, the event is never emitted which could break tooling.

Recommendation: Instead of setting the value of the variable directly, call `River._setBalanceToDeposit` function since it emits the event

```
/// @notice Sets the balance to deposit, but not yet committed
/// @param newBalanceToDeposit The new balance to deposit value
function _setBalanceToDeposit(uint256 newBalanceToDeposit) internal override(UserDepositManagerV1) {
    emit SetBalanceToDeposit(BalanceToDeposit.get(), newBalanceToDeposit);
    BalanceToDeposit.set(newBalanceToDeposit);
}
```

Liquid Collective: Fixed in PR 196.

Spearbit: Verified.

6.4.5 Reset the report data and increment the last epoch id before calling `River`'s `setConsensusLayerData` when a quorum is made

Severity: Low Risk

Context: Oracle.1.sol#L254-L263

Description: The current implementation of `reportConsensusLayerData` calls `river.setConsensusLayerData(report)` first when a quorum is made then resets the report variant and position data and also it increment the last epoch id afterward

```
// if adding this vote reaches quorum
if (variantVotes + 1 >= quorum) {
    // we push the report to river
    river.setConsensusLayerData(report);
    // we clear the reporting data
    _clearReports();
    // we increment the lastReportedEpoch to force reports to be on the last frame
    LastEpochId.set(lastReportedEpochValue + 1);
    emit SetLastReportedEpoch(lastReportedEpochValue + 1);
}
```

In the future version of the protocol there might be a possibility for an oracle member to call back into `reportConsensusLayerData` when `river.setConsensusLayerData(report)` is called and so it would open a reentrancy for compromised/malicious oracle members.

Recommendation: It would be best to clear the reporting data and increment the last epoch id first then call into the `River` contract

```
// if adding this vote reaches quorum
if (variantVotes + 1 >= quorum) {
    // we clear the reporting data
    _clearReports();
    // we increment the lastReportedEpoch to force reports to be on the last frame
    LastEpochId.set(lastReportedEpochValue + 1);

    // we push the report to river
    river.setConsensusLayerData(report);
    emit SetLastReportedEpoch(lastReportedEpochValue + 1);
} ...
```

Liquid Collective: Fixed in commit 4ab66c.

Spearbit: Verified.

6.4.6 Update BufferedExceedingEth before calling sendRedeemManagerExceedingFunds

Severity: Low Risk

Context: RedeemManager.1.sol#L157-L158

Description: In pullExceedingEth , River's sendRedeemManagerExceedingFunds is called before updating the RedeemManager's BufferedExceedingEth storage value

```
_river().sendRedeemManagerExceedingFunds{value: amountToSend}();
BufferedExceedingEth.set(BufferedExceedingEth.get() - amountToSend);
```

Recommendation: It would be best to update it before calling River to avoid future bookkeeping mistakes through unwanted reentrancy

```
BufferedExceedingEth.set(BufferedExceedingEth.get() - amountToSend);
_river().sendRedeemManagerExceedingFunds{value: amountToSend}();
```

pullCoverageFunds follow this suggestion and also only updates and send funds if amountToSend is non-zero.

Liquid Collective: Fixed in commit b92b87.

Spearbit: Verified.

6.4.7 Any oracle member can censor almost quorum report variants by resetting its address

Severity: Low Risk

Context: Oracle.1.sol#L189-L201

Description: The admin or an oracle member can DoS or censor almost quorum reports by calling setMember endpoint which would reset the report variants and report positions.

The admin also can reset the/clear the reports by calling other endpoints by that should be less of an issue compared to just an oracle member doing that.

Recommendation: The protocol could

- Rate limit this endpoint or
- Make the endpoint only callable by a more privileged account
- Disallow resetting an oracle member address close to the next upcoming frame epoch.

If no action is taken to address this issue, this scenario should be documented and monitored.

Liquid Collective: In commit aadbfb report data is not cleared anymore when setMember is called.

Spearbit: Verified.

6.4.8 Incentive mechanism that encourages operators to respond quickly to exit requests might diminish under certain condition

Severity: Low Risk

Context: Operators.2.sol#L158

Description:

```
File: Operators.2.sol
153:     /// @notice Retrieve all the active and fundable operators
154:     /// @dev This method will return a memory array of length equal to the number of operator, but
↪   only
155:     /// @dev populated up to the fundable operator count, also returned by the method
156:     /// @return The list of active and fundable operators
157:     /// @return The count of active and fundable operators
158:     function getAllFundable() internal view returns (CachedOperator[] memory, uint256) { //
↪   @audit-ok
..SNIP..
172:         uint32[] storage stoppedValidatorCounts = getStoppedValidators();
173:
174:         for (uint256 idx = 0; idx < operatorCount;) {
175:             if (
176:                 _hasFundableKeys(r.value[idx])
177:                     && _getStoppedValidatorCountAtIndex(stoppedValidatorCounts, idx) >=
↪   r.value[idx].requestedExits
178:             ) {
```

`r.value[idx].requestedExits` is the accumulative number of requested validator exits by the protocol

`_getStoppedValidatorCountAtIndex(stoppedValidatorCounts, idx)` function is a value reported by oracle members which consist of both exited and slashed validator counts

It was understood the rationale of having the `_getStoppedValidatorCountAtIndex(stoppedValidatorCounts, idx) >= r.value[idx].requestedExits` conditional check at Line 177 above is to incentivize operators to respond quickly to exit requests if they want new stakes from deposits. In other words, an operator with a `requestedExits` value larger than the `_getStoppedValidatorCountAtIndex` count indicates that an operator did not submit exit requests to the Consensus Layer (CL) in a timely manner or the exit requests have not been finalized in CL.

However, it was observed that the incentive mechanism might not work as expected in some instances. Consider the following scenario:

Assuming an operator called *A* has `5` slashed validators and `0` exited validators, the `_getStoppedValidatorCountAtIndex` function will return `5` for *A* since this function takes into consideration both stopped and slashed validators. Also, assume that the `requestedExits` of *A* is `5`, which means that *A* has been instructed by the protocol to submit `5` exit requests to CL.

In this case, the incentive mechanism seems to diminish as *A* will still be considered a fundable operator even if *A* does not respond to exit requests since the number of slashed validators is enough to "help" to push up the stopped validator count to satisfy the condition, giving the wrong impression that *A* has already submitted the exit requests. As such, *A* will continue to be selected to stake new deposits.

Recommendation: Consider excluding the number of slashed validators from the total number of stopped validator counts when performing the conditional check at Line 177 of the `Operator.getAllFundable` function so that one can more accurately determine the number of exit requests that the operators have responded.

Liquid Collective: Separating exits from slashes would add a lot of complexity to the reporting process. Here we have a mechanism to catch up between stopped and requested exits so they should be accounted for in the exiting process as well in case of involuntary slash or exit when nothing was asked.

Spearbit: Acknowledged.

6.4.9 `RedeemManager. _claimRedeemRequests` transaction sender might be tricked to pay more eth in transaction fees

Severity: Low Risk

Context: RedeemManager.1.sol#L451

Description: The `_claimRedeemRequests` function is designed to allow anyone to claim ETH on behalf of another party who has a valid redeem request. The function iterates through the `redeemRequestIds` list and fulfills each request individually. However, it is important to note that the transfer of ETH to the recipients is only limited by the 63/64 rule, which means that it is possible for a recipient to take advantage of a heavy fallback function and potentially cause the sender to pay a significant amount of unwanted transaction fees.

Recommendation: To mitigate this issue, we recommend limiting the amount of gas forwarded to the call in `_claimRedeemRequests`. This can be achieved by explicitly specifying the gas amount for the `.call` function. Implementing these measures will also inherently prevent any potential reentrancy attack vectors, although none have been discovered thus far.

Liquid Collective: This issue is acknowledged and it will be up to the integrator in charge of sponsoring the gas to ensure that the recipients are in a known set or are EOAs. Otherwise, users will always have the ability to claim redemption requests by themselves.

Spearbit: Acknowledged.

6.4.10 Claimable LsETH on the Withdraw Stack could exceed total LsETH requested on the Redeem Queue

Severity: Low Risk

Context: RedeemManager.1.sol#L136

Description: Let the total amount of claimable LsETH on the Withdraw Stack be x and the total amount of LsETH requested on the Redeem Queue be y.

The following points are extracted from the Withdrawals Smart Contract Architecture documentation:

- The design ensures that $x <= y$. Refer to page 15 of the documentation.
- It is impossible for a redeem request to be claimed before at least one Oracle report has occurred, so it is impossible to skip a slashing time penalty. Refer to page 16 of the documentation.

Based on the above information, the main purpose of the design ($x <= y$) is to avoid favorable treatment of LsETH holders that would request a redemption before others following a slashing incident.

However, this constraint ($x <= y$) is not being enforced in the contract. The reporter could continue to report withdrawal via the `RedeemManager.reportWithdraw` function till the point $x > y$.

If $x > y$, LsETH holders could request a redemption before others following a slashing incident to gain an advantage.

Recommendation: Ensure that the constraint ($x <= y$) is enforced in the contract.

Liquid Collective: Fixed in PR 197.

Spearbit: Verified.

6.4.11 An oracle member can resubmit data for the same epoch multiple times if the `quorum` is set to `1`

Severity: Low Risk

Context: Oracle.1.sol#L232-L237, Oracle.1.sol#L261

Description: If the `quorum` is set to `1` and the difference between the report's epoch e and `LastEpochId.get()` is Δe, an oracle member will be able to call `reportConsensusLayerData` $\Delta e + 1$ times to push its report for epoch e to the protocol and with different data each time (only restriction on successive reports is that the difference of underlying balance between reports would need to be negative since the `maxIncrease` will be `0`).

Note that in `reportConsensusLayerData` the first storage write to `LastEpochId` will be overwritten later due to quorum of one:

```
x = LastEpochId -> report.epoch -> x + 1
```

Recommendation: When the `quorum` is reached update `LastEpochId` to be equal to `report.epoch + 1`

```
// if adding this vote reaches quorum
if (variantVotes + 1 >= quorum) {
    // we push the report to river
    river.setConsensusLayerData(report);
    // we clear the reporting data
    _clearReports();

    // --- changes below ---
    // we increment the LastEpochId to force the next reports to be on the next frame
    LastEpochId.set(report.epoch + 1);
    emit SetLastReportedEpoch(report.epoch + 1);
}
```

Liquid Collective: Fixed in `commit 4ab66c`.

Spearbit: Verified.

6.4.12 Report's `validatorsCount`'s historical non-decreseness does not get checked

Severity: Low Risk

Context: OracleManager.1.sol#L318-L321, IOracleManager.1.sol#L109, IOracleManager.1.sol#L122

Description: Once the Oracle members come to a quorum for a selected report variant, the validators count is stored in the storage. Note that `validatorsCount` is supposed to represent the total cumulative number of validators ever funded on consensus layer (even if they have been slashed or exited at some point). So this value is supposed to be a non-decreasing function of reported epochs. But this invariant has never been checked in `setConsensusLayerData`.

Recommendation: Make sure to add an `if` block to revert in case the reported validator count is smaller than the stored one from the previous report.

Liquid Collective: Fixed in `commit 8b058b`.

Spearbit: Verified.

6.4.13 The report's `slashingContainmentMode` and `bufferRebalancingMode` are decided by the oracle members which affects the exiting strategy of validators

Severity: Low Risk

Context: OracleManager.1.sol#L429-L435, River.1.sol#L545-L554

Description: The current protocol leaves it up to the oracle members to come to a quorum to set either of the `report.slashingContainmentMode` or `report.bufferRebalancingMode` to `true` or `false`.

That means the oracle members have the power to decide off-chain whether validators should be exited and whether some of the deposit balance should be reallocated for redeeming (vs an algorithmic decision by the protocol on-chain).

A potential bad scenario would be oracle members deciding to not signal for new validators to exit and from the time for the current epoch to the next report some validators get penalized or slashed which would reduce the underlying value of the shares. If those validators would have exited before getting slashed or penalized, the redeemers would have received more `ETH` back for their investment.

Recommendation: Document the off-chain decision making process for setting `report.slashingContainmentMode` and `report.bufferRebalancingMode` or implement a trustless process on-chain.

Liquid Collective: This decision was made in parallel to releasing a public specification of the Oracle report crafting algorithm, allowing anyone to audit the reports based on the aggregated public data.

The `bufferRebalancingMode` is still not 100% defined on our side but the value will be set based on the lengths of entry and exit queues, that we don't have on-chain.

This is the intended behavior, we want the oracle to be precisely defined by a spec (wip on our side) so that anyone could audit and rebuild the reported data due to its ability to be replayed from public data.

Spearbit: Acknowledged.

6.4.14 Anyone can call `depositToConsensusLayer` and potentially push wrong data on-chain

Severity: Low Risk

Context: ConsensusLayerDepositManager.1.sol#L81

Description: Anyone can call `depositToConsensusLayer` and potentially push wrong data on-chain.

An example is when an operator would want to remove a validator key that is not-funded yet but has an index below the operator `limit` and will be picked by the strategy if `depositToConsensusLayer` is called. Then anyone can front run the removal call by the operator and force push this validator's info to the deposit contract.

Recommendation: It would be best to restrict the calls to `depositToConsensusLayer` to a privileged entity.

Liquid Collective: This is the downside of putting this method back to public. The main goal here is to limit the required actions from an admin entity to operate the system. Now everything required to run it in its current operator registry state is the oracle report.

This is the accepted behavior, the main goal is to concentrate critical decisions around the Oracle, making it the main bottleneck, upon which we will work to further decentralize its quorum participation / verification logic.

Spearbit: Acknowledged.

6.5 Gas Optimization

6.5.1 Calculation of `currentMaxCommittableAmount` can be simplified

Severity: Gas Optimization

Context: River.1.sol#L603-L607

Description: `currentMaxCommittableAmount` is calculated as:

```
// we adapt the value for the reporting period by using the asset balance as upper bound
uint256 underlyingAssetBalance = _assetBalance();
uint256 currentBalanceToDeposit = BalanceToDeposit.get();
...
uint256 currentMaxCommittableAmount = LibUint256.min(
    LibUint256.min(underlyingAssetBalance, (currentMaxDailyCommittableAmount * period) / 1 days),
    currentBalanceToDeposit
);
```

But `underlyingAssetBalance` is $B_u = B_v + B_d + B_c + B_r + 32(C_d - C_r)$ which is greater than `currentBalanceToDeposit` B_d since the other components are non-negative values.

parameter	description
B_v	`LastConsensusLayerReport.get().validatorsBalance`
B_d	`BalanceToDeposit.get()`
B_c	`CommittedBalance.get()`
B_r	`BalanceToRedeem.get()`
C_d	`DepositedValidatorCount.get()`
C_r	`LastConsensusLayerReport.get().validatorsCount`
M	`currentMaxCommittableAmount`
m	`currentMaxDailyCommittableAmount * period) / 1 days`
B_u	`underlyingAssetBalance`

Note that the fact that $C_d \geq C_r$ is an invariant that is enforced by the protocol.

and so currently we are computing M as:

$$M = \min(B_u, B_d, m) = \min(B_d, m)$$

since $B_u \geq B_d$.

Recommendation: Based on the definitions and the invariant checks, the above minimum calculation can be simplified to:

```
uint256 currentMaxCommittableAmount = LibUint256.min(currentMaxDailyCommittableAmount * period) / 1
↪  days, currentBalanceToDeposit);
```

Liquid Collective: Fixed in `commit 8f0f29`.

Spearbit: Verified.

6.5.2 Remove redundant array length check and variable to save gas.

Severity: Gas Optimization

Context: ConsensusLayerDepositManager.1.sol#L103, ValidatorKeys.sol#L83-L84

Description: When someone calls `ConsensusLayerDepositManager.depositToConsensusLayer`, the contract will verify that the `receivedSignatureCount` matches the `receivedPublicKeyCount` returned from `_getNextValidators`.

This is unnecessary as the code always creates them with the same length.

Recommendation: Remove the check and associated variable to save some gas:

```
- uint256 receivedSignatureCount = signatures.length;
- if (receivedSignatureCount != receivedPublicKeyCount) {
-   revert InvalidSignatureCount();
- }
```

Liquid Collective: Fixed in PR 196.

Spearbit: Verified.

6.5.3 Duplicated events emitted in `River` and `RedeemManager`

Severity: Gas Optimization

Context: RedeemManager.1.sol#L159, River.1.sol#L481

Description: The amount of ETH pulled from the redeem contract when `setConsensusData` is called by the oracle is notified with events in both `RedeemManager` and `River` contracts.

Recommendation: Consider removing `SentExceedingEth` from `RedeemManager` to save gas and using only `PulledRedeemManagerExceedingEth` from River.

Liquid Collective: Fixed in `commit ec310f`.

Spearbit: Fixed.

6.5.4 `totalRequestedExitsValue`'s calculation can be simplified

Severity: Gas Optimization

Context: OperatorsRegistry.1.sol#L678-L692

Description: In the `for` loop in this context, `totalRequestedExitsValue` is updated for every operator that satisfies `_getActiveValidatorCountForExitRequests(operators[idx]) == highestActiveCount`. Based on the used increments, their sum equals to `optimalTotalDispatchCount`.

Recommendation: One can increment `totalRequestedExitsValue` in one go outside of this `for` loop

```
// for loop without `totalRequestedExitsValue` updates

totalRequestedExitsValue += optimalTotalDispatchCount; // <--- new line
_count -= optimalTotalDispatchCount;
```

Liquid Collective: Fixed in `commit 50af4e`.

Spearbit: Verified.

6.5.5 Report's `bufferRebalancingMode` and `slashingContainmentMode` are only used during the reporting transaction process

Severity: Gas Optimization

Context: OracleManager.1.sol#L340-L341

Description: `report.bufferRebalancingMode` and `report.slashingContainmentMode` are only used during the transaction and their previous values are not used in the protocol. They can be removed from being added to the stored report.

Note that their historical values can be queried by listening to the `ProcessedConsensusLayerReport(report, vars.trace)` events.

Recommendation: The implementation can be changed so that `bufferRebalancingMode` and `slashingContainmentMode` only occupy stack or storage space and do not get added to storage.

Liquid Collective: The main reason was to be able to query the entire latest report from view calls.

We will keep these in storage as currently the `slashingContainmentMode` flag is used in the oracle process (if active, the logics to deactivate are different so we need the last report value). Not the case for `bufferRebalancingMode` but as the spec of the oracle might evolve it is better to keep both.

Spearbit: Acknowledged.

6.6 Informational

6.6.1 Add more comments/documentation for `ConsensusLayerReport` and `StoredConsensusLayerReport` structs

Severity: Informational

Context: IOracleManager.1.sol#L102-L125

Description: The `ConsensusLayerReport` and `StoredConsensusLayerReport` structs are defined as

```
/// @notice The format of the oracle report
struct ConsensusLayerReport {
    uint256 epoch;
    uint256 validatorsBalance;
    uint256 validatorsSkimmedBalance;
    uint256 validatorsExitedBalance;
    uint256 validatorsExitingBalance;
    uint32 validatorsCount;
    uint32[] stoppedValidatorCountPerOperator;
    bool bufferRebalancingMode;
    bool slashingContainmentMode;
}

/// @notice The format of the oracle report in storage
struct StoredConsensusLayerReport {
    uint256 epoch;
    uint256 validatorsBalance;
    uint256 validatorsSkimmedBalance;
    uint256 validatorsExitedBalance;
    uint256 validatorsExitingBalance;
    uint32 validatorsCount;
    bool bufferRebalancingMode;
    bool slashingContainmentMode;
}
```

Comments regarding their specified fields are lacking.

Recommendation: Add more documentation, specifically it would be great to mention

- The requirements for a valid `epoch` that is reported or stored
- The fact that the `validatorsSkimmedBalance`, `validatorsExitedBalance`, and `validatorsCount` are non-decreasing values
- What `bufferRebalancingMode` and `slashingContainmentMode` flags are and how they get decided to be picked by the oracle members
- Define exactly how and at what specific moment these values are calculated

And then make sure the current and future implementations are following these specifications.

Liquid Collective: Fixed in `commit 07c57e`.

Spearbit: Verified.

6.6.2 `postUnderlyingBalanceIncludingExits` and `preUnderlyingBalanceIncludingExits` can be removed from `setConsensusLayerData`

Severity: Informational

Context: OracleManager.1.sol#L349, OracleManager.1.sol#L360, OracleManager.1.sol#L380

Description: Both `postUnderlyingBalanceIncludingExits` (B_u^{post}) and `preUnderlyingBalanceIncludingExits` (B_u^{pre}) include the accumulated skimmed and exited amounts overtime which part of them might have exited the protocol through redeeming (or skimmed back to CL and penalized). Their delta is almost the same as the delta of `vars.postReportUnderlyingBalance` and `vars.preReportUnderlyingBalance` (almost if one adds a check for non-decreases of validator counts).

- B_u^{post}: `postUnderlyingBalanceIncludingExits`
- B_u^{pre}: `preUnderlyingBalanceIncludingExits`
- ΔB_u: $B_u^{post} - B_u^{pre}$
- $B_u^{report,post}$: `vars.postReportUnderlyingBalance`
- $B_u^{report,pre}$: `vars.preReportUnderlyingBalance`
- ΔB_u^{report}: $B_u^{report,post} - B_u^{report,pre}$
- B_v^{prev}: previous reported/stored value for total validator balances in CL `LastConsensusLayerReport.get().validatorsBalance`
- B_v^{curr}: current reported value of total validator balances in CL `report.validatorsBalance`
- ΔB_v: $B_v^{curr} - B_v^{prev}$ (can be negative)
- B_s^{prev}: `LastConsensusLayerReport.get().validatorsSkimmedBalance`
- B_s^{curr}: `report.validatorsSkimmedBalance`
- ΔB_s: $B_s^{curr} - B_s^{prev}$ (always non-negative, this is an invariant that gets checked).
- B_e^{prev}: `LastConsensusLayerReport.get().validatorsExitedBalance`
- B_e^{curr}: `report.validatorsExitedBalance`
- ΔB_e: $B_e^{curr} - B_e^{prev}$ (always non-negative, this is an invariant that gets checked).
- $C^{prev} $: `LastConsensusLayerReport.get().validatorsCount`
- C^{curr}: `report.validatorsCount`
- ΔC: $C^{curr} - C^{prev}$ (this value should be non-negative, note this invariant has not been checked in the codebase)
- $C^{deposit}$: `DepositedValidatorCount.get()`
- B_d: `BalanceToDeposit.get()`

- B_c: `CommittedBalance.get()`
- B_r: `BalanceToRedeem.get()`

Note that the above values are assumed to be in their form before the current `report` gets stored in the storage. Then we would have

$$B_u^{post} = B_v^{curr} + B_s^{curr} + B_e^{curr} = B_u^{pre} + \Delta B_v + \Delta B_s + \Delta B_e - 32\Delta C$$

and so:

$$\Delta B_u = \Delta B_v + \Delta B_s + \Delta B_e - 32\Delta C = \Delta B_u^{report}$$

Recommendation: Defining the 2 variables B_u^{post} and B_u^{pre} seems unnecessary and one could check for the required boundary conditions, calculate the reward and `vars.availableAmountToUpperBound` only using ΔB_u^{report}:

$$\Delta B_v + \Delta B_s + \Delta B_e - 32\Delta C$$

And that can also be simplified to reduce reading multiple times from storage.

The only places that `postUnderlyingBalanceIncludingExits` and `preUnderlyingBalanceIncludingExits` are actually used are in revert parameters when boundary conditions are not met (1, 2).

Related issue: Report's validatorsCount's historical non-discreteness does not get checked

Liquid Collective: Addressed in `commit 528a77d6`.

Spearbit: Verified.

6.6.3 The formula or the parameter names for calculating `currentMaxDailyCommittableAmount` can be made more clear

Severity: Informational

Context: River.1.sol#L597-L602

Description: `currentMaxDailyCommittableAmount` is calculated using the below formula:

```
// we compute the max daily committable amount by taking the asset balance without the balance to
↪  deposit into account
uint256 currentMaxDailyCommittableAmount = LibUint256.max(
    dcl.maxDailyNetCommittableAmount,
    (uint256(dcl.maxDailyRelativeCommittableAmount) * (underlyingAssetBalance -
    ↪  currentBalanceToDeposit))
        / LibBasisPoints.BASIS_POINTS_MAX
);
```

Therefore its value is the maximum of two potential maximum values.

Recommendation: It might make sense to use the minimum of the two potential maximum values to make sure the final result is always less than the two suggested upper bound.

Liquid Collective: `max` was chosen here because the net amount is seen as a "bootstrapping" amount when the underlying balance is low, then it switches to relative value once the underlying value increases.

Spearbit: It would be best to document this choice and also pick a different parameter name for `dcl.maxDailyNetCommittableAmount` as the current name might be slightly confusing.

Comments have been added in PR 198.

And also this commit `bfd3b5` ensures that `dcl.maxDailyRelativeCommittableAmount` does not exceed `BASIS_-POINTS_MAX`.

6.6.4 `preExitingBalance` is a rough estimate for signalling the number of validators to request to exit

Severity: Informational

Context: River.1.sol#L556-L574

Description: `exitingBalance` and `preExitingBalance` might be trying to compensate for the same portion of balance (non-stopped validators which have been signaled to exit and are in the CL exit queue). That means the number of `validatorCountToExit` calculated to accommodate for the redeem demand is actually lower than what is required.

The important portion of `preExitingBalance` is for the validators that were singled to exit in the previous reporting round but the operators have not registered them for exit in CL. Also `totalStoppedValidatorCount` can include slashed validator counts which again lowers the required `validatorCountToExit` and those values should not be accounted for here.

Perhaps the oracle members should also report the slashing counts of validators so that one can calculate these values more accurately.

Recommendation: It would be best to add more documentation around the key parameters used in these calculations to make sure it is known what they exactly represent.

Liquid Collective: As soon as we detect a validator starting its exit process, the count of stopped validators is increased for the operator (so it is accounted for in the totalStoppedValidatorCount). So this value accounts for exits that would be expected from the operators but haven't been detected yet and are not yet accounted into the stopped count and exiting balance.

For the differentiation between exited and slashed validators, this is something we can do but I think it would be better to have a common and unique flow for validators that are stopped, no matter the reason. It will indeed create a delta with the demand (ex: we expect 32 ETH and we receive ~30) but the system is able to re-compute the demand and ask for additional exits in such scenarios.

Spearbit: The fact that the stopped validator count starts when the validator starts its exit process should be documented.

In `commit 2b79e1` comments have been added in regards to the usage of `exitingBalance`.

6.6.5 More documentation can be added regarding the `currentMaxDailyCommittableAmount` calculation

Severity: Informational

Context: River.1.sol#L597-L602

Description: `currentMaxDailyCommittableAmount` calculated as

```
// we compute the max daily committable amount by taking the asset balance without the balance to
↪  deposit into the account
uint256 currentMaxDailyCommittableAmount = LibUint256.max(
    dcl.maxDailyNetCommittableAmount,
    (uint256(dcl.maxDailyRelativeCommittableAmount) * (underlyingAssetBalance -
    ↪  currentBalanceToDeposit))
        / LibBasisPoints.BASIS_POINTS_MAX
);
```

We can add further to the comment:

Since before the `_commitBalanceToDeposit` hook is called we have skimmed the remaining to redeem balance to `BalanceToDeposit`, `underlyingAssetBalance - currentBalanceToDeposit` represent the funds allocated for CL (funds that are already in CL, funds that are in transit to CL or funds committed to be deposited to CL).

It is important that the redeem balance is already skimmed for this upper bound calculation, so for future code changes we should pay attention to the order of hook callbacks otherwise the upper bounds would be different.

Recommendation: Document/comment on the above for better clarity around the calculation of `currentMaxDailyCommittableAmount`.

Liquid Collective: More comments have been added in

- commit 9b69bc
- commit 769bda

Spearbit: Verified.

6.6.6 BalanceToRedeem is only non-zero during a report processing transaction

Severity: Informational

Context: River.1.sol#L440-L445, River.1.sol#L468-L470, River.1.sol#L511, River.1.sol#L551, River.1.sol#L580-L588

Description: BalanceToRedeem is only ever posses a non-zero value during the report processing when a quorum has been made for the oracle member votes (setConsensusLayerData). And at the very end of this process its value gets reset back to 0.

Recommendation: It would be best to document the above fact and if it is not required to keep this value in the storage, one can utilise the stack or memory space.

Liquid Collective: This is something we discussed and we decided to keep it in storage for now for possible future update on the reporting flow where exit funds might be kept between reports (and not automatically balanced back to deposits). But yes in this implementation case, balanceToRedeem == 0 outside of state transitions.

Spearbit: Verified.

6.6.7 Improve clarity on bufferRebalancingMode variable

Severity: Informational

Context: OracleManager.1.sol#L441, River.1.sol#L545, River.1.sol#L580-L588

Description: According to the documentation, the bufferRebalancingMode flag passed by the oracle should allow or disallow the rebalancing of funds between the Deposit and Redeem buffers. The flag correctly disables rebalancing in the DepositBuffer to RedeemBuffer direction as can be seen here

```
if (depositToRedeemRebalancingAllowed && availableBalanceToDeposit > 0) {
    uint256 rebalancingAmount = LibUint256.min(
        availableBalanceToDeposit, redeemManagerDemandInEth - exitingBalance - availableBalanceToRedeem
    );
    if (rebalancingAmount > 0) {
        availableBalanceToRedeem += rebalancingAmount;
        _setBalanceToRedeem(availableBalanceToRedeem);
        _setBalanceToDeposit(availableBalanceToDeposit - rebalancingAmount);
    }
}
```

but it is not used at all when pulling funds in another way

```
// if funds are left in the balance to redeem, we move them to the deposit balance
_skimExcessBalanceToRedeem();
```

Recommendation: Consider renaming the variable and updating the documentation so it is clear that the flag is only supposed to control rebalancing to the redeem buffer.

Liquid Collective: This variable has been renamed to rebalanceDepositToRedeemMode in PR 196.

Spearbit: Verified.

6.6.8 Fix code style consistency issues

Severity: Informational

Context: Most of the diff breaches the code style

Description: There is a small code styling mismatch between the new code under audit and the style used through the rest of the code. Specifically, function parameter names are supposed to be prepended with _ to differentiate them from variables defined in the function body.

Recommendation: Prepend function parameter names with _ to match the rest of the codebase.

Liquid Collective: Fixed in PR 198.

Spearbit: Fixed.

6.6.9 Remove unused constants

Severity: Informational

Context: Oracle.1.sol#L26

Description: `DENOMINATION_OFFSET` is unused and can be removed from the codebase.

Recommendation: If the constants in this context are not going to be used, it would be best to remove them from the codebase to keep it project cleaner.

Liquid Collective: Fixed in `commit 7f15e6`. `OracleV1.ONE_YEAR` has also been removed.

Spearbit: Verified.

6.6.10 Document what `TotalRequestedExits` can potentially represent

Severity: Informational

Context: OperatorsRegistry.1.sol#L708-L711, TotalRequestedExits.sol#L8

Description: Documentation is lacking for `TotalRequestedExits`. This parameter represents a quantity that is a mix of exited (or to be exited) and slashed validators for an operator. Also, in general, this is a rough quantity since we don't have a finer reporting of slashed and exited validators (they are reported as a sum).

Recommendation: It would be great to add more documentation for this value and perhaps change its name (the parameter name is a bit misleading).

Liquid Collective: `TotalRequestedExits` has been removed from the codebase in PR 195. So this issue does not apply anymore.

Spearbit: Verified.

6.6.11 Operators need to listen to `RequestedValidatorExits` and exit their validators accordingly

Severity: Informational

Context: OperatorsRegistry.1.sol#L700

Description: Operators need to listen to `RequestedValidatorExits` and exit their validators accordingly

```
emit RequestedValidatorExits(operators[idx].index, requestedExits + operators[idx].picked);
```

Note that `requestedExits + operators[idx].picked` represents the upper bound for the index of the funded validators that need to be exited by the operator.

Recommendation: Make sure the daemon clients are set up properly to execute the required actions upon receiving these events.

Liquid Collective: Acknowledged.

Spearbit: Acknowledged.

6.6.12 Oracle members would need to listen to `ClearedReporting` and report their data if necessary

Severity: Informational

Context: Oracle.1.sol#L302

Description: Oracle members would need to listen to `ClearedReporting` event and report their data if necessary

Recommendation: Off-chain daemon needs to be configured to listen to this event and perform the required actions.

Liquid Collective: Acknowledged.

Spearbit: Acknowledged.

6.6.13 The only way for an oracle member to change its report data for an epoch is to reset the reporting process by changing its address

Severity: Informational

Context: Oracle.1.sol#L238-L241, Oracle.1.sol#L189-L201

Description: If an oracle member has made a mistake in its CL report to the `Oracle` or for some other reason would like to change its report, it would not be able to due to the following `if` block:

```
// we retrieve the voting status of the caller, and revert if already voted
if (ReportsPositions.get(uint256(memberIndex))) {
    revert AlreadyReported(report.epoch, msg.sender);
}
```

The only way for the said oracle member to be able to report different data is to reset its address by calling `setMember`. This would cause all the report variants and report positions to be cleared and force all the other oracle members to report their data again.

Related:

- Any oracle member can censor almost quorum report variants by resetting its address.

Recommendation: Document the scenario in this issue and if desired make it possible for an oracle member to change its report before a quorum is made (it can also be a time-based editing window).

Liquid Collective: Yes, this is a side effect of the oracle voting system, where in case of no quorum, a new frame would be the only solution to make things move forward in the votes.

This is not the case anymore after remediations (`commit aadbfb`), now there are no ways to change the vote and we should wait for the next frame to be able to vote again.

Spearbit: Acknowledged.

6.6.14 For a quorum making CL report the epoch restrictions are checked twice.

Severity: Informational

Context: Oracle.1.sol#L228-L231, OracleManager.1.sol#L268-L271

Description: When an oracle member reports to the Oracle's reportConsensusLayerData, the requirements for a valid epoch is checked once in reportConsensusLayerData:

```
// checks that the report epoch is not invalid
if (!river.isValidEpoch(report.epoch)) {
    revert InvalidEpoch(report.epoch);
}
```

and once again in setConsensusLayerData

```
// we start by verifying that the reported epoch is valid based on the consensus layer spec
if (!_isValidEpoch(cls, report.epoch)) {
    revert InvalidEpoch(report.epoch);
}
```

Note that only the Oracle can call the setConsensusLayerData endpoint and the only time the Oracle makes this call is when the quorum is reached in reportConsensusLayerData.

Recommendation: The second check inside setConsensusLayerData can be removed, but a comment should be made that the check has been performed in reportConsensusLayerData to make sure future changes would not cause this check to be skipped.

Liquid Collective: We would prefer keeping it in both

- On the Oracle side, this prevents members from erasing reports by reporting invalid epochs that are greater than the current frame first epoch.
- On River, in the same spirit as what we decided to do by bringing all sanity checks inside the core contract, I would prefer being able to ensure a report is valid without assuming the behavior of the oracle contract as this contract might change in the future.

Spearbit: Acknowledged.

6.6.15 Clear report variants and report position data during the migration to the new contracts

Severity: Informational

Context: Oracle.1.sol#L36

Description: Upon migration to the new contract with a new type of reporting data the old report positions and variants should be cleared by calling _clearReports() on the new contract or an older counterpart on the old contract.

Note that the report variants slot will be changed from:

```
bytes32(uint256(keccak256("river.state.reportsVariants")) - 1)
```

to:

```
bytes32(uint256(keccak256("river.state.reportVariants")) - 1)
```

Recommendation: Make sure the reporting data is cleaned before the next round of reporting process after the migration.

Liquid Collective: Fixed in PR 193 by clearing the report data upon reinitialization and also the report variants slot has been restored to its original slot.

Spearbit: Verified. Need to make sure oracle.initOracleV1_1() is called in the migration sequence to trigger the report cleanup.

6.6.16 Remove unused functions from `Oracle`

Severity: Informational

Context:

- Oracle.1.sol#L115
- Oracle.1.sol#L120
- Oracle.1.sol#L125
- Oracle.1.sol#L130
- Oracle.1.sol#L140
- Oracle.1.sol#L145
- Oracle.1.sol#L150
- Oracle.1.sol#L155
- Oracle.1.sol#L160

Description: The following functions are unused and can be removed from the `Oracle`'s implementation

- `isValidEpoch`
- `getTime`
- `getExpectedEpochId`
- `getLastCompletedEpochId`
- `getCurrentEpochId`
- `getCLSpec`
- `getCurrentFrame`
- `getFrameFirstEpochId`
- `getReportBounds`

Recommendation: It might be best to remove these endpoints to keep the codebase clean. These endpoints are just wrapped around the corresponding endpoint from the `River` contract.

Liquid Collective: This is for off-chain and was a request to keep the oracle contract similar to its old version from the view call point of view.

Functions have been removed in `commit f73551`.

Spearbit: Verified.

6.6.17 `RedeemManager. _claimRedeemRequests` - Consider adding the `recipient` to the revert message in case of failure

Severity: Informational

Context: RedeemManager.1.sol#L454

Description: The purpose of the `_claimRedeemRequests` function is to facilitate the claiming of ETH on behalf of another party who has a valid redeem request. It is worth noting that if any of the calls to recipients fail, the entire transaction will revert.

Although it is impossible to conduct a denial-of-service (DoS) attack in this scenario, as the worst-case scenario only allows the transaction sender to specify a different array of `redeemRequestIds`, it may still be challenging to determine the specific redemption request that caused the transaction to fail.

Recommendation: Consider adding the `recipient` to the revert message.

Liquid Collective: Fixed in `commit 28602e`.

Spearbit: Verified.

6.6.18 `River.1.sol` should not have a permission to call `requestRedeem(uint256 lsETHAmount)`

Severity: Informational

Context: RedeemManager.1.sol#L114

Description: The function `RedeemManagerAddress .requestRedeem(uint256 lsETHAmount)` is not being called by the `River.1.sol` contract currently. In accordance with the principle of least privilege, any permissions or privileges that are not strictly necessary should be removed. Therefore, it is recommended to remove the permission for the `River.1.sol` contract to call this function. This will help to minimize the attack surface and reduce the risk of unauthorized access or misuse of the function.

Liquid Collective: Fixed in `commit e76fd5` by removing the unnecessary permission from `RedeemManagerAddress.requestRedeem(uint256 lsETHAmount)`.

Spearbit: Verified.

6.6.19 Exit validator picking strategy does not consider slashed validator between reported epoch and current epoch

Severity: Informational

Context: Operators.2.sol#L281, OperatorsRegistry.1.sol#L615

Description: The current picking strategy in the `OperatorsRegistry._pickNextValidatorsToExitFromActiveOperators` function relies on a report from an old `epoch`. In between the reported epoch and the current epoch, validators might have been slashed and so the strategy might pick and signal to the operators those validators that have been slashed. As a result, the suggested number of validators to exit the protocol to compensate for the redemption demand in the next round of reports might not be exactly what was requested.

Similarly, the `OperatorsV2._hasExitableKeys` function only evaluates based on a report from an old `epoch`. In between the reported epoch and the current epoch, validators might have been slashed. Thus, some returned operators might not have exitable keys in the current epoch.

Recommendation: It might not be possible to obtain real-time information on the slashed validators while picking the validator to exit due to various technical limitations (e.g. delay between components). Consider documenting this limitation.

Liquid Collective: We have taken the decision to not separate slashed from stopped validators. This has an impact on the amount received due to exits but It's accepted and the system can handle the problem by requesting another exit if the amount does not match the exit demand. Also, the OperatorsRegistry is able to "catch up" in case of stopped > exitRequests.

Spearbit: Acknowledged.

6.6.20 Duplicated functions

Severity: Informational

Context: Operators.2.sol#L142, OperatorsRegistry.1.sol#L484

Description: The `Operator.2._getStoppedValidatorCountAtIndex` and `OperatorsRegistry.1._getStoppedValidatorsCountFromRawArray` functions are the same.

```
File: Operators.2.sol
    function _getStoppedValidatorCountAtIndex(uint32[] storage stoppedValidatorCounts, uint256
↪   index)
        internal
        view
        returns (uint32)
    {
        if (index + 1 >= stoppedValidatorCounts.length) {
            return 0;
        }
        return stoppedValidatorCounts[index + 1];
    }
```

```
File: OperatorsRegistry.1.sol
    function _getStoppedValidatorsCountFromRawArray(uint32[] storage stoppedValidatorCounts,
↪   uint256 operatorIndex)
        internal
        view
        returns (uint32)
    {
        if (operatorIndex + 1 >= stoppedValidatorCounts.length) {
            return 0;
        }
        return stoppedValidatorCounts[operatorIndex + 1];
    }
```

Recommendation: Consider removing one of the functions.

Liquid Collective: Fixed in PR 195.

Spearbit: Verified.

6.6.21 Funds might be pulled from `CoverageFundV1` even when there has been no slashing incident.

Severity: Informational

Context: CoverageFund.1.sol#L18-L19, OracleManager.1.sol#L416-L422

Description: `vars.availableAmountToUpperBound` might be positive even though no validators have been slashed. In this case, we still pull funds from the coverage funds contract to get closer to the upper bound limit:

```
// if we have available amount to upper bound after pulling the exceeding eth buffer, we attempt to
↪   pull coverage funds
if (vars.availableAmountToUpperBound > 0) {
    // we pull the funds from the coverage recipient
    vars.trace.pulledCoverageFunds = _pullCoverageFunds(vars.availableAmountToUpperBound);
    // we do not update the rewards as coverage is not considered rewards
    // we do not update the available amount as there are no more pulling actions to perform afterwards
}
```

So it is possible the slashed coverage funds get used even when there has been no slashing to account for.

Recommendation: Either update the NatSpec comment for `CoverageFundV1` to document the above case or a finer implementation needs to be introduced that account for the slashed validators directly.

Liquid Collective: Pulling funds from CoverageFunds is an independent procedure from the Oracle report. Following a slash, an off-chain resolution for reimbursement will occur, and reimbursement can happen a long time after the slash happens.

The CoverageFund should be seen as an independent component allowing the injection of ETH into the system without minting LsETH.

Spearbit: Acknowledged. Pulling funds from CoverageFunds happens when oracle members come to a quorum, so it is directly related to the reporting process. But when funds get deposited to `CoverageFundV1` by allowed parties is a different procedure.

It would be best to document the depositing aspect and also if extra funds have been injected into `CoverageFundV1` for possible future slashing events, before those events happen it is possible for the funds to get used as mentioned in this issue.

6.6.22 Update inline documentation

Severity: Informational

Context:

- OracleManager.1.sol#L61
- OracleManager.1.sol#L472
- OracleManager.1.sol#L483
- OracleManager.1.sol#L495
- IOracle.1.sol#L204
- Oracle.1.sol#L189

Description:

- `OracleManager.1.sol` functions highlighted in *Context* are missing the `@return` natspec.
- `IOracle.1.sol#L204`'s highlighted comment is outdated. `setMember` can now also be called by the member itself. Also, there is a typo: `adminitrator` -> `administrator`.

```
File: IOracle.1.sol
204:     /// @dev Only callable by the adminitrator @audit typo and outdated
209:     function setMember(address _oracleMember, address _newAddress) external;

File: Oracle.1.sol
28:     modifier onlyAdminOrMember(address _oracleMember) {
29:         if (msg.sender != _getAdmin() && msg.sender != _oracleMember) {
30:             revert LibErrors.Unauthorized(msg.sender);
31:         }
32:         _;
33:     }
...
189:     function setMember(address _oracleMember, address _newAddress) external
↪   onlyAdminOrMember(_oracleMember) {
```

Recommendation: Update the above inline documentation.

Liquid Collective: Fixed in PR 198.

Spearbit: Verified.

6.6.23 Document/mark unused (would-be-stale) storage parameters after migration

Severity: Informational

Context:

- CLValidatorCount.sol#L10
- CLValidatorTotalBalance.sol#L10-L11
- LastOracleRoundId.sol#L10-L11
- Operators.1.sol#L10
- ReportVariants.sol#L8-L9

Description: The following storage parameters will be unused after the migration of the protocol to `v1`

- `CLValidatorCount`
- `CLValidatorTotalBalance`
- `LastOracleRoundId.sol`
- `OperatorsV1`, this will be more than one slot (it occupies regions of storage)
- `ReportVariants`, the slot has been changed (that means right after migration `ReportVariants` will be an empty array by default):

```
- bytes32 internal constant REPORTS_VARIANTS_SLOT =
↪  bytes32(uint256(keccak256("river.state.reportsVariants")) - 1);
+ bytes32 internal constant REPORT_VARIANTS_SLOT  =
↪  bytes32(uint256(keccak256("river.state.reportVariants")) - 1);
```

Recommendation: Document/mark unused (would-be-stale) storage parameters after migration.

Liquid Collective: The reports variant slot has been restored to its original slot, so this issue does not apply to it anymore

- `commit bcbb09`.

The other slots have been marked in

- `commit c32fc9`.

Spearbit: Verified.

6.6.24 `pullEth`, `pullELFees` and `pullExceedingEth` do not check for a non-zero amount before sending funds to `River`

Severity: Informational

Context:

- ELFeeRecipient.1.sol#L31-L32
- RedeemManager.1.sol#L157-L158
- Withdraw.1.sol#L42-L43
- CoverageFund.1.sol#L46-L49

Description: `pullCoverageFunds` makes sure that the amount sending to `River` is non-zero before calling its corresponding endpoint. This behavior differs from the implementations of

- `pullELFees`
- `pullExceedingEth`
- `pullEth`

Not checking for a non-zero value has the added benefit of saving gas when the value is non-zero, while the check for a non-zero value before calling back `River` saves gas for cases when the amount could be `0`.

Recommendation: It would be best to keep the implementation of these 4 pulling mechanisms consistent by either having or excluding the non-zero amount check.

Liquid Collective: Fixed in `commit 01078a`.

Spearbit: Verified.