PUBLIC

Code Assessment

of the yETH Governance Smart Contracts

November 3, 2023

Produced for

by

Contents

1 Executive Summary

Dear Yearn team,

Thank you for trusting us to help Yearn with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of yETH Governance according to Scope to support you in forming an opinion on their security risks.

Yearn implements an on-chain governance system for `yETH` and the new contracts. They allow `st-yETH` holders to vote for generic proposals and Pool parameter changes.

The most critical subjects covered in our audit are access control and functional correctness. All raised issues have been addressed accordingly. The most critical issue found in the assessment was related to incorrectly counted votes in `InclusionVote` (see Blank Votes Not Counted).

In summary, we find that the codebase now provides a good level of security. It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	1
• Code Corrected	1
Medium-Severity Findings	5
• Code Corrected	5
Low-Severity Findings	4
• Code Corrected	4

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the yETH Governance repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	25 September 2023	d1ed4007f71982629906b51d219ee2e2d9560e8c	Initial Version
2	31 October 2023	2a6bf23ce399beac5bef72b412dc76f2ded44c20	Version 2

For the vyper smart contracts, the compiler version `0.3.7` was chosen.

The following contracts are in the scope of the review:

```
contracts/governance/
    DelegateMeasure.vy
    Executor.vy
    GenericGovernor.vy
    InclusionIncentives.vy
    InclusionVote.vy
    LaunchMeasure.vy
    OwnershipProxy.vy
    PoolGovernor.vy
    SnapshotToken.vy
    WeightIncentives.vy
    WeightVote.vy
```

2.1.1 Excluded from scope

Any contracts not explicitly listed above are out of the scope of this review. The Vyper compiler and standard library are out of the scope of this review. Any other yETH contracts not listed, such as `Bootstrap` or `Staking`, are out of the scope of this review.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

The assessed contacts are part of Yearn's ecosystem and move the management role of yETH on-chain. Until now, the management power was distributed over different accounts and contracts that had different roles. This set of management roles that have powers within the protocol to, e.g., set rates, add assets, etc. will be replaced by the `OwnershipProxy`. The `OwnershipProxy` will become the new owner of all these management roles (excluding the proxy's own). The `OwnershipProxy` is not a `delegatecall`

proxy contract. It acts as a relay contract that relays arbitrary calls, and executes them with `OwnershipProxy` as `msg.sender`. The `OwnershipProxy` has one privileged role, (`management`) that is allowed to call `execute()` on the `OwnershipProxy`. In this way, calls can be made from the proxy's `management` to contracts for which the `OwnershipProxy` has admin power. It is assumed that the `management` role will be held by the `Executor` contract. The `Executor` maintains a set of governors. Accounts or contracts with the governor role can call the `Executor`'s `execute()` function. Additionally, the contract has a white- and blacklist allowing to define a very precise access control by setting the functions and the target that a specific governor account is allowed to call - or not to call. The `Executor` is self-governing, meaning the `OwnershipProxy` is supposed to fulfill its `management` role.

There are currently two contracts that are intended to act as governors. The `GenericGovernor` and the `PoolGovernor`. Both contracts allow to call `execute()` on the `Executor` via their own `execute` function if a proposal has passed successfully. The `GenericGovernor` is used for arbitrary calls and allows to add proposals and vote on them directly. The users' voting weight is determined by querying either the `DelegateMeasure` or the `LaunchMeasure` contract. The `DelegateMeasure` is similar to the `LaunchMeasure`, but allows the delegation of voting power to another address. In the first 3 weeks of an epoch, the `GenericGovernor` accepts proposals from anyone with sufficient voting weight. Proposals come in the form of a script, which will be executed on the `Executor` if passed. In the final week of the epoch, users vote in favor of - or against each proposal. At the end of the epoch, all proposals that pass a threshold of relative votes in favor will have passed. They become executable by anyone after a delay. The author of the proposal can retract their proposal, and `management` can cancel any before it is executed.

The `PoolGovernor` has two attached contracts that contain the voting mechanisms: `WeightVote` and `InclusionVote`. The `PoolGovernor` is more explicit (compared to the generic governor) in the actions it can execute. These are adding a new asset to a pool and/or changing the target weights of assets in the `Pool`. The voting process of the `PoolGovenor` is handled in separate contracts depending on the operation to be executed. The two contracts used are `WeightVote` and `InclusionVote`. They handle the voting process and the Governor will handle the execution for both if a vote is successful. To incentivize voting to increase a certain asset's weight, or to add a certain asset, users can deposit incentives in `WeightIncentives` and `InclusionIncentives`. In the first three weeks, anyone can apply (upon paying a fee) for a token to be whitelisted in `InclusionVote`. The `operator` role has the task of setting rate providers for each of the applying tokens. An application with a rate provider automatically becomes whitelisted, meaning it can be voted on. The token with the most votes will be added to the pool. One of the vote options is a `blank` vote. If that option has the most votes, no new asset will be added to the pool this epoch. In `WeightVote` users can vote to increase the weight an asset has in the `Pool`.

All voting takes place in the last week of a four-week-long epoch (each starting on a Thursday `00:00` UTC).

2.2.1 OwnershipProxy

The contract will eventually be assigned the `management` role of all relevant contracts. It is not a `delegatecall` proxy, but a relay contract that has an `execute()` function to execute arbitrary calls (including calling itself), to mainly manage other Yearn contracts. This function can only be called by a `management` role, which can be set via `set_management()`. The contract meant to hold this `management` role is the `Executor` contract.

2.2.2 Executor

The contract allows the execution of arbitrary calls via the `OwnershipProxy`'s `execute` function, by either calling `execute_single()` or `execute()` (multi-call option). When multiple calls are executed in one transaction, the data needs to be parsed correctly and is called in a script in this context. The view function `script` helps compose and format the script.

To execute a call, the calling address needs to be a governor. The governor role can be set in `set_governor`. E.g., the `GenericGovernor` and the `PoolGovernor` will have this role. Additionally, a

governor can be limited in the actions they can execute. Via `set_access`, a flag can be set to indicate what kind of access restrictions are set. This should be either a white- or blacklist, or none. To white- or blacklist, the management role can call `whitelist()` or `blacklist()`. The Executor's `management` role can be changed using `set_management` and `accept_management`. All the mentioned administrative functions can only be called by the management role, which is expected to be held by the `OwnershipProxy`.

2.2.3 LaunchMeasure

To execute a proposal the `GenericGovernor` and `PoolGovernor` are used. Both query the voting weight of a user from a measure contract. The `LaunchMeasure` has only two relevant functions. `vote_weight` and `total_vote_weight`. The total vote weight is the total supply of the staking contract, and a user's `vote_weight` is the user's `vote_weight` in the staking contract plus their bootstrap contribution. The additional `vote_weight` from the bootstrap is equal to the bootstrap contract's staking `vote_weight` multiplied by the user's share of the bootstrap contract.

2.2.4 DelegateMeasure

The delegate Measure also implements the two functions `vote_weight` and `total_vote_weight`, which can be queried by the `GenericGovernor` and `PoolGovernor`. Additionally, it adds the functionality to delegate voting power to a specific address. However, the delegation address cannot be set by users. It needs to be set by the contract's management role. The feature is intended to be used by protocols that aggregate the voting weight of multiple users. When using delegation, the voting weight is not a decay function based on time, as it is for individual users that have deposited to the staking contract. Instead, the `vote_weight` is equal to the staking balance of the address, which is the voting power that would usually be achieved after holding the tokens for an infinite amount of time. However, there is a multiplier that can be set by the contract's management, which multiplies the voting power of delegated funds by a fixed factor (e.g., `0.5`). The multiplier can be set via `set_multiplier`. The management role can be set via `set_management` and `accept_management`. The management is assumed to be the `OwnershipProxy`. Voting power in the bootstrap contract cannot be delegated.

2.2.5 DelegatedStaking

The `DelegatedStaking` contract is a wrapper and helper contract that allows users (mainly protocols) to delegate their voting power. To delegate the voting power, the functions `deposit` or `mint` can be called. When `DelegateMeasure``calculates the vote weight, it will call ``DelegatedStaking.vote_weight` and weigh it with a scaling factor. By calling `withdraw` and `redeem` users can exit the delegation contract. The contract implements the ERC4626 interface including standard ERC20 functionality.

2.2.6 GenericGovernor

The generic governor has the proposal and voting functionality for users. To vote, a user can either call `vote_yea()` or `vote_nay()` to vote with the full voting weight for a proposal or call `vote()` to allocate percentages of the voting weight to yes or no. A successful proposal can be executed by calling `enact`. Note that there was no quorum needed in version 1, a proposal could pass with 1 yes vote. This was changed. Voting is carried out in the last week of a four-week-long epoch. In the first three weeks, proposals can be created by calling `propose()`. To create a proposal, the proposer needs to have at least `propose_min_weight` voting weight. Proposals contain a script, which can be executed by the Executor. At the end of the epoch, all proposals that pass a threshold of relative votes in favor will have passed. After a delay, their scripts become executable by anyone by calling `enact`. The author of the proposal can retract their proposal by calling `retract` as long as it is proposed, and management can call `cancel` to cancel any proposal before it is executed. In case management is the `OwnershipProxy`, it would be impossible to call cancel as another vote would need to succeed to call it. But this function intends to have the current/first holder of management roles as a governor and remove it after a few epochs via governance vote. During this period the `cancel` functionality can be used in emergencies. It could potentially be used in the future by introducing additional governors, although Yearn states, that no such plans exist currently. The state of a proposal can be updated via `update_proposal_state()`. Internally, the contract uses the packed values `packed_quorum` and `packed_delay`. They have the following bit-layout (from least significant to most significant bit): `current (120) | previous (120) | epoch (16)`.

The contract's `management` (`OwnershipProxy`) can set the following configuration parameters:

- `set_measure` - to set the contract responsible for defining the voting weights a user has
- `set_executor` - to set the address of the executor contract that is called when a proposal is executed
- `set_delay` - sets the delay after a proposal passes before it can be executed
- `set_majority` - Sets the threshold (e.g. 50 percent would be 5000) for a proposal to pass
- `set_propose_min_weight` - Defines the minimal voting weight a user needs to submit proposals
- `set_management` - Sets a pending management address, that will be able to call the administrative functions
- `accept_management` - Called by the pending management address to become the management role

2.2.7 PoolGovernor

This contract is intended to manage Yearn's `yETH` pool. It will execute successfully proposed weight changes and add new tokens to the `Pool`. The proposal and voting process is handled in the two separate contracts `WeightVote` and `IncentiveVote`. The `PoolGovernor` is the shared execution handler for the two contracts. The `PoolGovernor` will evaluate the votes from both contracts and execute a successful proposal by calling `execute_single` on the `Executor` contract. There is an `operator` role, which is tasked with calling the `execute()` function and choosing parameters that minimize the arbitrage opportunities created from adding a new asset to the `Pool`.

The contract has the following configuration parameters that can be set by the management contract (`OwnershipProxy`):

- `set_target_amplification` - Sets the target amplification that can be applied to the pool.
- `set_executor` - Sets the address of the executor contract that is called when a proposal is executed.
- `set_operator` - Sets the operator.
- `set_initial_weight` - Sets the initial weight for newly added assets.
- `set_ramp_weight` - Set the ramp target weight for newly added assets.

- `set_redistribute_weight` - Set the weight that can be redistributed each epoch.
- `set_ramp_duration` - Set the ramp duration.
- `set_inclusion_vote` - Set the inclusion vote contract that determines which assets should be added to the pool.
- `set_weight_vote` - Set the weight vote contract that determines the redistribution of weights over the assets.
- `set_management` - Sets a pending management address, that will be able to call the administrative functions
- `accept_management` - Called by the pending management address to become the management role

2.2.8 WeightVote

This contract is used to indicate to the `PoolGovernor` how to update the target weight of assets in the pool. It will accept votes in the last week of every epoch. Users can vote with their voting weight, which is determined by a `Measure` contract. They can vote for any number of assets in the pool, as well as the `blank` option, which keeps the current target weights. The maximum amount of weight change per epoch is defined by `set_redistribute_weight``in ``PoolGovenor`.

2.2.9 InclusionVote

This contract is used to indicate to the pool governor contract whether or not an asset should be added to the pool. It will accept votes in the last week of every epoch. Outside of the voting period, users can apply to add any ERC20 token to the pool, paying an application fee. If the token has already been applied for in a previous epoch, a different fee is applied than if it is an initial application. The contract's `operator` is expected to add a suitable rate provider contract for the token, which returns the `ETH` value per token. Users can vote with their voting weight, which is determined by a `Measure` contract. They can vote for any number of assets, as well as the `blank` option, which does not add an asset.

2.2.10 WeightIncentives

This contract allows users to deposit incentives in any ERC20 token to reward voters who vote for a certain asset in `WeightVote`. The reward per asset is always paid to voters, proportional to the vote weight they used to vote for the incentivized asset. For example, if 2 voters A and B voted for the asset, and A used twice as much voting power as B, then A will receive `2/3` of the reward, while B will receive `1/3`. After the voting is over, the voters can claim their share of the rewards. Rewards left unclaimed for a configurable amount of epochs can be swept by the treasury.

2.2.11 InclusionIncentives

This contract allows users to deposit incentives in any ERC20 token to incentivize voters who vote in `InclusionVote` to include a given candidate token or vote for the blank option. If the incentivized target wins, the rewards are allocated to all voters in proportion to their voting weight, regardless of how they voted. After the voting is over, either the depositor can get a refund on their deposit (if their target did not win), or the voters can claim their share of the rewards. Rewards left unclaimed or unrefunded for a configurable amount of epochs can be swept by the treasury.

2.2.12 Roles and trust assumptions

The system has one main role. The `OwnershipProxy`. It has the management role in these contracts:

- `DelegateMeasure`
- `Executor`

- `GenericGovernor`
- `InclusionIncentives`
- `InclusionVote`
- `PoolGovernor`
- `WeightIncentives`
- `WeightVote`

It is assumed that the management contract of the `OwnershipProxy` is the Executor. It is the only account that can call the `OwnershipProxy`. As long as this is the case, there are no additional trust assumptions needed besides the trust assumptions for the `Executor`. The `Executor`'s `execute` function can only be triggered by governors. Consequently, the trust assumptions propagate to this role. We assume the governors to be fully trusted. Usually, these should be contracts that are controlled through a voting process. These votes need to be proposed, assessed and voted on by the token holders. This means that the decisions made by the majority of token holders are assumed to be fully trusted.

We assume the deploying account to be fully trusted, up until the moment when it gives all its privileged roles to other addresses.

The `PoolGovernor` is operated by an `operator`, which is a role tasked with adding assets to the pool while minimizing the arb opportunities. The role is trusted to choose only appropriate and well-tested parameters for pool changes. Especially, the `operator` needs to take care to plan the ramping so that it will not harm the pool and be in sync with the new epoch. The power is limited as the role cannot change weights at will or add arbitrary assets.

The `operator` role in the `InclusionVote` contract is trusted to set an appropriate, non-malicious, and well-tested rate provider. The rate providers are trusted and assumed to work correctly at all times.

We assume pool tokens that are added to be non-malicious and not have problematic properties, e.g., rebasing, or tokens with double entry points.

We assume the staking contract does not return different `vote_weights` during an ongoing epoch. We also assume deposits to the bootstrap contract are no longer possible.

2.2.13 Changes in Version 2

A quorum has been added to the GenericGovernor, meaning that a minimum number of yes + no votes is now required for a proposal to pass.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

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

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation
- Trust: Violations to the least privilege principle

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	1

- Blank Votes Not Counted Code Corrected

Medium-Severity Findings	5

- Balance Used Instead of Voting Weight in DelegateMeasure Code Corrected
- Griefing by Flooding Malicious Proposals Code Corrected
- InclusionVote Operator Trust Code Corrected
- Proposals Can Be Enacted After More Than One Epoch Code Corrected
- Voters Trust Proposal Author Not to Retract Code Corrected

Low-Severity Findings	4

- Access Control Can Have Invalid Value Code Corrected
- Delegation Could Allow Double Voting Code Corrected
- Number of Assets Could Change During Vote Code Corrected
- Race Condition in GenericGovernor Code Corrected

Informational Findings	4

- Majority Parameter Can Be Less Than Fifty Percent Code Corrected
- Sanity Checks Code Corrected
- Missing Events Specification Changed
- Should Governance Be Able to Evict the Treasury Specification Changed

6.1 Blank Votes Not Counted

Correctness High Version 1 Code Corrected

CS-YEGOV-013

In InclusionVote, the winner is determined as follows:

```
if votes > winner_votes:
        candidate: address = self.candidates[epoch][i]
        if self.rate_providers[candidate] in [empty(address), APPLICATION_DISABLED]:
            # operator could have unset rate provider after
            continue
        winner = candidate
        winner_votes = votes
```

The zero (blank) candidate will not have a rate provider set. The condition for `continue` will be fulfilled and the winner_votes will not be set to the blank votes.

As a result, the blank votes are ignored and a candidate with fewer votes than the blank votes can become the `winner`.

Code corrected:

A special case has been added for *candidate == 0x0*. Now, the votes of the zero address are counted. Additionally, the zero address's rate_provider is not set to *APPLICATION_DISABLED* when the zero address is the winner.

6.2 Balance Used Instead of Voting Weight in `DelegateMeasure`

Correctness Medium Version 1 Code Corrected

CS-YEGOV-018

When computing the voting weight of an account, if the account has been delegated to, the following formula is used to compute the additional weight.

```
weight += Staking(staking).balanceOf(delegated) * self.delegate_multiplier / DELEGATE_SCALE
```

Since the balance can be altered without delay simply by acquiring the staking token on the spot, the call to `balanceOf` is prone to manipulation.

This issue was found during the review. It was also reported independently by Yearn while the review was still ongoing.

Code corrected:

This was fixed in Version 2 by storing delegated stake in a separate vault, which only updates `vote_weight` at the end of the week.

6.3 Griefing by Flooding Malicious Proposals

Design Medium Version 1 Code Corrected

CS-YEGOV-015

In `GenericGovernor`, as long as an attacker holds at least `propose_min_weight` tokens, they can submit as many proposals as they want, paying only gas.

If these proposals would hurt the protocol, other users are forced to vote `nay` each time, to ensure the proposal does not pass. There is no quorum needed to pass a proposal.

It may also be problematic if the same proposal is submitted multiple times. Voters will need to coordinate and choose which of these they want to pass, while rejecting the others. See also: Voters trust proposal author not to retract.

Code corrected:

A quorum has been added to the GenericGovernor, meaning that a minimum number of yes + no votes is now required for a proposal to pass. Governance functions for setting the quorum, as well as view functions to read it have also been added.

6.4 InclusionVote Operator Trust

Trust Medium Version 1 Code Corrected

CS-YEGOV-016

The InclusionVote contract has an `operator` role, which is tasked with setting rate providers for proposed tokens.

In the current implementation, the `operator` can change the rate provider at any time. In particular, it can change the rate provider even after voters have already voted.

This design results in the `operator` role needing to be fully trusted to set a correct rate provider.

If an alternative design was chosen where the rate provider can no longer change between the beginning of the voting period and `finalize_epoch()`, the voters could ensure that they are voting for a correct rate provider and that it cannot change after their vote. This would reduce the trust assumption on the `operator` role.

Code corrected:

Now, the `operator` can only change a rate provider that was previously set if:

1. The voting period of the current epoch has not started
2. The previous epoch has already been finalized

If the rate provider has never been set (still 0x0), it can still be added at any time.

This means that voters can now independently check that the `operator` has set a correct rate provider, and can be sure that it will not change after they vote. This reduces the trust required in the `operator`.

6.5 Proposals Can Be Enacted After More Than One Epoch

Correctness Medium Version 1 Code Corrected

CS-YEGOV-014

To enact proposals in the `GenericGovernor` via `enact`, the proposal state is checked and asserted to be `PASSED` by calling `_proposal_state()`. The function `_proposal_state` explicitly returns `PASSED` only if `current_epoch == vote_epoch + 1`. Consequently, a proposal must be enacted one epoch after `vote_epoch`.

However, by calling `update_proposal_state()` on a proposal that just passed, it is possible to set the state of this proposal to `PASSED` in storage. In this case, it is possible to circumvent the condition that a proposal needs to be enacted one epoch after `vote_epoch`, because `_proposal_state()` returns `PASSED` from now on due to: `if state != STATE_PROPOSED return state`.

This will allow the execution of the proposal forever, even though it should revert if it is not executed in the epoch after passing.

Code corrected:

The state is now reevaluated when calling `_proposal_state()`, even if the storage was set to PASSED.

6.6 Voters Trust Proposal Author Not to Retract

Trust Medium Version 1 Code Corrected

CS-YEGOV-012

In `GenericGovernor`, a proposal can be retracted by its author at any point up until the end of the voting period. This means that the author can grief voters by retracting maliciously.

Consider the following situation:

1. The community decides off-chain that a certain proposal is something they want to vote on.
2. Alice has `propose_min_weight` votes and anonymously submits the proposal.
3. The proposal receives 99% `yea` votes.
4. One hour before the vote period ends, Alice retracts the proposal.
5. Now the proposal will not be executable and it will take at least another epoch until it can be voted on again and pass.

To avoid this, the proposal author needs to be trusted by the voters.

As a possible countermeasure, the same proposal could be submitted multiple times by different authors. However, this could be problematic if the proposal does something which should not happen multiple times, (e.g., send some tokens) and more than one of the proposals pass.

Code corrected:

Proposals can now no longer be retracted once the voting period has begun.

6.7 Access Control Can Have Invalid Value

Design Low Version 1 Code Corrected

CS-YEGOV-009

The access control in Executor is set using the Access enum. When something should have a whitelist, the enum is set to a value of `1`, when it should have a blacklist, it is set to a value of `2`. If neither is true, it should be set to the default value of `0`.

However, in Vyper it is also possible to set enum in such a way that multiple "flags" are set at once, not just one. `set_access()` has no sanity check for the access argument. As a result, `set_access()` could be called by the management with a value of 3, which is a valid value in Vyper and represents the states whitelist and blacklist being true at the same time.

However, the contract is not designed to handle this value and will treat it the same as `0`.

Code corrected:

A check has been added that disallows values that are greater than 2. Now the only possible enum values are default, whitelist and blacklist.

6.8 Delegation Could Allow Double Voting

Design | Low | Version 1 | Code Corrected

CS-YEGOV-011

In DelegateMeasure, an address that has given a delegation to another address, has a `vote_weight` of `0`, which means it can no longer vote directly.

However, the `delegate()` function does not check if the address that is giving delegation has previously voted during the current epoch. As a result, it is possible that an address first votes with its own `vote_weight`, then `delegate()` is called. This would allow the voting power to be used a second time by the address receiving the delegation.

Note that `delegate()` can only be called by the `management` role, which is expected to be used through the `GenericGovernor`. In this case, the issue can be avoided by calling `enact()` before the `VOTE_PERIOD` starts, given that the `delay` is smaller than `VOTE_START`.

Code corrected:

This was fixed in Version 2 by storing delegated stake in a separate vault, which only updates `vote_weight` at the end of the week.

6.9 Number of Assets Could Change During Vote

Design | Low | Version 1 | Code Corrected

CS-YEGOV-010

In WeightVote, the number of assets in the Pool is queried once when the first vote in a voting period happens. The value is cached and not updated for the rest of the epoch.

If the number of assets changes within the voting period, it will be impossible to vote for the newly voted asset. This would only happen if the `execute` function of PoolGovernor is called late (in the last week of the epoch) by the `operator`.

Code corrected:

Yearn removed the caching of the number of tokens and now queries them directly from the pool.

6.10 Race Condition in `GenericGovernor`

Design | Low | Version 1 | Code Corrected

CS-YEGOV-005

If a proposal is passed that stops another proposal in the same epoch from being enacted, whether by explicitly canceling it or by modifying common parameters such as `majority`, then a race condition occurs whereby depending on the order in which the proposals are enacted, the end result is different.

Note that `enact()` can be called by anyone, thus this ordering is also subject to MEV.

Yearn found and reported this issue while the review was ongoing.

Code corrected:

In Version 2, `enact()` uses the values of `majority` and `delay` snapshotted at the end of the previous epoch.

6.11 Majority Parameter Can Be Less Than Fifty Percent

Informational | Version 1 | Code Corrected

CS-YEGOV-006

In `GenericGovernor`, the `majority` parameter can counterintuitively be set to less than `50%`.

This would mean that a proposal with more no votes than yes votes can pass.

Code corrected:

Yearn now enforces a range betweeen `VOTE_SCALE / 2` and `VOTE_SCALE` for `majority`.

6.12 Missing Events

Informational | Version 1 | Specification Changed

CS-YEGOV-008

The constructors of `DelegateMeasure`, `Executor`, `GenericGovernor`, `InclusionIncentives`, `InclusionVote`, `OwnershipProxy`, `PoolGovernor`, `WeightIncentives` and `WeightVote` do not emit the `SetManagement()` event.

Specfication changed

Yearn answered:

```
This is intentional, as it would require to also emit events for a lot of other parameters
during the constructor to be fully consistent. For example, in the
generic governor constructor we set a value for measure, delay, quorum, majority and delay.
```

6.13 Sanity Checks

Informational | Version 1 | Code Corrected

CS-YEGOV-007

Multiple functions do not sanitize their input. E.g., the `Executor` contract in `set_access()`, `set_governor()`, `whitelist()` and `blacklist()` do not check for address zero. We advise reviewing which functions would benefit from a sanity check, even if they are permissioned.

Code corrected:

Yearn changed the listed functions and implemented sanity checks. We additionally assume Yearn checked all other potential functions and added checks.

6.14 Should Governance Be Able to Evict the Treasury

Informational | Version 1 | Specification Changed

CS-YEGOV-017

In the `setTreasury()` function of `InclusionVote`, `InclusionIncentives` and `WeightIncentives`, the management role has the power to change the treasury address to an arbitrary value. The yETH protocol is designed to be governed by st-yETH holders. At the same time, YIP-72 says that the treasury should be the "Yearn Treasury or an autonomous splitter contract directed by yBudget." Is it intended that holders are able to direct the treasury revenue away from Yearn?

Specification changed

This described behavior was originally intended. But after being raised and careful consideration, Yearn decided that only the treasury shall be allowed to call `setTrasury` and changed the code accordingly.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Gas Optimisations

Informational | Version 1 | Code Partially Corrected

CS-YEGOV-003

We discovered the following potential gas optimizations:

1. The Proxy interface in Executor uses `Bytes[65536]` as data argument, but the OwnershipProxy only supports `Bytes[2048]`. The `calldata` variable in `execute()` also uses this large Array size. In Vyper, arrays reserve memory slots for their maximum size, even when many of the elements are zero. As a result, the memory will be extended by `65536` Bytes as soon as another variable is placed in memory after the array. This is very expensive.
2. `uint` could be used instead of boolean values. E.g., as governor flag in `Executor`.

Code partially corrected

Yearn decided to decrease the overall max script size to `Bytes[2048]`. In the rare case that a proposal requires a script larger than this, they can work around it by deploying a one-time use contract that is granted a temporary governor role during execution.

7.2 PoolGovernor Can Skip Epochs

Informational | Version 1

CS-YEGOV-001

The PoolGovernor's `execute` function always executes the vote results for `epoch - 1`. This means that if `execute()` is not called during an epoch, the preceding epoch's vote results will never be executed.

The `winner` in InclusionVote has its `rate_provider` set to `APPLICATION_DISABLED`, so if an asset wins but then the execution of the winning epoch is skipped, that asset cannot be proposed again unless the `operator` of InclusionVote sets the rate_provider again.

The `execute` function can only be called by the `operator` of PoolGovernor. If the `operator` is unavailable or malicious, it may not be called.

7.3 Unused Code

Informational | Version 1 | Code Partially Corrected

CS-YEGOV-004

The following code is not used:

- `WeightVote`: the interface definition of `Measure.total_vote_weight`
- `InclusionVote`: the interface `Measure.total_vote_weight`

- `InclusionIncentives`: the interface `voting.candidates_map` and the constants `VOTE_START` and `VOTE_LENGTH`.
- `WeightIncentives`: the constants `VOTE_LENGTH` and `VOTE_START`
- `GenericGovernor`: the interface definition `Measure.total_vote_weight`

Code partially corrected

The unused interfaces were removed. The unused constants still exist in `WeightIncentives`.

7.4 `DelegatedStaking` Does Not Strictly Conform to ERC-4626

Informational **Version 1**

CS-YEGOV-002

`maxDeposit()` and `maxMint()` return $2^{256} - 1$. Per ERC-4626, "MUST NOT be higher than the actual maximum that would be accepted". The balance is eventually stored packed in only 240-bits. Therefore, the theoretical maximum is $2^{240} - 1$. However, this is not enforced in the code, rather the supply of ETH is assumed to upper-bound the system.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Epoch Boundary Agreement

Note Version 1

To prevent double voting, `VOTE_LENGTH` should always be at most one week, `EPOCH_LENGTH` should always be a multiple of one week, and `genesis` should be set to a multiple of one week. This is to be consistent with the current `Staking` contract which provides voting weights.

8.2 Governance Proposal Passes In The Event Of A Tie

Note Version 1

In `GenericGovernor`, the condition for a proposal to be treated as passed is as follows:

```
if votes > 0 and yea * VOTE_SCALE >= votes * self.majority:
    return STATE_PASSED
```

Assuming majority is 50% and a proposal has one yea and one nay vote, it will pass.

8.3 Limted Number of Pool Tokens

Note Version 1

Pools have 32 slots. This sets a cap to the maximum number of tokens to add. Once included, a token can never be removed from the protocol. Removing tokens from a pool would need a redeploy.

In PoolGovernor, the `execute` function will get the `winner` of the InclusionVote and try to add it to the Pool.

If there are already 32 assets in the Pool and InclusionVote has a `winner`, `execute()` will revert. This will also make it impossible to change the weights during that epoch.

The `management` of InclusionVote should call `disable()` once there are 32 assets to avoid this.

8.4 Power of the PoolGovernor Operator

Note Version 1

The specifications currently say that the operator of PoolGovernor has limited power. This is true but the operator role is still extremely powerful as it must be trusted to set the pool values like amplification and ramping in a non-exploitable way. The parameters the operator role can set are critical in a yETH pool and related to other parameters. Hence, as mentioned in the system assumptions, ths role needs to be fully trusted.

8.5 Ramp_Duration Should Be Chosen Carefully

Note Version 1

The `ramp_duration` variable in `PoolGovernor` should be chosen carefully. If it is too short, it may be possible to make profitable sandwich attacks.

It should also not be too long. In particular, it must be shorter than the length of an epoch, as assets cannot be added to the `Pool` while there is an ongoing ramp. The `operator` of `PoolGovernor` should call `execute()` at least `ramp_duration` before the end of the epoch, so that the ramp ends by the time `execute()` is callable again.

8.6 Rebasing and Fee-On-Transfer Tokens Cannot Be Used as Incentives

Note Version 1

Both `InclusionIncentives` and `WeightIncentives` keep internal balances for tokens used as incentives. This is done in such a way that, if the contract ends up with more tokens than expected, then the leftover amount will be lost. If the contract ends up with fewer tokens than expected, then `transfer()` will fail and the last user to claim will not be able to receive the incentives they are owed.