yAudit

May 2026

Prepared for
Yearn

Audited by
Panda
Watermelon

Yearn stYFI - YBC

Smart Contract Security Assessment

Contents

1 Review Summary

1.1 Protocol Overview

The system introduces stYFI and llYFI middlewares that sync staked balances to a WeightAggregator with a weight ramping mechanism. It also implements the Yearn Blue Chip (YBC) collective, featuring a dedicated WeightAggregator for member-only reward tracking and voting, a reward distribution system, and an internal election contract for membership management.

1.2 Audit Scope

This audit covers 9 smart contracts totaling approximately 1040 lines of code across 4 days of review.

```
├── ybc
│   ├── YBC.vy
│   ├── YBCBonusRecipient.vy
│   ├── YBCElection.vy
│   ├── YBCRewardDistributor.vy
│   └── YBCWeightAggregator.vy
├── LiquidLockerMiddleware.vy
├── SnapshotMeasure.vy
├── StakingMiddleware.vy
└── WeightAggregator.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	1
Medium	1
Low	1
Informational	1

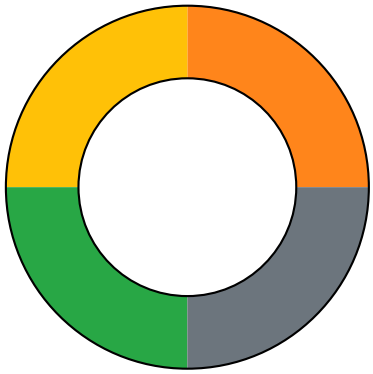

Figure 1: Distribution of security findings by impact level

1.5 Overall Assessment

The codebase is clean, modular, and builds upon mature stYFI primitives. It features robust role-based access control and solid test coverage. While the mathematics are generally accurate, minor edge cases exist in the voting weight ramping logic. Documentation and monitoring are well-implemented, ensuring a high-quality integration with Yearn infrastructure.

2 Audit Overview

2.1 Project Information

Protocol Name: Yearn
Repository: https://github.com/yearn/stYFI/
Commit Hash: 26001f97757e092fb65979665679b49f713c45da
Commit URL:
https://github.com/yearn/stYFI/tree/26001f97757e092fb65979665679b49f713c45da

2.2 Audit Team

Panda, Watermelon

2.3 Audit Timeline

The audit was conducted from April 27 to May 01, 2026.

2.4 Audit Resources

- Code repositories and documentation
- Additional resources: whitepaper, previous audits, etc.

Category	Mark	Description
Access Control	Good	Robust role-based access control, though some management functions could be more restrictive.
Mathematics	Good	Accurate calculations generally, with minor edge cases in voting weight ramping logic.
Complexity	Excellent	Clean and modular architecture, keeping the system easy to reason about.
Libraries	Excellent	Effective use of standard Vyper patterns and established Yearn infrastructure.
Decentralization	Good	Balanced governance model with typical Yearn management controls.
Documentation	Good	Clear code and reasonably well-documented interfaces and logic.
Testing and verification	Good	Solid test coverage supported by functional proofs of concept for reported issues.

Table 2: Code Evaluation Matrix

2.5 Critical Findings

None.

2.6 High Findings

2.6.1 Single ramping bucket lets hooks distort voting weight

Technical Details

`WeightAggregator` and `YBCWeightAggregator` track each account's immature stake with a single `(ramping, time)` bucket. On every increment, `_update_staked()` decays the existing `ramping` value, adds the incoming amount, and then sets `time = block.timestamp` for the entire combined remainder.
This means any increment hook can rewrite the ramp schedule for stake that was already immature. The newly added amount should start ramping from the current timestamp, but the account's previous immature balance should keep its existing ramp schedule. Zero or net-neutral hooks should not refresh prior ramping state at all.
This creates two related attack vectors:

1. **Same-epoch self-inflation:** a user can trigger a zero-amount deposit/mint, or stake and unstake dust in the same epoch. If the user's net stake is not greater than the epoch-start stake, `weight()` uses the refreshed current packed state, allowing post-snapshot ramp progress to increase current-epoch voting weight.
2. **Future-epoch dust griefing:** a third party can transfer dust staking shares to a victim before maturation. The victim cannot reject the inbound transfer, and the hook refreshes the timestamp for the victim's whole remaining immature balance, delaying future voting-weight maturation.

Example:

1. Alice has `100 YFI` still ramping.
2. Alice can use a zero or net-neutral hook after the epoch snapshot to inflate her current-epoch weight.
3. Bob can instead transfer `1 wei` of staking shares to Alice before later snapshots, resetting the remaining ramp for Alice's existing immature balance and suppressing her future weight.

Affected paths:

- `WeightAggregator.on_transfer()` and `on_stake()` call `_update_staked(..., INCREMENT)` for recipients and stakers.
- `WeightAggregator._update_staked()` rewrites `time = block.timestamp` for the combined ramping amount.
- `YBCWeightAggregator.on_transfer()` and `on_stake()` repeat the same pattern for YBC voting weight.

Impact

High. Voting weight can be distorted without a meaningful stake change. Users can increase their own current-epoch weight above the intended snapshot, while third parties can suppress a victim's future global or YBC voting weight with dust transfers.

Recommendation

Do not collapse all immature stake into one timestamped bucket. Track new inflows separately from previously immature stake, for example with per-epoch or per-deposit ramp buckets, and compute weight as the sum of those buckets. This ensures an inbound transfer only ramps the transferred amount and cannot reset prior vesting progress.
Also ignore zero-amount hooks before mutating aggregator state, and add regression tests for same-epoch zero/net-neutral hooks and dust transfers into both aggregators.

Developer Response

Because of this and other issues independently found with the ramping mechanism we have opted to replace it entirely and store weights for 4 sequential epochs instead, see commit 0f0f779

2.7 Medium Findings

2.7.1 `YBCRewardDistributor.sweep()` shouldn't be allowed to sweep rewards

Technical Details

`YBCRewardDistributor.sweep()` allows management to transfer out any token held by the contract, including the configured reward token.
The reward token balance is part of the distributor's accounting. `last_balance` tracks the amount of reward tokens backing already-accounted rewards, and `pending_rewards` tracks member claims against that balance. If management sweeps the reward token while the distributor is still live, the on-chain token balance can become lower than `last_balance` and/or lower than the amount needed to satisfy pending member claims.
This can cause future claims to revert due to insufficient token balance. It can also cause future reward synchronization to revert when `_sync_integral()` computes `balance - self.last_balance` after the reward token balance has been reduced externally.
Sweeping the reward token is only safe after the distributor has been killed, because `kill()` permanently disables new reward claims from the YBC while preserving access to rewards that were already accounted.

Impact

Management can accidentally or intentionally remove reward tokens that are still needed for live distribution, causing member claims and reward synchronization to fail.

Recommendation

Prevent sweeping the configured reward token unless `killed` is true.
For example, add a guard in `sweep()`:

```
assert _token != token.address or self.killed
```

If sweeping reward tokens after kill is intended to recover only excess funds, consider also documenting the operational requirement to avoid sweeping tokens needed for already-accrued member claims.

Developer Response

Fixed in bed5c0c.

2.8 Low Findings

2.8.1 `set_ramp_length` dilutes already accrued ramping for existing members

Technical Description

Both `WeightAggregator.set_ramp_length` and `YBCWeightAggregator.set_ramp_length` update the global `ramp_length` value that is read live from storage by `_staked` and `_update_staked` rather than being snapshotted at the time of each member's last stake update.
Both contracts' `_staked` method computes a member's weighted balance by scaling the still-ramping portion by the ratio of elapsed time over `ramp_length`. `_update_staked` similarly recomputes the residual ramping against the current `ramp_length` on every stake change. Since both methods read the live global value, any update to `ramp_length` immediately changes the result of these computations for every member who is still mid-ramp. As a consequence, if `management` increases `ramp_length`, the elapsed-time fraction shrinks for every active member, retroactively diluting ramping that has already been accrued.

Impact

Low.

Recommendation

Implement a ramp length snapshotting mechanism in both contracts so that members continue to ramp under the value in effect at the time of their most recent stake update, while new stake actions adopt the updated `ramp_length`.
A possible approach is to extend the per-account packed state to include the `ramp_length` that applied at the time of the last `_update_staked` call, and read that snapshotted value within `_staked` and on the next `_update_staked` invocation rather than the current global value.

Developer Response

This is no longer relevant, because of issues found with the ramping mechanism we have opted to replace it entirely and store weights for 4 sequential epochs instead, see commit 0f0f779.

2.9 Gas Savings Findings

2.9.1 Epoch boundary arithmetic overhead

Technical Details

veYFI unlock times are rounded to weekly boundaries. The stYFI `genesis` is also week-aligned, and each stYFI epoch is two weeks, so epoch boundaries are a subset of veYFI unlock boundaries.
`SnapshotMeasure._weight()` still computes the next epoch boundary on each `balanceOf()` call before checking `unlock_time >= epoch_end`.

Impact

Gas savings.

Recommendation

Remove the unnecessary code.

Developer Response

ACK, but this contract is only used for offchain purposes (voting on snapshot.org) so we are not concerned with gas usage

2.10 Informational Findings

2.10.1 Inconsistent naming of the `Hooks` interface across contracts

Technical Details

The same `Hooks` interface is referred to as `IHooks` in most contracts, but as `IStakedHooks` in the YBC contracts. Specifically, the interface is declared as `IHooks` in:

- `StakingMiddleware.vy#L15`
- `LiquidLockerMiddleware.vy#L11`
- `DelegatedStakingRewardDistributor.vy#L15`
- `StakedYFI.vy#L18`
- `WeightAggregator.vy#L18`
- `LiquidLockerRewardDistributor.vy#L16`

- `LiquidLockerDepositor.vy#L20`
- `StakingRewardDistributor.vy#L16`
- `DelegatedStakedYFI.vy#L21`

While it is declared as `IStakedHooks` in:

- `ybc/YBCRewardDistributor.vy#L23`
- `ybc/YBCWeightAggregator.vy#L17`

Using two different names for the same interface forces readers to mentally reconcile the two identifiers when navigating the codebase, and obscures the fact that YBC contracts implement and consume the very same hooks contract used by the rest of the system.

Impact

Informational.

Recommendation

Adopt a single name for the interface across all contracts. Renaming the `IStakedHooks` declarations in `YBCRewardDistributor.vy` and `YBCWeightAggregator.vy` to `IHooks` will align the YBC contracts with the convention already used throughout the rest of the codebase.

Developer Response

Updated in b4440bc.

Yearn stYFI - YBC

Completed 2026-05-01