Aura Finance contest
Findings & Analysis Report

2022-07-26

Table of contents

- L-03 Function reverts if called a second time
- L-04 `pragma experimental ABIEncoderV2` is deprecated
- L-05 `safeApprove()` is deprecated
- L-06 Missing checks for `address(0x0)` when assigning values to `address` state variables
- N-01 Unused file
- N-02 Call `For` / `From` variants instead of copying an pasting code
- N-03 Remove tautological code
- N-04 Adding a `return` statement when the function defines a named return variable, is redundant
- N-05 `override` function arguments that are unused should have the variable name removed or commented out to avoid compiler warnings
- N-06 `public` functions not called by the contract should be declared `external` instead
- N-07 `type(uint<n>).max` should be used instead of `uint<n>(-1)`
- N-08 `constant` s should be defined rather than using magic numbers
- N-09 Redundant cast
- N-10 Numeric values having to do with time should use time units for readability
- N-11 Missing event for critical parameter change
- N-12 Use a more recent version of solidity
- N-13 Use a more recent version of solidity
- N-14 Use a more recent version of solidity
- N-15 Constant redefined elsewhere
- N-16 Inconsistent spacing in comments
- N-17 Non-library/interface files should use fixed compiler versions, not floating ones
- N-18 Typos
- N-19 File is missing NatSpec

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the Aura Finance smart contract system written in Solidity. The audit contest took place between May 11—May 25 2022.

Wardens

109 Wardens contributed reports to the Aura Finance contest:

1. csanuragjain
2. cccz
3. IIIIIII
4. 0xjuicer
5. hyh
6. kirk-baird
7. catchup
8. QuantumBrief (pedroais, GermanKuber, and fatherOfBlocks)
9. WatchPug (jtp and ming)
10. kenzo
11. Chom
12. Kumpa
13. 0x52
14. 0xsomeone
15. xiaoming90
16. MaratCerby

17. BowTiedWardens (BowTiedHeron, BowTiedPickle, m4rio_eth, Dravee, and BowTiedFirefox)
18. Aits
19. reassor
20. TerrierLover
21. 0xkatana
22. SmartSek (0xDjango and hake)
23. defsec
24. robee
25. 0xNazgul
26. 0x4non
27. joestakey
28. c3phas
29. Hawkeye (0xwags and 0xmint)
30. Tomio
31. hansfriese
32. kenta
33. MiloTruck
34. CertoraInc (egjlmn1, OriDabush, ItayG, and shakedwinder)
35. sashik_eth
36. _Adam
37. fatherOfBlocks
38. 0x1f8b
39. Funen
40. 0xf15ers (remora and twojoy)
41. Kaiziron
42. delfin454000
43. simon135

44. Waze
45. ellahi
46. mics
47. FSchmoede
48. bobirichman
49. cthulhu_cult (badbird and seanamani)
50. unforgiven
51. Ruhum
52. Tadashi
53. oyc_109
54. asutorufos
55. sach1r0
56. sikorico
57. NoamYakov
58. samruna
59. GimelSec (rayn and sces60107)
60. JC
61. Kthere
62. SooYa
63. z3s
64. jayjonah8
65. zmj
66. tintin
67. berndartmueller
68. cryptphi
69. Nethermind
70. PPrieditis
71. Rolezn

72. sorrynotsorry
73. BouSalman
74. p_crypt0
75. sseefried
76. 242
77. 0xNineDec
78. AlleyCat
79. ch13fd357r0y3r
80. JDeryl
81. hubble (ksk2345 and shri4net)
82. Cityscape
83. 0xKitsune
84. UnusualTurtle
85. rfa
86. 0v3rf10w
87. DavidGialdi
88. Fitraldys
89. Randyyy
90. antonttc
91. minhquanym
92. marcopaladin
93. orion

This contest was judged by LSDan.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 23 unique vulnerabilities. Of these vulnerabilities, 1 received a risk rating in the category of HIGH severity and 22

received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 76 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 66 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Aura Finance contest repository, and is composed of 44 smart contracts written in the Solidity programming language and includes 6,034 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (1)

[H-01] User can forfeit other user rewards

Submitted by csanuragjain

ExtraRewardsDistributor.sol#L127

User can forfeit other user rewards by giving a higher _startIndex in getReward function.

Proof of Concept

1. Assume User B has not received any reward yet so that his userClaims[_token][User B]=0
2. User A calls getReward function with _account as User B and _startIndex as 5
3. This eventually calls _allClaimableRewards at ExtraRewardsDistributor.sol#L213 which computes epochIndex =5>0?5:0 = 5
4. Assuming tokenEpochs is 10 and latestEpoch is 8, so reward will computed from epoch 5 till epoch index 7 and _allClaimableRewards will return index as 7
5. _getReward will simply update userClaims[_token][User B] with 7
6. This is incorrect because as per contract User B has received reward from epoch 0-7 even though he only received reward for epoch 5-7

Recommended Mitigation Steps

Do not allow users to call getReward function for other users.

0xMaharishi (Aura Finance) confirmed, but disagreed with severity and commented:

This is a valid report, however, considering it is only related to the distribution of reward tokens, I have a hard time classifying this as high risk.

LSDan (judge) commented:

I agree with the high risk rating on this one. A third party could cause significant loss of expected reward funds for users across the entire protocol if so inclined.

0xMaharishi (Aura Finance) resolved:

code4rena aurafinance/aura-contracts#84
All code4rena fixes code-423n4/2022-05-aura#6

Medium Risk Findings (22)

[M-01] `BaseRewardPool4626` is not IERC4626 compliant

Submitted by 0xjuicer

BaseRewardPool4626.sol

BaseRewardPool4626 is not IERC4626 compliant.
This makes the BaseRewardPool4626 contract irrelevant as it is for now since projects won't be able to integrate with BaseRewardPool4626 using the eip-4626 standard.

Recommended Mitigation Steps

You can choose to remove the BaseRewardPool4626 and save on some deployment gas or review the necessary `functions` and `emits` required on eip-4626 and add it to BaseRewardPool4626.

0xMaharishi (Aura Finance) confirmed, but disagreed with severity and commented:

Valid report. Probably should be severity 1 though.. no funds are ever at risk under any scenario.

LSDan (judge) commented:

I agree with medium risk here.

0xMaharishi (Aura Finance) resolved:

code-423n4/2022-05-aura#5

[M-02] `CrvDepositorWrapper.sol` relies on oracle that isn't frequently updated

Submitted by 0x52

CrvDepositorWrapper.sol#L56-L65

Unpredictable slippage, sandwich vulnerability or frequent failed transactions

Proof of Concept

CrvDepostiorWrapper uses the TWAP provided by the 20/80 WETH/BAL. The issue is that this pool has only handled ~15 transactions per day in the last 30 days, which means that the oracle frequently goes more than an hour without updating. Each time a state changing operation is called, the following code in the balancer pool takes a snapshot of the pool state BEFORE any operation changes it:

OracleWeightedPool.sol#L156-L161

This could result in the price of the oracle frequently not reflecting the true value of the assets due to infrequency of update. Now also consider that the pool has a trading fee of 2%. Combine an inaccurate oracle with a high fee pool and trades can exhibit high levels of "slippage". To account for this outputBps in AuraStakingProxy needs to be set relatively low or risks frequent failed transactions when calling distribute due to slippage conditions not being met. The lower outputBps is set the more vulnerable distribute becomes to sandwich attacks.

Recommended Mitigation Steps

Consider using chainlink oracles for both BAL and ETH to a realtime estimate of the LP value. A chainlink LP oracle implementation can be found here.

0xMaharishi (Aura Finance) confirmed and commented:

Valid finding and agree with the severity generally. Vector here is either function reverting or potentially getting sandwiched.

To mitigate this currently, there is a `keeper` address added and the tx would be sent via flashbots, however agree that other steps could be taken to allow it to operate more fluidly.

0xMaharishi (Aura Finance) resolved and commented:

Resolution for now is to use the CrvDepositorWrapper price as a guideline and let the `keeper` of AuraStakingProxy provide a minOut.

[M-03] Improperly Skewed Governance Mechanism

Submitted by 0xsomeone

AuraLocker.sol#L594-L609
AuraLocker.sol#L611-L618

File	Lines	Type	
AuraLocker.sol	L594-L609, L611-L618	Governance Susceptibility	

Description

The balance checkpointing system exposed by the contract for governance purposes is flawed as it does not maintain voting balances properly. In detail, the total supply of votes is tracked as the sum of all locked balances, however, the total voting power of an individual only tracks delegated balances. As a result, governance percentage thresholds will be significantly affected and potentially unmet.

Impact

The governance module may be unusable due to the significant discrepancy between “circulating” voting power supply and the actual voting power of each individual summed up.

Solution (Recommended Mitigation Steps)

We advise the total voting supply to properly track the delegated balances only as otherwise, any system relying on proportionate checkpointed balances will fail to function properly.

Proof of Concept

Issue is deducible by inspecting the relevant lines referenced in the issue and making note of the calculations within the `getPastVotes` individual voting power

function as well as the `getPastTotalSupply` cumulative voting power function.

0xMaharishi (Aura Finance) disputed and commented:

This is intended behaviour. There is no incentive for users not to delegate their votes. And even if there were, not delegating is the equivalent to having voting power but not voting. Therefore this is not a relevant issue.

LSDan (judge) decreased severity to Medium and commented:

I'm going to leave this one in play and downgrade the severity. The warden's report is accurate; however, if the required percentages of voting cannot be met, the DAO would simply have to go on a campaign to get people to delegate their votes. This would be annoying but not critically destructive. That said, medium severity makes sense because a bad actor could potentially gather voting power and intentionally disrupt things by not delegating it. I'd recommend implementing the fix suggested by the warden.

[M-04] `AuraLocker` kick reward only takes last locked amount into consideration, instead of whole balance

Submitted by kenzo

The issue occurs in AuraLocker, when expired locks are processed via kicking, and if all the user locks have expired.
In this scenario, to calculate the kick reward, `_processExpiredLocks` multiplies the last locked amount by the number of epochs between the last lock's unlock time and the current epoch.
A comment in this section mentions `"wont have the exact reward rate that you would get if looped through"`. However, there's no reason not to multiply *user's whole locked balance* by the number of epochs since the *last lock's* unlock time, *instead of only the last locked amount*.
While this will still not be as accurate as looping through, this will give a more accurate kick reward result, which is still bounded by the full amount that would have been calculated if we had looped through.

Impact

The reward calculation is inaccurate and lacking for no reason.
Kickers receive less rewards than they should.
Giving them a bigger, more accurate reward, will incentivize them better.

Proof of Concept

This is the section that calculates the kick reward if all locks have expired:

```
//check for kick reward
//this wont have the exact reward rate that you would get
//but this section is supposed to be for quick and easy l
//we'll assume that if the reward was good enough someone
if (_checkDelay > 0) {
    uint256 currentEpoch = block.timestamp.sub(_checkDela
    uint256 epochsover = currentEpoch.sub(uint256(locks[l
    uint256 rRate = AuraMath.min(kickRewardPerEpoch.mul(e
    reward = uint256(locks[length - 1].amount).mul(rRate)
}
```

This flow is for low gas processing, so the function is not looping through all the locks (unlike the flow where some locks have not expired yet).
In this flow, the function is just calculating the reward for the last lock.

Instead of doing this, it can multiply the *total amount locked by the user* (`locked` , already saved) by the *number of epochs between the last unlock time and current epoch*.
The reward will still be smaller than if we had looped through all the rewards (since then each lock amount would be multiplied by more than just the last lock's number of expired epochs).
But it would be more accurate and give better incentive for kicking.

Recommended Mitigation Steps

Change the last line in the code above to:

```
reward = uint256(locked).mul(rRate).div(denominator);
```

This will keep the low gas consumption of this flow, while giving a more accurate result.

0xMaharishi (Aura Finance) confirmed, but disagreed with severity and commented:

Valid, but unsure if it should be classified as medium risk. Probably 1.

LSDan (judge) commented:

I'm going to leave this one as medium because there is unnecessary fund loss over time. Good suggestions.

0xMaharishi (Aura Finance) resolved:

code4rena aurafinance/aura-contracts#84
All code4rena fixes code-423n4/2022-05-aura#6

[M-05] Users can grief reward distribution

Submitted by lllllll

Users can grief reward distributions by spending dust.

Proof of Concept

If a reward is targeted for an epoch in the past, a user can front-run the txn in the mempool and call `addRewardToEpoch()` with a dust amount at an epoch after the one in question. This will cause the transaction in the mempool to revert

```
File: contracts/ExtraRewardsDistributor.sol   #1

74              require(len == 0 || rewardEpochs[_token][len - 1] <
```

ExtraRewardsDistributor.sol#L74

Recommended Mitigation Steps

Allow the backdating of rewards, which will cost more gas

0xMaharishi (Aura Finance) acknowledged, but disagreed with severity and commented:

Fair finding; however, this is a peripheral contract and only affects user reward claiming. In the Aura system, rewards are only added to the current epoch so should be fine.

0xMaharishi (Aura Finance) resolved:

All code4rena fixes code-423n4/2022-05-aura#6
code4rena aurafinance/aura-contracts#84

[M-06] Rewards distribution can be delayed/never distributed on `AuraLocker.sol#L848`

Submitted by 0xjuicer, also found by csanuragjain

Rewards distribution can be delayed/never distributed on AuraLocker.sol#L848

Issue

Someone malicious can delay the rewards distribution for non `cvxCrv` tokens distributed on AuraLocker.sol.

1: Attacker will send one wei of token that are distributed on the AuraLocker.sol to AuraStakingProxy.

2: Attacker will call distributeOther.
The function will call notifyRewardAmount that calls _notifyReward

When calling _notifyReward the rewards left to distribute over the 7 days are redistributed throughout a new period starting immediately.

```
uint256 remaining = uint256(rdata.periodFinish).sub(block.timestamp);
uint256 leftover = remaining.mul(rdata.rewardRate);
```

```
rdata.rewardRate = _reward.add(leftover).div(rewardsDuration).to96();
```

Example: If the reward rate is 1 token (10**18) per second and 3.5 days are left (302400 seconds), we get a leftover of 302400 tokens. this is then divided by 604800, the reward rate is now 0.5 and the user of the protocol will have to wait one week for tokens that were supposed to be distributed over 3.5 days. This can be repeated again and again so that some rewards are never distributed.

Recommended Mitigation Steps

I can see that queueNewRewards has some protective mechanism. A new period is started only if the token that is added on top of the already distributed tokens during the duration is over 120%.

I suggest adding a similar check to queueNewRewards

0xMaharishi (Aura Finance) confirmed, but disagreed with severity

0xMaharishi (Aura Finance) resolved:

code4rena aurafinance/aura-contracts#84
code-423n4/2022-05-aura#6

[M-07] Reward may be locked forever if user doesn't claim reward for a very long time such that too many epochs have been passed

Submitted by Chom

ExtraRewardsDistributor.sol#L233-L240
AuraLocker.sol#L334-L337

Reward may be locked forever if user doesn't claim reward for a very long time such that too many epochs have been passed. The platform then forced to reimburse reward to the user that got their reward locked. Causing huge economics loss.

Proof of Concept

Can be done by reverse engineering from the affected code

```
for (uint256 i = epochIndex; i < tokenEpochs; i++) {
    //only claimable after rewards are "locked in"
    if (rewardEpochs[_token][i] < latestEpoch) {
        claimableTokens += _claimableRewards(_account, _token
        //return index user claims should be set to
        epochIndex = i + 1;
    }
}
```

From this line you will see a loop from epochIndex to tokenEpochs which loop tokenEpochs - epochIndex times.
If tokenEpochs - epochIndex value goes high, it will consume too much gas which go beyond the limit of the chain and cause the transaction to be always failed. As a result, reward may be locked forever.

```
uint256 latestEpoch = auraLocker.epochCount() - 1;
// e.g. tokenEpochs = 31, 21
uint256 tokenEpochs = rewardEpochs[_token].length;

// e.g. epochIndex = 0
uint256 epochIndex = userClaims[_token][_account];
// e.g. epochIndex = 27 > 0 ? 27 : 0 = 27
epochIndex = _startIndex > epochIndex ? _startIndex : epochIn
```

- epochIndex is the maximum of _startIndex and latest index of rewardEpochs that user has claim the reward
- tokenEpochs is the number of epochs that has reward, can be added through `addRewardToEpoch` function up to latest epoch count of auraLocker
- latestEpoch is epoch count of auraLocker

If you specified too high _startIndex, the reward may be skipped and these skipped reward are lost forever as the _getReward function set latest epoch that user has claim to the lastest index of rewardEpochs that can be claimed.

the aura locker epoch can be added by using `checkpointEpoch` function which will automatically add epochs up to current timestamp. Imagine today is 100 years from latest checkpoint and rewardsDuration is 1 day, the total of around 36500 epochs needed to be pushed into the array in single transaction which always failed due to gasLimit. The code that responsible for pushing new epochs below (in AuraLocker file)

```
while (epochs[epochs.length - 1].date != currentEpoch) {
    uint256 nextEpochDate = uint256(epochs[epochs.length
    epochs.push(Epoch({ supply: 0, date: uint32(nextEpochD
}
```

Even if these line are passed because the nature that checkpointEpoch is likely to be called daily and reward are added daily. if user doesn't claim the reward for 100 years, `rewardEpochs[_token].length = 36500 where epochIndex = 0. Which cause an impossible loop that run 36500 times` . In this case this transaction will always be failed due to gas limit. In the worst case, If this problem cause staking fund to be frozen, the only way is to trash the reward and use `emergencyWithdraw` to withdraw staked fund.

From above statement, we can proof that there exists a case that user reward may be locked forever due to looping too many times causing gas to be used beyond the limit thus transaction always failed.

Tools Used

Reverse engineering using the help of IDE.

Recommended Mitigation Steps

User should be able to supply endEpochIndex to the claim reward functions. And only calculate reward from startIndex to min(auraLocker.epochCount() - 1, endEpochIndex). And also add support for partial reward claiming.

0xMaharishi (Aura Finance) acknowledged, but disagreed with severity and commented:

Valid report although given these are reward tokens and the max amount of entries is one per week, it would take some years for this to run over gas limit, during which time the contract could easily be changed.

LSDan (judge) decreased severity to Medium and commented:

I'm downgrading this to medium severity. It is unreasonable to expect contracts to be future proof to the tune of a hundred years or more, but if the frequency had been unreasonably fast this issue could have kicked in.

0xMaharishi (Aura Finance) resolved and commented:

code-423n4/2022-05-aura#6
code4rena aurafinance/aura-contracts#84

[M-08] Locking up AURA Token does not increase voting power of individual

Submitted by xiaoming90

Per the documentation, AURA tokens can be locked in the AuraLocker to recieve vlAURA. **vlAURA is voting power** in the AURA ecosystem.

It is also possible for the users to delegate their voting power to a specific address by calling the `AuraLocker.delegate(address account)` function.

However, after users locked up their AURA tokens in exchange for vlAURA tokens, their voting power did not increase.

Proof of Concept

The following shows an example of Alice attempting to get some voting power by locking up her AURA tokens, but her voting power did not increase:

1. At this point, Alice has not locked any AURA token into the AuraLocker yet. Thus, when `AuraLocker.getVotes(Alice.address)` is called, it returned "0" (No voting power. This is expected).

2. Alice decided to get some voting power. So, Alice locked 100 AURA tokens by calling the `AuraLocker._lock()` function, and gain 100 vlAURA in return.
3. Alice understand that as per the design, voting power will be 0 after depositing until the next epoch. So, she waited for around 1 week.
4. After a week has passed, the `AuraLocker.getVotes(Alice.address)` is called again. Alice expected it to return"100", but it still returned "0" (Still no voting power).
5. Alice has locked up her AURA tokens for a week and hold 100 vlAURA, yet she has no voting power.

The following snippet of test script demonstrates the above issue, showing that the vote power remains the same after locking up the AURA tokens for a week.

```
it("(Debug) allows users to lock aura", async () => {
    const cvxBalance = await phase4.cvx.balanceOf(stakerAddress);
    const lockBefore = await phase4.cvxLocker.lockedBalances(stakerAd
    console.log("(Debug) User Locked Balance Record = Total %s CVX (U

    console.log("(Debug) User is going to lock %s CVX", cvxBalance)
    await phase4.cvx.connect(staker.signer).approve(phase4.cvxLocker.
    await phase4.cvxLocker.connect(staker.signer).lock(stakerAddress,

    const lockAfter = await phase4.cvxLocker.lockedBalances(stakerAdd
    console.log("(Debug) User Locked Balance Record = Total %s CVX (U

    expect(lockAfter.locked.sub(lockBefore.locked)).eq(cvxBalance);
});
it("(Debug) check user has votes after locking", async () => {
    const votesBefore = await phase4.cvxLocker.getVotes(stakerAddress
    const lock = await phase4.cvxLocker.lockedBalances(stakerAddress)
    console.log("(Debug) votesBefore = %s, locked CVX = %s", votesBef
    console.log("(Debug) Properly locked tokens as of the most recent

    await increaseTime(ONE_WEEK);
    console.log("After 1 week")

    const votesAfter = await phase4.cvxLocker.getVotes(stakerAddress)
    console.log("(Debug) votesAfter = %s, locked CVX = %s", votesBefo
    console.log("(Debug) Properly locked tokens as of the most recent
```

```
        expect(votesAfter.sub(votesBefore)).eq(lock.locked);

    });
    it("(Debug) check user lock balance and votes after 20 weeks", async
        const TWENTY_WEEKS = BN.from(60 * 60 * 24 * 7 * 20);
        await increaseTime(TWENTY_WEEKS);
        console.log("(Debug) After 20 weeks")

        const lockAfter20 = await phase4.cvxLocker.lockedBalances(stakerA
        console.log("(Debug) User Locked Balance = Total %s CVX (Unlockab
        console.log("(Debug) Properly locked tokens as of the most recent

        expect(lockAfter20.unlockable).eq(lockAfter20.total); // all lock
    });
```

Following is the output of the test script.

1. The first section shows that user has 800563688188805506352 vlAURA after locking up their AURA tokens
2. The second section shows that after a week, the user has 0 voting power even though the user has `800557536376417310407` vlAURA tokens. Note that these vlAURA tokens are all properly locked tokens that have not been expired.

(Note: vlAURA == vlCVX and AURA == CVX in this context)

```
          aura locker
    (Debug) User Locked Balance Record = Total 0 CVX (Unlockable = 0 CVX,
    (Debug) User is going to lock 800563688188805506352 CVX
    (Debug) User Locked Balance Record = Total 800563688188805506352 CVX
            ✓ (Debug) allows users to lock aura

    (Debug) votesBefore = 0, locked CVX = 800563688188805506352
    (Debug) Properly locked tokens as of the most recent eligible epoch =
    After 1 week
    (Debug) votesAfter = 0, locked CVX = 800563688188805506352
    (Debug) Properly locked tokens as of the most recent eligible epoch =
            1) (Debug) check user has votes after locking

    (Debug) After 20 weeks
    (Debug) User Locked Balance = Total 800563688188805506352 CVX (Unlock
```

```
(Debug) Properly locked tokens as of the most recent eligible epoch =
          ✓ (Debug) check user lock balance and votes after 20 weeks
```

Aura Finance has implemented a checkpointing mechanism for determine user's voting power. Therefore, accounting for the votes will only happen during checkpoint when `AuraLocker.checkpointDelegate()` function is being called. Therefore, the `AuraLocker.getVotes()` function will only consider the locked AURA tokens that have been "checkpointed" as votes. In other words, if the locked AURA tokens have not been "checkpointed" yet, it will simply remain as a balance in the AuraLocker contract, and the user's locked AURA tokens effectively have no voting power.

Based on the source code, the root cause of this issue is that if a user does not have a delegatee, the system will not perform any checkpointing, and user's locked AURA token will not be accounted as voting power.

Following code from `AuraLocker._lock()` shows that checkpointing will only be performed if the user has a delegatee. Otherwise, no checkpointing will be performed when users locked their AURA tokens.

```
function _lock(address _account, uint256 _amount) internal {
    ..SNIP..
    address delegatee = delegates(_account);
    if (delegatee != address(0)) {
        delegateeUnlocks[delegatee][unlockTime] += lockAmount;
        _checkpointDelegate(delegatee, lockAmount, 0);
    }
    // @audit - No checkpointing performed for the rest of the code i
    ..SNIP..
}
```

The only way for Alice could get back her voting power is to delegate to herself after locking her AURA tokens. This is a workaround. `AuraLocker.delegate()` sole purpose should only serve to delegate one's voting power to another user, and should not be used as a workaround to force the system to perform checkpointing to gain voting power.

For Alice to get back her voting power, she must call the `AuraLocker.delegate(Alice.address)` function, which will delegate to herself. This function will in turn call the `AuraLocker._checkpointDelegate()` function, which will "checkpointed" Alice's locked tokens to become votes. Only after this step, Alice's voting power will be updated and calling `AuraLocker.getVotes(Alice.address)` should return "100" now.

Additionally, documentation did not mention that a user is required to delegate to oneself in order to get the voting power. Thus, it is very likely that majority of the users would not know how to get their voting power unless they review the source code or is aware of this workaround.

Impact

The impact of this issue is that users might miss the opportunity to vote on critical protocol decisions or flow of incentives (Gauge voting) due to lack of voting power as voting power is not assigned to them after locking up AURA tokens.

If the users only realised this issue in the current epoch, they would miss the chance to vote in current epoch. This is because by calling the `AuraLocker.delegate(address account)` function to fix the issue, the votes will only be effective in the next epoch.

The outcome of the governance or gauge voting might be impacted and might not reflect the true consensus of the community as affected users are not able to participate in the vote or have inaccurate voting power, thus affecting the protocol.

Recommended Mitigation Steps

In Convex Finance, users lock their CVX tokens by calling `CvxLocker._lock()` function and voting power will be allocated to the users immediately. Similar strategy should be adopted.

It is recommended to update the `AuraLocker._lock()` function so that the user's locked AURA tokens are "checkpointed" and converted to voting power immediately after locking up if a user has not assigned a delegatee yet. This will trigger the accounting for votes and translate the newly locked tokens into voting power immediately.

Original Code

```
function _lock(address _account, uint256 _amount) internal {
    ..SNIP..
    address delegatee = delegates(_account);
    if (delegatee != address(0)) {
        delegateeUnlocks[delegatee][unlockTime] += lockAmount;
        _checkpointDelegate(delegatee, lockAmount, 0);
    }
    ..SNIP..
}
```

Suggested Modification

```
function _lock(address _account, uint256 _amount) internal {
    ..SNIP..
    address delegatee = delegates(_account);
    if (delegatee != address(0)) {
        delegateeUnlocks[delegatee][unlockTime] += lockAmount;
        _checkpointDelegate(delegatee, lockAmount, 0);
    } else {
        // If there is no delegatee,
        // then automatically delegate to the account to trigger the
        delegateeUnlocks[_account][unlockTime] += lockAmount;
        _checkpointDelegate(_account, lockAmount, 0);
    }
    ..SNIP..
}
```

0xMaharishi (Aura Finance) disputed and commented:

Users must simply delegate to themselves to receive voting power

LSDan (judge) commented:

Valid issue. Fix the documentation or the code. If all users need to do is delegate to themselves, then auto-delegating newly minted votes to the user would solve the issue.

[M-09] Reward can be vested even after endTime

Submitted by csanuragjain

AuraVestedEscrow.sol#L96

Reward vesting should end once endTime is reached, this is not done currently.

Proof of Concept

1. Observe the fund function
2. Observe that there is no check to disallow funding once endTime has been reached

Recommended Mitigation Steps

Add below check

```
require(block.timestamp<=endTime, "Reward vesting period over");
```

0xMaharishi (Aura Finance) acknowledged, but disagreed with severity and commented:

This report is kind of invalid, because there would be no utility in this.. it's specifically left open ended. With that being said, adding a check to ensure that funding is made BEFORE START TIME would be good.

This should be a 0 or 1 at most.

LSDan (judge) commented:

As far as I can tell, this is totally valid. Funding in this state would cause a loss of funds in that they would never go towards a reward.

0xMaharishi (Aura Finance) resolved:

code4rena aurafinance/aura-contracts#84
All code4rena fixes code-423n4/2022-05-aura#6

[M-10] Increase voting power by tokenizing the address that locks the token

Submitted by Kumpa

AuraLocker.sol#L258-L295

Without restriction on the type of address that `lock` the token, a bad actor could lock the token through the smart contract. Doing so enable him to make the lockedToken becomes liquidate by tokenize his smart contract which defeat the purpose of the lockedToken that is supposed to be untransferable. Moreover, a bad actor could attract people to lock the token through his smart contract instead of directly locking with AuraLocker by injecting better short-term incentives to his wrapper token. This enable the bad actor to accumulate voting power that could dictate the future of the protocol.

Proof of Concept

- A bad actor creates a smart contract
- A contract calls `lock` in AuraLocker and locks the token
- A bad actor tokenizes the contract
- A bad actor attracts people to lock the token through his smart contract by offering a wrapper tokens or additional incentives like high apy etc.
- A bad actor dictates the smart contract to delegate its vote to his preferred address.

Recommended Mitigation Steps

It would be best to check whether the locker is the smart contract or the wallet and, if the protocol wants the smart contract to be the locker, it can implement the whitelist or blacklist.

0xMaharishi (Aura Finance) acknowledged, but disagreed with severity and commented:

Given no user funds are at risk I don't think this is a super high risk, but I do agree that there is a governance risk there and it's something to be concerned about if/when there is no multisig intermediary between aura voters and execution. With

that being said, I think a good solution would be to have a blacklist that the owner can set to block non-eoa's from making any further locks: `bool canLock = isEOA(address) || !isBlacklisted(address)`

LSDan (judge) decreased severity to Medium and commented:

I'll leave this in place as a medium risk because there are external factors involved. High risk is too severe.

0xMaharishi (Aura Finance) resolved:

code-423n4/2022-05-aura#6
code4rena aurafinance/aura-contracts#84

[M-11] Users may lose rewards to other users if rewards are given as fee-on-transfer tokens

Submitted by lllllll, also found by Ait, BowTiedWardens, and MaratCerby

If rewards are given in fee-on-transfer tokens, users may get no rewards, breaking functionality.

`Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or :::leak value with a hypothetical attack path with stated assumptions:::, but external requirements.` (emphasis mine)

The underlying BAL protocol support fee-on-transfer tokens, so should Aura.

Proof of Concept

File: contracts/ExtraRewardsDistributor.sol #1

```
87      function _addReward(
88          address _token,
89          uint256 _amount,
90          uint256 _epoch
91      ) internal nonReentrant {
92          // Pull before reward accrual
93          IERC20(_token).safeTransferFrom(msg.sender, address(this
```

```
94
95            //convert to reward per token
96            uint256 supply = auraLocker.totalSupplyAtEpoch(_epoch);
97            uint256 rPerT = (_amount * 1e20) / supply;
98            rewardData[_token][_epoch] += rPerT;
```

[ExtraRewardsDistributor.sol#L87-L98]

If a fee is charged the total amount available to be transferred later will be less than the `_amount` passed in.

Consider the following scenario:
User A holds 98% of the total supply of vlBAL (the system is being bootstrapped)
User B holds 1%
User C holds 1%

1. `_token` is given out as a reward. It is a fee-on-transfer token with a fee of 2%
2. Nobody claims the reward until it's fully available (to save gas on transaction fees)
3. User A is the first to claim his/her reward and gets 98% of the reward, leaving 0 wei of the token left (since the other 2% was already taken as a fee by the token itself)
4. User B tries to claim and the call reverts since there's no balance left
5. User C tries to claim and the call reverts for them too
6. Users B and C are angry and stop using Aura

```
File: contracts/ExtraRewardsDistributor.sol   #2

87        function _addReward(
88            address _token,
89            uint256 _amount,
90            uint256 _epoch
91        ) internal nonReentrant {
92            // Pull before reward accrual
93            IERC20(_token).safeTransferFrom(msg.sender, address(this
94
```

```
95            //convert to reward per token
96            uint256 supply = auraLocker.totalSupplyAtEpoch(_epoch);
97            uint256 rPerT = (_amount * 1e20) / supply;
98            rewardData[_token][_epoch] += rPerT;
```

ExtraRewardsDistributor.sol#L87-L98

Recommended Mitigation Steps

Measure the contract balance before and after the transfer, and use the difference as the amount, rather than the stated amount.

0xMaharishi (Aura Finance) disputed and commented:

This contract is optional to use. It is not supposed to support fee bearing tokens.

LSDan (judge) commented:

See my comment on issue #18: “There are several cases in the code reported where the token in question comes from an external (non-admin, non-protocol) source. One of these is the addReward functionality (ExtraRewards). This would indeed cause an accounting issue and allow a potential malicious actor to send rewards which cause distribution to fail due to lack of funds. Just because you don’t plan to use fee on transfer tokens, does not mean they will not be used. This should be protected against in the scenarios where it could cause an issue.

That said, this clearly requires external factors and relies on hypothetical attack motivation that seems unlikely to me. I think it should be included as a medium risk.”

[M-12] User will lose funds

Submitted by csanuragjain, also found by hyh and kirk-baird

AuraClaimZap.sol#L224-L226

It was observed that User will lose funds due to missing else condition.

Proof of Concept

1. User call claimRewards at ClaimZap.sol#L103 with Options.LockCvx as false
2. claimRewards internally calls _claimExtras
3. Everything goes good until AuraClaimZap.sol#L218

```
if (depositCvxMaxAmount > 0) {
            uint256 cvxBalance = IERC20(cvx).balanceOf(msg.sender).sul
            cvxBalance = AuraMath.min(cvxBalance, depositCvxMaxAmount
            if (cvxBalance > 0) {
                //pull cvx
                IERC20(cvx).safeTransferFrom(msg.sender, address(this
                if (_checkOption(options, uint256(Options.LockCvx))) 
                    IAuraLocker(locker).lock(msg.sender, cvxBalance);
                }
            }
        }
```

4. Since user cvxBalance>0 so cvxBalance is transferred from user to the contract.
5. Now since Options.LockCvx was set to false in options so if (_checkOption(options, uint256(Options.LockCvx))) does not evaluate to true and does not execute
6. This means User cvx funds are stuck in contract

Recommended Mitigation Steps

The condition should check if user has enabled lock for cvx, otherwise cvx should not be transferred from user

```
if (depositCvxMaxAmount > 0 && _checkOption(options, uint256(Options.
          uint256 cvxBalance = IERC20(cvx).balanceOf(msg.sender).sub(
          cvxBalance = AuraMath.min(cvxBalance, depositCvxMaxAmount);
          if (cvxBalance > 0) {
              //pull cvx
              IERC20(cvx).safeTransferFrom(msg.sender, address(this),

                  IAuraLocker(locker).lock(msg.sender, cvxBalance);
          }
      }
```

0xMaharishi (Aura Finance) confirmed, but disagreed with severity and commented:

This is valid, although it:

- relies on user function input
- does not affect user deposits
- requires pre-approval of tokens

Therefore, I don't think this should be a 3 severity. 2 at most.

LSDan (judge) decreased severity to Medium and commented:

This is a tough one, but I agree that medium severity makes more sense here since we're talking about a user acting on their own behalf in a very specific way. This does not open up an attack vector which would allow a malicious actor to lock a user's funds.

0xMaharishi (Aura Finance) resolved:

code4rena aurafinance/aura-contracts#84
All code4rena fixes code-423n4/2022-05-aura#6

[M-13] `ConvexMasterChef` : When `_lpToken` is cvx, reward calculation is incorrect

Submitted by cccz

In the ConvexMasterChef contract, a new staking pool can be added using the add() function. The staking token for the new pool is defined using the _lpToken variable. However, there is no additional checking whether the _lpToken is the same as the reward token (cvx) or not.

```
    function add(
        uint256 _allocPoint,
        IERC20 _lpToken,
```

```
        IRewarder _rewarder,
        bool _withUpdate
    ) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardBlock = block.number > startBlock
            ? block.number
            : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(
            PoolInfo({
                lpToken: _lpToken,
                allocPoint: _allocPoint,
                lastRewardBlock: lastRewardBlock,
                accCvxPerShare: 0,
                rewarder: _rewarder
            })
        );
    }
```

When the _lpToken is the same token as cvx, reward calculation for that pool in the updatePool() function can be incorrect. This is because the current balance of the _lpToken in the contract is used in the calculation of the reward. Since the _lpToken is the same token as the reward, the reward minted to the contract will inflate the value of lpSupply, causing the reward of that pool to be less than what it should be.

```
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return;
        }
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (lpSupply == 0) {
            pool.lastRewardBlock = block.number;
            return;
        }
        uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.
        uint256 cvxReward = multiplier
            .mul(rewardPerBlock)
            .mul(pool.allocPoint)
```

```
            .div(totalAllocPoint);
        //cvx.mint(address(this), cvxReward);
        pool.accCvxPerShare = pool.accCvxPerShare.add(
            cvxReward.mul(1e12).div(lpSupply)
        );
        pool.lastRewardBlock = block.number;
    }
```

Proof of Concept

ConvexMasterChef.sol#L96-L118
ConvexMasterChef.sol#L186-L206

Recommended Mitigation Steps

Add a check that _lpToken is not cvx in the add function or mint the reward token to another contract to prevent the amount of the staked token from being mixed up with the reward token.

0xMaharishi (Aura Finance) commented:

Could potentially require not to be the reward token. but I think this is just a relevant part of dao ownership.

0xMaharishi (Aura Finance) confirmed and resolved

[M-14] Integer overflow will lock all rewards in `AuraLocker`

Submitted by kirk-baird

AuraLocker.sol#L176-L177
AuraLocker.sol#L802-L814
AuraLocker.sol#L864

There is a potential overflow in the rewards calculations which would lead to `updateReward()` always reverting.

The impact of this overflow is that all reward tokens will be permanently locked in the contract. User's will be unable to call any of the functions which have the

`updateReward()` modifier, that is:

- `lock()`
- `getReward()`
- `_processExpiredLocks()`
- `_notifyReward()`

As a result the contract will need to call `shutdown()` and the users will only be able to receive their staked tokens via `emergencyWithdraw()`, which does not transfer the users the reward tokens.

Note that if one reward token overflows this will cause a revert on all reward tokens due to the loop over reward tokens.

This issue will always be present if the staked token is one with a low number of decimal places such as USDC or USDT which have 6 decimal places. This is because the `totalSupply` will be limited in size by the decimal places of the `stakingToken`.

Proof of Concept

The overflow may occur due to the base of values in `_rewardPerToken()`.

```
function _rewardPerToken(address _rewardsToken) internal view ret
    if (lockedSupply == 0) {
        return rewardData[_rewardsToken].rewardPerTokenStored;
    }
    return
        uint256(rewardData[_rewardsToken].rewardPerTokenStored).a
            _lastTimeRewardApplicable(rewardData[_rewardsToken].p
                .sub(rewardData[_rewardsToken].lastUpdateTime)
                .mul(rewardData[_rewardsToken].rewardRate)
                .mul(1e18)
                .div(lockedSupply)
        );
}
```

The return value of `_rewardPerToken()` is in terms of

```
(now - lastUpdateTime) * rewardRate * 10**18 / totalSupply
```

Here `(now - lastUpdateTime)` has a maximum value of `rewardDuration = 6 * 10**5`.

Now `rewardRate` is the `_reward.div(rewardsDuration)` as seen in `_notifyRewardAmount()` on line #864. Note that `rewardDuration` is a constant 604,800.

```
rewardDuration = 6 * 10**5
```

Thus, if we have a rewards such as AURA or WETH (or most ERC20 tokens) which have units 10**18 we can transfer 1 WETH to the reward distributor which calls `_notifyRewardAmount()` and sets the reward rate to,

```
rewardRate = 10**18 / (6 * 10**5) ~= 10**12
```

Finally, if this attack is run either by the first depositor they may `lock()` a single token which would set `totalSupply = 1`.

Therefore our equation in terms of units will become,

```
(now - lastUpdateTime) * rewardRate * 10**18 / totalSupply => 10**5 *
```

In since `rewardPerTokenStored` is a `uint96` it has a maximum value of `2**96 ~= 7.9 * 10**28`. Hence there will be an overflow in `newRewardPerToken.to96()`. Since we are unable to add more total supply due to `lock()` reverting there will be no way to circumvent this revert except to `shutdown()`.

```
uint256 newRewardPerToken = _rewardPerToken(token);
rewardData[token].rewardPerTokenStored = newRewardPer
```

Note this attack is described when we have a low `totalSupply`. However it is also possible to apply this attack on a larger `totalSupply` when there are reward tokens which have decimal places larger than 18 or tokens which such as SHIB which have small token value and so many of the tokens can be bought for cheap.

Recommended Mitigation Steps

To mitigate this issue it is recommended to increase the size of the `rewardPerTokenStored`. Since updating this value will require another slot to be used we recommend updating this to either `uint256` or to update both `rewardRate` and `rewardPerTokenStored` to be `uint224`.

0xMaharishi (Aura Finance) confirmed, but disagreed with severity and commented:

Given that the staked token will have 18 decimals (it's the aura token) and there will be at least 1e21 units in there before any rewards come, it would take a number of tokens equal to 7.9e49 to be distributed to get this overflow.

I think that while this is certainly a possibility, it would take an orchestrated governance attack and wouldn't necessarily put any funds at risk. That said, a solid mitigation would be to enforce `rewardRate < 1e17` in the notifyRewardAmount, therefore it would never be possible for this to happen.

IMO this should be a medium risk.

LSDan (judge) decreased severity to Medium and commented:

Agree with sponsor about the downgrade to medium. This requires external factors to be an issue, including potential governance collusion in the attack.

0xMaharishi (Aura Finance) resolved:

code-423n4/2022-05-aura#6
code4rena aurafinance/aura-contracts#84

[M-15] `ConvexMasterChef : safeRewardTransfer` can cause loss of funds

Submitted by cccz

Same as https://github.com/code-423n4/2022-02-concur-findings/issues/244

All calculations are rounded down, since a lack of tokens in the contracts cannot be rounding errors' fault. So the function is redundant.
On the other hand, if the contract is undersupplied with cvx tokens, this will cause depositors to be sent less tokens than needed (or none). This is especially unsafe because the tokens that were lacking are not resembled in accountings at all. Thus a depositor may invoke the safeRewardTransfer and not receive tokens they were supposed to.

Proof of Concept

ConvexMasterChef.sol#L299-L306

Recommended Mitigation Steps

Use usual safeTransfer instead of safeRewardTransfer.

0xMaharishi (Aura Finance) acknowledged, but disagreed with severity and commented:

Reward tokens are transferred here before rewards start.

LSDan (judge) commented:

I agree with this report. The fallback situation in this function specifically prioritizes loss of funds over bricking the contract, which while laudable, results in what is effectively a silent failure case.

0xMaharishi (Aura Finance) resolved:

code-423n4/2022-05-aura#6

[M-16] DDOS in `BalLiquidityProvider`

Submitted by QuantumBrief

DDOS to liquidity providers in BalLiquidityProvider.

Proof of Concept

- bal is equal to the contract's balance of the asset: BalLiquidityProvider.sol#L56
- bal is required to be equal to the input parameter _request.maxAmountsIn[i]: BalLiquidityProvider.sol#L57

An attacker can front-run liquidity providers by sending 1 Wei of the asset to make the balance not equal to the input. This can be repeated and be used to impede the liquidity provider from using the function which will always revert since bal != _request.maxAmountsIn[i]

Recommended Mitigation Steps

Balances shouldn't be required to be equal to an input variable. An attacker can always make the balance a little bigger. This check should be removed or changed to require (bal >= _request.maxAmountsIn[i]).

0xMaharishi (Aura Finance) acknowledged, but disagreed with severity and commented:

Fair report 👍

0xMaharishi (Aura Finance) resolved:

code-423n4/2022-05-aura#6
code4rena aurafinance/aura-contracts#84

[M-17] `ConvexMasterChef` 's deposit and withdraw can be reentered drawing all reward funds from the contract if reward token allows for transfer flow control

Submitted by hyh

Reward token accounting update in deposit() and withdraw() happens after reward transfer. If reward token allows for the control of transfer call flow or can be

upgraded to allow it in the future (i.e. have or can introduce the _beforetokentransfer, _afterTokenTransfer type of hooks; or, say, can be upgraded to ERC777), the current implementation makes it possible to drain all the reward token funds of the contract by directly reentering deposit() or withdraw() with tiny _amount.

Setting the severity to medium as this is conditional to transfer flow control assumption, but the impact is the full loss of contract reward token holdings.

Proof of Concept

Both withdraw() and deposit() have the issue, performing late accounting update and not controlling for reentrancy:

ConvexMasterChef.sol#L209-L221

```
function deposit(uint256 _pid, uint256 _amount) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    updatePool(_pid);
    if (user.amount > 0) {
        uint256 pending = user
            .amount
            .mul(pool.accCvxPerShare)
            .div(1e12)
            .sub(user.rewardDebt);
        safeRewardTransfer(msg.sender, pending);
    }
    pool.lpToken.safeTransferFrom(
```

ConvexMasterChef.sol#L239-L250

```
function withdraw(uint256 _pid, uint256 _amount) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    require(user.amount >= _amount, "withdraw: not good");
    updatePool(_pid);
    uint256 pending = user.amount.mul(pool.accCvxPerShare).div(1e
        user.rewardDebt
```

```
        );
        safeRewardTransfer(msg.sender, pending);
        user.amount = user.amount.sub(_amount);
        user.rewardDebt = user.amount.mul(pool.accCvxPerShare).div(1e
        pool.lpToken.safeTransfer(address(msg.sender), _amount);
```

Recommended Mitigation Steps

Consider adding a direct reentrancy control, e.g. nonReentrant modifier:

https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard

Also, consider finishing all internal state updates prior to external calls:

https://consensys.github.io/smart-contract-best-practices/attacks/reentrancy/#pitfalls-in-reentrancy-solutions

0xMaharishi (Aura Finance) confirmed and commented:

Protected by governance, but agree could be solved with simple reentrancy guard.

0xMaharishi (Aura Finance) resolved:

code-423n4/2022-05-aura#6
code4rena aurafinance/aura-contracts#84

[M-18] `AuraBalRewardPool` charges a penalty to all users in the pool if the `AuraLocker` has been shut down

Submitted by lllllll, also found by csanuragjain

Users are charged the penalty due to admin actions, and they have no way to avoid it

Proof of Concept

When claiming their rewards, users are charged a penalty if they take the reward directly, rather than by passing it into the `auraLocker`. Those are the only two

options:

```
File: contracts/AuraBalRewardPool.sol   #1

176       function getReward(bool _lock) public updateReward(msg.send
177           uint256 reward = rewards[msg.sender];
178           if (reward > 0) {
179               rewards[msg.sender] = 0;
180               if (_lock) {
181                   auraLocker.lock(msg.sender, reward);
182               } else {
183                   uint256 penalty = (reward * 2) / 10;
184                   pendingPenalty += penalty;
185                   rewardToken.safeTransfer(msg.sender, reward - p
186               }
```

AuraBalRewardPool.sol#L176-L186

If the pool has been shut down, the `auraLocker.lock()` call will always revert, which means the user must take the penalty path:

```
File: contracts/AuraLocker.sol   #2

258       function _lock(address _account, uint256 _amount) internal
259           require(_amount > 0, "Cannot stake 0");
260           require(!isShutdown, "shutdown");
```

AuraLocker.sol#L258-L260

Recommended Mitigation Steps

Don't charge the penalty if the locker has been shut down.

0xMaharishi (Aura Finance) acknowledged, but disagreed with severity and commented:

The auraBAL reward pool only runs for 2 weeks at the beginning of the protocol. It's highly unlikely the AuraLocker will be shut down.

[M-19] `CrvDepositor.sol` Wrong implementation of the 2-week buffer for lock

Submitted by WatchPug

[CrvDepositor.sol#L127-L134]

```
    uint256 unlockAt = block.timestamp + MAXTIME;
    uint256 unlockInWeeks = (unlockAt/WEEK)*WEEK;

    //increase time too if over 2 week buffer
    if(unlockInWeeks.sub(unlockTime) > 2){
        IStaker(staker).increaseTime(unlockAt);
        unlockTime = unlockInWeeks;
    }
```

In `_lockCurve()`, `unlockInWeeks - unlockTime` is being used as a number in weeks, while it actually is a number in seconds.

Thus, comparing it with `2` actually means a 2 seconds buffer instead of a 2 weeks buffer.

The intention is to wait for 2 weeks before extending the lock time again, but the current implementation allows the extension of the lock once a new week begins.

Recommended Mitigation Steps

Consider changing the name of `unlockTime` to `unlockTimeInWeeks`, and:

1. Change L94-102 to:

[CrvDepositor.sol#L94-L102]

```
    uint256 unlockAt = block.timestamp + MAXTIME;
    uint256 unlockInWeeks = unlockAt / WEEK;

    //release old lock if exists
    IStaker(staker).release();
```

```
//create new lock
uint256 crvBalanceStaker = IERC20(crvBpt).balanceOf(staker);
IStaker(staker).createLock(crvBalanceStaker, unlockAt);
unlockTimeInWeeks = unlockInWeeks;
```

2. Change L127-L134 to:

```
uint256 unlockAt = block.timestamp + MAXTIME;
uint256 unlockInWeeks = unlockAt / WEEK;

//increase time too if over 2 week buffer
if(unlockInWeeks.sub(unlockTime) > 2){
    IStaker(staker).increaseTime(unlockAt);
    unlockTimeInWeeks = unlockInWeeks;
}
```

0xMaharishi (Aura Finance) confirmed and resolved:

code-423n4/2022-05-aura#6

[M-20] `massUpdatePools()` is susceptible to DoS with block gas limit

Submitted by catchup

ConvexMasterChef.sol#L178-L183

massUpdatePools() is a public function and it calls the updatePool() function for the length of poolInfo. Hence, it is an unbounded loop, depending on the length of poolInfo.
If poolInfo.length is big enough, block gas limit may be hit.

Proof of Concept

https://consensys.github.io/smart-contract-best-practices/attacks/denial-of-service/#dos-with-block-gas-limit

Recommended Mitigation Steps

I suggest to limit the max number of loop iterations to prevent hitting block gas limit.

0xMaharishi (Aura Finance) disagreed with severity and commented:

Duplicate of #147

LSDan (judge) commented:

This is not a duplicate of Duplicate of #147 and is also clearly documented as a potential issue in the code itself. If the admin were to accidentally add too many pools the contract would be affected, but the likelihood of this is low and if it were to happen, the admin could still turn off the pools and migrate to another contract. This would, however, affect the protocol in a severely negative way. Not fully updating all of the pools would potentially cause accounting issue and lead to loss of earned rewards. Given the impact and likelihood together, I think medium is actually reasonable in this case.

IIIIIIII000 (warden) commented:

@LSDan- The `massUpdatePool()` function was found to be non-critical in previous contests (https://github.com/code-423n4/2022-02-concur-findings/issues/161) and when I filed the issue with Convex for their bug bounty, they rejected it saying it was a "non-issue" and didn't meet their criteria for a bounty. Furthermore, ConvexMasterChef.sol is not listed as in scope for this contest: https://github.com/code-423n4/2022-05-aura#contracts-of-interest

dmitriia (warden) commented:

@IIIIIIII000- Actually the scope was all non-test contracts, https://github.com/code-423n4/2022-05-aura#repo

LSDan (judge) commented:

@IIIIIIII000- Unlike the other contest, `massUpdatePools()` is used in this contract. I'm going to keep this as medium.

0xMaharishi (Aura Finance) confirmed and resolved

[M-21] `ConvexMasterChef` : When using `add()` and `set()` , it should always call `massUpdatePools()` to update all pools

Submitted by cccz

Same as IDX-003 in https://public-stg.inspex.co/report/Inspex_AUDIT2021024_LuckyLion_Farm_FullReport_v2.0.pdf
The totalAllocPoint variable is used to determine the portion that each pool would get from the total reward, so it is one of the main factors used in the rewards calculation. Therefore, whenever the totalAllocPoint variable is modified without updating the pending reward first, the reward of each pool will be incorrectly calculated.
For example, when _withUpdate is false, in the add() shown below, the totalAllocPoint variable will be modified without updating the rewards (massUpdatePools()).

```
function add(
    uint256 _allocPoint,
    IERC20 _lpToken,
    IRewarder _rewarder,
    bool _withUpdate
) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    uint256 lastRewardBlock = block.number > startBlock
        ? block.number
        : startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolInfo.push(
        PoolInfo({
            lpToken: _lpToken,
            allocPoint: _allocPoint,
            lastRewardBlock: lastRewardBlock,
            accCvxPerShare: 0,
            rewarder: _rewarder
        })
    );
```

```
        }
```

Proof of Concept

ConvexMasterChef.sol#L96-L138

Recommended Mitigation Steps

Removing the _withUpdate variable in the add() and set() functions and always calling the massUpdatePools() function before updating totalAllocPoint variable.

0xMaharishi (Aura Finance) confirmed and commented:

We didn't change this from the Convex implementation. I believe it is there to protect the contract from bricking in case there are too many pools added. The choice here is between giving admin the ability to brick, and giving admin the responsibility of adding the correct alloc points. I think we should remove as advised, because we are only likely to have a few pools.

0xMaharishi (Aura Finance) resolved:

code4rena aurafinance/aura-contracts#84
All code4rena fixes code-423n4/2022-05-aura#6

[M-22] Duplicate LP token could lead to incorrect reward distribution

Submitted by csanuragjain, also found by cccz

ConvexMasterChef.sol#L96

It was observed that add function is not checking for duplicate lpToken which allows 2 or more pools to have exact same lpToken. This can cause issue with reward distribution

In case of duplicate lpToken, lpSupply will become incorrect (ConvexMasterChef.sol#L160), hence rewards will be calculated incorrectly

Proof of Concept

1. Owner call add function and uses lpToken as A
2. Owner again call add function and mistakenly provides lpToken as A
3. Now 2 pools will be created with lpToken as A
4. This becomes a problem while reward calculation or updatePool function which uses pool.lpToken.balanceOf(address(this)). Since both pool have same lpToken so lpSupply will be calculated as same which is wrong. Since lpSupply defines the rewardRate so this directly impact reward calculation

Recommended Mitigation Steps

Add a global variable keeping track of all lpToken added for pool. In case of duplicate lpToken add function should fail.

0xMaharishi (Aura Finance) acknowledged, but disagreed with severity and commented:

Given the result of this would be a net negative to everyone (due to overall increased lp token supply) there doesn't seem to be any incentive for anyone to do this. Considering that the owner is a distributed 4 of 7 multisig, i think it is an acceptable scenario.

LSDan (judge) commented:

I'm going to let this one stand. Multisigs make mistakes and it would be trivial to prevent this one.

Low Risk and Non-Critical Issues

For this contest, 76 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **IIIIIII** received the top score from the judge.

The following wardens also submitted reports: MaratCerby, reassor, BowTiedWardens, TerrierLover, SmartSek, 0x4non, 0xNazgul, hyh, robee, tintin, catchup, defsec, Hawkeye, joestakey, _Adam, 0x1f8b, fatherOfBlocks, Funen, berndartmueller, cryptphi, hansfriese, kenta, Nethermind, PPrieditis,

QuantumBrief, Rolezn, sorrynotsorry, 0xf15ers, bobirichman, BouSalman, c3phas, cccz, cthulhu_cult, FSchmoede, Kaiziron, kenzo, mics, MiloTruck, p_crypt0, Ruhum, sseefried, Tadashi, unforgiven, WatchPug, 0xkatana, CertoraInc, csanuragjain, delfin454000, ellahi, GimelSec, JC, Kthere, sashik_eth, sikorico, simon135, Waze, oyc_109, 242, 0xNineDec, AlleyCat, asutorufos, ch13fd357r0y3r, Chom, jayjonah8, JDeryl, kirk-baird, NoamYakov, sach1r0, samruna, SooYa, z3s, hubble, Cityscape, Kumpa, *and* zmj.

Summary

Low Risk Issues

	Issue	Instances
L-01	Wrong amounts sent if arrays don't match	1
L-02	Incorrect/misleading NatSpec	1
L-03	Function reverts if called a second time	1
L-04	`pragma experimental ABIEncoderV2` is deprecated	1
L-05	`safeApprove()` is deprecated	36
L-06	Missing checks for `address(0x0)` when assigning values to `address` state variables	103

Total: 143 instances over 6 issues

Non-critical Issues

	Issue	Instances
N-01	Unused file	1
N-02	Call `For` / `From` variants instead of copying an pasting code	1
N-03	Remove tautological code	1
N-04	Adding a `return` statement when the function defines a named return variable, is redundant	3
N-05	`override` function arguments that are unused should have the variable name removed or commented out to avoid compiler warnings	1

	Issue	Instances
N-06	`public` functions not called by the contract should be declared `external` instead	18
N-07	`type(uint<n>).max` should be used instead of `uint<n>(-1)`	8
N-08	`constant` s should be defined rather than using magic numbers	47
N-09	Redundant cast	2
N-10	Numeric values having to do with time should use time units for readability	4
N-11	Missing event for critical parameter change	24
N-12	Use a more recent version of solidity	1
N-13	Use a more recent version of solidity	26
N-14	Use a more recent version of solidity	1
N-15	Constant redefined elsewhere	38
N-16	Inconsistent spacing in comments	80
N-17	Non-library/interface files should use fixed compiler versions, not floating ones	12
N-18	Typos	29
N-19	File is missing NatSpec	6
N-20	NatSpec is incomplete	21
N-21	Event is missing `indexed` fields	66

Total: 390 instances over 21 issues

[L-01] Wrong amounts sent if arrays don't match

The caller may make a copy-paste error where they provide all amounts, but miss one of the recipients in the middle of the list they're copying. This will cause all recipients after that mistake to get the wrong amounts, and the function will not revert

There is 1 instance of this issue:

```
File: contracts/AuraVestedEscrow.sol   #1
```

```
96:     function fund(address[] calldata _recipient, uint256[] calld
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraVestedEscrow.sol#L96

[L-02] Incorrect/misleading NatSpec

The function retrieves the number of votes at the end of an *epoch*, not at the end of a block. Furthermore, `blockNumber` is not an actual variable name

There is 1 instance of this issue:

```
File: contracts/AuraLocker.sol   #1

595:      * @dev Retrieve the number of votes for `account` at the en
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L595

[L-03] Function reverts if called a second time

`safeApprove()` reverts if called a second time without fist calling `safeApprove(0)`

There is 1 instance of this issue:

```
File: contracts/CrvDepositorWrapper.sol   #1

/// @audit `setApprovals()` is an external function that calls this fu
51      function _setApprovals() internal {
52          IERC20(WETH).safeApprove(address(BALANCER_VAULT), type(u
53          IERC20(BAL).safeApprove(address(BALANCER_VAULT), type(ui
54:     }
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/CrvDepositorWrapper.sol#L51-L54

[L-04] `pragma experimental ABIEncoderV2` is deprecated

Use `pragma abicoder v2` instead

There is 1 instance of this issue:

```
File: contracts/AuraLocker.sol   #1

3:   pragma experimental ABIEncoderV2;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L3

[L-05] `safeApprove()` is deprecated

Deprecated in favor of `safeIncreaseAllowance()` and `safeDecreaseAllowance()`. If only setting the initial allowance to the value that means infinite, `safeIncreaseAllowance()` can be used instead

There are 36 instances of this issue. For details, see the warden's full report.

[L-06] Missing checks for `address(0x0)` when assigning values to `address` state variables

There are 103 instances of this issue. For details, see the warden's full report.

[N-01] Unused file

The file is never imported by any other file

There is 1 instance of this issue:

```
File: convex-platform/contracts/contracts/interfaces/BoringMath.sol
```

```
0:    // SPDX-License-Identifier: MIT
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/interfaces/BoringMath.sol#L0

[N-02] Call `For` / `From` variants instead of copying an pasting code

Duplicating code can lead to errors when a change is made to only one of the locations

There is 1 instance of this issue:

```
File: contracts/AuraBalRewardPool.sol   #1

/// @audit This function should call `stakeFor(msg.sender, _amount)` :
120       function stake(uint256 _amount) public updateReward(msg.send
121           require(_amount > 0, "RewardPool : Cannot stake 0");
122
123           _totalSupply = _totalSupply.add(_amount);
124           _balances[msg.sender] = _balances[msg.sender].add(_amoun
125
126           stakingToken.safeTransferFrom(msg.sender, address(this),
127           emit Staked(msg.sender, _amount);
128
129           return true;
130:      }
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraBalRewardPool.sol#L120-L130

[N-03] Remove tautological code

There is 1 instance of this issue:

```
File: convex-platform/contracts/contracts/CrvDepositor.sol   #1

/// @audit `_lockIncentive` is always greater than or equal to zero,
75:           if(_lockIncentive >= 0 && _lockIncentive <= 30){
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/CrvDepositor.sol#L75

[N-04] Adding a `return` statement when the function defines a named return variable, is redundant

There are 3 instances of this issue:

```
File: contracts/AuraLocker.sol   #1

678:           return amount;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L678

```
File: contracts/AuraLocker.sol   #2

778:           return userRewards;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L778

```
File: convex-platform/contracts/contracts/VoterProxy.sol   #3

196:           return balance;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/VoterProxy.sol#L196

[N-05] `override` function arguments that are unused should have the variable name removed or commented out to avoid compiler warnings

There is 1 instance of this issue:

```
File: convex-platform/contracts/contracts/BaseRewardPool4626.sol   #1

134:     function maxDeposit(address owner) public view virtual over
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/BaseRewardPool4626.sol#L134

[N-06] `public` functions not called by the contract should be declared `external` instead

Contracts are allowed to override their parents' functions and change the visibility from `external` to `public`.

There are 18 instances of this issue. For details, see the warden's full report.

[N-07] `type(uint<n>).max` should be used instead of `uint<n>(-1)`

There are 8 instances of this issue:

```
File: convex-platform/contracts/contracts/interfaces/BoringMath.sol

25:         require(a <= uint128(-1), "BoringMath: uint128 Overflow

30:         require(a <= uint64(-1), "BoringMath: uint64 Overflow")

35:         require(a <= uint32(-1), "BoringMath: uint32 Overflow")
```

```
40:           require(a <= uint40(-1), "BoringMath: uint40 Overflow")

45:           require(a <= uint112(-1), "BoringMath: uint112 Overflow

50:           require(a <= uint224(-1), "BoringMath: uint224 Overflow

55:           require(a <= uint208(-1), "BoringMath: uint208 Overflow

60:           require(a <= uint216(-1), "BoringMath: uint216 Overflow
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/interfaces/BoringMath.sol#L25

[N-08] `constant` s should be defined rather than using magic numbers

There are 47 instances of this issue. For details, see the warden's full report.

[N-09] Redundant cast

The type of the variable is the same as the type to which the variable is being cast

There are 2 instances of this issue:

```
File: contracts/AuraLocker.sol   #1

/// @audit uint256(_epoch)
654:          uint256 epochStart = uint256(epochs[0].date).add(uint25
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L654

```
File: contracts/AuraLocker.sol   #2

/// @audit uint256(_epoch)
```

```
718:          uint256 epochStart = uint256(epochs[0].date).add(uint25
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L718

[N-10] Numeric values having to do with time should use time units for readability

There are units for seconds, minutes, hours, days, and weeks

There are 4 instances of this issue:

```
File: contracts/AuraLocker.sol   #1

/// @audit 86400
81:       uint256 public constant rewardsDuration = 86400 * 7;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L81

```
File: contracts/CrvDepositorWrapper.sol   #2

/// @audit 3600
60:           queries[0].secs = 3600; // last hour
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/CrvDepositorWrapper.sol#L60

```
File: convex-platform/contracts/contracts/CrvDepositor.sol   #3

/// @audit 86400
26:       uint256 private constant MAXTIME = 1 * 364 * 86400;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/CrvDepositor.sol#L26

```
File: convex-platform/contracts/contracts/CrvDepositor.sol   #4

/// @audit 86400
27:       uint256 private constant WEEK = 7 * 86400;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/CrvDepositor.sol#L27

[N-11] Missing event for critical parameter change

There are 24 instances of this issue. For details, see the warden's full report.

[N-12] Use a more recent version of solidity

Use a solidity version of at least 0.8.12 to get `string.concat()` to be used instead of `abi.encodePacked(<str>,<str>)`

There is 1 instance of this issue:

```
File: contracts/AuraMerkleDrop.sol   #1

2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraMerkleDrop.sol#L2

[N-13] Use a more recent version of solidity

Use a solidity version of at least 0.8.13 to get the ability to use `using for` with a list of free functions

There are 26 instances of this issue. For details, see the warden's full report.

[N-14] Use a more recent version of solidity

Use a solidity version of at least 0.8.4 to get `bytes.concat()` instead of `abi.encodePacked(<bytes>,<bytes>)` Use a solidity version of at least 0.8.12 to get `string.concat()` instead of `abi.encodePacked(<str>,<str>)`

There is 1 instance of this issue:

```
File: convex-platform/contracts/contracts/DepositToken.sol   #1

2:    pragma solidity 0.6.12;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/DepositToken.sol#L2

[N-15] Constant redefined elsewhere

Consider defining in only one contract so that values cannot become out of sync when only one location is updated. A cheap way to store constants in a single location is to create an `internal constant` in a `library`. If the variable is a local cache of another contract's value, consider making the cache variable internal or private, which will require external users to query the contract with the source of truth, so that callers don't get out of sync.

There are 38 instances of this issue. For details, see the warden's full report.

[N-16] Inconsistent spacing in comments

Some lines use `// x` and some use `//x`. The instances below point out the usages that don't follow the majority, within each file

There are 80 instances of this issue. For details, see the warden's full report.

[N-17] Non-library/interface files should use fixed compiler versions, not floating ones

There are 12 instances of this issue:

```
File: contracts/AuraClaimZap.sol

2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraClaimZap.sol#L2

```
File: contracts/AuraMinter.sol

2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraMinter.sol#L2

```
File: contracts/ExtraRewardsDistributor.sol

2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/ExtraRewardsDistributor.sol#L2

```
File: contracts/AuraMerkleDrop.sol

2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraMerkleDrop.sol#L2

```
File: contracts/AuraPenaltyForwarder.sol
```

```
2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraPenaltyForwarder.sol#L2

```
File: contracts/AuraBalRewardPool.sol

2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraBalRewardPool.sol#L2

```
File: contracts/AuraLocker.sol

2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L2

```
File: contracts/ClaimFeesHelper.sol

2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/ClaimFeesHelper.sol#L2

```
File: contracts/Aura.sol
```

```
2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/Aura.sol#L2

```
File: contracts/AuraStakingProxy.sol

2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraStakingProxy.sol#L2

```
File: contracts/AuraVestedEscrow.sol

2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraVestedEscrow.sol#L2

```
File: contracts/BalLiquidityProvider.sol

2:    pragma solidity ^0.8.11;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/BalLiquidityProvider.sol#L2

[N-18] Typos

There are 29 instances of this issue. For details, see the warden's full report.

[N-19] File is missing NatSpec

There are 6 instances of this issue:

```
File: contracts/Interfaces.sol
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/Interfaces.sol

```
File: convex-platform/contracts/contracts/Interfaces.sol
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/Interfaces.sol

```
File: convex-platform/contracts/contracts/interfaces/IGaugeController
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/interfaces/IGaugeController.sol

```
File: convex-platform/contracts/contracts/interfaces/IProxyFactory.so
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/interfaces/IProxyFactory.sol

```
File: convex-platform/contracts/contracts/interfaces/IRewardHook.sol
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-

platform/contracts/contracts/interfaces/IRewardHook.sol

```
File: convex-platform/contracts/contracts/interfaces/IRewarder.sol
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/interfaces/IRewarder.sol

[N-20] NatSpec is incomplete

There are 21 instances of this issue. For details, see the warden's full report.

[N-21] Event is missing `indexed` fields

Each `event` should use three `indexed` fields if there are three or more fields

There are 66 instances of this issue. For details, see the warden's full report.

0xMaharishi (Aura Finance) acknowledged

Gas Optimizations

For this contest, 66 reports were submitted by wardens detailing gas optimizations. The report highlighted below by IIIIIII received the top score from the judge.

The following wardens also submitted reports: BowTiedWardens, 0xkatana, Tomio, TerrierLover, defsec, 0xKitsune, c3phas, joestakey, catchup, CertoraInc, hansfriese, kenta, MaratCerby, MiloTruck, robee, sashik_eth, UnusualTurtle, _Adam, 0xf15ers, 0xNazgul, delfin454000, fatherOfBlocks, Kaiziron, simon135, WatchPug, Waze, 0x1f8b, 0x4non, ellahi, reassor, rfa, 0v3rf10w, asutorufos, DavidGialdi, mics, oyc_109, sach1r0, Fitraldys, FSchmoede, Funen, Hawkeye, NoamYakov, Randyyy, samruna, sikorico, antonttc, bobirichman, csanuragjain, cthulhu_cult, GimelSec, hyh, minhquanym, QuantumBrief, SmartSek, SooYa, unforgiven, z3s, jayjonah8, JC, Kthere, marcopaladin, orion, Ruhum, Tadashi, *and* zmj.

Summary

	Issue	Instances
G-01	Remove or replace unused state variables	1
G-02	Multiple `address` mappings can be combined into a single `mapping` of an `address` to a `struct`, where appropriate	8
G-03	State variables only set in the constructor should be declared `immutable`	6
G-04	State variables can be packed into fewer storage slots	3
G-05	Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas	6
G-06	State variables should be cached in stack variables rather than re-reading them from storage	60
G-07	`<x> += <y>` costs more gas than `<x> = <x> + <y>` for state variables	5
G-08	`internal` functions only called once can be inlined to save gas	4
G-09	`<array>.length` should not be looked up in every loop of a `for`-loop	13
G-10	`++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for`- and `while`-loops	13
G-11	`require()` / `revert()` strings longer than 32 bytes cost extra gas	1
G-12	`keccak256()` should only need to be called on a specific string literal once	1
G-13	Not using the named return variables when a function returns, wastes deployment gas	10
G-14	Using `bool`s for storage incurs overhead	18
G-15	Use a more recent version of solidity	28
G-16	Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` statement	23
G-17	It costs more gas to initialize variables to zero than to let the default of zero be applied	26
G-18	`++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i` / `i--` too)	24
G-19	Splitting `require()` statements that use `&&` saves gas	15
G-20	Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead	99

	Issue	Instances
G-21	`abi.encode()` is less efficient than `abi.encodePacked()`	2
G-22	Using `private` rather than `public` for constants, saves gas	30
G-23	Don't compare boolean expressions to boolean literals	9
G-24	Don't use `SafeMath` once the solidity version is 0.8.0 or greater	2
G-25	Duplicated `require()` / `revert()` checks should be refactored to a modifier or function	32
G-26	Multiplication/division by two should use bit shifting	5
G-27	Stack variable used as a cheaper cache for a state variable is only used once	1
G-28	`require()` or `revert()` statements that check input arguments should be at the top of the function	11
G-29	Empty blocks should be removed or emit something	6
G-30	Use custom errors rather than `revert()` / `require()` strings to save deployment gas	101
G-31	Functions guaranteed to revert when called by normal users can be marked `payable`	37
G-32	`public` functions not called by the contract should be declared `external` instead	18

Total: 618 instances over 32 issues

[G-01] Remove or replace unused state variables

Saves a storage slot. If the variable is assigned a non-zero value, saves Gsset (20000 gas). If it's assigned a zero value, saves Gsreset (2900 gas). If the variable remains unassigned, there is no gas savings unless the variable is `public`, in which case the compiler-generated non-payable getter deployment cost is saved. If the state variable is overriding an interface's public function, mark the variable as `constant` or `immutable` so that it does not use a storage slot

There is 1 instance of this issue:

```
File: convex-platform/contracts/contracts/RewardFactory.sol   #1
```

```
28:       mapping(address => uint256[]) public rewardActiveList;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/RewardFactory.sol#L28

[G-02] Multiple `address` mappings can be combined into a single `mapping` of an `address` to a `struct`, where appropriate

Saves a storage slot for the mapping. Depending on the circumstances and sizes of types, can avoid a Gsset (20000 gas) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they both fit in the same storage slot

There are 8 instances of this issue:

```
File: contracts/ExtraRewardsDistributor.sol

20        mapping(address => mapping(uint256 => uint256)) public rewa
21        // token -> epochList
22        mapping(address => uint256[]) public rewardEpochs;
23        // token -> account -> last claimed epoch index
24:       mapping(address => mapping(address => uint256)) public user
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/ExtraRewardsDistributor.sol#L20-L24

```
File: contracts/AuraBalRewardPool.sol

44        mapping(address => uint256) public userRewardPerTokenPaid;
45        mapping(address => uint256) public rewards;
46:       mapping(address => uint256) private _balances;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraBalRewardPool.sol#L44-L46

```
File: contracts/AuraLocker.sol

91          mapping(address => Balances) public balances;
92          mapping(address => LockedBalance[]) public userLocks;
93
94          // Voting
95          //     Stored delegations
96          mapping(address => address) private _delegates;
97          //     Checkpointed votes
98          mapping(address => DelegateeCheckpoint[]) private _checkpoi
99          //     Delegatee balances (user -> unlock timestamp -> amou
100:        mapping(address => mapping(uint256 => uint256)) public dele
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L91-L100

```
File: contracts/AuraVestedEscrow.sol

35          mapping(address => uint256) public totalLocked;
36:         mapping(address => uint256) public totalClaimed;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraVestedEscrow.sol#L35-L36

```
File: convex-platform/contracts/contracts/VoterProxy.sol

35          mapping (address => bool) private stashPool;
36:         mapping (address => bool) private protectedTokens;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/VoterProxy.sol#L35-L36

```
File: convex-platform/contracts/contracts/BaseRewardPool.sol

80        mapping(address => uint256) public userRewardPerTokenPaid;
81        mapping(address => uint256) public rewards;
82:       mapping(address => uint256) private _balances;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/BaseRewardPool.sol#L80-L82

```
File: convex-platform/contracts/contracts/VirtualBalanceRewardPool.so

97        mapping(address => uint256) public userRewardPerTokenPaid;
98:       mapping(address => uint256) public rewards;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/VirtualBalanceRewardPool.sol#L97-L98

```
File: convex-platform/contracts/contracts/RewardFactory.sol

27        mapping (address => bool) private rewardAccess;
28:       mapping(address => uint256[]) public rewardActiveList;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/RewardFactory.sol#L27-L28

[G-03] State variables only set in the constructor should be declared `immutable`

Avoids a Gsset (20000 gas) in the constructor, and replaces each Gwarmacces (100 gas) with a `PUSH32` (3 gas).

There are 6 instances of this issue:

```
File: contracts/AuraLocker.sol

117:      string private _name;

118:      string private _symbol;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L117

```
File: contracts/ClaimFeesHelper.sol

23:       IFeeDistributor public feeDistro;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/ClaimFeesHelper.sol#L23

```
File: convex-platform/contracts/contracts/TokenFactory.sol

21:       string public namePostfix;

22:       string public symbolPrefix;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/TokenFactory.sol#L21

```
File: convex-platform/contracts/contracts/BaseRewardPool4626.sol
```

```
26:       address public override asset;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/BaseRewardPool4626.sol#L26

[G-04] State variables can be packed into fewer storage slots

If variables occupying the same slot are both written the same function or by the constructor, avoids a separate Gsset (20000 gas). Reads of the variables can also be cheaper

There are 3 instances of this issue:

```
File: convex-platform/contracts/contracts/CrvDepositor.sol   #1

/// @audit Variable ordering with 5 slots instead of the current 6:
 uint256(32):lockIncentive, uint256(32):incentiveCrv, uint256(32):unl
29:       uint256 public lockIncentive = 10; //incentive to users who
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/CrvDepositor.sol#L29

```
File: convex-platform/contracts/contracts/ExtraRewardStashV3.sol   #2

/// @audit Variable ordering with 9 slots instead of the current 10:
 uint256(32):pid, mapping(32):historicalRewards, mapping(32):tokenInf
33:       uint256 public pid;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/ExtraRewardStashV3.sol#L33

```
File: convex-platform/contracts/contracts/Booster.sol   #3
```

```
/// @audit Variable ordering with 18 slots instead of the current 19:
 uint256(32):lockIncentive, uint256(32):stakerIncentive, uint256(32):
26:       uint256 public lockIncentive = 825; //incentive to crv stak
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/Booster.sol#L26

[G-05] Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas

When a function with a `memory` array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the `calldata` to the `memory` index. Each iteration of this for-loop costs at least 60 gas (i.e. `60 * <mem_array>.length`). Using `calldata` directly, obliviates the need for such a loop in the contract code and runtime execution. Structs have the same overhead as an array of length one

There are 6 instances of this issue:

```
File: contracts/Interfaces.sol

17:       function getTimeWeightedAverage(OracleAverageQuery[] memory

79:           JoinPoolRequest memory request

83:           SingleSwap memory singleSwap,

84:           FundManagement memory funds,

93:           ExitPoolRequest memory request
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/Interfaces.sol#L17

```
File: convex-platform/contracts/contracts/PoolManagerSecondaryProxy.s
```

```
68:       function setUsedAddress(address[] memory usedList) external
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/PoolManagerSecondaryProxy.sol#L68

[G-06] State variables should be cached in stack variables rather than re-reading them from storage

The instances below point to the second+ access of a state variable within a function. Caching will replace each Gwarmaccess (100 gas) with a much cheaper stack read. Less obvious fixes/optimizations include having local storage variables of mappings within state variable mappings or mappings within state variable structs, having local storage variables of structs within mappings, having local memory caches of state variable structs, or having local caches of state variable contracts/addresses.

There are 60 instances of this issue. For details, see the warden's full report.

[G-07] `<x> += <y>` costs more gas than `<x> = <x> + <y>` for state variables

There are 5 instances of this issue:

```
File: contracts/AuraMerkleDrop.sol

137:             pendingPenalty += penalty;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraMerkleDrop.sol#L137

```
File: contracts/AuraBalRewardPool.sol

184:                 pendingPenalty += penalty;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraBalRewardPool.sol#L184

```
File: contracts/AuraLocker.sol

363:           lockedSupply -= amt;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L363

```
File: contracts/Aura.sol

130:           minterMinted += _amount;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/Aura.sol#L130

```
File: contracts/AuraVestedEscrow.sol

66:            require(totalTime >= 16 weeks, "!short");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraVestedEscrow.sol#L66

[G-08] `internal` functions only called once can be inlined to save gas

Not inlining costs 20 to 40 gas because of two extra `JUMP` instructions and additional stack operations needed for function calls.

There are 4 instances of this issue:

```
File: contracts/AuraClaimZap.sol   #1

171       function _claimExtras( // solhint-disable-line
172           uint256 depositCrvMaxAmount,
173           uint256 minAmountOut,
174           uint256 depositCvxMaxAmount,
175           uint256 removeCrvBalance,
176           uint256 removeCvxBalance,
177:          uint256 options
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraClaimZap.sol#L171-L177

```
File: convex-platform/contracts/contracts/VoterProxy.sol   #2

230:      function _withdrawSome(address _gauge, uint256 _amount) int
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/VoterProxy.sol#L230

```
File: convex-platform/contracts/contracts/ExtraRewardStashV3.sol   #3

124       function checkForNewRewardTokens() internal {
125:          for(uint256 i = 0; i < maxRewards; i++){
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/ExtraRewardStashV3.sol#L124-L125

```
File: convex-platform/contracts/contracts/Booster.sol   #4

572:      function _earmarkRewards(uint256 _pid) internal {
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/Booster.sol#L572

[G-09] `<array>.length` should not be looked up in every loop of a `for`-loop

The overheads outlined below are *PER LOOP*, excluding the first loop

- storage arrays incur a Gwarmaccess (100 gas)
- memory arrays use `MLOAD` (3 gas)
- calldata arrays use `CALLDATALOAD` (3 gas)

Caching the length changes each of these to a `DUP<N>` (3 gas), and gets rid of the extra `DUP<N>` needed to store the stack offset

There are 13 instances of this issue:

```
File: contracts/AuraClaimZap.sol

143:          for (uint256 i = 0; i < rewardContracts.length; i++) {

147:          for (uint256 i = 0; i < extraRewardContracts.length; i+

151:          for (uint256 i = 0; i < tokenRewardContracts.length; i+
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraClaimZap.sol#L143

```
File: contracts/AuraLocker.sol

696:          for (uint256 i = nextUnlockIndex; i < locks.length; i++
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L696

```
File: contracts/AuraVestedEscrow.sol

100:          for (uint256 i = 0; i < _recipient.length; i++) {
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraVestedEscrow.sol#L100

```
File: convex-platform/contracts/contracts/ArbitartorVault.sol

49:          for(uint256 i = 0; i < _toPids.length; i++){
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/ArbitartorVault.sol#L49

```
File: convex-platform/contracts/contracts/PoolManagerSecondaryProxy.s

69:           for(uint i=0; i < usedList.length; i++){
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/PoolManagerSecondaryProxy.sol#L69

```
File: convex-platform/contracts/contracts/BaseRewardPool.sol

214:          for(uint i=0; i < extraRewards.length; i++){

230:          for(uint i=0; i < extraRewards.length; i++){

262:          for(uint i=0; i < extraRewards.length; i++){
```

```
296:                for(uint i=0; i < extraRewards.length; i++){
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/BaseRewardPool.sol#L214

```
File: convex-platform/contracts/contracts/Booster.sol

379:            for(uint i=0; i < poolInfo.length; i++){

538:            for(uint256 i = 0; i < _gauge.length; i++){
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/Booster.sol#L379

[G-10] `++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops

The `unchecked` keyword is new in solidity version 0.8.0, so this only applies to that version or higher, which these instances are. This saves 30-40 gas ***PER LOOP***

There are 13 instances of this issue:

```
File: contracts/AuraClaimZap.sol

143:            for (uint256 i = 0; i < rewardContracts.length; i++) {

147:            for (uint256 i = 0; i < extraRewardContracts.length; i+

151:            for (uint256 i = 0; i < tokenRewardContracts.length; i+
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraClai

mZap.sol#L143

```
File: contracts/ExtraRewardsDistributor.sol

233:          for (uint256 i = epochIndex; i < tokenEpochs; i++) {
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/ExtraRewardsDistributor.sol#L233

```
File: contracts/AuraLocker.sol

174:              for (uint256 i = 0; i < rewardTokensLength; i++) {

306:          for (uint256 i; i < rewardTokensLength; i++) {

410:              for (uint256 i = nextUnlockIndex; i < length; i++)

664:          for (uint256 i = locksLength; i > 0; i--) {

696:          for (uint256 i = nextUnlockIndex; i < locks.length; i++

726:          for (uint256 i = epochIndex + 1; i > 0; i--) {

773:          for (uint256 i = 0; i < userRewardsLength; i++) {
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L174

```
File: contracts/AuraVestedEscrow.sol

100:          for (uint256 i = 0; i < _recipient.length; i++) {
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraVest

edEscrow.sol#L100

```
File: contracts/BalLiquidityProvider.sol

51:             for (uint256 i = 0; i < 2; i++) {
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/BalLiquidityProvider.sol#L51

[G-11] `require()` / `revert()` strings longer than 32 bytes cost extra gas

There is 1 instance of this issue:

```
File: contracts/AuraLocker.sol   #1

197:             require(_rewardsToken != address(stakingToken), "Cannot
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L197

[G-12] `keccak256()` should only need to be called on a specific string literal once

It should be saved to an immutable variable, and the variable used instead. If the hash is being used as a part of a function selector, the cast to `bytes4` should also only be done once

There is 1 instance of this issue:

```
File: convex-platform/contracts/contracts/Booster.sol   #1

562:             bytes memory data = abi.encodeWithSelector(bytes4(kecca
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/Booster.sol#L562

[G-13] Not using the named return variables when a function returns, wastes deployment gas

There are 10 instances of this issue:

```
File: contracts/AuraLocker.sol

603:                return 0;

649:            return balanceAtEpochOf(findEpochId(block.timestamp), _

708:            return (userBalance.locked, unlockable, locked, lockDat

708:            return (userBalance.locked, unlockable, locked, lockDat

708:            return (userBalance.locked, unlockable, locked, lockDat

708:            return (userBalance.locked, unlockable, locked, lockDat

713:            return totalSupplyAtEpoch(findEpochId(block.timestamp))

740:            return _time.sub(epochs[0].date).div(rewardsDuration);
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L603

```
File: contracts/AuraVestedEscrow.sol

159:                return 0;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraVestedEscrow.sol#L159

```
File: convex-platform/contracts/contracts/BaseRewardPool4626.sol

180:          return convertToShares(assets);
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/BaseRewardPool4626.sol#L180

[G-14] Using `bool`s for storage incurs overhead

```
// Booleans are more expensive than uint256 or any type that take
// word because each write operation emits an extra SLOAD to firs
// slot's contents, replace the bits taken up by the boolean, and
// back. This is the compiler's defense against contract upgrades
// pointer aliasing, and it cannot be disabled.
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27 Use `uint256(1)` and `uint256(2)` for true/false

There are 18 instances of this issue:

```
File: contracts/AuraMerkleDrop.sol

34:       mapping(address => bool) public hasClaimed;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraMerkleDrop.sol#L34

```
File: contracts/AuraLocker.sol

77:       mapping(address => mapping(address => bool)) public rewardD
```

```
114:      bool public isShutdown = false;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L77

```
File: contracts/AuraVestedEscrow.sol

33:       bool public initialised = false;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraVestedEscrow.sol#L33

```
File: convex-platform/contracts/contracts/PoolManagerV3.sol

22:       bool public protectAddPool;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/PoolManagerV3.sol#L22

```
File: convex-platform/contracts/contracts/CrvDepositor.sol

39:       bool public cooldown;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/CrvDepositor.sol#L39

```
File: convex-platform/contracts/contracts/PoolManagerSecondaryProxy.s

24:       bool public isShutdown;
```

```
26:       mapping(address => bool) public usedMap;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/PoolManagerSecondaryProxy.sol#L24

```
File: convex-platform/contracts/contracts/VoterProxy.sol

35:       mapping (address => bool) private stashPool;

36:       mapping (address => bool) private protectedTokens;

37:       mapping (bytes32 => bool) private votes;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/VoterProxy.sol#L35

```
File: convex-platform/contracts/contracts/BoosterOwner.sol

49:       bool public isSealed;

53:       bool public isForceTimerStarted;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/BoosterOwner.sol#L49

```
File: convex-platform/contracts/contracts/ExtraRewardStashV3.sol

40:       bool public hasRedirected;

41:       bool public hasCurveRewards;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/ExtraRewardStashV3.sol#L40

```
File: convex-platform/contracts/contracts/Booster.sol

54:       bool public isShutdown;

67:       mapping(address => bool) public gaugeMap;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/Booster.sol#L54

```
File: convex-platform/contracts/contracts/RewardFactory.sol

27:       mapping (address => bool) private rewardAccess;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/RewardFactory.sol#L27

[G-15] Use a more recent version of solidity

Use a solidity version of at least 0.8.0 to get overflow protection without `SafeMath` Use a solidity version of at least 0.8.2 to get compiler automatic inlining Use a solidity version of at least 0.8.3 to get better struct packing and cheaper multiple storage reads Use a solidity version of at least 0.8.4 to get custom errors, which are cheaper at deployment than `revert()/require()` strings Use a solidity version of at least 0.8.10 to have external calls skip contract existence checks if the external call has a return value

There are 28 instances of this issue. For details, see the warden's full report.

[G-16] Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` statement

This change saves 6 gas per instance

There are 23 instances of this issue:

```
File: contracts/AuraMerkleDrop.sol

122:          require(_amount > 0, "!amount");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraMerkleDrop.sol#L122

```
File: contracts/AuraPenaltyForwarder.sol

52:           require(bal > 0, "!empty");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraPenaltyForwarder.sol#L52

```
File: contracts/AuraBalRewardPool.sol

121:          require(_amount > 0, "RewardPool : Cannot stake 0");

139:          require(_amount > 0, "RewardPool : Cannot stake 0");

157:          require(amount > 0, "RewardPool : Cannot withdraw 0");

210:          require(rewardsAvailable > 0, "!balance");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraBalRewardPool.sol#L121

```
File: contracts/AuraLocker.sol
```

```
259:          require(_amount > 0, "Cannot stake 0");

359:          require(amt > 0, "Nothing locked");

385:          require(length > 0, "no locks");

431:          require(locked > 0, "no exp locks");

471:          require(len > 0, "Nothing to delegate");

822:          require(_rewards > 0, "No reward");

851:          require(_reward > 0, "No reward");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L259

```
File: contracts/Aura.sol

68:           require(_amount > 0, "Must mint something");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/Aura.sol#L68

```
File: contracts/BalLiquidityProvider.sol

70:           require(balAfter > 0, "!mint");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/BalLiquidityProvider.sol#L70

```
File: convex-platform/contracts/contracts/CrvDepositor.sol
```

```
169:          require(_amount > 0,"!>0");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/CrvDepositor.sol#L169

```
File: convex-platform/contracts/contracts/PoolManagerSecondaryProxy.s

104:          require(weight > 0, "must have weight");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/PoolManagerSecondaryProxy.sol#L104

```
File: convex-platform/contracts/contracts/interfaces/BoringMath.sol

20:           require(b > 0, "BoringMath: division by zero");

102:          require(b > 0, "BoringMath: division by zero");

123:          require(b > 0, "BoringMath: division by zero");

143:          require(b > 0, "BoringMath: division by zero");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/interfaces/BoringMath.sol#L20

```
File: convex-platform/contracts/contracts/BaseRewardPool.sol

211:          require(_amount > 0, 'RewardPool : Cannot stake 0');

227:          require(amount > 0, 'RewardPool : Cannot withdraw 0');
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/BaseRewardPool.sol#L211

[G-17] It costs more gas to initialize variables to zero than to let the default of zero be applied

There are 26 instances of this issue. For details, see the warden's full report.

[G-18] `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i` / `i--` too)

Saves 6 gas *PER LOOP*

There are 24 instances of this issue:

```
File: contracts/AuraClaimZap.sol

143:          for (uint256 i = 0; i < rewardContracts.length; i++) {

147:          for (uint256 i = 0; i < extraRewardContracts.length; i+

151:          for (uint256 i = 0; i < tokenRewardContracts.length; i+
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraClaimZap.sol#L143

```
File: contracts/ExtraRewardsDistributor.sol

233:          for (uint256 i = epochIndex; i < tokenEpochs; i++) {
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/ExtraRewardsDistributor.sol#L233

```
File: contracts/AuraLocker.sol

174:                 for (uint256 i = 0; i < rewardTokensLength; i++) {

306:             for (uint256 i; i < rewardTokensLength; i++) {

410:                 for (uint256 i = nextUnlockIndex; i < length; i++)

664:             for (uint256 i = locksLength; i > 0; i--) {

696:             for (uint256 i = nextUnlockIndex; i < locks.length; i++

726:             for (uint256 i = epochIndex + 1; i > 0; i--) {

773:             for (uint256 i = 0; i < userRewardsLength; i++) {
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L174

```
File: contracts/AuraVestedEscrow.sol

100:             for (uint256 i = 0; i < _recipient.length; i++) {
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraVestedEscrow.sol#L100

```
File: contracts/BalLiquidityProvider.sol

51:             for (uint256 i = 0; i < 2; i++) {
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/BalLiquidityProvider.sol#L51

File: convex-platform/contracts/contracts/ArbitartorVault.sol

49: for(uint256 i = 0; i < _toPids.length; i++){

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/ArbitartorVault.sol#L49

File: convex-platform/contracts/contracts/PoolManagerSecondaryProxy.s

69: for(uint i=0; i < usedList.length; i++){

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/PoolManagerSecondaryProxy.sol#L69

File: convex-platform/contracts/contracts/BoosterOwner.sol

144: for(uint256 i = 0; i < poolCount; i++){

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/BoosterOwner.sol#L144

File: convex-platform/contracts/contracts/ExtraRewardStashV3.sol

125: for(uint256 i = 0; i < maxRewards; i++){

199: for(uint i=0; i < tCount; i++){

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/ExtraRewardStashV3.sol#L125

```
File: convex-platform/contracts/contracts/BaseRewardPool.sol

214:            for(uint i=0; i < extraRewards.length; i++){

230:            for(uint i=0; i < extraRewards.length; i++){

262:            for(uint i=0; i < extraRewards.length; i++){

296:                for(uint i=0; i < extraRewards.length; i++){
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/BaseRewardPool.sol#L214

```
File: convex-platform/contracts/contracts/Booster.sol

379:            for(uint i=0; i < poolInfo.length; i++){

538:            for(uint256 i = 0; i < _gauge.length; i++){
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/Booster.sol#L379

[G-19] Splitting `require()` statements that use `&&` saves gas

See this issue which describes the fact that there is a larger deployment gas cost, but with enough runtime calls, the change ends up being cheaper

There are 15 instances of this issue:

```
File: contracts/ExtraRewardsDistributor.sol

171:            require(_index > 0 && _index < rewardEpochs[_token].len
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/ExtraRewardsDistributor.sol#L171

```
File: contracts/AuraStakingProxy.sol

90:           require(_outputBps > 9000 && _outputBps < 10000, "Inval

159:          require(_token != crv && _token != cvx && _token != cvx

203:          require(address(_token) != crv && address(_token) != cv
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraStakingProxy.sol#L90

```
File: contracts/BalLiquidityProvider.sol

48:           require(_request.assets.length == 2 && _request.maxAmou

57:               require(bal > 0 && bal == _request.maxAmountsIn[i],
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/BalLiquidityProvider.sol#L48

```
File: convex-platform/contracts/contracts/StashFactoryV2.sol

83:           require(!isV1 && !isV2 && !isV3,"stash version mismatch
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/StashFactoryV2.sol#L83

```
File: convex-platform/contracts/contracts/PoolManagerSecondaryProxy.s

111:          require(!usedMap[_lptoken] && !usedMap[_gauge], "cant f
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/PoolManagerSecondaryProxy.sol#L111

```
File: convex-platform/contracts/contracts/Booster.sol

220:          require(lockRewards != address(0) && rewardFactory != a

222:          require(_feeToken != address(0) && _feeDistro != addres

278:          require(_lockFees >= 300 && _lockFees <= 1500, "!lockFe

279:          require(_stakerFees >= 300 && _stakerFees <= 1500, "!st

280:          require(_callerFees >= 10 && _callerFees <= 100, "!call

313:          require(msg.sender==poolManager && !isShutdown, "!add")

314:          require(_gauge != address(0) && _lptoken != address(0),
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/Booster.sol#L220

[G-20] Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead

When using elements that are smaller than 32 bytes, your contract's gas usage may be higher. This is because the EVM operates on 32 bytes at a time. Therefore, if the element is smaller than that, the EVM must use more operations in order to reduce the size of the element from 32 bytes to the desired size.

https://docs.soliditylang.org/en/v0.8.11/internals/layout_in_storage.html Use a larger size then downcast where needed

There are 99 instances of this issue. For details, see the warden's full report.

[G-21] `abi.encode()` is less efficient than `abi.encodePacked()`

There are 2 instances of this issue:

```
File: contracts/CrvDepositorWrapper.sol   #1

93:                   abi.encode(IVault.JoinKind.EXACT_TOKENS_IN_FOR_
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/CrvDepositorWrapper.sol#L93

```
File: convex-platform/contracts/contracts/StashFactoryV2.sol   #2

88:           bytes memory data = abi.encode(rewarded_token);
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/StashFactoryV2.sol#L88

[G-22] Using `private` rather than `public` for constants, saves gas

If needed, the value can be read from the verified contract source code. Savings are due to the compiler not having to create non-payable getter functions for deployment calldata, and not adding another entry to the method ID table

There are 30 instances of this issue. For details, see the warden's full report.

[G-23] Don't compare boolean expressions to boolean literals

`if (<x> == true) => if (<x>), if (<x> == false) => if (!<x>)`

There are 9 instances of this issue:

```
File: contracts/AuraMerkleDrop.sol

123:          require(hasClaimed[msg.sender] == false, "already claim
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraMerkleDrop.sol#L123

```
File: convex-platform/contracts/contracts/ArbitartorVault.sol

54:           require(shutdown==false,"pool closed");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/ArbitartorVault.sol#L54

```
File: convex-platform/contracts/contracts/VoterProxy.sol

107:          require(operator == address(0) || IDeposit(operator).is
168:          if(protectedTokens[_token] == false){
171:          if(protectedTokens[_gauge] == false){
190:          require(protectedTokens[address(_asset)] == false, "pro
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/VoterProxy.sol#L107

```
File: convex-platform/contracts/contracts/Booster.sol

400:          require(pool.shutdown == false, "pool is closed");

574:          require(pool.shutdown == false, "pool is closed");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/Booster.sol#L400

```
File: convex-platform/contracts/contracts/RewardFactory.sol

72:           require(msg.sender == operator || rewardAccess[msg.send
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/RewardFactory.sol#L72

[G-24] Don't use `SafeMath` once the solidity version is 0.8.0 or greater

Version 0.8.0 introduces internal overflow checks, so using `SafeMath` is redundant and adds overhead

There are 2 instances of this issue:

```
File: contracts/AuraBalRewardPool.sol   #1

5:    import { SafeMath } from "@openzeppelin/contracts-0.8/utils/mat
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraBalRewardPool.sol#L5

```
File: contracts/AuraStakingProxy.sol   #2

7:    import { SafeMath } from "@openzeppelin/contracts-0.8/utils/mat
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraStakingProxy.sol#L7

[G-25] Duplicated `require()` / `revert()` checks should be refactored to a modifier or function

Saves deployment costs

There are 32 instances of this issue. For details, see the warden's full report.

[G-26] Multiplication/division by two should use bit shifting

`<x> * 2` is equivalent to `<x> << 1` and `<x> / 2` is the same as `<x> >> 1`. The `MUL` and `DIV` opcodes cost 5 gas, whereas `SHL` and `SHR` only cost 3 gas

There are 5 instances of this issue:

```
File: contracts/AuraMerkleDrop.sol

136:              uint256 penalty = address(auraLocker) == address(0)
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraMerkleDrop.sol#L136

```
File: contracts/AuraBalRewardPool.sol

183:                  uint256 penalty = (reward * 2) / 10;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraBalRewardPool.sol#L183

```
File: contracts/AuraMath.sol

36:             return (a / 2) + (b / 2) + (((a % 2) + (b % 2)) / 2);

36:             return (a / 2) + (b / 2) + (((a % 2) + (b % 2)) / 2);

36:             return (a / 2) + (b / 2) + (((a % 2) + (b % 2)) / 2);
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraMath.sol#L36

[G-27] Stack variable used as a cheaper cache for a state variable is only used once

If the variable is only accessed once, it's cheaper to use the state variable directly that one time

There is 1 instance of this issue:

```
File: contracts/AuraLocker.sol   #1

328:             uint256 epochindex = epochs.length;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L328

[G-28] `require()` or `revert()` statements that check input arguments should be at the top of the function

Checks that involve constants should come before checks that involve state variables

There are 11 instances of this issue:

```
File: contracts/AuraMerkleDrop.sol

69:           require(_expiresAfter > 2 weeks, "!expiry");

122:          require(_amount > 0, "!amount");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraMerkleDrop.sol#L69

```
File: contracts/AuraBalRewardPool.sol

77:           require(_startDelay < 2 weeks, "!delay");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraBalRewardPool.sol#L77

```
File: contracts/AuraLocker.sol

472:          require(newDelegatee != address(0), "Must delegate to s

822:          require(_rewards > 0, "No reward");

851:          require(_reward > 0, "No reward");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraLocker.sol#L472

```
File: contracts/Aura.sol

68:           require(_amount > 0, "Must mint something");
```

```
69:           require(_minter != address(0), "Invalid minter");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/Aura.sol#L68

```
File: contracts/AuraStakingProxy.sol

129:          require(_incentive <= 100, "too high");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraStakingProxy.sol#L129

```
File: convex-platform/contracts/contracts/BaseRewardPool.sol

127:          require(_reward != address(0),"!reward setting");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/BaseRewardPool.sol#L127

```
File: convex-platform/contracts/contracts/Booster.sol

281:          require(_platform <= 200, "!platform");
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/Booster.sol#L281

[G-29] Empty blocks should be removed or emit something

The code should be refactored such that they no longer exist, or the block should do something useful, such as emitting an event or reverting. If the block is an empty if-

statement block to avoid doing subsequent checks in the else-if/else conditions, the else-if/else conditions should be nested under the negation of the if-statement, because they involve different classes of checks, which may lead to the introduction of errors when the code is later modified (`if(x){}else if(y){...}else{...}` => `if(!x){if(y){...}else{...}}`)

There are 6 instances of this issue:

```
File: convex-platform/contracts/contracts/VoterProxy.sol

312:           }catch{}
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/VoterProxy.sol#L312

```
File: convex-platform/contracts/contracts/ExtraRewardStashV3.sol

116                 try IRewardHook(rewardHook).onRewardClaim(){
117:                }catch{}

117:                }catch{}
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/ExtraRewardStashV3.sol#L116-L117

```
File: convex-platform/contracts/contracts/Booster.sol

361           try IStaker(staker).withdrawAll(pool.lptoken,pool.gauge
362:          }catch{}

362:          }catch{}

389:              }catch{}
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/Booster.sol#L361-L362

[G-30] Use custom errors rather than `revert()` / `require()` strings to save deployment gas

Custom errors are available from solidity version 0.8.4. The instances below match or exceed that version

There are 101 instances of this issue. For details, see the warden's full report.

[G-31] Functions guaranteed to revert when called by normal users can be marked `payable`

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are `CALLVALUE` (2), `DUP1` (3), `ISZERO` (3), `PUSH2` (3), `JUMPI` (10), `PUSH1` (3), `DUP1` (3), `REVERT` (0), `JUMPDEST` (1), `POP` (2), which costs an average of about 21 gas per call to the function, in addition to the extra deployment cost

There are 37 instances of this issue. For details, see the warden's full report.

[G-32] `public` functions not called by the contract should be declared `external` instead

Contracts are allowed to override their parents' functions and change the visibility from `external` to `public` and can save gas by doing so.

There are 18 instances of this issue:

```
File: contracts/ExtraRewardsDistributor.sol

117:      function getReward(address _account, address _token) public

127       function getReward(
128           address _account,
```

```
129             address _token,
130:            uint256 _startIndex
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/ExtraRewardsDistributor.sol#L117

```
File: contracts/AuraMerkleDrop.sol

114         function claim(
115             bytes32[] calldata _proof,
116             uint256 _amount,
117             bool _lock
118:        ) public returns (bool) {

149         function forwardPenalty() public {
150:            uint256 toForward = pendingPenalty;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraMerkleDrop.sol#L114-L118

```
File: contracts/AuraPenaltyForwarder.sol

47          function forward() public {
48:             require(block.timestamp > lastDistribution + distributi
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraPenaltyForwarder.sol#L47-L48

```
File: contracts/AuraBalRewardPool.sol

138:        function stakeFor(address _for, uint256 _amount) public upd

152         function withdraw(
```

```
153             uint256 amount,
154             bool claim,
155             bool lock
156:        ) public updateReward(msg.sender) returns (bool) {

195         function forwardPenalty() public {
196:            uint256 toForward = pendingPenalty;
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/AuraBalRewardPool.sol#L138

```
File: contracts/BalLiquidityProvider.sol

46:         function provideLiquidity(bytes32 _poolId, IVault.JoinPoolR
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/contracts/BalLiquidityProvider.sol#L46

```
File: convex-platform/contracts/contracts/ConvexMasterChef.sol

96          function add(
97              uint256 _allocPoint,
98              IERC20 _lpToken,
99              IRewarder _rewarder,
100             bool _withUpdate
101:        ) public onlyOwner {

121         function set(
122             uint256 _pid,
123             uint256 _allocPoint,
124             IRewarder _rewarder,
125             bool _withUpdate,
126             bool _updateRewarder
127:        ) public onlyOwner {

209:        function deposit(uint256 _pid, uint256 _amount) public {
```

```
239:      function withdraw(uint256 _pid, uint256 _amount) public {
```

```
283:      function emergencyWithdraw(uint256 _pid) public {
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/ConvexMasterChef.sol#L96-L101

```
File: convex-platform/contracts/contracts/VoterProxy.sol
```

```
151:      function isValidSignature(bytes32 _hash, bytes memory) publ
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/VoterProxy.sol#L151

```
File: convex-platform/contracts/contracts/BaseRewardPool.sol
```

```
191       function stakeFor(address _for, uint256 _amount)
192           public
193:          returns(bool)
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/BaseRewardPool.sol#L191-L193

```
File: convex-platform/contracts/contracts/VirtualBalanceRewardPool.so
```

```
178       function withdraw(address _account, uint256 amount)
179           public
180:          updateReward(_account)
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/VirtualBalanceRewardPool.sol#L178-L180

```
File: convex-platform/contracts/contracts/Booster.sol

493:      function withdrawAll(uint256 _pid) public returns(bool){
```

https://github.com/code-423n4/2022-05-aura/blob/4989a2077546a5394e3650bf3c224669a0f7e690/convex-platform/contracts/contracts/Booster.sol#L493

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top