Livepeer Onchain Treasury Upgrade Findings & Analysis Report

2023-10-17

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit outlined in this document, C4 conducted an analysis of the Livepeer Onchain Treasury Upgrade smart contract system written in Solidity. The audit took place between August 31—September 6 2023.

Wardens

30 Wardens contributed reports to the Livepeer Onchain Treasury Upgrade:

1. BanditxOx
2. ladboy233
3. Vagner
4. bronze_pickaxe
5. Krace
6. VAD37
7. ether_sky
8. rvierdiiev
9. ADM
10. Sathish9098
11. catellatech
12. twicek
13. HChang26
14. JayShreeRAM
15. Proxy

16. c3phas
17. kaveyjoe
18. 0x3b
19. favelanky
20. nadin
21. DavidGiladi
22. 0x11singh99
23. SAQ
24. hunter_w3b
25. 0xta
26. turvy_fuzz
27. ReyAdmirado
28. JCK
29. lsaudit
30. K42

This audit was judged by hickuphh3.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 5 unique vulnerabilities. Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity and 3 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 7 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 13 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Livepeer Onchain Treasury Upgrade repository, and is composed of 3 interfaces and 7 smart contracts written in

the Solidity programming language and includes 2,602 lines of Solidity code.

In addition to the known issues identified by the project team, a Code4rena bot race was conducted at the start of the audit. The winning bot, **henry** from warden hihen, generated the [Automated Findings report] and all findings therein were classified as out of scope.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on [the C4 website], specifically our section on [Severity Categorization].

High Risk Findings (2)

[H-01] Underflow in `updateTranscoderWithFees` can cause corrupted data and loss of winning tickets

Submitted by [bronze_pickaxe], *also found by* [VAD37], [ether_sky], *and* [Krace]

`updateTranscoderWtihFees` can underflow because MathUtils is used instead of PreciseMathUtils.

Proof of Concept

According to [LIP-92] the initial `treasuryRewardCutRate` will be set to `10%`.

`treasuryRewardCutRate` is set with the `setTreasuryRewardCutRate()` function, which calls the internal function `_setTreasuryRewardCutRate()`.

```
file: 2023-08-livepeer/contracts/bonding/BondingManager.sol

function _setTreasuryRewardCutRate(uint256 _cutRate) internal {
        require(PreciseMathUtils.validPerc(_cutRate), "_cutRate
```

In this function the value will be checked if it's a valid `PreciseMathUtils` percentage (< `100%` specified with 27-digits precision):

```
file: 2023-08-livepeer/contracts/libraries/PreciseMathUtils.sol
library PreciseMathUtils {
// ...
    // Divisor used for representing percentages
    uint256 public constant PERC_DIVISOR = 10**27;
        function validPerc(uint256 _amount) internal pure return
        return _amount <= PERC_DIVISOR;
    }
// ...
```

However, in `updateTranscoderWithFees`, to calculate `treasuryRewards`, `MathUtils` is used instead of `PreciseMathUtils`.

```
file: 2023-08-livepeer/contracts/bonding/BondingManager.sol

function updateTranscoderWithFees(
        address _transcoder,
        uint256 _fees,
        uint256 _round
    ) external whenSystemNotPaused onlyTicketBroker {
// ...
uint256 treasuryRewards = MathUtils.percOf(rewards, treasuryRewa
rewards = rewards.sub(treasuryRewards);
// ...
}
```

`MathUtils` uses a `PREC_DIVISOR` of `1000000` instead of `10 ** 27` from the `PreciseMathUtils`:

```
file: 2023-08-livepeer/contracts/libraries/MathUtils.sol
library MathUtils {
// ...
    uint256 public constant PERC_DIVISOR = 1000000;
// ...
```

This leads to `treasuryRewards` value being bigger than expected. Here is a **gist of the POC**. Running the POC it shows that the current usage of `MathUtils` when calculating `treasuryRewards` will always cause an underflow in the next line of code.

`updateTranscoderWithFees` is called every time a winning ticket is redeemed. Whenever the transcoder has skipped the previous round reward call, this function has to re-calculate the rewards, as documented in **LIP-92**. This re-calculation will always fail due to the underflow shown above.

Impact

This will lead to accounting errors, unexpected behaviours and can cause a loss of winning tickets.

Firstly, the accounting errors and unexpected behaviours: these are all the storage values getting updated in `updateTranscoderWithFees`:

```
file: 2023-08-livepeer/contracts/bonding/BondingManager.sol

function updateTranscoderWithFees( address _transcoder,
        uint256 _fees,
        uint256 _round
    ) external whenSystemNotPaused onlyTicketBroker {
// ...
// transcoder & earningsPool.data
L314: Transcoder storage t = transcoders[_transcoder];
L321: EarningsPool.Data storage earningsPool = t.earningsPoolPer

//accounting updates happen here
L377: t.cumulativeFees = t.cumulativeFees.add(transcoderRewardSt
          .add(transcoderCommissionFees);
```

```
L382: earningsPool.updateCumulativeFeeFactor(prevEarningsPool,de
L384: t.lastFeeRound = currentRound;
```

- Let `currentRound() - 1` be the previous round where the transcoder skipped the reward call
- Let `currentRound()` be current round
- Let `currentRound() + 1` be the next round

During `currentRound()` it won't be possible to update the `Transcoder` storage or `earningsPool.data` storage because of the underflow that will happen because `currentRound() - 1` reward call has been skipped by the transcoder.

During `currentRound() + 1` it will be possible to call `updateTranscoderWithFees`, however, L382 will only update the `prevEarningsPool`, which in this case will be `currentRound()`, not `currentRound - 1`. Therefor, the `EarningsPool.data.cumulativeRewardFactor` won't be updated for `currentRound() - 1`.

Lastly, the validity of a ticket is two rounds as per the specs. This means that a transcoder that receives a winning ticket in `currentRound() - 1` should be able to redeem it in `currentRound() - 1` and `currentRound()`. However, a transcoder that receives a winning ticket in `currentRound() - 1` wont be able to redeem it in `currentRound()` because of the underflow that happens while redeeming a winning ticket in `currentRound()`. The transcoder wont be able to redeem it after `currentRound + 1..N` because the ticket will be expired.

Recommended Mitigation Steps

Use `PreciseMathLib` instead of `MathLib`:

```
file: 2023-08-livepeer/contracts/bonding/BondingManager.sol

L355:
- uint256 treasuryRewards = MathUtils.percOf(rewards, treasuryRe
+ uint256 treasuryRewards = PreciseMathUtils.percOf(rewards, tre
```

Assessed type

Library

victorges (Livepeer) commented:

> Can confirm this issue!

victorges (Livepeer) mitigated:

> https://github.com/livepeer/protocol/pull/624

[H-02] By delegating to a non-transcoder, a delegator can reduce the tally of someone else's vote choice without first granting them any voting power

Submitted by BanditxOx

A delegate can subtract their own voting weight from the voting choice of another delegate, even if that user isn't a transcoder. Since they are not a transcoder, they don't have their votes initially increased by the amount delegated to them, voting weight is still subtracted from the tally of their vote choice.

Maliciously, this could be used to effectively double one's voting power, by delegating their votes to a delegator who is about to vote for the choice which they don't want. It can also occur accidentally, for example when somebody delegates to a transcoder who later switches role to delegate.

Proof of Concept

When a user is not a transcoder, their votes are determined by the amount they have delegated to the delegatedAddress, and does not increase when a user delegates to them:

```
if (bond.bondedAmount == 0) {
    amount = 0;
} else if (isTranscoder) {
    amount = bond.delegatedAmount;
} else {
```

```
                amount = delegatorCumulativeStakeAt(bond, _round);
            }
        }
```

Let's say that this delegator (Alice) has 100 votes and votes `For`, then another delegator(Bob) has delegated 1000 votes to Alice. As stated above, Alice doesn't get the voting power of Bob's 1000 votes, so the `For` count increases by 100.

Bob now votes, and `_handleVotesOverrides` is called. In this function, the first conditional, `if isTranscoder` will return false as Bob is not self-delegating.

Then, there is a check that the address Bob has delegated to has voted. Note that there is a missing check of whether the delegate address is a transcoder. Therefore the logic inside `if (delegateVoter.hasVoted)` is executed:

```
if (delegateVoter.hasVoted) {
// this is a delegator overriding its delegated transcoder vote,
// we need to update the current totals to move the weight of
// the delegator vote to the right outcome.
VoteType delegateSupport = delegateVoter.support;

                if (delegateSupport == VoteType.Against) {
                    _tally.againstVotes -= _weight;
                } else if (delegateSupport == VoteType.For) {
                    _tally.forVotes -= _weight;
                } else {
                    assert(delegateSupport == VoteType.Abstain);
                    _tally.abstainVotes -= _weight;
                }
            }
```

The logic reduces the tally of whatever choice Alice voted for by Bob's weight. Alice initially voted `For` with 100 votes, and then the For votes is reduced by Bob's `1000 votes`. Lets say that Bob votes `Against`. This will result in an aggregate 900 vote reduction in the `For` tally and +1000 votes for `Agaisnt` after Alice and Bob has finished voting.

If Alice was a transcoder, Bob will be simply reversing the votes they had delegated to Alice. However since Alice was a delegate, they never gained the voting power that

was delegated to her.

Bob has effectively gained the ability to vote against somebody else's votes (without first actually increasing their voting power since they are not a transcoder) and can vote themselves, which allows them to manipulate governance.

Recommended Mitigation Steps

There should be a check that a delegate is a transcoder before subtracting the tally. Here is some pseudocode:

```
if (delegateVoter.hasVoted && ---delegate is transcoder ---)
```

This is an edit of the conditional of the function `_handleOverrides`. This ensures that the subtraction of vote tally is only performed when the delegate is a voter AND the delegate is a transcoder. This should fix the accounting/subtraction issue of vote tally for non-transcoder delegates.

Assessed type

Invalid Validation

victorges (Livepeer) confirmed

victorges (Livepeer) mitigated:

> **https://github.com/livepeer/protocol/pull/625**
> **https://github.com/livepeer/protocol/pull/626**

Medium Risk Findings (3)

[M-01] The logic in `_handleVoteOverride` to determine if an account is the transcoder is not consistent with the logic in the `BondManager.sol`

Submitted by **ladboy233**

In the current implementation, when voting, the function **_countVote is triggered**, this function is overridden in the function GovernorCountingOverridable.sol

```
_weight = _handleVoteOverrides(_proposalId, tally, voter, _a
```

This is calling:

```
function _handleVoteOverrides(
    uint256 _proposalId,
    ProposalTally storage _tally,
    ProposalVoterState storage _voter,
    address _account,
    uint256 _weight
) internal returns (uint256) {

    uint256 timepoint = proposalSnapshot(_proposalId);

    address delegate = votes().delegatedAt(_account, timepoi

    // @audit
    // is transcoder?
    bool isTranscoder = _account == delegate;

    if (isTranscoder) {
        // deduce weight from any previous delegators for th
        // make a vote
        return _weight - _voter.deductions;
    }
```

The logic to determine if an account is the transcoder is too simple in this **line of code**:

```
// @audit
// is transcoder?
bool isTranscoder = _account == delegate;
```

And does not match the logic that determine if the address is an registered transcorder and an active transcoder in the bondManager.sol.

In BondManager.sol, the function that used to check if a transcoder is registered is in this line of code:

```
/**
 * @notice Return whether a transcoder is registered
 * @param _transcoder Transcoder address
 * @return true if transcoder is self-bonded
 */
function isRegisteredTranscoder(address _transcoder) public
    Delegator storage d = delegators[_transcoder];
    return d.delegateAddress == _transcoder && d.bondedAmoun
}
```

The function that is used to check if a transcoder is active is in this line of code.

```
function isActiveTranscoder(address _transcoder) public view
    Transcoder storage t = transcoders[_transcoder];
    uint256 currentRound = roundsManager().currentRound();
    return t.activationRound <= currentRound && currentRound
}
```

Missing the check in the delegator's bond amount (delegators[_transcoder].bondeAmount > 0).

The code incorrectedly counts regular delegator as transcoder and does not update the deduction power correctly.

Recommended Mitigation Steps

Reuse the function isRegisteredTranscoder and isActiveTranscoder to determine if an account is a registered and active transcoder when counting the voting power.

Assessed type

Governance

victorges (Livepeer) confirmed and commented:

> There is in fact an issue with the inconsistency with the `isRegisteredTranscodeer` function. This report didn't manage to go specifically into that issue, but still pointed a valid problem which is in a sense the root cause there. FTR There is no problem with `isActiveTranscoder` though, since we don't give voting power only to active transcoders. That part of this report is invalid.

victorges (Livepeer) mitigated:

> https://github.com/livepeer/protocol/pull/626

[M-02] Fully slashed transcoder can vote with 0 weight messing up the voting calculations

Submitted by Vagner (1, 2), also found by ladboy233

If a transcoder gets slashed fully he can still vote with 0 amount of `weight` making any other delegated user that wants to change his vote to subtract their `weight` amount from other delegators/transcoders.

Proof of Concept

In `BondingManager.sol` any transcoder can gets slashed by a specific percentage, and that specific transcoder gets resigned and that specific percentage gets deducted from his `bondedAmount`.
https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L394-L411

If any `bondedAmount` will remain then the penalty will also gets subtracted from the `delegatedAmount`, if not, nothing happens.
https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L412-L417

After that the `penalty` gets burned and the fees gets paid to the finder, if that is the case.
https://github.com/code-423n4/2023-08-

livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L420-L440

The problem relies in the fact that a fully slashed transcoder, even if it gets resigned, he is still seen as an active transcoder in the case of voting. Let's assume this case:

- A transcoder gets fully slashed and gets resigned from the transcoder pools, getting his `bondedAmount` to 0, but he still has `delegatedAmount` to his address since nothing happens this variable.
  https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L402-L418
- Transcoder vote a proposal and when his weighting gets calculated here, it will use the `_getVotes` from `GovernorVotesUpgradeable.sol`.
  https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/f34a3a7e5a1d698d87d517fda698d48286310bee/contracts/governance/extensions/GovernorVotesUpgradeable.sol#L55-L61
- `_getVotes` calls `getPastVotes` on `BondingVotes.sol`
  https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingVotes.sol#L167-L170 which returns the amount of weight specific to this transcoder and as you can see, because the transcoder has a `bondedAmount` equal to 0, the first if statement will be true and 0 will be returned.
  https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingVotes.sol#L372-L373
- 0 weight will be passed into `_countVote` which will then be passed into `_handleVoteOverrides`.
  https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/treasury/GovernorCountingOverridable.sol#L151
- Then it will check if the caller is a transcoder, which will be true in our case, because nowhere in the `slashTranscoder` function, or any other function the transcoder `delegateAddress` gets changed, so this if statement will be true
  https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/trea

sury/GovernorCountingOverridable.sol#L182-L184, which will try to deduct the weight from any previous delegators. https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/treasury/GovernorCountingOverridable.sol#L184-L189

- If any delegator already overridden any amount this subtraction would revert, but if that is not the case, 0 weight will be returned, which is then used to vote `for`, `against`, `abstain`, but since 0 weight is passed no changed will be made.
- Now the big problem arise, if any delegator that delegated their votes to this specific transcoder want to change their vote, when his weight gets calculated, `delegatorCumulativeStakeAt` gets called https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingVotes.sol#L459-L487 which will return most of the time his `bondedAmount`, amount which is greater than 0, since he didn't unbound.
- Because of that when `_handleVoteOverrides` gets called in `_countVote`, to override the vote, this if statement will be true, since the transcoder voted already, but with 0 weight https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/treasury/GovernorCountingOverridable.sol#L195 and the delegator weight gets subtracted from the support that the transcoder used in his vote.
- The system in this case expects the transcoder to vote with the whole `delegatedAmount`, which will make the subtraction viable, since the weight of the delegator should be included in the full `delegatedAmount` of that specific transcoder, but since the transcoder voted with 0 weight, the subtraction would be done from other delegators/transcoders votes.
- Also this can be abused by a transcoder by voting a category which he knows will not get a lot of votes, if let's say a transcoder used his 0 weight to vote for `abstain` and every other voter will vote on `for` or `against`, every time one of his delegators want to change the vote the subtraction can revert, which will force those delegators out of the vote, until they will change their transcoder.

Recommended Mitigation Steps

If a transcoder gets fully slashed and resigned, delete his address from `delegateAddress` so he will not appear as a transcoder in the mechanism of counting the votes. If he still wants to participate in the system he can act as a

delegator to another transcoder. Another solution would be to not let 0 weight votes happen, since they don't modify the vote state at all.

Assessed type

Governance

victorges (Livepeer) confirmed and commented:

> Slashing is turned off in the protocol, so this specific issue is not that important. It did point towards an issue in `BondingVotes` though in the way that the `getBondingState` function checks if transcoder is a transcoder or not and defaults to 0 when `bondedAmount` is `0`. There are other ways that this logic can be triggered (not through `slashTranscoder`) that can show inconsistencies as well, so I'm flagging this as valid for the underlying issue that it hints to, even though it does not describe a realistic scenario about how this issue could happen.

hickuphh3 (judge) commented:

> I found the POC hard to follow. A coded instance would have tremendously helped, but what I gather is:
>
> - fully slashed transcoder that votes will do so with 0 weight.
> - any delegator that delegated their votes to this specific transcoder intending to change their vote may face an issue with sub overflow because `_handleVoteOverrides()` uses the slashed transcoder's `bondedAmount` instead of the 0 weight. so this issue seems valid, although its premise is that the slashing functionality is in place.

> The README did not indicate that slashing was turned off, though I see it's mentioned a couple of times in the Discord channel. Should've been added in the "Known caveats / limitations" section.

> Hence, IMO it isn't entirely fair to wardens if I invalidate their issue because they weren't aware of this fact.

> Leaning towards a downgrade to M because of the external requirement of the active slashing mechanism, but am open to discussion.

hickuphh3 (judge) decreased severity to Medium

hickuphh3 (judge) commented:

> Revisiting this issue, IMO it's unreasonable to expect the sponsor to explicitly list out all the gotchas / limitations of the protocol. Some will only pop into mind through discussions with wardens, which seemed to be the case here when the slashing deprecation was mentioned a couple of times in the Discord channel.
>
> On the flip side, the fact remains that the slashing mechanism is part of the audit scope, and was counted towards the sLOC, plus the points I raised above. There wasn't an indication in the code that slashing had been deprecated.
>
> It's a difficult decision here, considering the different parties' POV. There could be some wardens who didn't submit issues related to slashing because of what was brought up in the Discord channel, and those that did (as one can see referenced above) because they didn't monitor the channel.

victorges (Livepeer) mitigated:

> https://github.com/livepeer/protocol/pull/625

[M-03] `withdrawFees` does not update checkpoint

Submitted by ADM, *also found by* rvierdiiev, twicek, *and* HChang26

BondingVotes may have stale data due to missing checkpoint in `BondingManager#withdrawFees()`.

Proof of Concept

The withdrawFee function has the autoClaimEarnings modifier:

```
function withdrawFees(address payable _recipient, uint256 _a
```

which calls _autoClaimEarnings:

```
    modifier autoClaimEarnings(address _delegator) {
            _autoClaimEarnings(_delegator);
            _;
```

which calls updateDelegatorWithEarnings:

```
    function _autoClaimEarnings(address _delegator) internal {
            uint256 currentRound = roundsManager().currentRound();
            uint256 lastClaimRound = delegators[_delegator].lastClai
            if (lastClaimRound < currentRound) {
                updateDelegatorWithEarnings(_delegator, currentRound
            }
        }
```

During updateDelegatorWithEarnings, both delegator.lastClaimRound and delegator.bondedAmount can be assigned new values.

```
            del.lastClaimRound = _endRound;
            // Rewards are bonded by default
            del.bondedAmount = currentBondedAmount;
```

However, during the lifecycle of all these functions, _checkpointBondingState is never called either directly or through the autoCheckpoint modifier resulting in lastClaimRound & bondedAmount's values being stale in BondingVotes.sol.

Recommended Mitigation Steps

Add autoCheckpoint modifier to the withdrawFees function.

victorges (Livepeer) confirmed

victorges (Livepeer) mitigated:

> https://github.com/livepeer/protocol/pull/623

Low Risk and Non-Critical Issues

For this audit, 7 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **Proxy** received the top score from the judge.

The following wardens also submitted reports: **rvierdiiev**, **ladboy233**, **favelanky**, **nadin**, **BanditxOx**, *and* **DavidGiladi**.

[01] There is no way to decrease the max number of active transcoders

`setNumActiveTranscoders()` sets the maximum number of active transcoders however there is no way to decrease the number since `setMaxSize()` does not allow it.

```
function setNumActiveTranscoders(uint256 _numActiveTranscoders) {
    transcoderPool.setMaxSize(_numActiveTranscoders);

    emit ParameterUpdate("numActiveTranscoders");
}
```

```
function setMaxSize(Data storage self, uint256 _size) public {
    require(_size > self.maxSize, "new max size must be greater

    self.maxSize = _size;
}
```

[02] When removing transcoders `Transcoder.activationRound` should be set to 0 for inactive

Function `tryToJoinActiveSet()` removes a transcoder if `transcoderPool.isFull()` however it does not set `Transcoder.activationRound` of the transcoder to 0 for inactive. In `Transcoder` struct it says if a transcoder is inactive `activationRound` should be 0.

```
uint256 activationRound; // Round in which the transcoder became
```

```
transcoderPool.remove(lastTranscoder);
transcoders[lastTranscoder].deactivationRound = _activationRound
pendingNextRoundTotalActiveStake = pendingNextRoundTotalActiveSt
```

[03] No function implemented for `setMaxEarningsClaimsRounds()`

The constructor natspec says that `setMaxEarningsClaimsRounds()` should be called to initialize state variables post-deployment. However there is no such function implemented anywhere.

```
/**
 * @notice BondingManager constructor. Only invokes constructor
 * @dev This constructor will not initialize any state variables
 * should be used to initialize state variables post-deployment:
 * - setUnbondingPeriod()
 * - setNumActiveTranscoders()
 * - setMaxEarningsClaimsRounds()
 * @param _controller Address of Controller that this contract w
 */
```

[04] Typos

- GovernorCountingOverriable.sol (#L35 , #L59):

```
    // @audit Typo - instead of "specicic" use "specific"
35: // @dev Tracks state of specicic voters in a single proposal

    // @audit Typo - instead of "abstain" use "against"
59: // @notice The required percentage of "for" votes in relatio
```

- BondingVotes.sol (#L65-66):

```
    // @audit Typo - instead of "Notce that..." use "Notice that
65: // @dev Stores a list of checkpoints for the total active st
66: // differently from bonding checkpoints, it's only accessibl
```

[05] Functions need a more descriptive name

Some functions say they set one thing but set something different and some function names are too ambiguous.

- `setTreasuryRewardCutRate()` and `_setTreasuryRewardCutRate()` set `nextRoundTreasuryRewardCutRate` not `treasuryRewardCutRate`
- `transcoder()` and `transcoderWithHint()` are too ambiguous

victorges (Livepeer) confirmed and commented:

> [01]
> This is known and expected behavior, which has an explicit check in the code to guarantee consistency. An improvement could be made to the code but the current one works as expected.

> [02]
> Valid point on doc that oversimplifies the behavior of the field and can be confusing. In the way these fields are read, that 0 value wouldn't matter though: https://github.com/livepeer/protocol/blob/ca3f5bb7a65dedba0f6be6a341d184c48553be98/contracts/bonding/BondingManager.sol#L1159

> [03]
> Misleading docs. Probably a deprecated property that doesn't exist anymore. Here's the deploy code of that contract for reference: https://github.com/code-423n4/2023-08-livepeer/blob/23bd30274c4d426c4bb01da661ad3ef2480c6494/deploy/deploy_contracts.ts#L215-L230

> [04]
> Good docs improvements.

> [05]
> Valid points, but expected behavior, might not fix.

hickuphh3 (judge) commented:

> [01]: Recommendation
> [02]: Recommendation

> [03]: Low
> [04]: Non-Critical
> [05]: Recommendation

Gas Optimizations

For this audit, 13 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **c3phas** received the top score from the judge.

The following wardens also submitted reports: Sathish9098, kaveyjoe, 0x11singh99, SAQ, hunter_w3b, 0xta, turvy_fuzz, ReyAdmirado, JCK, lsaudit, 0x3b, *and* K42.

Notes from the Auditor

Auditor's Disclaimer

While we try our best to maintain readability in the provided code snippets, some functions have been truncated to highlight the affected portions.

It's important to note that during the implementation of these suggested changes, developers must exercise caution to avoid introducing vulnerabilities. Although the optimizations have been tested prior to these recommendations, it is the responsibility of the developers to conduct thorough testing again.

Code reviews and additional testing are strongly advised to minimize any potential risks associated with the refactoring process.

Note on Gas Estimates

We've tried to give the exact amount of gas being saved from running the included tests, whenever the function is within the test coverage.

All suggested changes have been tested per function to ensure we don't encounter stack too deep error.
Most of the changes involve rewriting functions and as such changes are suggested on a function basis.
We would love to look at this code again if they end up implementing the changes on this report.

When doing the refactoring, we've avoided including any changes that were reported by the bot and we encourage you to go through the automated findings to maximum the amount of gas being saved.

[G-01] Function `processRebond()` can be more optimized (Save 207 Gas on average)

Gas benchmarks based on function `rebondWithHint()` which calls our function of interest.

	Min	Max	Avg
Before	229425	231632	230529
After	229218	231425	230322

```
File: /contracts/bonding/BondingManager.sol
1564:     function processRebond(
1565:         address _delegator,
1566:         uint256 _unbondingLockId,
1567:         address _newPosPrev,
1568:         address _newPosNext
1569:     ) internal autoCheckpoint(_delegator) {
1570:         Delegator storage del = delegators[_delegator];
1571:         UnbondingLock storage lock = del.unbondingLocks[_unl

1573:         require(isValidUnbondingLock(_delegator, _unbonding
```

Of interest to us is the require statement on line 1573:
`require(isValidUnbondingLock(_delegator, _unbondingLockId), "invalid unbonding lock ID");`

The statement makes a function call to `isValidUnbondingLock()` which we can see its implementation on https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L1167-L1170.

```
File: /contracts/bonding/BondingManager.sol
1167:     function isValidUnbondingLock(address _delegator, uint2
```

```
1168:         // A unbonding lock is only valid if it has a non-ze
1169:         return delegators[_delegator].unbondingLocks[_unbon
1170:     }
```

Note the return statement in the above function `return delegators[_delegator].unbondingLocks[_unbondingLockId].withdrawRound > 0;`.
We make some state reads by reading `delegators[_delegator]`.

Back to our original function context, note we also made the same state read on https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L1570.

```
1570:         Delegator storage del = delegators[_delegator];
```

The require statement ends up doing the following:
`require(delegators[_delegator].unbondingLocks[_unbondingLockId].withdrawRound > 0,"invalid unbonding lock ID");`

Note the variable `delegators[_delegator].unbondingLocks[_unbondingLockId]` is equivalent to the variable `lock` declared here `UnbondingLock storage lock = del.unbondingLocks[_unbondingLockId];`

If we inline the function `isValidUnbondingLock()` we can avoid making this two state reads and take advantage of the already declared variable.

```
diff --git a/contracts/bonding/BondingManager.sol b/contracts/bo
index 93e06ba..2482d64 100644
--- a/contracts/bonding/BondingManager.sol
+++ b/contracts/bonding/BondingManager.sol
@@ -1569,8 +1569,7 @@ contract BondingManager is ManagerProxyTar
     ) internal autoCheckpoint(_delegator) {
         Delegator storage del = delegators[_delegator];
         UnbondingLock storage lock = del.unbondingLocks[_unbond
-
-        require(isValidUnbondingLock(_delegator, _unbondingLock
```

```
+           require(lock.withdrawRound > 0,"invalid unbonding lock
```

https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L249-L257

[G-02] Function `withdrawStake()` can be more optimized (Save 443 Gas on average)

	Min	Max	Avg		
Before	104389	121489	110093		
After	103946	121046	109650		

```
File: /contracts/bonding/BondingManager.sol
249:     function withdrawStake(uint256 _unbondingLockId) externa
250:         Delegator storage del = delegators[msg.sender];
251:         UnbondingLock storage lock = del.unbondingLocks[_unb

253:         require(isValidUnbondingLock(msg.sender, _unbondingL
254:         require(
255:             lock.withdrawRound <= roundsManager().currentRou
256:             "withdraw round must be before or equal to the c
257:         );
```

Similar to the previous explanations with some additional refactoring

After the refactoring explained in the previous case, the variable `lock.withdrawRound` was found to be called 3 times, with the 3rd call being a memory store ie `uint256 withdrawRound = lock.withdrawRound;` . We can optimize the code further by making this memory store call before we make our validations using the require statements and use the memory variable in the require statement instead of reading the `lock.withdrawRound` . In the worst case(sad path) where we fail on the 1st require check, we would only waste ~6 gas for `mstore/mload` . The benefit outweighs the cons thus we should refactor.

```
diff --git a/contracts/bonding/BondingManager.sol b/contracts/bo
index 93e06ba..5f0e073 100644
```

```
--- a/contracts/bonding/BondingManager.sol
+++ b/contracts/bonding/BondingManager.sol
@@ -249,15 +249,15 @@ contract BondingManager is ManagerProxyTar
     function withdrawStake(uint256 _unbondingLockId) external wh
         Delegator storage del = delegators[msg.sender];
         UnbondingLock storage lock = del.unbondingLocks[_unbond
-
-        require(isValidUnbondingLock(msg.sender, _unbondingLock
+        uint256 withdrawRound = lock.withdrawRound;
+        require(withdrawRound > 0,"invalid unbonding lock ID");
         require(
-            lock.withdrawRound <= roundsManager().currentRound(
+            withdrawRound <= roundsManager().currentRound(),
             "withdraw round must be before or equal to the curre
         );

         uint256 amount = lock.amount;
-        uint256 withdrawRound = lock.withdrawRound;
+
         // Delete unbonding lock
         delete del.unbondingLocks[_unbondingLockId];
```

https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L485-L502

[G-03] Function `transcoderWithHint()` can be more optimized (Save 4749 Gas on average)

	Min	Max	Avg	
Before	-	-	276355	
After	-	-	271606	

```
File: /contracts/bonding/BondingManager.sol
485:     function transcoderWithHint(

490:     ) public whenSystemNotPaused currentRoundInitialized {
491:         require(!roundsManager().currentRoundLocked(), "can'
492:         require(MathUtils.validPerc(_rewardCut), "invalid re
493:         require(MathUtils.validPerc(_feeShare), "invalid fee
```

```
494:         require(isRegisteredTranscoder(msg.sender), "transco

496:         Transcoder storage t = transcoders[msg.sender];
497:         uint256 currentRound = roundsManager().currentRound(

499:         require(
500:             !isActiveTranscoder(msg.sender) || t.lastRewardR
501:             "caller can't be active or must have already cal
502:         );


507:         if (!transcoderPool.contains(msg.sender)) {
508:             tryToJoinActiveSet(
509:                 msg.sender,
510:                 delegators[msg.sender].delegatedAmount,
```

The require statement on line 499 makes a call to the function `isActiveTranscoder(msg.sender)` which has the following implementation https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L1145-L1149.

```
File: /contracts/bonding/BondingManager.sol
1145:     function isActiveTranscoder(address _transcoder) public
1146:         Transcoder storage t = transcoders[_transcoder];
1147:         uint256 currentRound = roundsManager().currentRound
1148:         return t.activationRound <= currentRound && current
1149:     }
```

Note the first two lines of this implementation:

```
1146:         Transcoder storage t = transcoders[_transcoder];
1147:         uint256 currentRound = roundsManager().currentRound
```

If we look at the calling function, `transcoderWithHint()` we are making the same calls.

```
496:         Transcoder storage t = transcoders[msg.sender];
```

```
497:         uint256 currentRound = roundsManager().currentRound(
```

Additional optimizations

On line 494, we have a check `require(isRegisteredTranscoder(msg.sender), "transcoder must be registered");` that makes a function call to `isRegisteredTranscoder` which has the following implementation:

```
1156:     function isRegisteredTranscoder(address _transcoder) pul
1157:         Delegator storage d = delegators[_transcoder];
1158:         return d.delegateAddress == _transcoder && d.bonded
1159:     }
```

In our context we pass `msg.sender` as the value for `_transcoder`. If we could inline this function, we can avoid making another state read to `delegators[msg.sender]` on the line 510 in the main function. we read it here `delegators[msg.sender].delegatedAmount,`.
Since we already cache it as `d`, referencing `d` would save us some gas.

```
diff --git a/contracts/bonding/BondingManager.sol b/contracts/bo
index 93e06ba..3fc436b 100644
--- a/contracts/bonding/BondingManager.sol
+++ b/contracts/bonding/BondingManager.sol
@@ -491,13 +491,12 @@ contract BondingManager is ManagerProxyTar
         require(!roundsManager().currentRoundLocked(), "can't u
         require(MathUtils.validPerc(_rewardCut), "invalid rewar
         require(MathUtils.validPerc(_feeShare), "invalid feeSha
-        require(isRegisteredTranscoder(msg.sender), "transcoder
+        Delegator storage d = delegators[msg.sender];
+        require(d.delegateAddress == msg.sender && d.bondedAmou

         Transcoder storage t = transcoders[msg.sender];
         uint256 currentRound = roundsManager().currentRound();
-
-        require(
-            !isActiveTranscoder(msg.sender) || t.lastRewardRoun
+        require(!(t.activationRound <= currentRound && currentR
             "caller can't be active or must have already called
         );

@@ -507,7 +506,7 @@ contract BondingManager is ManagerProxyTarge
```

```
            if (!transcoderPool.contains(msg.sender)) {
                tryToJoinActiveSet(
                    msg.sender,
-                   delegators[msg.sender].delegatedAmount,
+                   d.delegatedAmount,
                    currentRound.add(1),
                    _newPosPrev,
                    _newPosNext
```

https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L842-L900

[G-04] Function `rewardWithHint()` optimization (Save 4716 Gas on average)

	Min	Average	Median	Max
Before	-	453410	-	-
After	-	448694	-	-

```
File: /contracts/bonding/BondingManager.sol
842:     function rewardWithHint(address _newPosPrev, address _ne

847:     {
848:         uint256 currentRound = roundsManager().currentRound(

850:         require(isActiveTranscoder(msg.sender), "caller must
851:         require(
852:             transcoders[msg.sender].lastRewardRound != curre
853:             "caller has already called reward for the curren
854:         );

856:         Transcoder storage t = transcoders[msg.sender];
```

We look into the implementation of the function `isActiveTranscoder` which is being called in the require statement. The function has the following implementation:

```
1145:     function isActiveTranscoder(address _transcoder) public
```

```
1146:         Transcoder storage t = transcoders[_transcoder];
1147:         uint256 currentRound = roundsManager().currentRound
1148:         return t.activationRound <= currentRound && current
1149:     }
```

Put the above implementation in the context of our main function and we see two similar lines in both functions.

```
1146:         Transcoder storage t = transcoders[_transcoder];
1147:         uint256 currentRound = roundsManager().currentRound
```

This means we are making this calls twice, one in the function being called, and the other one by the calling function.
To avoid this, we can inline the function `isActiveTranscoder()` and refactor the code as follows:

```
diff --git a/contracts/bonding/BondingManager.sol b/contracts/bo
index 93e06ba..34c2832 100644
--- a/contracts/bonding/BondingManager.sol
+++ b/contracts/bonding/BondingManager.sol
@@ -846,14 +846,13 @@ contract BondingManager is ManagerProxyTar
         autoCheckpoint(msg.sender)
     {
         uint256 currentRound = roundsManager().currentRound();
-
-        require(isActiveTranscoder(msg.sender), "caller must be
+        Transcoder storage t = transcoders[msg.sender];
+        require(t.activationRound <= currentRound && currentRou
         require(
-            transcoders[msg.sender].lastRewardRound != currentR
+            t.lastRewardRound != currentRound,
             "caller has already called reward for the current r
         );
-
-        Transcoder storage t = transcoders[msg.sender];
+
         EarningsPool.Data storage earningsPool = t.earningsPool

         // Set last round that transcoder called reward
```

https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L1307-L1345

[G-05] Function `increaseTotalStakeUncheckpointed()` can be more optimized (save 148 Gas on average)

Gas benchmarks based on function `rewardWithHint` that calls our internal function.

	Min	Average	Median	Max
Before	-	453410	-	-
After	-	453262	-	-

```
File: /contracts/bonding/BondingManager.sol
1307:     function increaseTotalStakeUncheckpointed(

1312:     ) internal {

1318:         if (isRegisteredTranscoder(_delegate)) {
1319:             uint256 currRound = roundsManager().currentRoun
1320:             uint256 nextRound = currRound.add(1);

1343:         // Increase delegate's delegated amount
1344:         delegators[_delegate].delegatedAmount = newStake;
1345:     }
```

The if statement makes a call to a function `isRegisteredTranscoder(_delegate)` whose implementation is as follows:

```
1156:     function isRegisteredTranscoder(address _transcoder) pul
1157:         Delegator storage d = delegators[_transcoder];
1158:         return d.delegateAddress == _transcoder && d.bonded
1159:     }
```

Note in the function above, we cache `delegators[_transcoder]` with `_transcoder` being the address passed to the function, so in the context of our function it would look like `Delegator storage d = delegators[_delegate];` On

the calling function line 1344, we write to the state variable `delegators[_delegate].delegatedAmount`, as we already cached the variable `delegators[_delegate]` we don't need to call it directly but instead we should use the cached reference. As the cached one is simply a pointer to the original one, we would still be writing to the state variable as required.

```
-         if (isRegisteredTranscoder(_delegate)) {
+         Delegator storage d = delegators[_delegate];
+         if (d.delegateAddress == _delegate && d.bondedAmount > 
              uint256 currRound = roundsManager().currentRound();
              uint256 nextRound = currRound.add(1);

@@ -1341,8 +1341,7 @@ contract BondingManager is ManagerProxyTar
          }

          // Increase delegate's delegated amount
-         delegators[_delegate].delegatedAmount = newStake;
-     }
+         d.delegatedAmount = newStake;//@audit Refactor the call
```

https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L745-L784

[G-06] Refactor the function `unbondWithHint()` (Save 1452 Gas)

	Min	Average	Median	Max
Before	-	509309	-	-
After	-	507857	-	-

```
File: /contracts/bonding/BondingManager.sol
745:     function unbondWithHint(

749:     ) public whenSystemNotPaused currentRoundInitialized aut
750:         require(delegatorStatus(msg.sender) == DelegatorStat

752:         Delegator storage del = delegators[msg.sender];
```

The require statement calls the function `delegatorStatus(msg.sender)` which has the implementation:

```
956:    function delegatorStatus(address _delegator) public view
957:        Delegator storage del = delegators[_delegator];

959:        if (del.bondedAmount == 0) {
960:            // Delegator unbonded all its tokens
961:            return DelegatorStatus.Unbonded;
962:        } else if (del.startRound > roundsManager().currentR
963:            // Delegator round start is in the future
964:            return DelegatorStatus.Pending;
965:        } else {

969:            return DelegatorStatus.Bonded;
970:        }
971:    }
```

Note we make a state read `Delegator storage del = delegators[_delegator];.`

In the calling function , we pass `msg.sender` as the function parameter which means our state read in the function being called is `Delegator storage del = delegators[msg.sender];` which is the same state read we make in the main function. This means we are making this call twice(too much gas).

For this to work, we introduce a new state variable to keep track of the status as shown below.

```
diff --git a/contracts/bonding/BondingManager.sol b/contracts/bo
index 93e06ba..3ec3b73 100644
--- a/contracts/bonding/BondingManager.sol
+++ b/contracts/bonding/BondingManager.sol
@@ -33,6 +33,7 @@ contract BondingManager is ManagerProxyTarget,

     // Time between unbonding and possible withdrawl in rounds
     uint64 public unbondingPeriod;
+        DelegatorStatus delegatorStatus_;

     // Represents a transcoder's current state
     struct Transcoder {
@@ -742,20 +743,34 @@ contract BondingManager is ManagerProxyTar
```

```
         * @param _newPosPrev Address of previous transcoder in poo
         * @param _newPosNext Address of next transcoder in pool if
         */
+
        function unbondWithHint(
            uint256 _amount,
            address _newPosPrev,
            address _newPosNext
        ) public whenSystemNotPaused currentRoundInitialized autoCl
-           require(delegatorStatus(msg.sender) == DelegatorStatus.

            Delegator storage del = delegators[msg.sender];
-
            require(_amount > 0, "unbond amount must be greater tha
            require(_amount <= del.bondedAmount, "amount is greater
+           uint256 currentRound = roundsManager().currentRound();
+           if (del.bondedAmount == 0) {
+               // Delegator unbonded all its tokens
+               delegatorStatus_ = DelegatorStatus.Unbonded;
+           } else if (del.startRound > currentRound) {
+               // Delegator unbonded all its tokens
+               delegatorStatus_ = DelegatorStatus.Pending;
+           } else {
+               // Delegator round start is now or in the past
+               // del.startRound != 0 here because if del.startRou
+               // would trigger the first if clause
+               delegatorStatus_ = DelegatorStatus.Bonded;
+           }
+
+           require(delegatorStatus_ == DelegatorStatus.Bonded, "ca

            address currentDelegate = del.delegateAddress;
-           uint256 currentRound = roundsManager().currentRound();
+
            uint256 withdrawRound = currentRound.add(unbondingPerio
            uint256 unbondingLockId = del.nextUnbondingLockId;
```

https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L394-L418

[G-07] Function `slashTranscoder()` can be optimized

Note: not covered by the tests.

```
File: /contracts/bonding/BondingManager.sol
394:     function slashTranscoder(

399:     ) external whenSystemNotPaused onlyVerifier autoClaimEar
400:         Delegator storage del = delegators[_transcoder];

402:         if (del.bondedAmount > 0) {
403:             uint256 penalty = MathUtils.percOf(delegators[_t

413:             // If still bonded decrease delegate's delegated
414:             if (delegatorStatus(_transcoder) == DelegatorSta
415:                 delegators[del.delegateAddress].delegatedAmo
416:                     penalty
417:                 );
418:             }
```

The function call `delegatorStatus(_transcoder)` ends up making a similar state read like the one in the calling function ie `Delegator storage del = delegators[_transcoder];`.
We can refactor the code to avoid this as follows.

```
     // Time between unbonding and possible withdrawl in rounds
     uint64 public unbondingPeriod;
+    DelegatorStatus delegatorStatus_;

+
     function slashTranscoder(
         address _transcoder,
         address _finder,
@@ -400,7 +402,7 @@ contract BondingManager is ManagerProxyTarge
         Delegator storage del = delegators[_transcoder];

         if (del.bondedAmount > 0) {
-            uint256 penalty = MathUtils.percOf(delegators[_tran
+            uint256 penalty = MathUtils.percOf(del.bondedAmount

             // If active transcoder, resign it
             if (transcoderPool.contains(_transcoder)) {
@@ -410,8 +412,21 @@ contract BondingManager is ManagerProxyTarg
             // Decrease bonded stake
             del.bondedAmount = del.bondedAmount.sub(penalty);

+            if (del.bondedAmount == 0) {
```

```
+                // Delegator unbonded all its tokens
+                delegatorStatus_= DelegatorStatus.Unbonded;
+            } else if (del.startRound > roundsManager().currentl
+                // Delegator round start is in the future
+                delegatorStatus_ =  DelegatorStatus.Pending;
+            } else {
+                // Delegator round start is now or in the past
+                // del.startRound != 0 here because if del.star
+                // would trigger the first if clause
+                delegatorStatus_ =  DelegatorStatus.Bonded;
+            }
+
             // If still bonded decrease delegate's delegated am
-            if (delegatorStatus(_transcoder) == DelegatorStatus
+            if (delegatorStatus_ == DelegatorStatus.Bonded) {
                 delegators[del.delegateAddress].delegatedAmount
                     penalty
                 );
```

https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingVotes.sol#L258-L294

[G-08] BondingVotes.sol: We can optimize the function `checkpointBondingState`

Note: no gas estimates as it was not covered by the tests.

```
File: /contracts/bonding/BondingVotes.sol
258:     function checkpointBondingState(

266:     ) external virtual onlyBondingManager {

273:         BondingCheckpoint memory previous;
274:         if (hasCheckpoint(_account)) {
275:             previous = getBondingCheckpointAt(_account, _sta
276:         }

278:         BondingCheckpointsByRound storage checkpoints = bond
```

The if statement on line 274 makes two function calls `hasCheckpoint(_account)` and `getBondingCheckpointAt(_account, _startRound)`.

The implementation of `hasCheckpoint(_account)` is as below:

```
315:     function hasCheckpoint(address _account) public view ret
316:         return bondingCheckpoints[_account].startRounds.leng
317:     }
```

We can see the return statement reads `bondingCheckpoints[_account]` which we already had cached in the main function as `checkpoints`. Inline the `hasCheckpoint` function would help us use the cached value.

The next function `getBondingCheckpointAt()` has an implementation that has the following state read:

```
BondingCheckpointsByRound storage checkpoints =
bondingCheckpoints[_account];
```

Which is equivalent to the variable we cached in the main function. Inlining this function would avoid making two state reads which are quite expensive.

We should refactor the code as follows:

```
diff --git a/contracts/bonding/BondingVotes.sol b/contracts/bond
index 2408c66..ca92f91 100644
--- a/contracts/bonding/BondingVotes.sol
+++ b/contracts/bonding/BondingVotes.sol
@@ -271,26 +271,44 @@ contract BondingVotes is ManagerProxyTarge
         }

         BondingCheckpoint memory previous;
-        if (hasCheckpoint(_account)) {
-            previous = getBondingCheckpointAt(_account, _startR
-        }
-
         BondingCheckpointsByRound storage checkpoints = bonding
+        if (checkpoints.startRounds.length > 0){
+            if (_startRound > clock() + 1) {
+                revert FutureLookup(_startRound, clock() + 1);
+            }
```

```
+            // Most of the time we will be calling this for a t
+            // On those cases we will have a checkpoint for exa
+            BondingCheckpoint storage bond = checkpoints.data[_
+            if (bond.bondedAmount > 0) {
+                previous = bond;
+            }
+
+            uint256 startRoundIdx = checkpoints.startRounds.fin
+            if (startRoundIdx == checkpoints.startRounds.length
+                // No checkpoint at or before _round, so return
+                // are no checkpoints for _account. The voting
+                previous = bond;
+            }
+
+            uint256 startRound = checkpoints.startRounds[startR
+            previous = checkpoints.data[startRound];
+
+            }

-        BondingCheckpoint memory bond = BondingCheckpoint({
+        BondingCheckpoint memory _bond = BondingCheckpoint({
            bondedAmount: _bondedAmount,
            delegateAddress: _delegateAddress,
            delegatedAmount: _delegatedAmount,
            lastClaimRound: _lastClaimRound,
            lastRewardRound: _lastRewardRound
        });
-       checkpoints.data[_startRound] = bond;
+       checkpoints.data[_startRound] = _bond;

        // now store the startRound itself in the startRounds a
        // to find it and lookup in the above mapping
        checkpoints.startRounds.pushSorted(_startRound);

-       onBondingCheckpointChanged(_account, previous, bond);
+       onBondingCheckpointChanged(_account, previous, _bond);
    }
```

[G-09] Optimizing check order for cost efficient function execution

Checks that involve constants should come before checks that involve state variables, function calls, and calculations. By doing these checks first, the function is able to

revert before wasting a Gcoldsload (2100 gas) in a function that may ultimately revert in the unhappy case.

https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L745-L784

[G-09-01] Validate the function parameter `_amount` first before making any state reads/writes

```
File: /contracts/bonding/BondingManager.sol
745:     function unbondWithHint(

749:     ) public whenSystemNotPaused currentRoundInitialized aut
750:         require(delegatorStatus(msg.sender) == DelegatorStat

752:         Delegator storage del = delegators[msg.sender];

755:         require(_amount > 0, "unbond amount must be greater
```

```
diff --git a/contracts/bonding/BondingManager.sol b/contracts/bo
index 93e06ba..c43f3e3 100644
--- a/contracts/bonding/BondingManager.sol
+++ b/contracts/bonding/BondingManager.sol
@@ -747,11 +747,12 @@ contract BondingManager is ManagerProxyTar
         address _newPosPrev,
         address _newPosNext
     ) public whenSystemNotPaused currentRoundInitialized autoCl
+        require(_amount > 0, "unbond amount must be greater tha
+
         require(delegatorStatus(msg.sender) == DelegatorStatus.

         Delegator storage del = delegators[msg.sender];

-        require(_amount > 0, "unbond amount must be greater tha
         require(_amount <= del.bondedAmount, "amount is greater
```

[G-10] Change the model of how events are emitted

Currently if we look at the event declaration we see something like this `event ParameterUpdate(string param);` and the actual emit as `emit ParameterUpdate("unbondingPeriod");`.
This method seems quite interesting especially for readability as we can clearly tell what the event is about. However, this method is quite expensive and an anti pattern. We can actually achieve the same readability by using named parameters for our event argument and naming the events using a descriptive name.

https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L155-L159

[G-10-01] BondingManager.sol.setUnbondingPeriod(): emit `_unbondingPeriod` instead of `"unbondingPeriod"` (Save 593 Gas on average)

	Min	Max	Avg
Before	-	-	61189
After	-	-	60596

```
File: /contracts/bonding/BondingManager.sol
155:     function setUnbondingPeriod(uint64 _unbondingPeriod) ext
156:         unbondingPeriod = _unbondingPeriod;

158:         emit ParameterUpdate("unbondingPeriod");
159:     }
```

```
diff --git a/contracts/bonding/BondingManager.sol b/contracts/bo
index 93e06ba..482d4e9 100644
--- a/contracts/bonding/BondingManager.sol
+++ b/contracts/bonding/BondingManager.sol
@@ -136,6 +136,7 @@ contract BondingManager is ManagerProxyTarge
         _;
         _checkpointBondingState(_account, delegators[_account],
     }
+    event setUnbondingPeriodNewValue(uint256 unbondingPeriod);

     /**
      * @notice BondingManager constructor. Only invokes constru
```

```
@@ -155,7 +156,7 @@ contract BondingManager is ManagerProxyTarge
     function setUnbondingPeriod(uint64 _unbondingPeriod) externa
         unbondingPeriod = _unbondingPeriod;

-        emit ParameterUpdate("unbondingPeriod");
+        emit setUnbondingPeriodNewValue(_unbondingPeriod);
     }
```

https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L186-L190

[G-10-02] BondingManager.sol.setNumActiveTranscoders(): Emit `_numActiveTranscoders` instead of `"numActiveTranscoders"` (Save 584 Gas on average)

	Min	Max	Avg
Before	47192	64292	55742
After	46608	63708	55158

```
File: /contracts/bonding/BondingManager.sol
186:     function setNumActiveTranscoders(uint256 _numActiveTrans
187:         transcoderPool.setMaxSize(_numActiveTranscoders);

189:         emit ParameterUpdate("numActiveTranscoders");
190:     }



+             event setNumActiveTranscodersCeilingNewValue(uint25
+

     /**
      * @notice BondingManager constructor. Only invokes constru
@@ -186,7 +188,7 @@ contract BondingManager is ManagerProxyTarge
     function setNumActiveTranscoders(uint256 _numActiveTranscod
         transcoderPool.setMaxSize(_numActiveTranscoders);

-        emit ParameterUpdate("numActiveTranscoders");
+        emit setNumActiveTranscodersCeilingNewValue(_numActiveT
     }
```

The following are not covered by the tests thus we couldn't give an exact amount of gas being saved.

https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L176-L180

[G-10-03] BondingManager.sol.setTreasuryBalanceCeiling(): Emit `_ceiling` instead of `"treasuryBalanceCeiling"`

```
File: /contracts/bonding/BondingManager.sol
176:     function setTreasuryBalanceCeiling(uint256 _ceiling) ext
177:         treasuryBalanceCeiling = _ceiling;

179:         emit ParameterUpdate("treasuryBalanceCeiling");
180:     }
```

```
+        event settreasuryBalanceCeilingNewValue(uint256 treasur
+
     /**
      * @notice BondingManager constructor. Only invokes constru
      * @dev This constructor will not initialize any state vari
@@ -176,7 +178,7 @@ contract BondingManager is ManagerProxyTarge
     function setTreasuryBalanceCeiling(uint256 _ceiling) extern
         treasuryBalanceCeiling = _ceiling;

-        emit ParameterUpdate("treasuryBalanceCeiling");
+        emit settreasuryBalanceCeilingNewValue(_ceiling);
```

https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L462-L469

[G-10-04] BondingManager.sol.setCurrentRoundTotalActiveStake(): Emit `treasuryRewardCutRate` instead of `"treasuryRewardCutRate"`

```
File: /contracts/bonding/BondingManager.sol
462:     function setCurrentRoundTotalActiveStake() external onlyl
```

```
463:         currentRoundTotalActiveStake = nextRoundTotalActiveS

465:         if (nextRoundTreasuryRewardCutRate != treasuryReward
466:             treasuryRewardCutRate = nextRoundTreasuryRewardCu
467:             // The treasury cut rate changes in a delayed fa
468:             emit ParameterUpdate("treasuryRewardCutRate");
469:         }
```

```
+    event setCurrentRoundTotalActiveStakeNewValue(uint256 treasu
```

```
     /**
      * @notice BondingManager constructor. Only invokes constru
@@ -465,7 +466,7 @@ contract BondingManager is ManagerProxyTarge
         if (nextRoundTreasuryRewardCutRate != treasuryRewardCutR
             treasuryRewardCutRate = nextRoundTreasuryRewardCutRa
             // The treasury cut rate changes in a delayed fashi
-            emit ParameterUpdate("treasuryRewardCutRate");
+            emit setCurrentRoundTotalActiveStakeNewValue(treasu
         }
```

https://github.com/code-423n4/2023-08-livepeer/blob/a3d801fa4690119b6f96aeb5508e58d752bda5bc/contracts/bonding/BondingManager.sol#L1176-L1182

[G-10-05] BondingManager.sol._setTreasuryRewardCutRate(): Emit `_cutRate` instead of `"nextRoundTreasuryRewardCutRate"`

```
File: /contracts/bonding/BondingManager.sol
1176:     function _setTreasuryRewardCutRate(uint256 _cutRate) in
1177:         require(PreciseMathUtils.validPerc(_cutRate), "_cutR

1179:         nextRoundTreasuryRewardCutRate = _cutRate;

1181:         emit ParameterUpdate("nextRoundTreasuryRewardCutRate
1182:     }
```

```
+    event setTreasuryRewardCutRateNewValue(uint256 nextRoundTrea
     /**
      * @notice BondingManager constructor. Only invokes constru
      * @dev This constructor will not initialize any state vari
```

```
@@ -1178,7 +1178,7 @@ contract BondingManager is ManagerProxyTar
         nextRoundTreasuryRewardCutRate = _cutRate;

-        emit ParameterUpdate("nextRoundTreasuryRewardCutRate");
+        emit setTreasuryRewardCutRateNewValue(_cutRate);
     }
```

Conclusion

It is important to emphasize that the provided recommendations aim to enhance the efficiency of the code without compromising its readability. We understand the value of maintainable and easily understandable code to both developers and auditors.

As you proceed with implementing the suggested optimizations, please exercise caution and be diligent in conducting thorough testing. It is crucial to ensure that the changes are not introducing any new vulnerabilities and that the desired performance improvements are achieved. Review code changes, and perform thorough testing to validate the effectiveness and security of the refactored code.

Should you have any questions or need further assistance, please don't hesitate to reach out.

victorges (Livepeer) acknowledged

Audit Analysis

For this audit, 7 analysis reports were submitted by wardens. An analysis report examines the codebase as a whole, providing observations and advice on such topics as architecture, mechanism, or approach. The **report highlighted below** by **catellatech** received the top score from the judge.

The following wardens also submitted reports: **Sathish9098**, **JayShreeRAM**, **ADM**, **BanditxOx**, **0x3b**, *and* **Krace**.

Description overview of Livepeer Onchain Treasury Upgrade

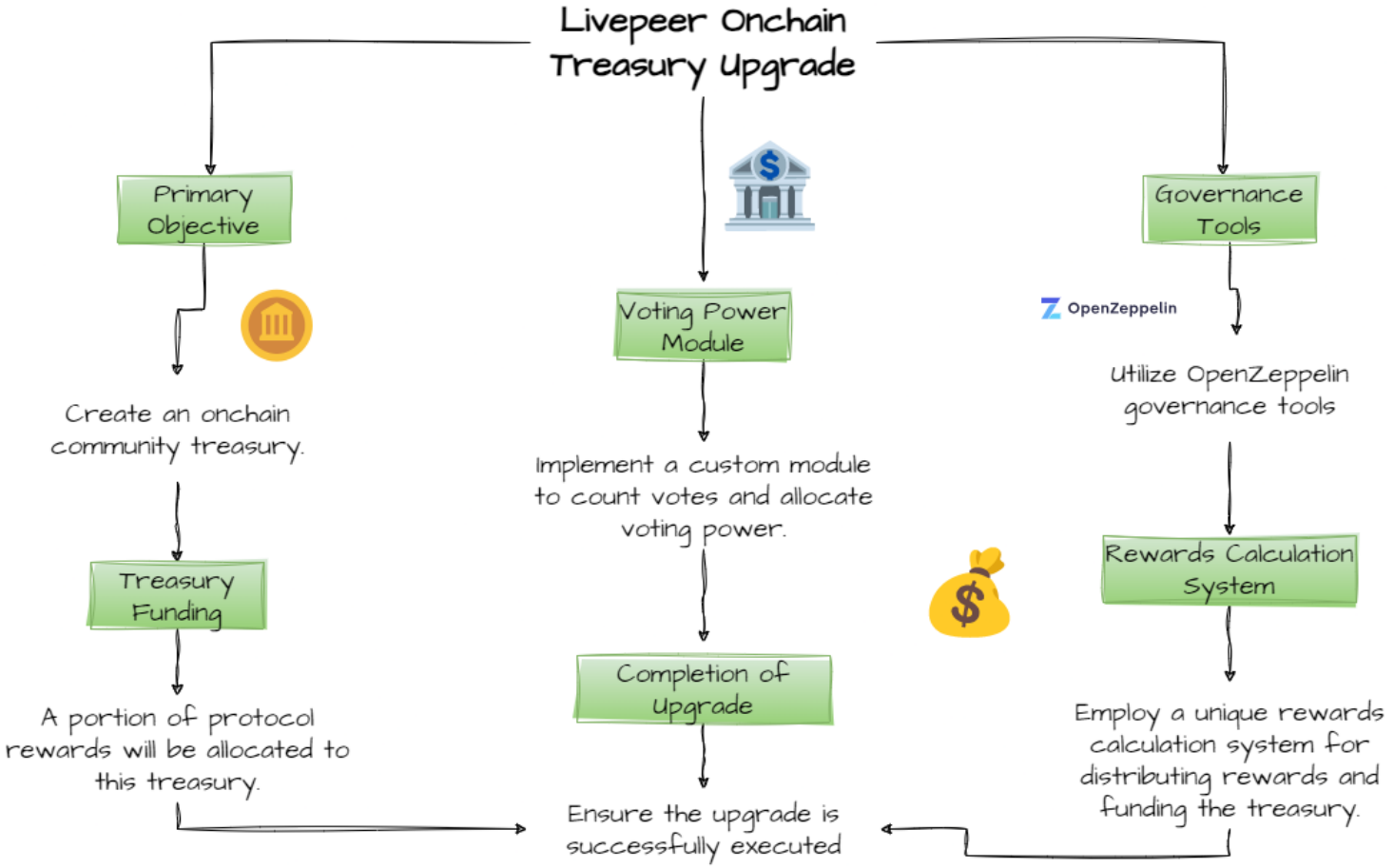
The `Livepeer Onchain Treasury Upgrade` is a significant enhancement to the `Livepeer protocol`, which is a decentralized video infrastructure platform used in

`web3's` social media and media applications. The primary objective of this upgrade is to create an `onchain community treasury`. This treasury is funded through a percentage of protocol rewards generated by the network.

To achieve this, `OpenZeppelin governance` tools are leveraged, and a custom voting power module is implemented to facilitate onchain voting. The upgrade process is detailed in various technical documents such as LIP-91, LIP-89, and LIP-92, which provide a comprehensive blueprint for the upgrade's implementation.

A critical aspect of this upgrade is ensuring that there are no significant changes to the behavior of the `core protocol contract`, known as the `BondingManager`, except for the specific modifications required to accommodate the community treasury. Attention is also paid to maintaining the security and integrity of the protocol to avoid introducing new vulnerabilities.

Additionally, it's worth noting that Livepeer employs a unique `staking rewards` calculation system, which is used not only for distributing rewards to participants but also for funding the community treasury.



1- System Overview

Scope

- BondingManager.sol: An existing contract that oversees the management of protocol bonds, often referred to as staking, as well as rewards. This upgrade introduces the capability to facilitate state checkpointing and the creation of treasury rewards. This contract holds paramount significance within the audit due to its pivotal role in the current protocol.
- BondingVotes.sol: This contract stores checkpoints of the BondingManager state to facilitate an IERC5805Upgradeable (IVotes) implementation for the OpenZeppelin Governor.
- EarningsPoolLIP36.sol: This library offers utility functions to calculate staking rewards utilizing the LIP-36 cumulative earnings algorithm.
- SortedArrays.sol: This library offers assistance for managing and searching within sorted arrays. It extends the functionality of Arrays.findUpperBound to include an equivalent findLowerBound for checkpoint retrieval.
- GovernorCountingOverridable.sol: Abstract contract that implements an OpenZeppelin Governor counting module, enabling delegators to override their delegates' (transcoders) votes on a proposal.
- Treasury.sol: This contract inherits from TimelockControllerUpgradeable to enable initialization. It is used by the governor to hold funds and execute proposals.
- LivepeerGovernor.sol: This contract serves as the practical Governor implementation, bringing together both the OZ built-in and custom modules and extensions into a tangible contract that can be instantiated and utilized. It primarily handles the framework necessary to assemble these components.

Privileged Roles

Some privileged roles exercise powers over the Controller contract:

- **TicketBroker**

```
// Check if sender is TicketBroker
modifier onlyTicketBroker() {
    _onlyTicketBroker();
    _;
}
```

- **RoundManager**

```
// Check if sender is RoundsManager
modifier onlyRoundsManager() {
    _onlyRoundsManager();
    _;
}
```

- Verifier

```
// Check if sender is Verifier
modifier onlyVerifier() {
    _onlyVerifier();
    _;
}
```

We asked the sponsors about who will be responsible for these roles, and this was their response:

> victorges — 09/2/2023 at 12:20
> Hey warden!
> The roles are managed by a special controller owner address whic

> dob | Livepeer — 09/2/2023 at 3:29
> I have seen a few questions come through about the Verifier role
>
> Only the registered "Verifier" in the Controller contract can ca

Given the level of control that these roles possess within the system, users must place full trust in the fact that the entities overseeing these roles will always act correctly and in the best interest of the system and its users.

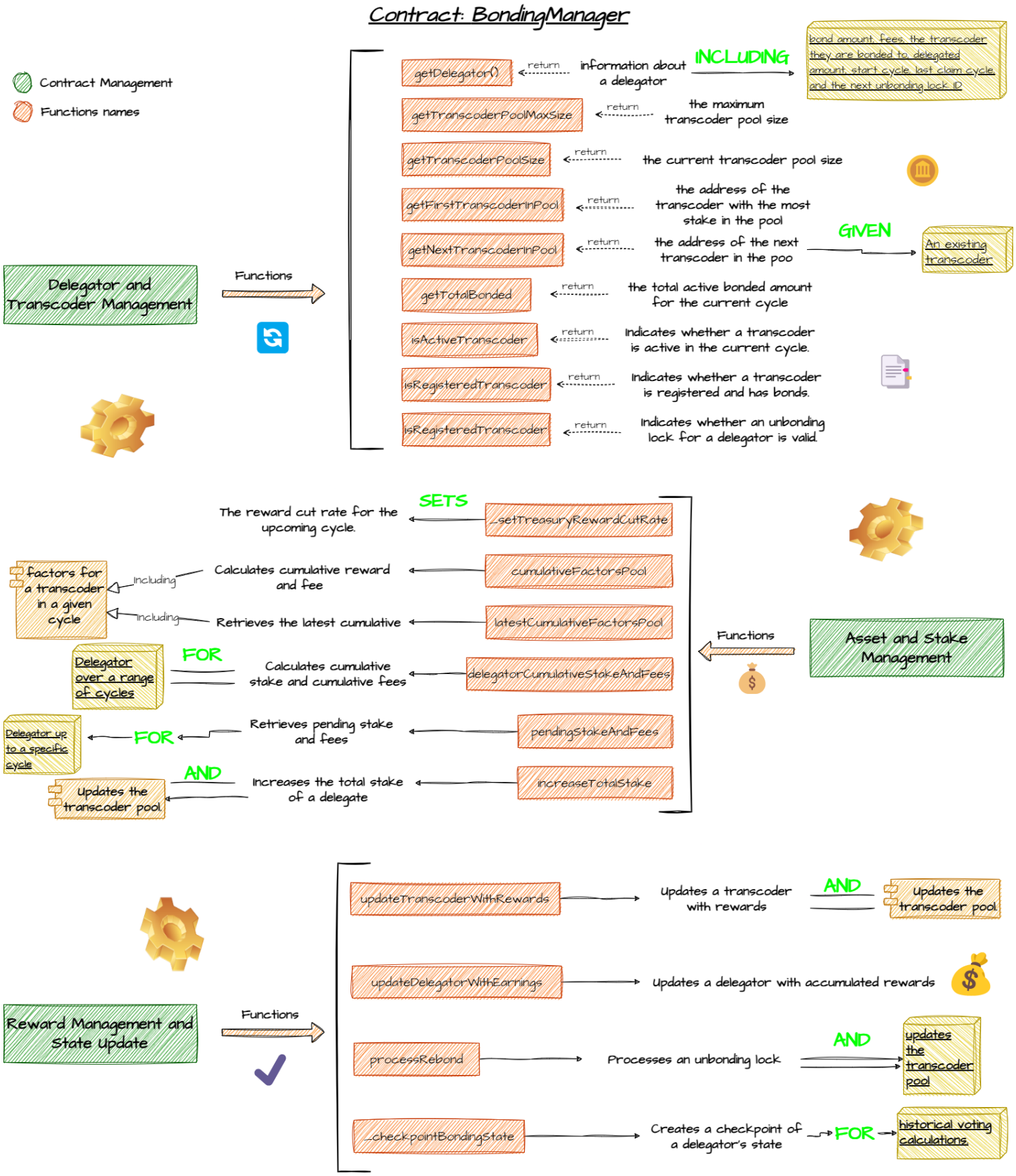
2- Codebase analysis through diagrams.

The main most important contracts of `Livepeer Onchain Treasury Upgrade`.

- In this audit, the protocol provided `5 contracts` and `2 libraries`. Here's a detailed diagrams of the essential components of each ones:

Contracts:

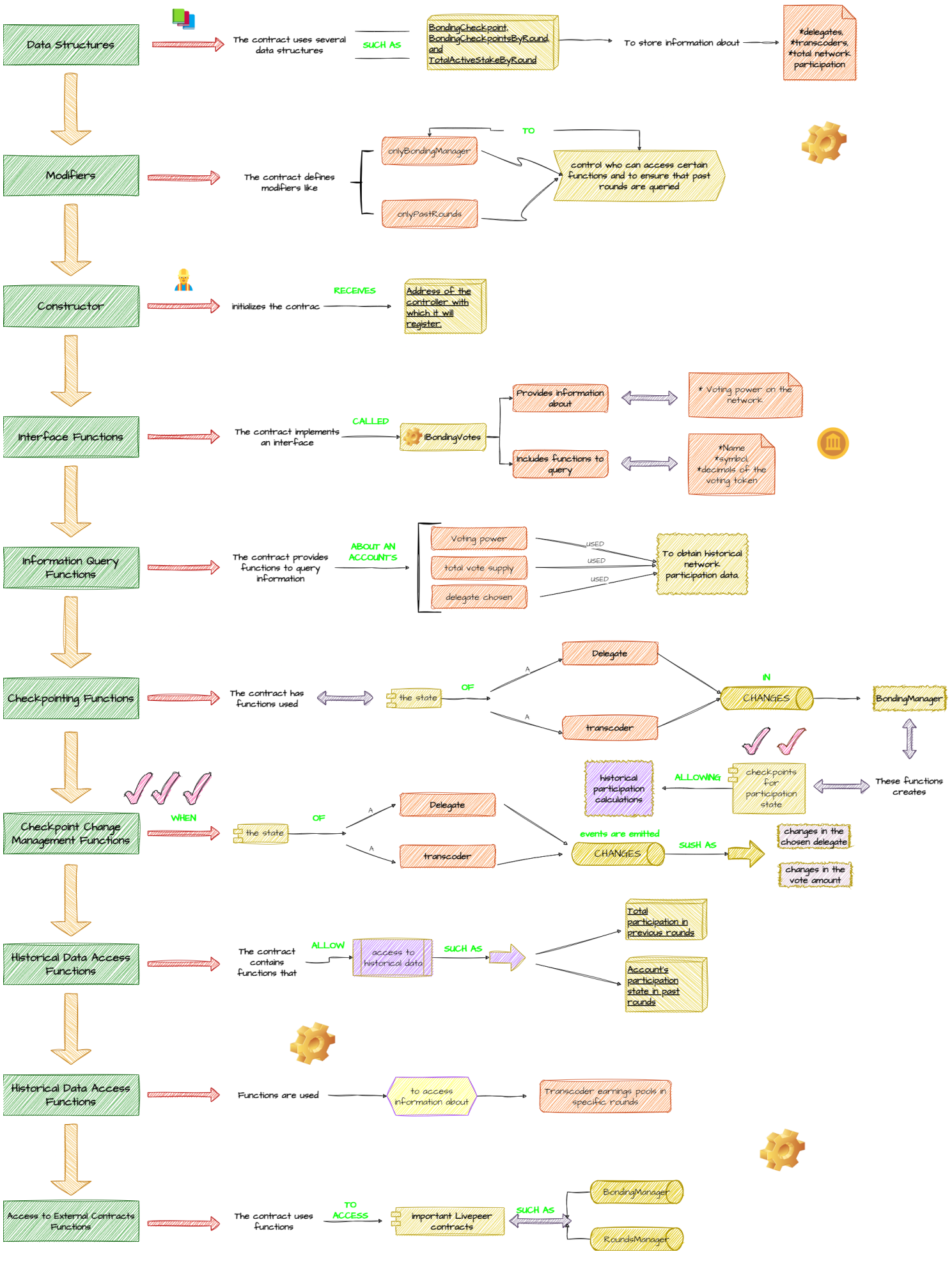
1. BondingManager:



We have created a diagram organized into sections to facilitate understanding for both `developers` and `auditors` of the main contract. This contract is responsible for managing three fundamental areas: `Delegator and Transcoder Management`, `Asset and Stake Management`, and `Reward Management and State Update`.

The contract plays a critical role in managing protocol `bonds` and `rewards`. A significant upgrade adds support for creating state checkpointing and generating rewards for the protocol's treasury. Additionally, the code includes sections related to the management of a decentralized video transcoding system, where delegates actively participate, and transcoders compete for rewards.

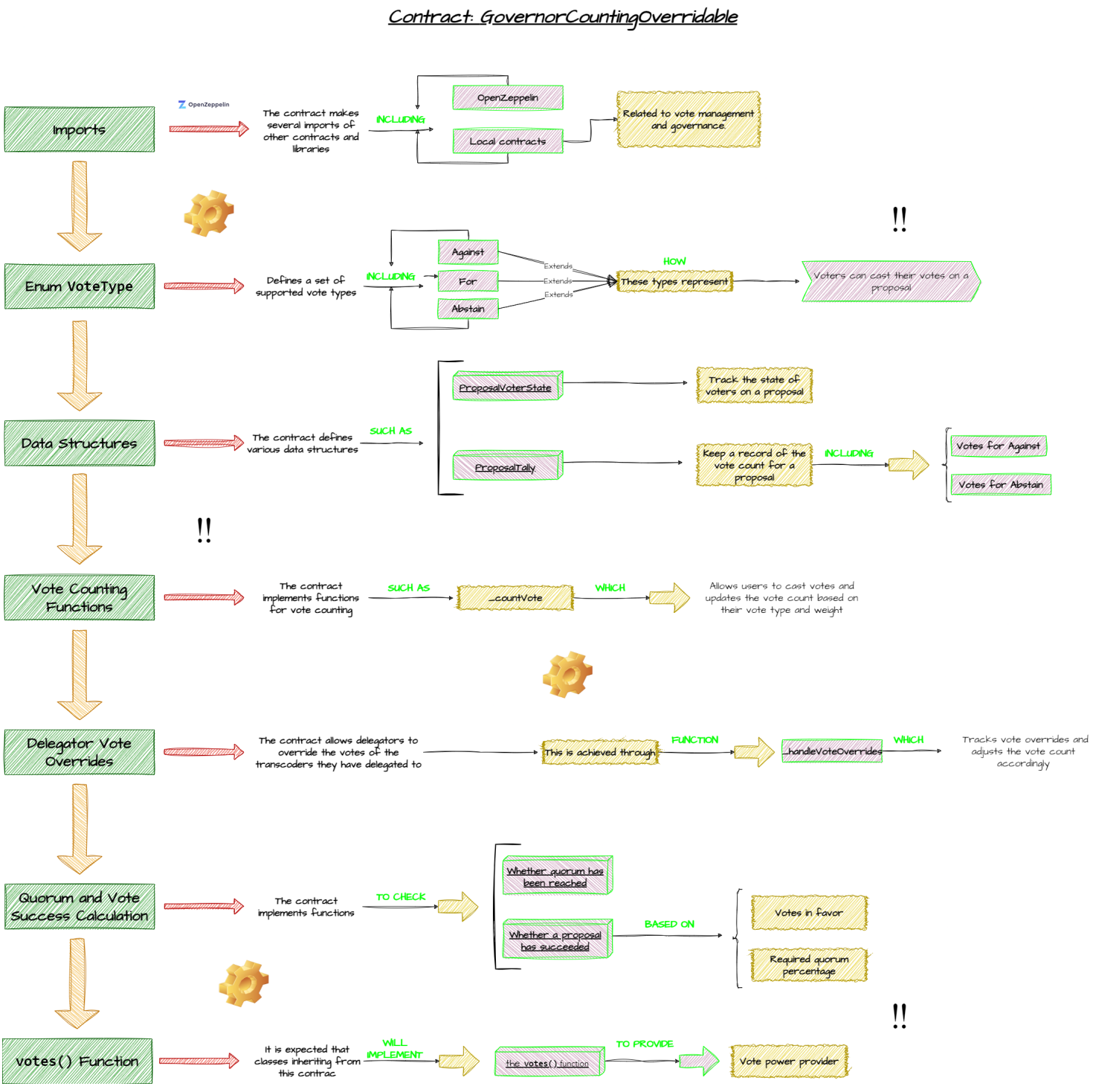
2. BondingVotes:



In this `BondingVotes` diagram, we provide a concise description of each function implemented by the contract. `BondingVotes` is responsible for the `checkpointing`

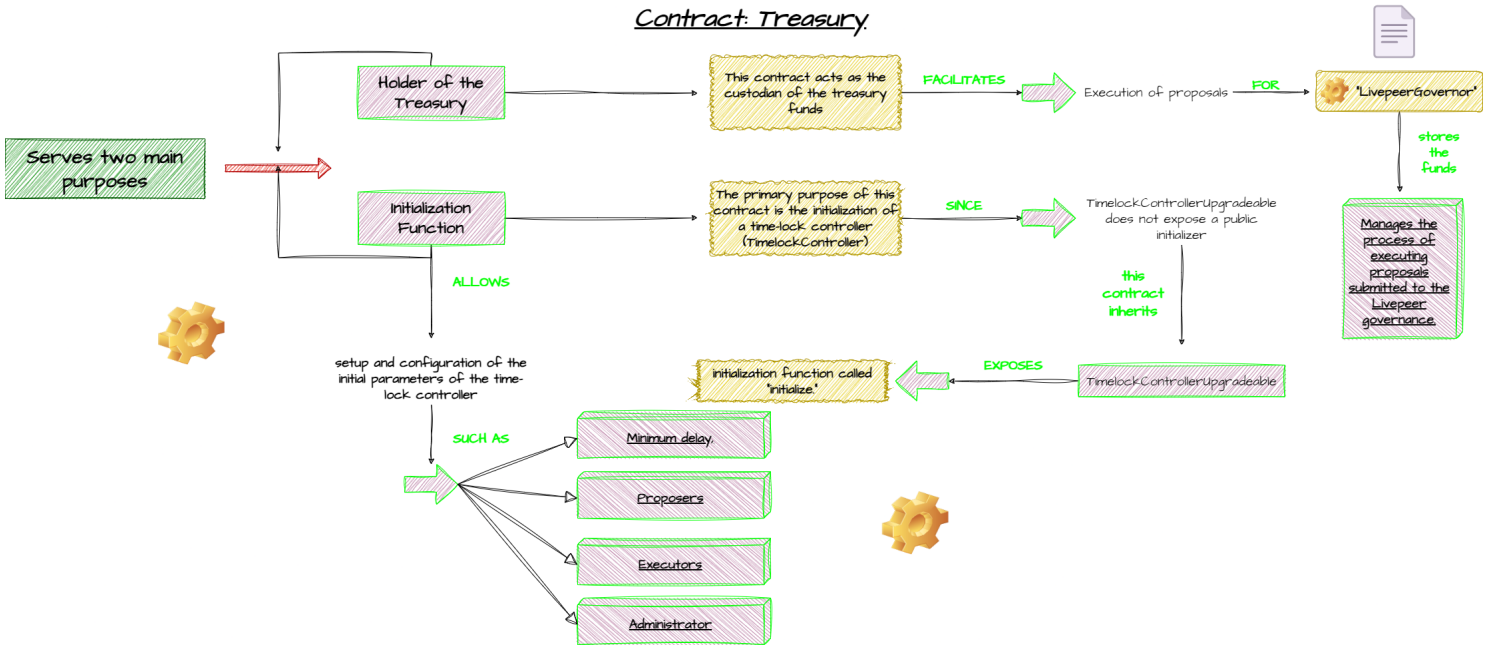
logic for the state of the `BondingManager` and its primary purpose is to perform `historical calculations` related to participation in the `Livepeer` network.

3. GovernorCountingOverridable:



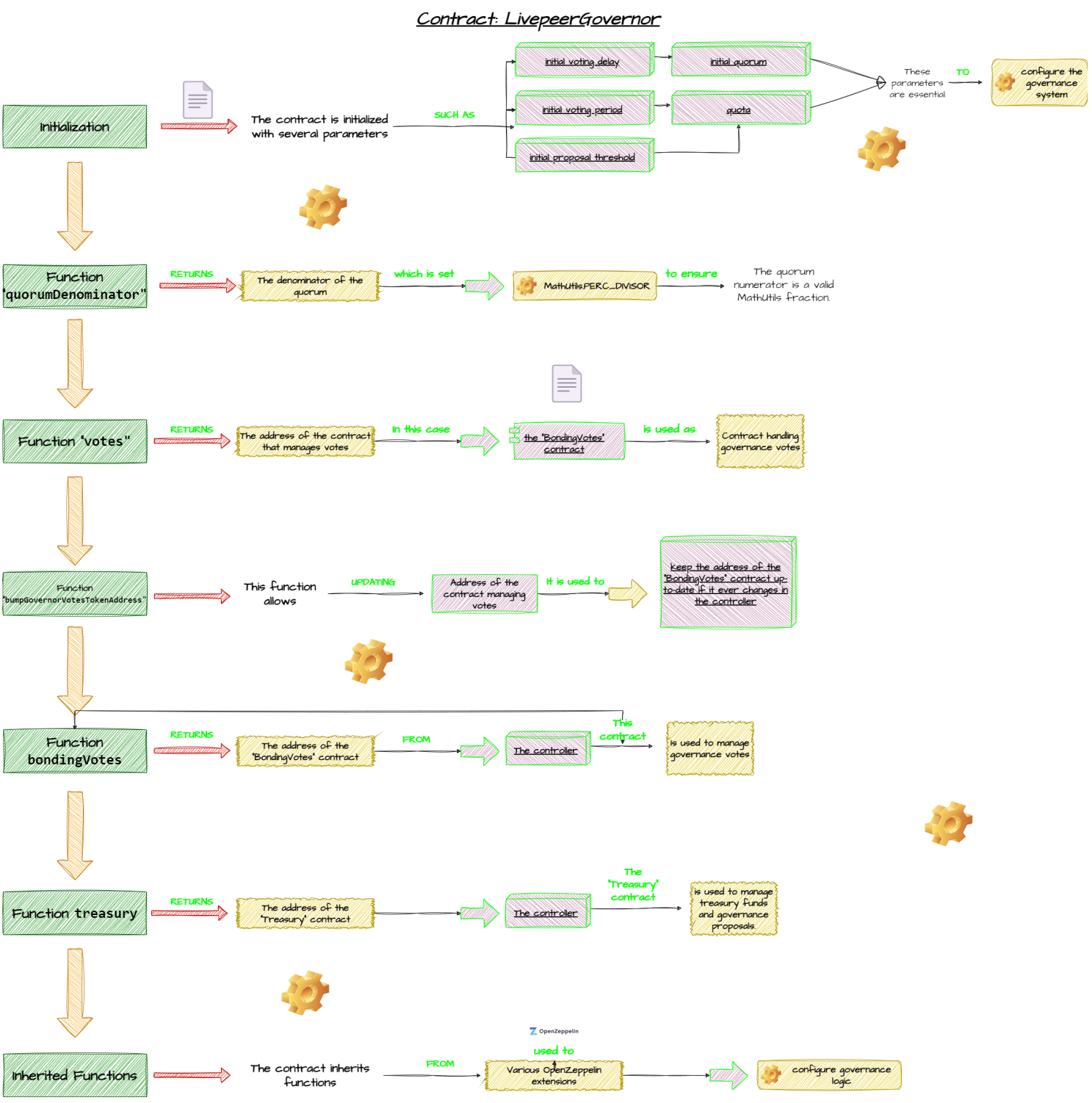
In this `diagram`, we review the `functions` performed in the contract and their respective purposes. The `GovernorCountingOverridable` contract serves as a foundation for implementing `decentralized governance systems` that allow `delegates` to override the votes of the `transcoders` they have delegated to. It also provides logic for `vote counting` and determining the `success of proposals`.

4. Treasury:



The `Treasury` contract is used to `manage treasury funds` and `governance proposals` in `Livepeer`, and its `initialization` function is `responsible` for `configuring` the `time-lock controller` for `governance`. The two main purposes of this contract are summarized in the provided `diagram`.

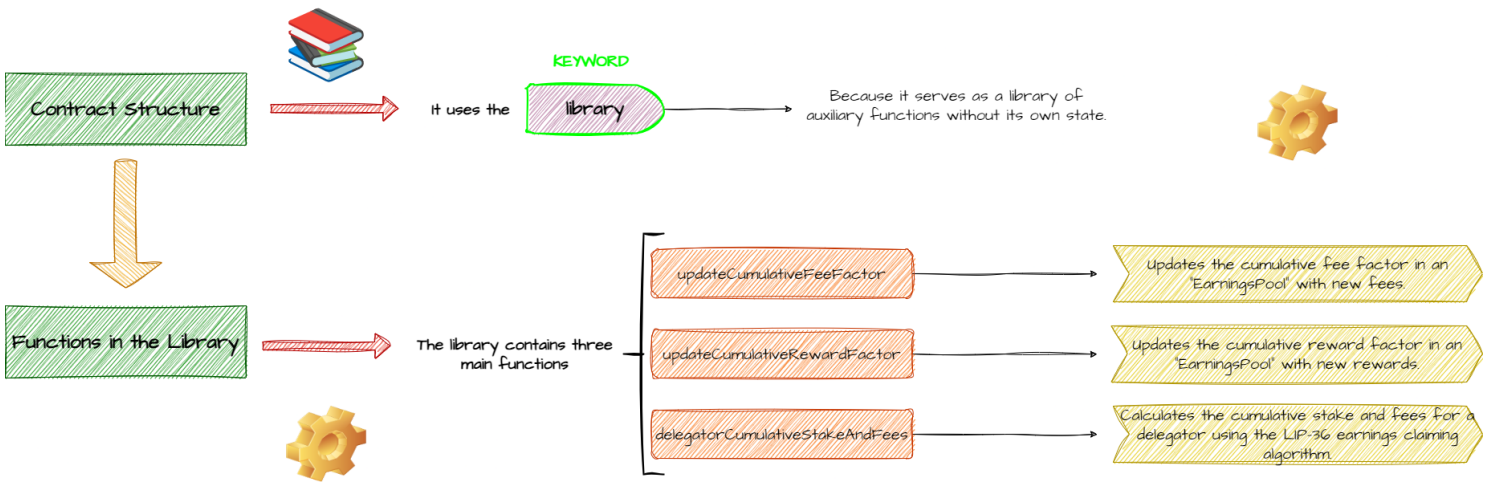
5. LivepeerGovernor:



In this `diagram`, we illustrate how the `LivepeerGovernor` contract acts as the `treasury governor` in `Livepeer`. It utilizes `extensions` and other `contracts` to implement `governance logic` and `vote management`.

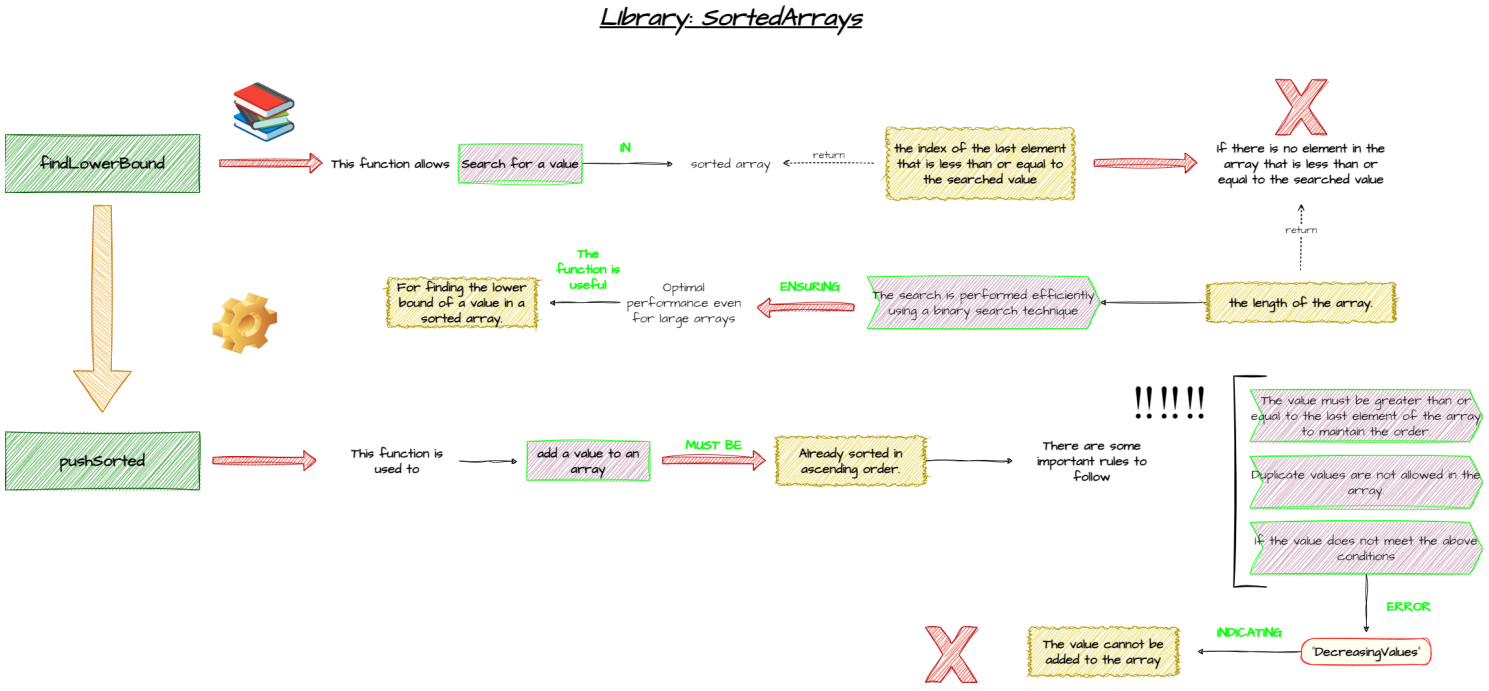
Libraries

1. EarningsPoolLIP36:



This `library` provides functions to update and calculate cumulative `fee` and `reward` factors in an `EarningsPool` and to `calculate` the cumulative `stake` and `fees` of a `delegator` based on these factors. The `diagram` includes all the `functions` in the library.

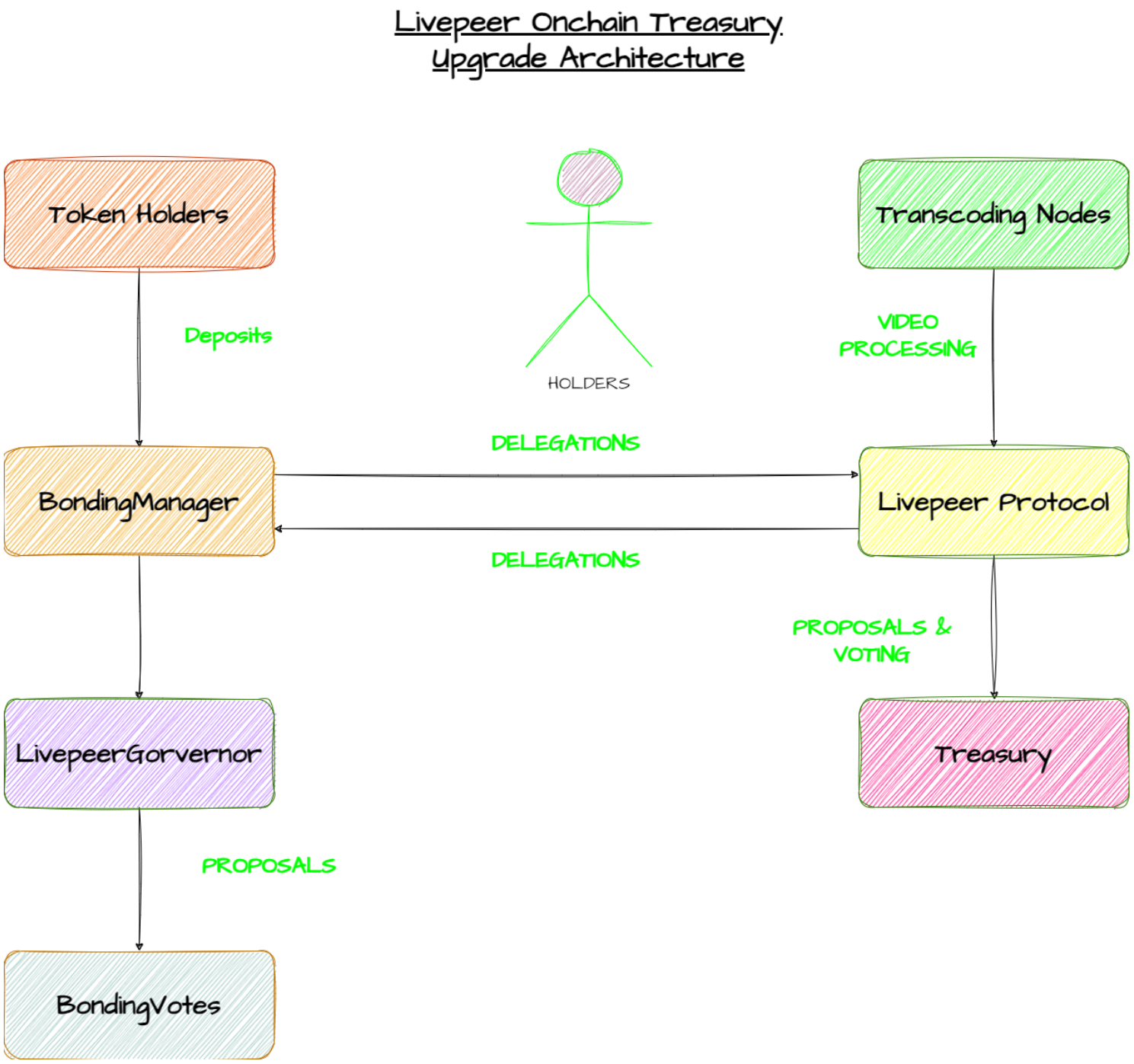
2. SortedArrays:



This `library` is useful for working with `arrays` of integers sorted in ascending order. In the `diagram`, we specify both the `findLowerBound` function, which is particularly useful for searching for a value in the array and finding the `index` of the last element that is less than or equal to the searched value, and the `pushSorted` function, which allows you to add values to the array while maintaining order and avoiding duplicates.

3- Livepeer Onchain Treasury Upgrade Architecture

This `diagram` illustrates the interaction among the key components of the `Livepeer protocol`. The `LivepeerGovernor` serves as the main governance system. The `BondingManager` is responsible for managing user participation, while `BondingVotes` is used to handle the voting logic. Lastly, the `Treasury` manages treasury funds and governance proposals related to the treasury. It also shows how a `user` engages with the system, all succinctly summarized in the diagram.



- **Some potential areas of improvement or consideration could include:**

    - **User Experience:** Enhancing the user experience for both transcoders and token holders can attract more participants. User-friendly interfaces, improved documentation, and educational resources can be beneficial.

- **Governance Flexibility**: Evaluating the governance mechanisms to ensure they are adaptable and can accommodate changes as the ecosystem evolves.
- **Interoperability**: Exploring interoperability with other blockchain networks or protocols can expand Livepeer's capabilities and reach.
- **Documentation and Education**: Comprehensive documentation and educational materials can help users understand and navigate the Livepeer ecosystem more effectively.

4- Documents

- The documentation of the `Livepeer Onchain Treasury Upgrade` project is quite comprehensive and detailed, providing a solid overview of how `Livepeer` is structured and how its various aspects function. However, we have noticed that there is room for additional details, such as `diagrams`, to gain a deeper understanding of how different contracts interact and the functions they implement. With considerable enthusiasm, we have dedicated several days to creating diagrams for each of the contracts. We are confident that these diagrams will bring significant value to the protocol as they can be seamlessly integrated into the existing `documentation`, enriching it and providing a more comprehensive and detailed understanding for `users`, `developers` and `auditors`.

5- Systemic & Centralization Risks

Here's an analysis of potential systemic and centralization risks in the provided contracts:

Systemic Risks:

- **Smart Contract Vulnerability Risk**: Smart contracts can contain vulnerabilities that can be exploited by attackers. If a smart contract has critical security flaws, such as `access errors` or `logic problems`, this could lead to `asset loss` or `system manipulation`. We strongly recommend that, once the protocol is audited, necessary actions be taken to `mitigate` any issues identified by `C4 Wardens`.
- **Third-Party Dependency Risk**: Contracts rely on external data sources, such as [@openzeppelin/contracts](), and there is a risk that if any `issues` are found with

these `dependencies` in your contracts, the `Livepeer` protocol could also be `affected`.

- We observed that `old versions` of `OpenZeppelin` are used in the project, and these should be updated to the `latest version`:

```
54: "@openzeppelin/contracts": "^4.9.2",
55: "@openzeppelin/contracts-upgradeable": "^4.9.2",
```

The latest `version` is `4.9.3` (as of July 28, 2023), while the project uses version `4.9.2`.

- **Update Risk**: If the smart contract needs to be updated, there is a risk that updates may be implemented incorrectly or that new versions introduce vulnerabilities. Additionally, updates can lead to divisions within the user community.

  - The protocol uses a `proxy`, and according to the `sponsor`, the type of `proxy` is:

```
   victorges | livepeer — 09/2/2023 at 11:22
   it's a custom proxy implemented/documented here
19 https:
20 it's more similar to a "transparent proxy" pattern than UUPS, with th
```

  If the logic controlling the `proxy` is not implemented correctly, there could be `vulnerabilities` that allow an attacker to modify the underlying contract or the `proxy` itself in an unintended way.

Centralization Risks:

- **Participation Centralization**: If a small group of `transcoders` or `delegators` controls a significant portion of `assets` or `participation` in the `system`, this could lead to significant `centralization`. This could undermine `decentralization` and `system security`.

- **Decision-Making Centralization:** If a small group of actors has disproportionate control over key decisions, such as `system rule changes` or `reward allocation`, this could lead to a `centralized system` where decisions are made for the benefit of a few rather than the entire `community`.
- **Transcoder Node Centralization:** If a small number of `transcoders` dominate the active set, this could lead to `centralization` in the provision of `transcoding services`, resulting in `higher fees` or the `exclusion` of smaller actors.

6- New insights and learning from this audit

Our latest insights from the `Livepeer Onchain Treasury Upgrade` protocol audit have been extensive. This audit has deepened our understanding of the `Livepeer` protocol `upgrade`, which introduces a community `treasury` funded by protocol `rewards`. The `upgrade` leverages `OpenZeppelin` governance primitives, underscoring the crucial role of auditing in upholding protocol integrity. It places particular emphasis on critical aspects like `reward` calculations and `highlights` security concerns, emphasizing the necessity of safeguarding against potential attacks to ensure ongoing system reliability. This audit has also showcased the protocol's adaptability, as it can seamlessly adjust its `reward` system and incorporate a `community treasury`.

Conclusion

Overall, the `Livepeer Onchain Treasury Upgrade` presents an interesting `architecture` aimed at establishing `decentralized` infrastructure for `video streaming` in `web3` social and media applications. The development team has done a good job with the code. However, as is common in `blockchain` projects, `Livepeer` faces risks related to the security of its `smart contracts` and the `centralization` of key actors such as `transcoders` and `delegators`. Ultimately, the continued success of `Livepeer` will depend on its ability to address these challenges, especially after potential security issues are disclosed by `C4 wardens`. This will involve maintaining `security` and `decentralization` and remaining an attractive choice for `web3 video streaming` applications.

Time Spent

A total of `3 days` were dedicated to completing this audit, distributed as follows:

1. 1st Day: Devoted to comprehending the protocol's functionalities and implementation.
2. 2nd Day: Initiated the analysis process, leveraging the documentation offered by the `Livepeer Onchain Treasury Upgrade`.
3. 3rd Day: Focused on conducting a thorough analysis, incorporating meticulously crafted diagrams derived from the contracts and information provided by the protocol.

Time spent:

15 hours

victorges (Livepeer) acknowledged and commented:

> The detailed designs of the system don't seem accurate, but the rest of the report and raised issues and suggestions are good.

Disclosures

C4 is an open organization governed by participants in the community.

C4 Audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top