yAudit YearnBoostedStaker Recheck Review

Review Resources:

None beyond the code repositories.

Auditors:

- invader-tak
- shakotan
- adriro

Table of Contents

Review Summary

Yearn BoostedStaker

Yearn BoostedStaker is a new contract template designed to be a more interesting staking system that rewards long-term alignment more-so than previous systems. The contract is meant to become the primary staking contract for Yearn locker products: yCRV and yPRISMA. The protocol is designed to boost the weight of deposits week by week until the max boost is reached.

Week	Weight	Boost
0 *(deposit week)*	50	0.5x boost
1	100	1x boost
2	150	1.5x boost
3	200	2x boost
4 *(final growth week)*	250	2.5x boost
5 *...n*	250	2.5x boost

The YearnBoostedStaker contract Repo were reviewed over 4 days. The code review was performed by 3 auditors between Jan 29 and Feb 1, 2024. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit a836d7bc6b105aec867c953c4611247d423d5e79 for the yPrisma repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
contracts
└── YearnBoostedStaker.sol
```

After the findings were presented to the Yearn Finance team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Yearn Finance and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Proper access control mechanisms are in place, ensuring that only authorized entities can execute sensitive functions.
Mathematics	Good	Arithmetic operations are securely handled, mitigating risks like overflow/underflow and ensuring precision.
Complexity	Average	The contract maintains a balance between functionality and complexity, minimizing potential attack vectors due to overly complex code.
Libraries	Average	Low usage of libraries that match the code's average complexity. Utilizes a well-tested external library once where it is appropriate.

Category	Mark	Description
Decentralization	Good	The contract design aligns with decentralization principles, avoiding single points of failure or control.
Code stability	Good	No changes where made to the contract during the review.
Documentation	Average	Comprehensive presentation is provided in the README, NatSpec comments are available for every function of the contract.
Monitoring	Good	Contract has proper events emitted on changes, facilitating real-time monitoring and tracking of activities.
Testing and verification	Low	Given the complexity of the design, more testing could help improve confidence in the codebase.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings
  - Findings that can improve the gas efficiency of the contracts.
- Informational
  - Findings including recommendations and best practices.

Critical Findings

None.

High Findings

None.

Medium Findings

None.

Low Findings

1. Low - MAX_STAKE_GROWTH_WEEKS can be set to 0 days

When deploying `YearnBoostedStaker`, `MAX_STAKE_GROWTH_WEEKS` immutable can be set to 0 days which can lead to unexpected behaviour.

Technical Details

It is possible to set `MAX_STAKE_GROWTH_WEEKS` when deploying the contract. This leads to interesting behaviour.

For example: Deploying `YearnBoostedStaker` with `MAX_STAKE_GROWTH_WEEKS` as 0.

Then Alice deposits 100e18 tokens at week 0. After 20 has passed, Alice weight is now 1050e18, which doesn't seem correct.

Impact

Low.

Recommendation

We suggest to guard against setting 0 days in constructor of the contract to avoid issues with weight calculation.

Developer Response

Added condition to require statement.

2. Low - `stakeToken.transferFrom()` in `_deposit()` method should be moved lower down by the call stack

`stakeToken.transferFrom()` is called at the very top of the `_deposit()` function. The same applies to the `depositAsWeighted()` function.

Technical Details

1. This breaks CEI pattern, because interaction is done on the very top of the callstack.
2. This will generate dust in the contract, because transfer is made before the rounding calculation is done.

Impact

Low.

Recommendation

Move `stakeToken.transferFrom()` in `_deposit()` function to the line after the rounding is being made. This will fix the dust problem.

Developer Response

Moved near bottom of function.

Gas Saving Findings

1. Gas - Unnecessary check when updating an account's week bitmap

The logic to update the week bitmap checks that weight is greater than zero, a condition that is always true.

Technical Details

In `_deposit()` and `depositAsWeighted()`, the `amount` is required to be greater than 1, and since `weight` is half of that, then `weight >= 1`.

Impact

Gas savings.

Recommendation

Remove the `weight > 0` check.

```
-   if (previous == 0 && weight > 0) {
+   if (previous == 0) {
        // We flip the far right bit to indicate an update occurred in current week.
        acctData.updateWeeksBitmap += 1;
    }


    // Only update storage bitmap if bit at target position needs to be flipped.
-   if (bitmap & mask != mask && weight > 0) {
+   if (bitmap & mask != mask) {
        acctData.updateWeeksBitmap = bitmap ^ mask; // Flip single bit.
    }
```

Developer Response

Updated both lines to remove weight check.

2. Gas - Incorrect optimization when loading previous weight in deposit

In `_deposit()`, the previous weight is conditionally loaded if the pending stake is greater than zero, which is always true as it is updated before the check.

Technical Details

The implementation of `_deposit()` loads the previous weight from `accountWeeklyRealized` only if the `pendingStake` is not zero (line 126).

```
118:         uint weight = _amount >> 1;
119:         _amount = weight << 1; // This helps prevent balance/weight
discrepancies.
120:
121:         acctData.pendingStake += uint112(weight);
122:         globalGrowthRate += uint112(weight);
123:
124:         uint realizeWeek = systemWeek + MAX_STAKE_GROWTH_WEEKS;
125:         uint previous;
126:         if (acctData.pendingStake != 0) previous =
accountWeeklyRealized[_account][realizeWeek]; // Only SLOAD if needed.
```

However, `pendingStake` is updated at line 121 with the weight for the current deposit, causing this variable to always be different from zero.

Impact

Gas savings.

Recommendation

Invert the order of the operations.

```
    uint weight = _amount >> 1;
    _amount = weight << 1; // This helps prevent balance/weight discrepancies.

+   uint realizeWeek = systemWeek + MAX_STAKE_GROWTH_WEEKS;
+   uint previous;
+   if (acctData.pendingStake != 0) previous = accountWeeklyRealized[_account]
[realizeWeek]; // Only SLOAD if needed.

    acctData.pendingStake += uint112(weight);
    globalGrowthRate += uint112(weight);

-   uint realizeWeek = systemWeek + MAX_STAKE_GROWTH_WEEKS;
-   uint previous;
-   if (acctData.pendingStake != 0) previous = accountWeeklyRealized[_account]
[realizeWeek]; // Only SLOAD if needed.
```

Developer Response

I don't really understand what is meant by "Invert the order of operations" for this. However, I've removed the 0 check.

3. Gas - Duplicate weight calculation in `depositAsWeighted()`

The base weight for a deposit is calculated twice in `depositAsWeighted()`.

Technical Details

The weight for the deposit (half the amount) is first calculated in line 167, and then re-calculated in line 179 when incrementing `globalGrowthRate`.

Impact

Gas savings.

Recommendation

Use `weight` to increment `globalGrowthRate`.

```
-   globalGrowthRate += uint112(_amount >> 1);
+   globalGrowthRate += uint112(weight);
```

Developer Response

Removed the double calc. Seems safe since `weight` doesn't change in lines above.

4. Gas - Account data is persisted twice to storage

Most functions persist the account data struct at the end of its implementation, which also happens when they first checkpoint.

Technical Details

In deposit (and all of its variants) and withdraw, the implementation first checkpoints the account, which saves the `accountData` struct to storage, while at the end of the implementation it updates the struct again to persist other potential changes that are part of the logic of each function.

Impact

Gas savings.

Recommendation

The internal `_checkpointAccount()` could avoid persisting the struct back to storage and delegate that responsibility to the caller.

Developer Response

Updated internal function not to write `acctData` to storage. Updated all callers of `_checkpointAccount()` to eventually write that value to storage.

5. Gas - Account checkpointing can be skipped if up to date

The implementation of `_checkpointAccount()` is executed even if the last update matches the current system week.

Technical Details

In `_checkpointAccount()`, the function checks that `_systemWeek >= lastUpdateWeek`, but still continues to execute all of its logic if these two values match. It can be seen that, if `_systemWeek == lastUpdateWeek`, then no changes will be made (none of the loops will execute).

Impact

Gas savings.

Recommendation

The function could early return if `lastUpdateWeek == systemWeek`.

Developer Response

Added an early return.

6. Gas - Bitmap handling optimizations

Turning on a bit in the bitmap can be optimized by using a bitwise `or` operation.

Technical Details

The implementation of `_deposit()` checks that the pending weight is zero before incrementing `acctData.updateWeeksBitmap` to flip the right-most bit.

```
133:         if (previous == 0 && weight > 0) {
134:             // We flip the far right bit to indicate an update occurred in
current week.
135:             acctData.updateWeeksBitmap += 1;
136:         }
```

This could be simplified, and improved in terms of security, by doing `bitmap | 1`.

Similarly, in `depositAsWeighted()`, the implementation checks that the associated bit is turned off before flipping it.

```
185:             uint8 mask = uint8(1 << _idx);
186:             uint8 bitmap = acctData.updateWeeksBitmap;
187:             // Only update storage bitmap if bit at target position needs to be
flipped.
188:             if (bitmap & mask != mask && weight > 0) {
189:                 acctData.updateWeeksBitmap = bitmap ^ mask; // Flip single bit.
190:             }
```

This could also be simplified as `bitmap | mask`.

Impact

Gas savings.

Recommendation

For `_deposit()`,

```
-   if (previous == 0 && weight > 0) {
-       // We flip the far right bit to indicate an update occurred in current week.
-       acctData.updateWeeksBitmap += 1;
-   }
+   acctData.updateWeeksBitmap |= 1;
```

For `depositAsWeighted()`,

```
    uint8 mask = uint8(1 << _idx);
-   uint8 bitmap = acctData.updateWeeksBitmap;
-   // Only update storage bitmap if bit at target position needs to be flipped.
-   if (bitmap & mask != mask && weight > 0) {
-       acctData.updateWeeksBitmap = bitmap ^ mask; // Flip single bit.
-   }
+   acctData.updateWeeksBitmap |= mask;
```

Developer Response

Implemented suggested bitwise operator change.

7. Gas - Storage load reordering

Some storage loads can be reordered to potentially save gas by avoiding some SLOADs.

Technical Details

In `getAccountWeightAt()`, the weight is loaded in line 404, but the early return in line 406 doesn't depend on it.

```
404:         uint weight = accountWeeklyWeights[_account][lastUpdateWeek];
405:
406:         if (lastUpdateWeek >= _week) return accountWeeklyWeights[_account]
[_week];
407:         if (pending == 0) return weight;
```

Impact

Gas savings.

Recommendation

Reorder storage loads to save gas.

Developer Response

Reordered as suggested.

8. Gas - Withdraw recalculates the original amount needed

In `_withdraw()`, the implementation modifies the amount of weight needed, only to recalculate it again.

Technical Details

The `amountNeeded` variable is first calculated in line 235, and then modified throughout the implementation to account for removed weight.

```
235:         uint amountNeeded = _amount >> 1;
```

To calculate the amount of pending weight that was removed (`pendingRemoved`), the implementation calculates again the original amount of needed weight to be removed.

```
268:         uint pendingRemoved = (_amount >> 1) - amountNeeded;
```

Impact

Gas savings.

Recommendation

Instead of using `amountNeeded` to track the removed weight, copy this value to a new `amountLeft` variable and use this variable to track removed weight. Then pending can be calculated as `pendingRemoved = amountNeeded - amountLeft`.

Developer Response

Ignored. I agree with the assessment, but choosing not to change the implementation.

9. Gas - `_withdraw()` function can use AccountData return value to avoid additional account data SLOAD

`_withdraw()` function ignores the return data of `_checkPointAccount()` method.

Technical Details

`_checkpointAccount()` method returns updated account data, which means `_withdraw()` can use it instead of doing an SLOAD.

Impact

Gas savings.

Recommendation

```
    function _withdraw(address _account, uint _amount, address _receiver) internal
returns (uint) {
        require(_amount > 1 && _amount < type(uint112).max, "invalid amount");
        uint systemWeek = getWeek();
        _checkpointAccount(_account, systemWeek);
       AccountData memory acctData = accountData[_account];
```

To avoid SLOAD for `acctData`.

Developer Response

Updated `_withdraw()` to remove extra SLOAD.

Informational Findings

1. Informational - `transferOwnership()` method is missing an event

Permissioned `transferOwnership()` method is missing an event.

Technical Details

Staker implements Ownable 2 step logic. But only one event gets emitted and only when ownership is accepted: `OwnershipTransferred`.

Impact

Informational.

Recommendation

Add `emit OwnershipTransferStarted(owner(), newOwner);` to indicate when ownership transfer started.

Developer Response

Chose not to implement.

2. Informational - Consider using custom errors

Replace `require()` statements with custom errors.

Technical Details

Given that the contracts have undergone extensive gas optimization, it seems reasonable to replace the require statements with custom errors.

Impact

Informational.

Recommendation

Consider using custom errors in place of `require` and string error messages.

Developer Response

Chose not to implement.

3. Informational - Consider using higher required amounts for deposit

Consider using higher required amounts for deposit, not 1 WEI.

Technical Details

In the deposit function, it is permitted to deposit a token amount of 1 wei, as indicated by the requirement `require(_amount > 1 && _amount < type(uint112).max, "invalid amount");`. However, it is unclear whether there is a practical use case for depositing such small amounts into the contract. While rounding errors are addressed, it might be wise to impose restrictions to prevent unintended interactions with the contract.

Impact

Informational.

Recommendation

Something like this can be used: `require(_amount > 10 ** stakeToken.decimals() && _amount < type(uint112).max, "invalid amount");`.

Developer Response

Chose not to implement.

4. Informational - `depositAsWeighted()` doesn't backfill weights for older weeks

When using `depositAsWeighted()`, weights are set for current system week and carried forward, but older periods are not backfilled.

Technical Details

The `depositAsWeighted()` function is a privileged action that allows whitelisted accounts to make deposits with a certain week maturity. As opposed to `deposit()`, the caller can specify a week `_idx` value that acts as a shift in the past, making the deposit look like it was deposited `_idx` weeks before the current system week.

The implementation adds the `instantWeight`, which is the boosted weight determined by `_idx`, to the current system week in the `accountWeeklyWeights` and `globalWeeklyWeights` mappings, and sets everything to carry the growth forward (unless `_idx` is at max growth which would just set the deposit as realized).

However, weights for previous weeks are not modified. A normal deposit would set weights accordingly for each period, setting 0.5x for the initial week, 1x for the next one, and so on. In contrast, deposits using `depositAsWeighted()` would miss weights for the period between `systemWeek - _idx` and `systemWeek`.

Impact

Informational.

Recommendation

Consumers of this contract may wrongly assume that `depositAsWeighted()` has the same effect as a `deposit()` in a past week. It is recommended to document this behavior, warning eventual integrations that `depositAsWeighted()` functions as a pre-boosted deposit that doesn't impact the weights of past weeks.

Developer Response

Added comment to function.

Final remarks

In our final assessment following the Yearn Finance team's response to our initial report, we revisited the protocol for a second review. Our findings indicate no severe issues, though we have identified some minor observations and opportunities for gas savings that could enhance the codebase. Notably, the complexity of the proposed staking solution, which diverges from typical designs in decentralized protocols, warranted additional scrutiny. This complexity stems from the requirement to efficiently manage weekly weights and boosts, leading to a range of optimizations and varied execution paths in critical functions such as deposit, withdrawal, and checkpointing. We conducted comprehensive testing, including edge case analysis and an invariant test suite comparing the results with a simplified re-implementation of the solution through state-fuzzing differential testing. These tests validated the robustness of the solution.

Following our review, the Yearn Finance team addressed the majority of the issues, further optimizing the code for gas efficiency without leading to any new findings.