yAudit

August 2026

Prepared for
Yearn

Audited by
Panda
Watermelon

stYFI Governance

Smart Contract Security Assessment

Contents

1 Review Summary **2**
- 1.1 Protocol Overview . . . 2
- 1.2 Audit Scope . . . 2
- 1.3 Risk Assessment Framework . . . 2
  - 1.3.1 Severity Classification . . . 3
- 1.4 Key Findings . . . 3
- 1.5 Overall Assessment . . . 4

2 Audit Overview **4**
- 2.1 Project Information . . . 4
- 2.2 Audit Timeline . . . 4
- 2.3 Audit Resources . . . 4
- 2.4 Critical Findings . . . 4
- 2.5 High Findings . . . 4
- 2.6 Medium Findings . . . 4
  - 2.6.1 Late YBC votes bypass decay when directing the pooled governance weight 4
  - 2.6.2 Vote-boost rewards can be captured with stake held only at the epoch boundary . . . 6
  - 2.6.3 Vetoed proposals continue accepting votes and grant participation rewards 8
- 2.7 Low Findings . . . 9
  - 2.7.1 A stale vote-boost cursor can temporarily block all delegated-staking operations . . . 9
  - 2.7.2 A single account can saturate proposal slots . . . 11
- 2.8 Gas Savings Findings . . . 12
- 2.9 Informational Findings . . . 12
  - 2.9.1 Proposals can pass without any minimum quorum . . . 12
  - 2.9.2 Early-exit reports may fail to reclaim all outstanding rewards . . . 13
  - 2.9.3 Inconsistent voting behavior documentation between `Voter` and `Voting` . 15

1 Review Summary

1.1 Protocol Overview

On-chain governance infrastructure is introduced within the stYFI system, granting voting power holders to submit and vote on governance proposals. To incentivize vote participation, the `VoteBoostRewardDistributor.vy` contract is used to keep track of a voter's participation frequency. The `Executor.vy` contract is the system's core fan-out contract, which will be executing the set of external calls attached to successfully passed proposals.

1.2 Audit Scope

This audit covers 7 smart contracts totaling approximately 1117 lines of code across 5 days of review.

```
contracts/
├── DelegatedStakingMiddleware.vy
├── DelegatedStakingRewardClaimer.vy
└── governance
    ├── Executor.vy
    ├── VoteBoostRewardDistributor.vy
    ├── Voter.vy
    ├── Voting.vy
    └── WeightMeasure.vy
```

1.3 Risk Assessment Framework

1.3.1 Severity Classification

Severity	Description	Potential Impact
Critical	Immediate threat to user funds or protocol integrity	Direct loss of funds, protocol compromise
High	Significant security risk requiring urgent attention	Potential fund loss, major functionality disruption
Medium	Important issue that should be addressed	Limited fund risk, functionality concerns
Low	Minor issue with minimal impact	Best practice violations, minor inefficiencies
Undetermined	Findings whose impact could not be fully assessed within the time constraints of the engagement. These issues may range from low to critical severity, and although their exact consequences remain uncertain, they present a sufficient potential risk to warrant attention and remediation.	Varies based on actual severity
Gas	Findings that can improve the gas efficiency of the contracts.	Increased transaction costs
Informational	Code quality and best practice recommendations	Reduced maintainability and readability

Table 1: severity classification

1.4 Key Findings

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	3
Low	2
Informational	3

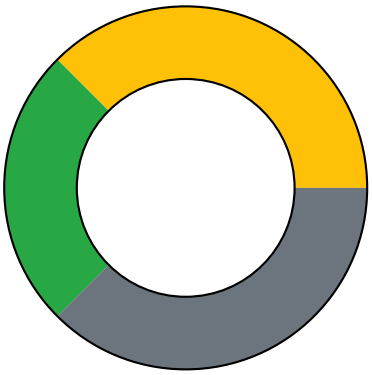

Figure 1: Distribution of security findings by impact level

1.5 Overall Assessment

The reviewed contracts implement full on-chain governance functionality within the stYFI ecosystem, granting voting power to veYFI locker holders, stYFI liquid wrappers and stYFI itself. The protocol allows for permissionless proposal submission, while retaining an authorized role which can veto selected proposals. The review uncovered a small set of medium, low and informational severity findings, which can pose issues under strict and rare circumstances.

2 Audit Overview

2.1 Project Information

Protocol Name: Yearn
Repository: https://github.com/yearn/stYFI
Commit Hash: a22c0ac31a55d2336c01cbbff63c46398cf31902
Final Commit Hash: 9395d5e6fffdfe21fda32af94d32fca1a4f7840b
Commit URL:
https://github.com/yearn/stYFI/tree/a22c0ac31a55d2336c01cbbff63c46398cf31902

2.2 Audit Timeline

The audit was conducted from July 29 to August 4, 2026.

2.3 Audit Resources

- Code repositories and documentation

2.4 Critical Findings

None.

2.5 High Findings

None.

2.6 Medium Findings

2.6.1 Late YBC votes bypass decay when directing the pooled governance weight

Technical Details

The configured late-vote decay is applied only to a YBC member's personal vote. The same member's influence over the pooled YBC and delegated-staking votes remains fully weighted, allowing a member to wait until near the end of an epoch and then move these potentially much larger voting positions without the intended penalty.

`Voter._vote()` calculates the current decay factor and passes it to `Voting.vote()` for the caller's personal vote:

```
scale: uint256 = PRECISION
decay_length: uint256 = self.decay_length
epoch_progress: uint256 = (block.timestamp - genesis) % EPOCH_LENGTH
if epoch_progress > EPOCH_LENGTH - decay_length:
    scale = PRECISION * (EPOCH_LENGTH - epoch_progress) // decay_length

user_weight: uint256 = extcall IVoting(_voting).vote(msg.sender, _idx, scale, _yea)
```

However, if the caller is a YBC member, their unscaled YBC weight is added to the blended vote:

```
weight: uint256 = staticcall self.ybc_weight_aggregator.weight(msg.sender)

votes: Votes = self.ybc_votes[_voting][_idx]
votes.weight += weight
if _yea > 0:
    votes.yea += weight
```

The resulting direction is then submitted for both pooled accounts at full scale:

```
ybc_yea: uint256 = PRECISION * votes.yea // votes.weight
extcall IVoting(_voting).vote(self.delegated_staking, _idx, PRECISION, ybc_yea)
extcall IVoting(_voting).vote(self.ybc.address, _idx, PRECISION, ybc_yea)
```

Consequently, `scale` has no effect on how much a late YBC member changes `ybc_yea`. For example, assume an earlier Yea voter has 3 units of YBC weight and a late Nay voter has 1 unit. Even if the late voter's decay factor is only 1%, the implementation changes the pooled position from 100% Yea to 75% Yea:

```
3 / (3 + 1) = 75%
```

If the same decay were applied to the late member's YBC weight, the pooled position would remain approximately 99.67% Yea:

```
3 / (3 + 1 * 1%) ~= 99.67%
```

For a combined YBC and delegated-staking voting position of 6 units, the late vote redirects 1.5 units to Nay instead of approximately 0.02 units. Meanwhile, only the member's personal vote is reduced to 1% of its normal weight.
This also differs from the protocol's `YBCElection` implementation, which applies its late-vote decay directly to the YBC weight before adding that weight to a proposal:

```
weight: uint256 = staticcall self.weight_aggregator.weight(msg.sender)
if epoch_progress > EPOCH_LENGTH - DECAY_LENGTH:
    weight = weight * (EPOCH_LENGTH - epoch_progress) // DECAY_LENGTH
```

Impact

Medium. A YBC member can retain full influence over the pooled YBC and delegated-staking positions when voting at the end of an epoch, bypassing the intended late-vote penalty and potentially changing whether a proposal passes.
The attacker must be a YBC member and can direct only their proportional share of the blended YBC decision. Nevertheless, the affected pooled positions may be substantially larger than the attacker's personal position.

Recommendation

Apply the same decay factor to the YBC weight before accumulating the member's contribution:

```
weight: uint256 = staticcall self.ybc_weight_aggregator.weight(msg.sender)
weight = weight * scale // PRECISION
if weight == 0:
    return user_weight
```

Use this effective weight for both `votes.weight` and `votes.yea`. Add a test combining YBC aggregation with a vote inside the decay window, including opposing early and late voters, to ensure the late member's influence over `ybc_yea` decays consistently with their personal vote.

Developer Response

This is intentional. YBC members contributors to Yearn and therefore are more trusted than regular participants, they are expected to not manipulate governance votes, that is why voting power is delegated to them by stYFIx. If they misbehave the other YBC members should vote for expulsion of this member.

2.6.2 Vote-boost rewards can be captured with stake held only at the epoch boundary

Technical Details

`VoteBoostRewardDistributor` treats the last recorded balance in an epoch as if that balance had participated for the entire epoch. A voter can therefore add a large transferable stYFI balance after voting, hold it only across the epoch boundary, and return or sell it immediately afterward while retaining the corresponding full-epoch vote-boost reward weight.
Each staking operation calls `_update_weights()`, which starts with the current epoch and overwrites the account's stored weight for that epoch:

```
epoch: uint256 = self._epoch()
...
for i: uint256 in range(6):
    ...
    weight: uint256 = staked
    ...
    weight = weight * num_votes // WEIGHT_PACKING_SCALE
    ...
    total = total - prev + weight
    ...
    packed_w = (packed_w & ~(WEIGHT_MASK << sh)) | (weight << sh)
```

On an stYFI transfer, the sender and recipient are recalculated using their post-transfer balances:

```
self._update_weights(_from, _prev_staked_from - _amount, False)
self._update_weights(_to, _prev_staked_to + _amount, False)
```

No time weighting or minimum holding period is applied. Once the epoch changes, a transfer back starts updating from the new epoch and cannot correct the previous epoch's weight. An attacker can therefore:

1. Cast a vote using any positive, already-mature voting position.

2. Acquire a large amount of transferable stYFI near the end of the voting epoch.
3. Allow the transfer hook to overwrite their vote-boost weight for the entire current epoch using the enlarged balance.
4. Return or sell the stYFI immediately after the boundary, leaving the previous epoch finalized with the enlarged weight.

An end-to-end proof of concept exercised the complete stYFI transfer-hook and reward-claim paths. A voter with only a dust voting position received 100 stYFI from a non-voting account one second before the boundary and returned it one second afterward. Against an honest voter who continuously held 10 stYFI, the temporary holder received more than 80% of the funded epoch reward despite retaining only the dust position at claim time. The source account earned no vote-boost reward because it had not voted.
The added stake does not need to have contributed to the attacker's governance vote. Governance weight for newly received stake ramps over four epochs in `WeightAggregator`, whereas `VoteBoostRewardDistributor` reads the raw staked balance. Consequently, a minimally weighted vote can qualify a much larger, momentarily held balance for rewards.

Impact

Medium. A voter can repeatedly capture a disproportionate share of an entire epoch's vote-boost rewards using stYFI held only for the brief interval around the epoch boundary. This directly dilutes voters who kept capital staked and participated throughout the epoch. The temporary stake also inflates the vote-boost component's finalized total weight. Because the root `RewardDistributor` uses that total when allocating rewards among components, the attack may additionally direct rewards away from other protocol reward components and into `VoteBoostRewardDistributor`, where the attacker captures most of the inflated allocation. The attack does not place protocol principal at risk and requires access to transferable stYFI plus reliable transactions around the boundary. Its profitability depends on the vote-boost reward amount exceeding borrowing costs, trading costs, and price risk.

Recommendation

Base an epoch's reward weight on stake that cannot be added only at the end of that epoch. Possible approaches include:

- maintaining a time-weighted stake integral throughout each epoch;
- maintaining an eligible balance for which increases become active only in the next epoch while decreases apply immediately to both active and pending balances; or
- applying the governance maturation schedule to increases while continuing to apply decreases immediately.

Applying decreases immediately is important so that the fix does not replace the current boundary-increase attack with a full-epoch reward for capital transferred out. Add tests that transfer stake immediately before and after an epoch boundary and verify that a short-lived balance cannot receive the same reward weight as stake held for the intended measurement period.

Developer Response

This is a known consequence of the way this reward distributor is designed. We consider it acceptable for two main reasons: 1) Obtaining a large temporary amount of stYFI right before

the deadline requires either borrowing it from someone, in which case it is basically the counterparty forfeiting their yield for that epoch or locking and unlocking themselves, in which case there is an unstaking delay. Neither of those options should be free 2) A naive integral approach is not possible here since the users weight (`num_votes * staked`) can change epoch-by-epoch without explicit interaction by the user since `num_votes` can change. An additional complicating matter is the decaying legacy boosts. Alternative implementations were therefore considered too complicated

2.6.3 Vetoed proposals continue accepting votes and grant participation rewards

Technical Details

The `Voting.veto` method marks a proposal as `vetoed`. When the proposal already has votes, it intentionally does not mark the proposal as `retracted` or invoke `hooks.on_retract`. However, `Voting.vote` only checks whether the proposal is retracted:

```
assert not self.proposals[_idx].retracted
```

It does not check `self.proposals[_idx].vetoed`. Consequently, accounts can continue voting on a vetoed proposal until the voting period ends.
Every new non-zero vote invokes `hooks.on_vote`. When the configured hook is `VoteBoostRewardDistributor`, its `on_vote` method credits the voter with participation for the current and following five epochs. An account can therefore obtain VBRD participation credit by voting on a proposal that the guardian has already permanently disabled.

Proof of Concept

The following unit test may be added to `tests/test_vote_boost_reward_distributor.py`:

```
def test_vote_after_veto_credits_participation(chain, deployer, alice, bob, genesis,
veyfi, verd, vbrd, voting, voter):
    # a vetoed proposal with an existing vote still accepts new votes
    unlock = genesis + 50 * EPOCH_LENGTH
    for account in (alice, bob):
        veyfi.set_locked(account, UNIT, unlock, sender=deployer)
        verd.set_snapshot(account, UNIT, 50, unlock, sender=deployer)
        verd.migrate(sender=account)

    voting.propose(IPFS_HASH, b"", sender=alice)
    chain.pending_timestamp = genesis + 6 * EPOCH_LENGTH - 48 * 60 * 60

    voter.vote_yea(voting, 0, sender=alice)
    voting.veto(0, "vetoed after voting began", sender=deployer)

    proposal = voting.proposals(0)
    assert proposal.vetoed
    assert not proposal.retracted
    votes_before = proposal.votes
    assert vbrd.num_votes(5, bob) == 0

    voter.vote_yea(voting, 0, sender=bob)

    assert voting.voted(bob, 0)
    assert voting.proposals(0).votes > votes_before
    assert vbrd.voted(bob, voting, 0)
    for epoch in range(5, 11):
        assert vbrd.num_votes(epoch, bob) == 1
```

Impact

Medium. Vetoed proposals continue accumulating votes, producing misleading governance signaling. More importantly, users can farm participation credit on dead proposals, increasing their share of VBRD rewards and diluting rewards received by users who only vote on active proposals.

Recommendation

Prevent voting on vetoed proposals by adding the following validation to `Voting.vote`:

```
 assert self.proposals[_idx].epoch == self._epoch()
 assert (block.timestamp - genesis) % EPOCH_LENGTH >= self.vote_start
 assert not self.proposals[_idx].retracted
+assert not self.proposals[_idx].vetoed
```

Developer Response

If a proposal has votes, it means some users will receive a boost from it. If the veto comes in after that it would be unfair towards the other users to not allow them to vote on it too and receive the boost. Therefore we still allow voting on vetoed proposals. The exception to that is when it gets vetoed before receiving any votes. in this case we can remove the proposal from the count in the vbrd without negative side effects

2.7 Low Findings

2.7.1 A stale vote-boost cursor can temporarily block all delegated-staking operations

Technical Details

The new `DelegatedStakingRewardClaimer` makes every reward synchronization for delegated stakers depend on `VoteBoostRewardDistributor.claim()`. If the vote-boost distributor is more than 32 epochs behind, that claim reverts and the failure propagates through the hook chain, causing stYFIx stake, unstake, and transfer operations to revert.
When delegated staking has an active supply, `DelegatedStakingRewardDistributor._sync_integral()` claims its upstream rewards during each user operation:

```
rewards: uint256 = extcall self.distributor.claim(self.distributor_claim)
```

After the PR's intended deployment, `self.distributor` is the new `DelegatedStakingRewardClaimer`. Its `claim()` synchronously claims both ordinary staking rewards and vote-boost rewards:

```
rewards += extcall staking_distributor.claim(delegated_staking)
rewards += extcall vote_boost_distributor.claim(delegated_staking)
```

The vote-boost leg requires its reward cursor to reach the current epoch:

```
assert self._sync_rewards(self._epoch())
```

However, `_sync_rewards()` processes at most 32 epochs:

```
for i: uint256 in range(32):
    if epoch == _current:
        break
    self.rewards[epoch] = (extcall self.distributor.claim())[2]
    epoch += 1

self.reward_epoch = epoch
return epoch == _current
```

If the cursor is more than 32 epochs behind, `_sync_rewards()` returns `False`. The assertion then reverts the entire transaction, including the cursor progress made inside `_sync_rewards()`. Because this call occurs within the stYFIx hook path, retrying the user operation cannot advance the cursor and will continue reverting.
The public `sync_rewards()` function does not assert full synchronization, so an external caller can recover the system by calling it enough times:

```
@external
def sync_rewards() -> bool:
    return self._sync_rewards(self._epoch())
```

A focused end-to-end test first synchronized the global `RewardDistributor` and `StakingRewardDistributor`, leaving only the vote-boost cursor more than 32 epochs behind. A subsequent stYFIx deposit reverted through the new claimer path. Two direct `VoteBoostRewardDistributor.sync_rewards()` calls caught the cursor up, after which the same deposit succeeded.
The same dependency also means the claimer bridge must not be activated before `VoteBoostRewardDistributor.activate()`, because a zero `reward_epoch` causes `_sync_rewards()` to revert.

Impact

Low. After more than 32 epochs of vote-boost inactivity, all delegated-staking operations can remain blocked until an external account manually synchronizes the stale cursor.
No funds are lost, and recovery is permissionless through a small number of direct synchronization transactions. The issue requires more than 32 epochs—approximately 448 days—without a claim or synchronization.

Recommendation

Allow `VoteBoostRewardDistributor.claim()` to preserve partial synchronization progress without reverting. For example, return zero before advancing the account's claim cursor when the global cursor is not yet current:

```
if not self._sync_rewards(self._epoch()):
    return 0
```

The next delegated-staking operation or keeper call can then continue catching up without blocking token operations or skipping account rewards. Additionally, enforce and test the deployment order: activate the vote-boost component before configuring the delegated-staking distributor to use the new claimer bridge.

Developer Response

Updated in 40c1a0e. Note that if the `RewardDistributor` also hasnt been interacted with for the same amount of time, the transaction will still revert since it requires being fully synced too.

2.7.2 A single account can saturate proposal slots

Technical Details

Each call to `VoteBoostRewardDistributor.on_propose` increments the proposal count for six consecutive epochs and reverts if any count would exceed `MAX_NUM_PROPOSALS_PER_EPOCH`, which is set to 64.
The default proposal cooldown is zero. Consequently, an account that satisfies the minimum proposal weight can submit 64 proposals during one epoch. These proposals fill all available slots for the following six-epoch window. Any subsequent proposal reverts in `on_propose`, including proposals submitted by other eligible accounts during the next five epochs.
The operator can free slots by flagging proposals only before they receive any votes. An attacker can prevent this after voting begins by immediately voting on every proposal. Doing so also causes `VoteBoostRewardDistributor.on_vote` to record 64 votes for the attacker across six epochs. Since reward weight is multiplied by the number of proposals on which an account voted, the attacker obtains the maximum participation multiplier while preventing legitimate proposals.

Proof of Concept
The following unit test may be added to `tests/test_vote_boost_reward_distributor.py`:

```
def test_proposal_slot_saturation(chain, deployer, alice, bob, genesis, veyfi, verd,
vbrd, voting, voter):
    # one proposer can fill the shared slots and farm the maximum vote boost
    unlock = genesis + 50 * EPOCH_LENGTH
    for account in (alice, bob):
        veyfi.set_locked(account, UNIT, unlock, sender=deployer)
        verd.set_snapshot(account, UNIT, 50, unlock, sender=deployer)
        verd.migrate(sender=account)

    for _ in range(64):
        voting.propose(IPFS_HASH, b"", sender=alice)

    assert voting.num_proposals() == 64
    for epoch in range(5, 11):
        assert vbrd.num_proposals(epoch) == 64

    # the attacker cannot submit a 65th proposal, but neither can another
    # eligible proposer in the following epoch because five slots overlap
    with reverts():
        voting.propose(IPFS_HASH, b"", sender=alice)

    chain.pending_timestamp += EPOCH_LENGTH
    with reverts():
        voting.propose(IPFS_HASH, b"", sender=bob)
    assert voting.num_proposals() == 64

    chain.pending_timestamp = genesis + 6 * EPOCH_LENGTH - 48 * 60 * 60
    voter.vote_yea(voting, 0, sender=alice)
    one_vote_weight = vbrd.weight(5, alice)
```

```
    assert one_vote_weight > 0

    for idx in range(1, 64):
        voter.vote_yea(voting, idx, sender=alice)

    for epoch in range(5, 11):
        assert vbrd.num_votes(epoch, alice) == 64
    max_vote_weight = vbrd.weight(5, alice)
    assert abs(max_vote_weight - 64 * one_vote_weight) < 64 * WEIGHT_PACKING_SCALE

    chain.pending_timestamp = genesis + 6 * EPOCH_LENGTH
    assert vbrd.sync_total_weight(5, sender=bob).return_value == max_vote_weight // 64
```

Impact

Low. An attacker can temporarily prevent governance proposals and increase their share of VoteBoostRewardDistributor rewards relative to users who do not vote on the spam proposals.

Recommendation

Enforce a nonzero per-account proposal cooldown or a separate per-account proposal limit so that one account cannot consume all shared proposal slots.

Developer Response

Added minimum cooldown of 1 day in 5b1ee9e.

2.8 Gas Savings Findings

None.

2.9 Informational Findings

2.9.1 Proposals can pass without any minimum quorum

Technical Details

Proposal passage depends only on the ratio of Yea votes to votes cast. There is no minimum turnout or minimum amount of Yea weight, so a single eligible holder can pass a proposal when nobody else votes.
`Voting._passed()` checks that at least one positive-weight vote exists and then applies the proposal's threshold solely to the votes cast:

```
votes: uint256 = self.proposals[_idx].votes
return votes > 0 and self.proposals[_idx].yea * PRECISION >= votes * self.proposals[_idx].threshold
```

For the default 50% threshold, any positive-weight voter voting entirely Yea satisfies this expression if there are no opposing votes. The `propose_min_weight` check does not provide quorum: it only controls who may create a proposal, and its default is one YFI of voting weight.

The proposal script may contain arbitrary calls that `Executor` performs with the protocol roles held by that contract. If `execute_guard` is disabled, any account can execute the passed script during its execution epoch. If the guard is enabled, the operator must submit the execution, but the proposal is still reported on-chain as passed based on the single vote.

Impact

Informational. An eligible proposer can authorize an arbitrary privileged proposal script with a single positive-weight Yea vote when no other holder participates, but the available documentation does not establish that a separate quorum is an intended requirement.
The long proposal lifecycle gives the community time to oppose the proposal, the operator can flag it before the first vote, the guardian can veto it before execution, and the default execution guard requires the operator to execute it. These controls materially reduce exploitability, but they are monitoring and trusted-role safeguards rather than an on-chain quorum. If permissionless execution is enabled or those safeguards fail to act, a low-turnout vote can affect any authority held by `Executor`. The behavior is therefore best treated as a governance-design consideration rather than a demonstrated vulnerability.

Recommendation

Require both:

1. a minimum quorum, expressed as an absolute amount or a fraction of a reliably snapshotted eligible voting supply; and
2. the existing Yea-to-cast-votes approval ratio.

Snapshot the quorum parameter when the proposal is created, as is already done for `threshold`, so management changes cannot alter an active proposal. If obtaining a safe total-supply snapshot is impractical, an absolute minimum Yea weight is still stronger than the current `votes > 0` condition.

Developer Response

This is intentional, quorums tend to only result in having to chase people to submit their votes. We expect that voter participation will be higher here becasue of the boost and the delegations.

2.9.2 Early-exit reports may fail to reclaim all outstanding rewards

`VoteBoostRewardDistributor.report()` permanently clears an account's cached veYFI lock even when the bounded reward-claim operation did not reach the requested timestamp. Consequently, an early exiter can retain part of the historical rewards that the report mechanism is intended to reclaim.

Technical Details

`report()` attempts to reclaim all of an early exiter's rewards through the end of the current epoch:

```
rewards: uint256 = self._claim(_account, genesis + (epoch + 1) * EPOCH_LENGTH)

self.ve_locks[_account] = empty(Lock)
staked: uint256 = staticcall self.weight_aggregator.staked(_account)
self._update_weights(_account, staked, False)
```

However, `_claim()` can process only 32 epoch transitions per invocation. If the account's claim cursor is farther behind, `_claim()` returns rewards for only the processed epochs and advances `last_claimed` to an intermediate timestamp:

```
for i: uint256 in range(32):
    epoch = unsafe_add(epoch, 1)
    ...

if synced:
    ...
    self.last_claimed[_account] = _time
else:
    self.last_claimed[_account] = genesis + (epoch + 2) * EPOCH_LENGTH
```

`_claim()` does not return whether it reached `_time`, and `report()` does not verify that `last_claimed[_account]` reached the end of the current epoch before clearing `ve_locks[_account]`.
For example, if the first unclaimed reward is for epoch 5 and the account is reported during epoch 40, one call processes epochs 5 through 37. The cached lock is then cleared, but rewards for epochs 38 and 39 remain associated with the account's historical packed weights. The account can subsequently obtain those rewards through the normal whitelisted claimer because regular claims do not require `ve_locks[_account]` to remain populated.
The account can trigger `report()` itself or front-run another reporter. Alternatively, an ordinary reporter can trigger the incomplete clawback unintentionally. Although an external caller can use `reclaim()` repeatedly before reporting, `report()` does not require this preparation. Moreover, `set_reward_expiration()` permits expiration periods of 32 epochs or more, which can prevent `reclaim()` from advancing the cursor close enough to the current epoch for a single report-time `_claim()` pass.
The sibling `VotingEscrowRewardDistributor` protects the equivalent flow by requiring:

```
assert synced or _time < block.timestamp
```

It also constrains the reward expiration period to fewer than 32 epochs.

Impact

A migrated veYFI holder who exits early can preserve and later claim a portion of their own unclaimed historical vote-boost rewards. These tokens should instead be transferred to the configured report recipient, optionally after paying the reporting bounty.
This is not a repeatable extraction strategy: the holder must genuinely exit the grandfathered veYFI position, cannot recreate the snapshotted position by relocking, and incurs the economic loss associated with early exit. The issue also does not allow the holder to claim another account's allocation. Therefore, intentionally performing an early exit solely to exploit this behavior is profitable only if the retained vote-boost rewards exceed the exit loss.
Once a holder has exited early for independent reasons, an incomplete report can still leave a residual claim. However, given the irreversible and penalized prerequisite, the inability to repeat the behavior with the snapshotted position, and the fact that only the holder's own allocation is affected, no practical profitable attack has been identified. This issue is therefore classified as Informational.

Recommendation

No change is recommended. The protocol can accept this one-time edge case because exploiting it requires an irreversible early exit at a loss and cannot be repeated by recreating the snapshotted veYFI position.

Developer Response

Acknowledged. For simplicity we keep as is

2.9.3 Inconsistent voting behavior documentation between `Voter` and `Voting`

Technical Details

`Voting.vote` documents that submitting a vote more than once overwrites the previous vote. However, both `Voter.vote_yea` and `Voter.vote_nay` call `Voter._vote`, which first asserts that the caller has not already voted for the proposal through `Voting.voted`.
Therefore, users can only submit one vote through `Voter`, despite `Voting.vote` documenting that votes may be overwritten. Since `Voting.vote` can only be called by the configured `Voter`, the documented overwrite behavior is unavailable to regular users.

Impact

Informational.
The inconsistent documentation may mislead integrators and users about whether votes can be changed after submission.

Recommendation

Update the documentation in `Voting.vote` to clarify that votes are overwritten when called by the `Voter`, while `Voter` prevents users from submitting multiple votes for the same proposal. Alternatively, remove the duplicate-vote check in `Voter` if vote changes are intended to be supported.

Developer Response

Updated documentation on `Voter` functions in 9395d5e.

stYFI Governance

Completed 2026-08-04