QuillAudits

Audit Report
March, 2023

For

ASTRA DAO

Table of Content

Executive Summary

Project Name	Astra DAO
Overview	Astra DAO is a decentralized and non-custodial automated crypto asset allocator. Astra DAO provides convenient and practical access to crypto-oriented investment strategies. ASTRA token is responsible for governance of the whole ecosystem which can be earned through various investment products/indices, participation units marketplace, user staking, harvesting investment strategies profits.
Timeline	December 7, 2022 to April 25, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze Astra Dao smart contract codebase for quality, security, and correctness. *https://github.com/astradao/astra-private/pull/37/commits/88e9f90a2ac669a7d550a40f37a5945fe927376a* *https://github.com/astradao/astra-private/tree/9c617cc41136455929116392 7a67def8b50e4d69* Commit hash: 9c617cc41136455929116392 7a67def8b50e4d69 Contracts:- - indicespayment.sol - itoken.sol - poolConfiguration.sol - swapv2.sol - itoken-staking.sol - chefv2.sol - uniswapAmount.sol - poolv2.sol - governance.sol - timelock.sol - batchVote.sol
Fixed In	*https://github.com/astradao/astra-private/commit/360a6c48bb55c5413d670c21e9d730f6257ddeff*

Executive Summary

Note: *https://github.com/astradao/astra-private/pull/41* has been Received for Audit on 17 Feb,2023

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	3
Partially Resolved Issues	0	0	0	0
Resolved Issues	2	15	30	29

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - IndicesPayment

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

A1. Hardcoded Index creation fee (astraAmount) is different from what mentioned in whitepaper

```
500     function initialize(address _ASTRTokenAddress, address _poolConf,address _exchangeContract, address _treasury) public initializer{
501                 require(_ASTRTokenAddress != address(0), "zero address");
502         require(_poolConf != address(0), "zero address");
503         require(_exchangeContract != address(0), "zero address");
504         require(_treasury != address(0), "zero address");
505         _setOwner(msg.sender);
506                 ReentrancyGuard.__init();
507         poolConf = _poolConf;
508         exchangeContract = _exchangeContract;
509                 ASTRTokenAddress = _ASTRTokenAddress;
510         treasury = _treasury;
511         astraAmount = 500 * uint256(10)**uint256(18);
512         }
513
```

The astraAmount initialized into code is different from the amount of astra tokens that is mentioned in whitepaper to create the index. In the whitepaper it is mentioned that “Creators need to pay 5,000,000,000 Astra tokens to create an index”, But the hardcoded amount is 500e18 which does not match 5000000000e18.

Recommendation

Verify and change the astraAmount to correct the token amount.

Status

Resolved

Astra Dao team’s comment: The amount was added for testing purpose and it will be changed to the expected one using setAstraAmount().

Low Severity Issues

A2. Ownership transfer should be two step process

Description

In indicesPayment contract transferOwnership() function takes newOwner and sets it to the _owner variable in _setOwner() function. But sometimes it can happen that the current owner might set a malicious account as owner which can be problematic for the project.

Remediation

Owner transfership process can be changed to two steps where the owner will set pendingOwner variable as new owner address and then the new owner who is going to take over the contract can call _setOwner to take over the contract.

Status

Resolved

Informational Issues

A3. Re-entrancy guard is unused

Description

In indicesPayments code ReentrancyGuard has been inherited and initialized but nonReentrant modifier has never been used for any of the functions.

Remediation

If not used it can be removed from the code.

Status

Resolved

B. Contract - ITokendeployer

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

B.1 Pushing same address in mapping

```
function addChefAddress(address _address) public returns(bool){
    require(managerAddress[msg.sender],"Only manager can add address");
    require(_address != address(0), "Zero Address");
    require(!whitelistedaddress[_address],"Already white listed");
    whitelistedaddress[_address] = true;
    alladdress.push(msg.sender);
    return true;
}
```

addChefAddress() function is used to add chef contract address to the code. Here msg.sender is getting pushed instead of _address variable which can cause the mapping to contain the same address.

Recommendation

To resolve the issue change the address value parameter to _address variable. Also as alladdress mapping is not used anywhere can be removed if not necessary.

Status

Resolved

Informational Issues

B.2 Spelling mistake in modifier

Description

onlyOwner() modifier has been misspelled as onlyOnwer()

Remediation

Please change the spelling from onlyOnwer to onlyOwner.

Status

Resolved

B.3 Unused code in contract

Description

_setupDecimals() and beforeTokenTransfer() functions are unused.

Remediation

The above mentioned functions _setupDecimals() and beforeTokenTransfer() can be removed.

Status

Resolved

C. Contract - PoolConfiguration

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

C.1 whitelistDAOaddress() function allows multiple DAOs to exist

In poolConfiguration contract whitelistDAOaddress() function allows whitelisting of DAO addresses to be interacted with. So for example there was supposed to be a change made to the DAO contract and is added again with the function. The older version of DAO contract still exists and can have or impact the decisions/values.

Recommendation

To remediate issue the value can be changed as a single variable to store DAO address instead of having mapping.

Status

Resolved

C.2 Set fix fee limit

updateEarlyExitFees(), updatePerfees(), updateSlippagerate() are taking uint values to set fees and rate, currently these functions are allowing less than 100 i.e maximum 99 percentage of fees and slippage rate. The strict amount should be hard coded as a limit which would be less than 99. so that in the worst cases it can’t be set 99 which is almost the amount.

Recommendation

set the maximum limit to less.

Status

Resolved

D. Contract - SwapV2

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

D.1 Add setter function to support new path

In contract the tokens are initialized in the initialize() function which acts as constructor. Consider if the pool happens to be for a token which gets low on liquidity then it won't be possible to work with that pool anymore for the protocol considering the pool exists.

Recommendation

To remediate the issue and to add a new path a setter function can be added.

Status

Resolved

D.2 Out of gas issue can occur on initializing the contract

The contract's initialize function takes _tokens address array which if is big enough can cause the transaction to go out of gas.

Recommendation

To remediate the issue please make sure that token length should be fixed when the contract is initialized.

Status

Resolved

Informational Issues

D.3 Unused code can be removed

```
constructor() public {}
```

Description

As the constructor is not taking any parameters, the line can be removed from the contract.

Remediation

The constructor can be removed.

Status

Resolved

D.4 Add a check for path with same tokens

Description

Swapv2 has some functions that are getting used by other contracts, this contains functions like getBestExchangeRate(), swapFromBestExchange(). these functions take tokenIn, tokenOut token parameters. The case can happen where tokenIn and tokenOut entered are the same and for these scenarios a check can be added so that the transaction can revert with a meaningful error message.

Remediation

Add check to revert the same tokenIn and tokenOut.

Status

Resolved

E. Contract - ITokenStaking

High Severity Issues

E.1 Calling restakeAstraReward() results in loss of funds for protocol

```
if (Dao(governanceAddress).getVotingStatus(msg.sender)) {
    astra.approve(astraStakingContract,claimableReward);
    IMasterChefV2(astraStakingContract).depositFromOtherContract(_pid1, claimableReward, SIX_MONTH_VAULT, msg.sender);
    safeAstraTransfer(msg.sender, claimableReward);
} else {
    slashedReward = slashedReward.add(
        claimableReward
    );
}
```

In contract itoken-staking if the user wants to withdraw tokens there is an option of staking the rewards for the deposit which is made. And if it is called, the call goes to the restakeAstraReward() function in which the function checks if the user has voted on the recent proposal. If yes then rewards will be restaked but astra tokens are getting transferred to masterchef contract which is chefv2 and to the user too. Which results in loss for the protocol.

Recommendation

To resolve the issue please remove the safeAstraTransfer() from the if block.

Status

Resolved

Medium Severity Issues

E.2 restakeAstraReward() can revert on wrong _pid

```
if (Dao(governanceAddress).getVotingStatus(msg.sender)) {
    astra.approve(astraStakingContract,claimableReward);
    IMasterChefV2(astraStakingContract).depositFromOtherContract(_pid1, claimableReward, SIX_MONTH_VAULT, msg.sender);
    safeAstraTransfer(msg.sender, claimableReward);
} else {
    slashedReward = slashedReward.add(
        claimableReward
    );
}
```

In contract itoken-staking user has option to restake astra rewards if he doesn't want to get slashed rewards. So while withdrawing, the user mentions their _pid from which the rewards will be restaked. As the rewards will be astra tokens they should be going to pool0 in chefv2 contract. But if it is something else than that the function will revert because astra tokens should always go to the pool0.

Recommendation

To resolve the issue first parameter can be kept as values 0 instead of _pid

Status

Resolved

Comment: The dev team said that pool0 will always be of astra tokens.

E.3 Pool should be restricted to one iToken only

Currently while depositing and withdrawing deposit() and withdraw() functions are taking itoken parameters this can increase the attack surface. malicious users can specify different itoken ids while depositing and withdrawing. which can result in loss of funds deposited by other users.

Recommendation

Consider directly taking the token address from PoolInfo.lpToken which can be added while creating a pool with add() similar to chefv2:add(). remove the _itokenId parameter in deposit()and withdraw()

Status

Resolved

E.4 LPToken should be added directly while adding pool

The Itoken and decimal value should be added directly while adding pool with add() instead of adding in `itokenInfo` array with addItoken() to avoid scenarios where users can specify different itoken ids while depositing and withdrawing, as mentioned in E.3.

Recommendation

Consider adding itoken address while adding pools.

Status

Resolved

E.5 _withdraw()sends itoken as well as astra of same amount

While withdrawing _withdraw() sends staked itoken amount back to the staker address on L800 but again it sends the same amount of astra back to the user using safeAstraTransfer() on L 804.

Recommendation

Consider verifying the business logic and remove the statement on L804 to send same amount of astra.

Status

Resolved

E.6 Restakes in incorrect vault

It is mentioned in whitepaper "5.7.Rewards Distribution System>Claiming Rewards - iTokens Staking & Liquidity Mining" on page 18 that "Claim 100% and restake in 90-days lockup" here in restakeAstraReward() the staking of claimableReward is happening on L375 but it is not staking the amount in 90 days (3 month) vault as mentioned in the whitepaper, instead it is staking in six month vault.

Recommendation

Verify and change the vault used.

Status

Resolved

E.7 In checkEligibleAmount function updateUserSlashingfees function totaldepositAmount, stkInfo.amount should be swapped

checkEligibleAmount() is calling updateUserAverageSlashingFees() when duration of staked vault is not passed while calling withdraw(). while calling updateUserAverageSlashingFees() it is passing totaldepositAmount and stkInfo.amount as previousDepositAmount and newDepositAmount respectively, But it is incorrect as previousDepositAmount should be the stkInfo.amount and newDepositAmount should be passed as totaldepositAmount (which is 0 in this case) while calling updateUserAverageSlashingFees(). Currently because of the incorrect sequence of these arguments, the averageStakedTime is getting increased for that user and because of that while claiming astra using claimAstra() the slashDays getting calculated is getting decreased.

Recommendation

Change the sequence of the argument as mentioned above.

Status

Resolved

E.8 Rewards are different for same deposit amount after changing decimals of iToken

In contract itokendeployer owner can change decimals of itoken. So if the itoken with 6 decimals is staked in itoken-staking contract and when the rewards are checked then it comes out to be less than the itoken with 18 decimals.
Eg.,
Amount(1000) itoken with 18 decimals was staked for 100 blocks then pending rewards = 4999.99

Amount(1000) itoken with 6 decimals was staked for 100 blocks the pending rewards = 3333.33

Recommendation

Please revisit the reward calculation formula after changing decimals if this is not the intended behavior.

Status

Resolved

Low Severity Issues

E.9 IToken staking gives the same reward for Different iTokens.

for depositing into different indexes different itokens get minted as a share of deposited amount.
Let's say for getting 100 itoken-1 you need to deposit 100 stablecoins in index-1, for getting 100 itoken-2 you need to deposit 200 stablecoins in index-2. Here the amount of stablecoins you deposited to get the same amount of itoken in index-2 is greater. When it comes to staking these itokens in itoken-staking, for different itoken it gives same amount of reward so the itokens for costly index will receive same rewards as compared to itokens for cheap index and vice versa.

Recommendation

Check the business logic and change the code logic accordingly.

Status

Resolved

E.10 Unnecessary use of addVault() function

In itoken-staking contract there are vaults decided for months 0, 3, 6, 9, 12. For months 0, 6, 9, 12 they are initialized in the initialize() function of the contract. Even the owner decided to add new month vaults such as 15, 18 still the multipliers which are related to them won't be added as there is no setter function which can be problematic and can cause issues in calculation of rewards.

Recommendation

To resolve the issue addVault() function can be removed and vault for 3 month can be initialized in initialize() function or if there are vaults which can be added such as 15, 18 or for any month then please add setter function to set their multiplier values too.

Status

Resolved

E.11 Cooldown period is not according to the whitepaper

On whitepaper page 14 "5.4.Astra Staking Model> Cool-down Period" it is mentioned that the cooldown period would be of 7 days and , if the user fails to confirm the unstake transaction in the 24h window, the cooldown period will be reset.

in the withdraw() logic on L747-748 the cooldown period is getting calculated with user.cooldowntimestamp.add(SECONDS_IN_DAY.mul(coolDownPeriodTime) here coolDownPeriodTime is getting set as 1 in initialize() and the contract does not have any function to update coolDownPeriodTime. the mentioned formula calculates cooldown period for 1 day only because coolDownPeriodTime is set to 1 in initialize.

Additionally if user fails to confirm withdraw() in 24 hrs window then cooldown perioid is not getting reset.

Recommendation

Consider changing the coolDownPeriodTime assignment of 1 to 7 in initialize() and add functionality to reset the cooldown period.

Status

Resolved

E.12 rewardDebt is getting overwritten in _withdraw() function

In the contract whenever rewards are accumulated they get stored in rewardsDebt variable and rewardDebt is updated while restaking with restakeAstraReward but is also getting overwritten in _withdraw() function.

Recommendation

To resolve the issue the variable should be updated in claim and restake only.

Status

Resolved

E.13 Handle cool down period in different way

While withdrawing, users need to wait till the cooldown period ends. Currently user needs to call withdraw() two times as first time the statement in if block will execute which will set user.cooldown to true and user.cooldowntimestamp to current timestamp, in second withdraw()call else block will execute which will check current block's timestamp is greater than user.cooldowntimestamp + cooldown period and then it calls internal _withdraw()to withdraw amount.

Here, it takes two calls to withdraw the amount, the first one to set the timestamp for cooldown period calculation. The functionality can be changed so the user can withdraw in one call and the cooldown period will get checked according to something that was recorded before e.g last deposited timestamp etc.

Recommendation

Add functionality as suggested in description.

Status

Resolved

Comments: This is updated in whitepaper.

Informational Issues

E.14 decimalValue variable is calculated twice

Description

In contract itoken-staking decimalValue variable calculation is done twice in the stakingScoreAndMultiplier() function which increases gas cost.

Remediation

To resolve the issue it can be declared locally once and use the assigned value later on for the calculations.

Status

Resolved

E.15 Variable has no impact on calculations

```
function getMultiplier(uint256 _from1, uint256 _to1)
    public
    view
    returns (uint256)
{
    if (_to1 <= bonusEndBlock) {
        return _to1.sub(_from1).mul(BONUS_MULTIPLIER);
    } else if (_from1 >= bonusEndBlock) {
        return _to1.sub(_from1);
    } else {
        return
            bonusEndBlock.sub(_from1).mul(BONUS_MULTIPLIER).add(
                _to1.sub(bonusEndBlock)
            );
    }
}
```

In contract itoken-staking BONUS_MULTIPLIER which is constant is set to 1. As the value is set to 1 it won't be having any impact on any of the calculations. Also it is constant so you can't even change the value. So that makes the if and else if block do the same work in the getMultiplier() function.

Recommendation

To resolve the issue please remove the variable which will help in saving gas or add a setter function to change the value of the variable.

Status

Resolved

F. Contract - ChefV2

High Severity Issues

No issues found

Medium Severity Issues

F.1 Withdraw sends astra tokens every time

While withdrawing deposited amount _withdraw() sends astra tokens everytime as it is using safeAstraTransfer() on L854. In the scenario where the user deposits any other token in the pool apart from the astra token, while withdrawing the astra tokens are getting sent back to that user which is incorrect.

Recommendation

Consider sending the token amount for the specific token that the user deposited.

Status

Resolved

F.2 Pool should be restricted to one itoken only

Currently while depositing and withdrawing deposit() and withdraw()functions are taking itoken parameters, this can increase the attack surface and may create issues as mentioned in itoken-staking.

Recommendation

Consider directly taking the token address from PoolInfo.lpToken which is getting added in add() function while adding pool.

Status

Resolved

F.3 In checkEligibleAmount function updateUserSlashingfees function totaldepositAmount, stkInfo.amount should be swapped

checkEligibleAmount() is calling updateUserAverageSlashingFees() when duration of staked vault is not passed while calling withdraw(). while calling updateUserAverageSlashingFees() it is passing totaldepositAmount and stkInfo.amount as previousDepositAmount and newDepositAmount respectively, But it is incorrect as previousDepositAmount should be the stkInfo.amount and newDepositAmount should be passed as totaldepositAmount (which is 0 in this case) while calling updateUserAverageSlashingFees(). Currently because of the incorrect sequence of these arguments, the averageStakedTime for these users is decreasing.

Recommendation

Change the sequence of the argument as mentioned above.

Status

Resolved

F.4 Withdrawing amount causes highestAstaStaker array to have some gaps/overriding values

Consider 100 highest users deposited amount to the contract. These users will be added to highestStakerInPool mapping. After sometime if say 65 th user decided to withdraw his stakes from pool the position at which his amount was in highestStakerInPool will be set to default values. ie. struct values when deleted are set to default, creating a gap in highestStakerInPool mapping. Now when new 101th user other than previous 100 users will deposit amount, at that time when addHighestStakedUser() will be called then it'll go in the else block of the code and will override the position of the 0th and when quicksort is called they will be sorted according to their amounts leaving the gap/default value at the 0th position

Recommendation

Consider changing the code logic where it will replace these gaps of default (deleted) values Or quickSort() can be called in removeHighestStakedUser() after deleting the value of highestStaker on L 1090.

Status

Resolved

Low Severity Issues

F.5 Cooldown period is not according to the whitepaper

On whitepaper page 14 "5.4.Astra Staking Model> Cool-down Period" it is mentioned that the cooldown period would be of 7 days and , if the user fails to confirm the unstake transaction in the 24h window, the cooldown period will be reset.
in the withdraw() logic on L747-748 the cooldown period is getting calculated with user.cooldowntimestamp.add(SECONDS_IN_DAY.mul(coolDownPeriodTime) here coolDownPeriodTime is getting set as 1 in initialize() and the contract does not have any function to update coolDownPeriodTime. The mentioned formula calculates cooldown period for 1 day only because coolDownPeriodTime is set to 1 in the initialize() function.
Additionally if the user fails to confirm withdraw() in the 24 hrs window then the cool down period is not getting reset.

Recommendation

Consider changing the coolDownPeriodTime assignment of 1 to 7 in initialize() and add functionality to reset the cooldown period.

Status

Resolved

F.6 Functions are missing return values

Functions in chefV2 like claimAstra() and restakeAstraReward() are missing return values but are declared in function definition.

Recommendation

Please return appropriate values for the above functions

Status

Resolved

F.7 Variable does not have any impact on calculations

The variable BONUS_MULTIPLIER is a constant and is set to 1 but is used in calculations anyhow it won't affect the values but is consuming gas only.

Recommendation

To resolve the issue please remove the variable as it consumes gas or don't keep it as constant if the multiplier can be changed.

Status

Resolved

F.8 Incorrect lp token status check in transferNFTandGetAmount()

transferNFTandGetAmount()has a require check to check if the lp token is added or not with addUniswapVersion3().While checking this on L685, it should check for lpTokensStatus[_token1][_token0]. Currently it is checking for lpTokensStatus[_token0][_token1] on both lines (L684,L685).

Recommendation

Change the token status check from token1 to token0 on L685 as mentioned in description.

Status

Resolved

F.9 rewardDebt is getting overwritten in _withdraw() function

In the contract whenever rewards are accumulated they get stored in rewardsDebt variable and rewardDebt is updated while restaking with restakeAstraReward but is also getting overwritten in _withdraw() function.

Recommendation

To resolve the issue the variable should be updated in claim and restake only.

Status

Resolved

F.10 Handle cool down period in different way

While withdrawing users need to wait till the cooldown period ends. Currently user needs to call withdraw() two times as first time the statement in if block will execute which will set user.cooldown to true and user.cooldowntimestamp to current_timestamp, in second withdraw()call else block will execute which will check current block's timestamp is greater than user.cooldowntimestamp + cooldown period and then it calls internal _withdraw()to withdraw amount.
Here, it takes two calls to withdraw the amount, the first one to set the timestamp for cooldown period calculation. The functionality can be changed so the user can withdraw in one call and the cooldown period will get checked according to something that was recorded before e.g last deposited timestamp etc.

Recommendation

Add functionality as suggested in description.

Status

Resolved

Comments: This is updated in whitepaper.

Informational Issues

F.11 Deposit can be called from any contract

Description

In chefV2 contract, there are 2 functions for deposit ie., deposit() and depositFromOtherContract(). The difference in both the functions is, msg.sender being passed to _deposit and a whitelist address check which can be called by an EOA or a contract hence current name depositFromOtherContract() can be confusing. Also the deposit() function can be called by any contract even if it is not whitelisted.

Remediation

If there is a need for no contract check, it is recommended to add a check in both deposits.

Status

Resolved

Comments: This function is kept to whitelist only Astra Dao contracts to deposit on some user's behalf.

F.12 Redundant code/statements

Description

1. L452 "&& slashDays >= 0" is redundant as slashDays is getting declared on L449 as an unsigned integer variable which is going to be 0 or greater than it when values are getting assigned.
2. In _deposit() on L776-778 there's redundant check and assignment because user.maxMultiplier is getting assigned on L779. Similarly on _withdraw() on L846-848 theres redundant check and assignment because user.maxMultiplier is getting assigned on L849.

Remediation

Consider checking and removing the redundant statements.

Status

Resolved

G. Contract - UniswapAmount

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

G.1 Unused code can be removed

Description

As the constructor is not taking any parameters, the line can be removed from the contract.

```
constructor() public {}
```

Also the following imports are not used inside the contract, hence they can be removed:

```
import "@uniswap/v3-core/contracts/libraries/FullMath.sol";
import "@uniswap/v3-core/contracts/libraries/UnsafeMath.sol";
import "@uniswap/v3-core/contracts/libraries/FixedPoint96.sol";
```

Remediation

Consider removing above mentioned things if not used.

Status

Resolved

H. Contract - PoolV2

High Severity Issues

No issues found

Medium Severity Issues

H.1 Couldn’t find depositFromDaaAndDAO() in master chef (chef v2)

Description

withdrawUserAmount() calls depositFromDaaAndDAO() on the chefv2 contract, but chefv2 doesn't has the function name depositFromDaaAndDAO() which results in failed transaction when stakeEarlyFees and/or stakePremium would be true.
Chefv2 contains a function named depositFromOtherContract() instead of depositFromDaaAndDAO().

Remediation

Consider changing the function name so that the call won’t fail.

Status

Resolved

Comment: Added a function called depositFromOtherContract in chefv2 contract.

H.2 Calculatefee() is unable to reduce the fee (feeRate) over time

Description

The calculatefee() is getting used to calculate the early exit fee by getting early exit fee percent (using getEarlyExitfees) from the PoolConfiguration contract. According to whitepaper the early exit fee should decrease over time and after 6 months the fee should be zero
The early exit fee is not getting decreased over a time and even after 6 months it deducts initial fee rate percent of the given amount.The issue here is caused by the feesValue variable in formula becoming 0 because of precision loss which then gets used for subtraction.
Example: For pending or current balance.
Let's say there are 100e18 (100000000000000000000) pending/total tokens. And if the fee rate is 2 (percent) then after withdrawing the amount of fee deducted would be 2e+18.
Now while withdrawing on L 966,967 the fee is getting calculated for pending and total amount. According to the formula let's say we are withdrawing after 591500 blocks then it should deduct half of fee rate. That is 2/2 =1 percent.

```
uint256 feeRate = 2;
uint256 Averageblockperday = 6500;
uint256 feeconstant = 182;
uint256 blocks = 591500;
uint feesValue = feeRate.mul(blocks).div(100); // 11830


// 0.01 = 0 (here feesValue would be 0 as solidity doesn't support decimals)
feesValue = feesValue.div(Averageblockperday).div(feeconstant);
feesValue = _amount.mul(feeRate).div(100).sub(feesValue); //_amount
```

Here the last feesValue would be 2% of 100e18 as it subtracts 0. It shows that it is unable to decrease the fee over time because of precision loss

Remediation

Fee rate can be multiplied with some temporary value like 1e12 which
can be again divided after getting the last answer to take care of fractional part.

Status

Resolved

Fixed In: https://github.com/astradao/astra-private/commit/fad0c03addf9464cd1d522bca1...

H.3 Fees are not getting calculated correctly

Description

In current code the fees are not getting calculated accordingly the formula and fees are not getting reduced correctly.

Remediation

Change the formula in the way where fees will get reduced. The changes may look like this:

```
function calculatefee(
    address _account,
    uint _amount,
    uint _poolIndex
) internal view returns (uint256) {
    // Calculate the early exit fees based on the formula mentioned above.
    uint256 feeRate = (IPoolConfiguration(_poolConf).getEarlyExitfees()) *
        1e12;
    uint256 startBlock = initialDeposit[_account][_poolIndex];
    uint256 withdrawBlock = block.number;
    uint256 Averageblockperday = 6500;
    uint256 feeconstant = 182;
    uint256 blocks = withdrawBlock.sub(startBlock);
    if (blocks >= 182 * 6500) {
        return 0;
    }
    uint feesValue = feeRate.mul(blocks).div(100);
    feesValue = feesValue.div(Averageblockperday).div(feeconstant);
    feesValue = (feeRate.div(100).sub(feesValue)).mul(_amount);
    return feesValue / 1e12;
}
```

Status

Resolved

H.4 Block elapsed are getting calculated from initial deposit

Description

Early exit fee is calculated from initial deposit which can be misused.
In calculatefee() on L736 the startblock is getting set as initialDeposit. Which is getting set in poolIn() L864 at the time of first deposit.A User can deposit some amount at the start and withdraw it instantly, initializing the initialDeposit time and existingUser. After that he can benefit from this for reducing the early exit fee.
For example, a user can deposit a big amount for a short period of time (less than 6 months) and could withdraw without deducting a fee if the difference between initial deposited block and current block is greater than 6 months, where initial deposited block set at at the first deposit.

Remediation

Calculate average staked time of the user for a pool and use it for calculating the fees.

Status

Resolved

H.5 Change itoken share calculation

Description

getItokenValue() calculates how much itoken shares to mint. In "if else" block on L843 the indexValue that is getting used is calculated by using getPoolValue(). This getPoolValue() function calculates it everytime for index token balances on L1218. tokenBalances are getting added in buytokens() on L765. So this amount of itoken will get updated first when pool threshold gets reached while someone is depositing and when that pool isn't rebalanced the poolIn() calls buytokens().
Now there can be other deposits after threshold reaches, so this amount will get added to the pending amount as it is not getting converted to index tokens. But now when anyone is depositing, the itoken share is getting calculated by fetching the best exchange rate for current index token balances of that pool and the pending amount is not getting considered while calculating this amount in getPoolValue().

Remediation

Consider adding the current pending amount while getting index value from getPoolValue().

Status

Resolved

Low Severity Issues

H.6 Rewards will always get slashed in these scenario

1. In the scenario where governance started in 1st month and other users start staking next month, now because these users have missed some proposals or they couldn't vote because they started late, the rewards of these users will get slashed as GovernorAlpha:getVotingStatus() will return false.
2. According to whitepaper for first 90 days the top 100 wallets can vote, in this case if there would be some proposals, users apart from these top 100 wallets will miss the governance and that's why the reward will get slashed for these users as getVotingStatus() will return false.

Recommendation

Verify the business logic.

Status

Resolved

H.7 No contract level check for paying to create index

White Paper states that ‘Creators need to pay 5,000,000,000 Astra tokens to create an index’. But there's no contract level check which checks the amount is paid. In this case the index can be created by anyone without paying astra tokens as addPublicPool() is a public function.

Recommendation

Add a contract level check to check if the user paid the astra token fee required for creating the index.

Status

Resolved

H.8 Take address and uint type instead of arrays

poolIn() is taking an array of _tokens and _values , the functionality of poolIn() requires that it _tokens and _values length should be less than 2 i.e 1 and it checks it by restricting length of these arrays to <2 in require check. As only one address and uint is needed and arrays can consume more gas than a single address type variable, input parameters of poolIn() can be changed to address and uint type instead of address[] and uint[].

Recommendation

Change the input parameters to address and uint types as suggested above.

Status

Resolved

Informational Issues

H.9 Remove isEnabled from PoolUser struct

Description

The is enabled boolean variable in PoolUser struct is not getting used and can be removed

Remediation

Remove redundant is enabled boolean variable.

Status

Resolved

H.10 Unused events

Description

Event WithdrawnToken() on L 587 is never used in contract hence is redundant.

Remediation

Consider removing the declared WithdrawnToken event if not getting used.

Status

Resolved

H.11 Redundant variables assignment

Description

1. These are some redundant variable assignments:
   On L 876 and L 877 redundant assignment of values to storage variables : returnedTokens and returnedAmounts is getting used and are declared on L 872, 873 . and are getting assigned to _TokensStable and _ValuesStable which are storage variables, after that these variables are getting used to push baseStableCoin address and weight respectively which is getting passed to swap() call on L889.
2. Redundant assignments in updatePool(): _tokens and _weights local variables are getting assigned to other local variables newTokens and newWeights. These are local to local variable assignments and are redundant.
3. L 760 in buytokens() assignment of "buf3" memory to "buf" storage type variable.
4. In withdraw() on L266 while calculating value of earlyfees, earlyfees local variable which would be 0 is getting added in calculation. As the earlyfees is getting declared in the function and never had any value assigned to it before,it is redundant addition before assignment.

```
uint256 earlyfees;
uint256 pendingEarlyfees;
// Check if user actually make profit or not.
if(_totalAmount>_balance){
        // Charge the performance fees on profit
        fees = _totalAmount.sub(_balance).mul(IPoolConfiguration(_poolConf).getperformancefees()
}

earlyfees = earlyfees.add(calculatefee(msg.sender,_totalAmount.sub(fees),_poolIndex));
```

Remediation

1. Here, these redundant assignments to storage variables can be removed and memory variables can be used instead.
2. Assignment of local to local variable can be removed and local variable can be used directly.
3. Check and remove assignment to storage variable if not needed.
4. Don’t add earlyfees to the calculated fee.

Status

Resolved

H.12 Redundant type casting

Description

1. In poolIn() on L897 , _tokens array and baseStableCoin are both address types and there's no need to cast them again to address type while comparing
2. In getItokenValue() on L840 in the condition the typecasting of 0 to uint is not needed and can be removed.

Remediation

Remove the address() and uint() used for typecasting as mentioned above.

Status

Resolved

H.13 Redundant function

Description

The withdrawPendingAmount() internal function is not getting used in the contract.

Remediation

Consider removing redundant functions.

Status

Resolved

H.14 Average number of blocks per day can vary

Description

It's not guaranteed that after exactly 6 months these blocks will reach the 6500*182 count as block creation time depends on mining process and network activity hence blocks per day can vary.

Remediation

Timestamp can be used instead of block.number to get an accurate measure of 6 months as timestamps can be trusted by +- 15 mins.

Status

Acknowledged

H.15 Limitation while setting/finding percentage

Description

In calculatefee() the fee is calculated by getting the fee percent value from PoolConfiguration using this on L735: 'IPoolConfiguration(_poolConf).getEarlyExitfees()'
PoolConfiguration has a function named updateEarlyExitFees() which can be used to set the fee percent value. It is required that the value should be less than 100 and in poolv2, the contract uses 100 as denominator which means the percentage is getting calculated on the scale of 100 (100 would be 100% of fee).

The fees cant be set less than 1 percent here as solidity does not support decimals so we need to multiply and divide with more than 2 zeros.
So here while setting fee in poolConfiguration less than 1000 can be allowed.
Let's say, for finding 0.5 percent fee of an amount (let's say 100). 5 can be set as a fee rate.
So, 100e18 * 5 / 1000 gives 5e+17 (0.5 in ether form)
In the formula amount will get multiplied with the fee rate and this will get divided by 1000.

```
733         function calculatefee(address _account, uint _amount,uint _poolIndex)internal view returns(uint256){
734                 // Calculate the early eit fees based on the formula mentioned above.
735                 uint256 feeRate = IPoolConfiguration(_poolConf).getEarlyExitfees();
736                 uint256 startBlock = initialDeposit[_account][_poolIndex];
737                 uint256 withdrawBlock = block.number;
738                 uint256 Averageblockperday = 6500;
739                 uint256 feeconstant = 182;
740                 uint256 blocks = withdrawBlock.sub(startBlock);
741                 uint feesValue = feeRate.mul(blocks).div(100);
742                 feesValue = feesValue.div(Averageblockperday).div(feeconstant);
743                 feesValue = _amount.mul(feeRate).div(100).sub(feesValue);
744                 return feesValue;
745         }
```

Remediation

Verify the business logic , in case there's need of setting fee in decimals, the denominator value can be increased as shown in description. Also to protect user finds, it is recommended to have a check where fees can't be set more than 10-20%.

Status

Resolved

H.16 require statement without error message

Description

L1122 has this check require(_tokens[i] != ETH_ADDRESS && _tokens[i] != WETH_ADDRESS) which checks if the token is not ETH and WETH address. The required function doesn't have an error message so if the transaction reverts, it would be difficult to debug.

Remediation

Consider adding an error message to avoid this confusion when the transaction reverts.

Status

Resolved

H.17 Require check can be added

Description

In case user deposits something and pendingAmount is not 0, then while withdrawing, the user can't use stakePremium=true because totalAmount for that user would be 0 and buyAstraToken() will try to call the getBestExchangeRate() on swap contract *(swapv2.getBestExchangeRate() => swapv2.getV3Rate() => uniswapV3Quoter.quoteExactInput)* which will revert as it will call swap() with 0 amount.
References: *v3-core/UniswapV3Pool.sol* at main

Remediation

Add require check which will revert the transaction when totalAmount will zero and stakePremium would be true while depositing.

Status

Resolved

I. Contract - Governance

High Severity Issues

I.1 Approved proposals maybe impossible to queue, cancel or execute

The propose() function of the GovernorAlpha contract allows proposers to submit proposals with an unbounded amount of actions. Specifically, the function does not impose a hard cap on the number of elements in the arrays passed as parameters (i.e., targets, values, signatures and calldata).
As a consequence, an approved proposal with a large number of actions can fail to be queued, canceled, or executed. This is due to the fact that the queue, cancel and execute functions iterate over the unbounded targets array of a proposal, which depending on the amount and type of actions, can lead to unexpected out-of-gas errors.

Recommendation

To avoid unexpected errors in approved proposals, consider setting a hard cap on the number of actions that they can include.

Status

Resolved

Medium Severity Issues

I.2 Votes can be double spent

Description

The function castVote() and castVoteBySig() uses chef's stakingScoreAndMultiplier() function to calculate the votes. In this function, a user can call this vote function multiple times which can keep adding up the votes multiple times for a single user. In case of castVoteBySig(), signatures can be copied from transaction data and used by the proposer to make the proposal win.

Remediation

It is recommended to check if the user has already voted on a proposal, the total votes should be overwritten instead of adding them if the whole amount is considered when calculating the votes.

Status

Resolved

Low Severity Issues

I.3 Flash-loan protections

While calling castVote(), users' votes are calculated using `stakingScoreAndMultiplier()` from chefV2 contract. As this function recalculates score on withdrawals and deposits, it is possible to manipulate it in a single transaction affecting the number of votes casted.

Recommendation

To prevent flash-loans of voting power to negatively affect governance dynamics, the governance contract should ensure withdrawal and deposits should not happen in the same block or make sure, the contract uses scores from the previous block instead of the current block.

Status

Resolved

I.4 Missing check for voters for first 90 days

It is mentioned in whitepaper that for 90 days, only top holders will be voting on DAO but no such check was found in contract:

During the first 90 days after the Astra network goes live, DAO governance will be performed by the top 100 wallets with the highest number of staked Astra tokens. After the first 90 days, there will be no limitations, and everyone can participate

Recommendation

It is recommended to check if the total time since deployment is less than or equal to 90 days, only top stakers should be able to vote. Missing this check can lead to external votes which can impact the proposal decision as well.

Status

Resolved

Comments: Astra Dao team mentioned that this condition is valid for first 90 day and the initial launch already happened in August 2022.

I.5 Anyone can cancel the proposal

The function cancel() in GovernorAlpha is an external function without msg.sender check.

Recommendation

It is recommended to check if the proposal is canceled either by the proposer or admin as AstraDAO takes decisions on fundamental changes with governance. Proposals can be manipulated which can lead to loss of trust from stakers.

Status

Resolved

I.6 Unused return value in castVoteBySig() function

In function castVoteBySig() function is returning _castVote() function but function definition of castVoteBySig() has not defined any return value.

Recommendation

Consider adding return value variable in function

Status

Resolved

I.7 Value do not match according to whitepaper

In governance contract proposalTokens value is set as 50000000*10**18 ie., 5e25 but in white paper the value is 5000000000*10**18 ie., 5e27

Recommendation

Please consider changing value in contract according to the whitepaper.

Status

Resolved

Informational Issues

I.8 Unused variable in contract

Description

There are some variables which are not used in the contract like:
- IHolders public topTraders;
- uint256 public startTime; (only initialized and not used in the contract context)

Remediation

It is recommended to remove unused variables from the contracts.

Status

Resolved

I.9 Transaction-Ordering Dependence

Description

Timlock executes the data passed from governance in the execute() function in the same order it was declared.

Remediation

It is important to take care of the order while creating a proposal to ensure it doesn't revert or get blocked due to this.

Status

Acknowledged

Comments: Astra Dao team mentioned this will be taken care of off-chain.

J. Contract - Timelock

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

K. Contract - BatchVote

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

K.1 Gas Limit and Loops

Description

The function castVoteBySigs() uses `for` loop without max length check.

Remediation

To avoid out of gas reverts, it is recommended to have a max length check.

Status

Resolved

L. indicesFeesSplit

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

L.1 Events can be added for critical actions.

Whenever certain significant privileged actions are performed within the contract, it is recommended to emit an event about it. When these events are thoroughly emitted in contracts, it makes it easier to query events on-chain.

Recommendation

Consider emitting an event whenever certain significant changes are made in the contracts which need to be notified or noted.

Status

Resolved

L.2 Redundant calculation

In checkUpkeep() function, subtraction and addition of remaining variables is happening on L95 and L96 with _currentContractBalance and rewardAdded respectively.
After every distribute() function call, the remaining variable’s value becomes 0 so in this way it's subtracting and adding 0 which is redundant.

Recommendation

Verify the logic and remove the redundant subtraction and addition of variables.

Status

Resolved

M. Common Issues

High Severity Issues

No issues found

Medium Severity Issues

M.1 Centralisation issue in contracts

Description

Contracts such as indicesPayments, itokendeployer, chefv2, itoken-staking, poolConfiguration contain functions which are only controlled by admin/owner. If for some reason a private key is leaked for the owner then a malicious user can take over and control some part of the project and cause issues to the protocol.

Remediation

To remediate the issue we suggest using multisig.

Status

Resolved

Comments: AstraDAO team will transfer the ownership to DAO and the community will have the control of the complete platform.

M.2 Use of old solidity version(s)

Description

Some contracts are using old solidity versions mentioned below :
- IndicesPayment, IToken, PoolConfiguration, PoolV2, governance, timelock, : 0.5.17
- SwapV2, ItokenStaking, chefv2,uniswapAmount, BatchVote: 0.6.12
Using an old version prevents access to new Solidity security checks.

Remediation

Use the latest solidity compiler version in order to avoid bugs introduced in older versions.

Status

Resolved

Low Severity Issues

M.3 Contracts without storage gap.

Some contracts are inheriting ReentrancyGuard , a context contract. This ReentrancyGuard, context doesn't have a storage gap. It may cause a storage collision while upgrading the logic contract that is inheriting this ReentrancyGuard.

For example: The problem can occur because there is no storage gap variable in the currently used ReentrancyGuard So in the case where in an updated/new logic contract newly used ReentrancyGuard introduces new storage variable(s) and the layout that proxy would be using would be according to the old logic contract where it will start other variable values in the layout after the first (and only one) variable value (_notEntered) of ReentrancyGuard.

Same can happen while adding more storage variables in context contract

Recommendation

Use updated ReentrancyGuardUpgradeable , ContextUpgradeable with storage gaps from OpenZeppelin. Also check other inherited contracts and use them from OpenZeppelin upgradeable contracts if the contract that is inheriting functionality is upgradeable contract logic.

Status

Resolved

M.4 Deposited amount should be greater than 0.

In chefv2 and itoken-staking, deposit functionality allows to deposit 0 amount in the pool. require check can be added to prevent from unintended outcomes if user deposits 0 amount.

Recommendation

Add require check to check deposited amount is greter than 0, if not it should revert.

Status

Resolved

Informational Issues

M.5 Variables can be changed to immutable to save gas

Description

Some variables are only changed once in contract but are public which takes a bit more gas than immutable.

a. IUniswapV2Router public sushiswapRouter, IUniswapV2Router public uniswapV2Router, IUniswapV3Router public uniswapV3Router in swapV2 contract
b. coolDownPeriodTime, coolDownClaimTime in chefV2 contract

Remediation

Consider changing variable types to immutable if only changed once in contract.

Status

Resolved

Comments: Due to different networks and initialisation in constructor, not possible to make it immutable or constant.

M.6 Comments are not matching with functionality

Description

Some comments in the code are not matching the smart contract functionality and which can create confusion sometimes

- buytokens() contains a comment that `this can only be called by poolIn function` but updatePool is also calling buyTokens().

Remediation

Consider changing comments to avoid confusion.

Status

Resolved

M.7 Missing event emission for important changes

Description

1. For functions updateEarlyExitfees(), updatePerfees(), updateMaxToken(), updateSippagerate(), addStable(), removeStable() there is no event emitted in poolConfiguration contract.

2. For functions set() in chefV2 contract

3. Add events for owner only state changing functions in itoken-staking, indicesPayment.

Remediation

Consider adding events to track changes made to the code.

Status

Resolved

M.8 Unused events

Description

Event WithdrawnToken() on L 587 is never used in contract hence is redundant.

Remediation

Consider removing the declared WithdrawnToken event if not getting used.

Status

Resolved

M.9 Inefficient logic for updating highest stakers

Description

While experimenting with deposit function, we found that as number of users went above 100, the gas limit reached 2 M which will take very high gas for deposits. Refer the screenshot attached.

Solc version: 0.5.17	Optimizer enabled: true	Runs: 200	Block limit: 10000000 gas

Methods

Contract	Method	Min	Max	Avg	# calls	usd (avg)
ERC20Upgradeable	approve	33674	53574	53215	214	-
ERC20Upgradeable	transfer	38930	65642	65265	214	-
MasterChefV2	add	-	-	171628	1	-
MasterChefV2	deposit	363387	2034465	1179707	201	-
MasterChefV2	setGovernanceAddress	-	-	53576	1	-
MasterChefV2	withdraw	73594	464681	269138	2	-

Deployments

				% of limit	
GovernanceMock	-	-	131683	1.3 %	-
MasterChefV2	-	-	4104493	41 %	-
Token	-	-	1896356	19 %	-

Remediation

Try to optimise it with a more efficient algorithms to avoid high gas cost for deposit on which the whole staking mechanism depends.

Status

Acknowledged

M.10 General recommendations

1. In Poolv2 initialDeposit can be renamed to initialdepositBlocknumber or something meaningful.

2. ! operator can be used instead of comparing the value with == false in indicesPayments contract in deposit function for checking stablecoin to save gas

3. Update the OZ contract version that is getting used is old in future problems may occur when it comes to using old contract/library which may have a bug in old version.

4. Many of the functions from contracts can be made external instead of keeping them as public.
   - swapFromBestExchange() from swapv2
   - addDaaaddress(), updateDecimalValue() from itokendeployer
   - addPublicPool(), updatePool() from poolv2

5. First (0th) pool should be astra's pool.
   - It is necessary that the 0th pool in chefv2 should be astra's pool as in some smart contracts it is assumed that 0 th pool would be astra's pool e.g in pool:stakeAstra() it deposits in 0 pool.
   - 0th pool for astra should be created in initialize() to avoid the possibility of creating the first pool of any other token instead of astra.

6. The updateRewardRate() is getting called twice while calling restake from withdraw. They can be called single time saving gas in both chefV2 and itoken-staking contracts.

7. In chefv2 and itoken-staking, following code can be removed:

```
if (user.maxMultiplier == 0) {
    user.maxMultiplier = MULTIPLIER_DECIMAL;
}
```

8. While rebalancing same token the check can be added to skip the swapping
   In the rebalancing process pool owners need to provide new token addresses so that old index tokens can be converted to new ones, if the same addresses are provided as new addresses the swap2() function swaps old token to base stablecoin and then back to new token which was similar to old one. The code logic can be added to skip the swapping of same tokens.

M.10 General recommendations

9. Contracts can be divided into multiple files(like itoken-staking and chefv2 were very similar, so fixes or changes can be missed hence recommended to use a common helper contract instead of same code in both the contracts

10. The contracts use block.timestamp and block.number, which are not good proxies for time because of issues with synchronization, miner manipulation and changing block times hence should not be relied on for precise calculations of time.

11. Follow the proper naming convention and style from the *Solidity style guide*, which will help readers to understand contract functionality more easily.

Functional Testing

Some of the tests performed are mentioned below

itoken-staking

- Test to check how balance will work out
- Check if difference in pool and amount yields in same rewards
- check rewards after changing decimals
- checking PID if it reverts

poolV2

- More than one address should be able to withdraw from single pool
- Multiple user should be able to deposit more than one time
- Should revert when reentered while depositing using ERC777 hooks
- Should be able to deposit with multiple pools
- Half of the fees are getting decreased when user is withdrawing after 3 months
- Fees are getting decreased when user is withdrawing ahead in the time (after 6 months)
- Should be able to buy and stake astra while withdrawing (stakePremium is true)
- Should be able to buy and stake astra while withdrawing (stakeEarlyFees is true)
- Should revert if totalAmount is zero and stakePremium is true while withdrawing

chefV2

- Checking restakeAstraReward() for correct multipliers
- Check max multiplier with restake
- Check max multiplier with nft restake
- Check max multiplier with nft restake with increased time
- Check max multiplier with nft restake for 12 months
- Check restakeAstraReward for 12 months
- Test max multiplier(NFT) when it will increase
- Test max multiplier with nft restake with increased time(Multiplier will increase)
- Checking restakeAstraReward for multiple users (multiplier will increase)

Automated Testing

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Astra Dao. We performed our audit according to the procedure described above.

Some issues of High,Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Astra Dao Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Astra Dao Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

700+
Audits Completed

$16B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
March, 2023

For

ASTRA DAO

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com