Cyberscope

Audit Report

aiPX

December 2023

Repository https://github.com/0xytocinOfficial/aipx-flare-core-contracts

Commit d71b5d65289bbabe2ea82e476a3c7948154b99a0

Audited by © cyberscope

Table of Contents

Review

Repository	https://github.com/0xytocinOfficial/aipx-flare-core-contracts
Commit	d71b5d65289bbabe2ea82e476a3c7948154b99a0

Audit Updates

Initial Audit	20 Dec 2023

Source Files

Filename	SHA256
LyAipx.sol	894ede7bfffb9c23c673bd7e898d3ade347 80fd198e592d8f7f0bdf10604b3ee
GovernancePowerDelegationERC20.sol	9af6668ebb8303bcb31178b4c87d864b15 170c8cc05a0c9f488841bc6ba9e02d
Erc20Reserve.sol	5136b7736a647ef35f964654ca3df2d5177 212a917817a947d510b54a74c7b82
AlpRewardDistributor.sol	90d905401b3acc17d36bb21fb1c4a817a7 e0cb0a1558eb84b118eb3250d36128
AipxToken.sol	2be01d46c5f33750016f0f0d48f9fdd0c3f5 9efe301e60c3210e33838b948746
AipxStaking.sol	7d1020ef7294dd3629b2b9a1aea40c8cdd 161c1feef30632e4c6d65d0f714da0
AipxReferralRegistry.sol	6fd3c806fe35382ac9655b0f725f0711284c 26c22cdd7731424766d0ce821b91
AipxReferralController.sol	96a40283bd7afd4cfea2c116e56375988ee cc56b39b293b26c75d6b6d4800134

AipxMaster.sol	c4916676ab917cf3c63f7e736f2488ebd6f7e495c694cbfcaae420c0693dbcfc
AipxGovernance.sol	6bd82ac7e188925d3bd8dcdde0d893197e375b68f23420931276e8b0aa899414
AgoStaking.sol	aa26e0714a17b291728d9b346f5234bb9ea246d743bf561e3e3b074d5a4863e2
AIPXStake.sol	bee08aae79df02ec5ebaf57a6460fadb37141115abbe6e7e6bd86cb6d8e99cb9
AIPXOracle.sol	dc4241467faba8b4c3fc221fde6182a3f110ab3a59e4c1d1534e80eb7f4e4f2e
tokens/LyAipx.sol	894ede7bfffb9c23c673bd7e898d3ade34780fd198e592d8f7f0bdf10604b3ee
tokens/AipxToken.sol	2be01d46c5f33750016f0f0d48f9fdd0c3f59efe301e60c3210e33838b948746
tokens/AipxGovernance.sol	6bd82ac7e188925d3bd8dcdde0d893197e375b68f23420931276e8b0aa899414
referral/AipxReferralRegistry.sol	6fd3c806fe35382ac9655b0f725f0711284c26c22cdd7731424766d0ce821b91
referral/AipxReferralController.sol	96a40283bd7afd4cfea2c116e56375988eecc56b39b293b26c75d6b6d4800134
oracle/AIPXOracle.sol	dc4241467faba8b4c3fc221fde6182a3f110ab3a59e4c1d1534e80eb7f4e4f2e
lib/GovernancePowerDelegationERC20.sol	9af6668ebb8303bcb31178b4c87d864b15170c8cc05a0c9f488841bc6ba9e02d
interfaces/IWETH.sol	c115b7a379d4e125ab5e022fd290d7417e6a1832c75b318d63bfdfbcc97a829f
interfaces/IRewarder.sol	4d6acb83c17800b8cd79bdaf337de2ae6c635bc58f9b21fd65e9226434efbbfc

interfaces/IPriceFeed.sol	159cadc9afc6b4a3d7a2e296828b09c7f885dcdc5e249500920d24daf2a90ea5
interfaces/IPool.sol	fe71928d0ea00e97a041f7ce667e6c5ec85fd759283458a1ab683a66c61a7447
interfaces/ILPToken.sol	80a7cc1d1c292b38333f7d99e528263dd145f96be6f8221c795676d47133baf6
interfaces/IGovernancePowerDelegationToken.sol	bda36291cd324e76ec6cbdb5404e4b999b19993632f531001738e1c2a3ccc850
interfaces/IETHUnwrapper.sol	38f80942effc64ca7fb46199d9aa7150bc1f8f1c88f3789817f7218b0f7c6be0
interfaces/IBurnableERC20.sol	ff16001feaa334410cb2be8cdabaf5abed38203fb137f24101348ff2ed8d5994
interfaces/IAlpRewardDistributor.sol	bb6fcca5e262727e06f360f474f29a4e0e9a85107c061b1017b560abe42bb7ad
interfaces/IAipxReferralRegistry.sol	f6b43af24ef9e40ac6b93eeaed191fd31acc7242e30e5f68a18e7eae67701d9b
interfaces/IAIPXStake.sol	5c997e9ef491473dc42fc7d97aa146fd72aa703ddc88d0bcc09153a514b0a3f9
interfaces/IAIPXOracle.sol	436fd2d07f306fda5655721ff072912dacf9e6a702f9aede95b3ab017083a488
interfaces/IAGOToken.sol	bf78fc43d7f60a984e85629655cecd645bce8612b0118171bda9960f6e61d013
fund/Erc20Reserve.sol	5136b7736a647ef35f964654ca3df2d5177212a917817a947d510b54a74c7b82
fund/AlpRewardDistributor.sol	90d905401b3acc17d36bb21fb1c4a817a7e0cb0a1558eb84b118eb3250d36128
farm/AipxStaking.sol	7d1020ef7294dd3629b2b9a1aea40c8cdd161c1feef30632e4c6d65d0f714da0

farm/AipxMaster.sol	c4916676ab917cf3c63f7e736f2488ebd6f7e495c694cbfcaae420c0693dbcfc
farm/AgoStaking.sol	aa26e0714a17b291728d9b346f5234bb9ea246d743bf561e3e3b074d5a4863e2
farm/AIPXStake.sol	bee08aae79df02ec5ebaf57a6460fadb37141115abbe6e7e6bd86cb6d8e99cb9

Overview

aiPX stands as a revolutionary decentralized, non-custodial perpetual DEX, that is redefining the landscape of decentralized trading. At its core, aiPX focuses on offering an unparalleled trading experience for perpetual contracts, a domain traditionally dominated by centralized platforms. This DEX stands out for its commitment to risk management and its innovative approach to liquidity provision, catering to both traders and liquidity providers.

One of the standout features of aiPX is its decentralized perpetual exchanges. This functionality allows users to engage in perpetual trading contracts directly on the blockchain, bypassing the need for intermediaries. This direct approach not only empowers users with full control over their funds but also enhances the security and transparency of the trading process.

Risk management is a cornerstone of the aiPX platform. Users are equipped with a custom risk management system, enabling them to set and manage their own risk parameters. This level of customization ensures that traders can adopt strategies that align with their individual risk appetites, making aiPX a versatile platform for traders of varying experiences and styles.

In addition to its trading solutions, aiPX introduces an innovative liquidity solution for liquidity providers. This system ensures deep and liquid markets across various trading pairs, significantly reducing price impact and slippage. As a result, traders enjoy a more seamless and efficient trading experience, while liquidity providers benefit from passive earnings opportunities.

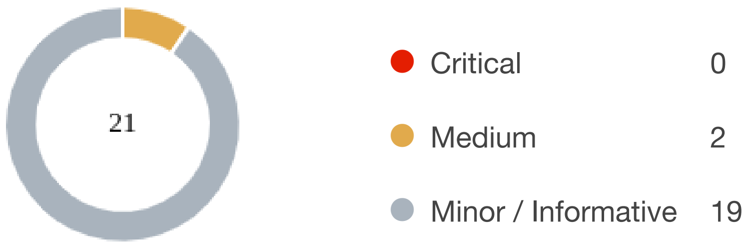
The ecosystem of aiPX is further enriched by the AIPX token, which plays a pivotal role in enabling a robust, real yield ecosystem. This token not only incentivizes liquidity provision but also opens up avenues for capital-efficient hedging with near-zero market impact, a feature particularly beneficial for sophisticated traders.

In summary, aiPX, offers a cutting-edge solution for decentralized perpetual trading. Its focus on risk management, liquidity provision, and an expanding suite of trading options positions it as a leading choice for traders seeking a decentralized, efficient, and versatile trading platform.

Audit scope

The current audit report is specifically focused on the contract files provided. The functionality of these contracts is significantly dependent on several external contracts, namely `aipxPool` , `weth` , `pool` , `rewardToken` and `ethUnwrapper` . These external contracts are initialized during the initialization phase of the audited contracts. Consequently, the addresses and the internal workings of `aipxPool` , `weth` , `pool` , `rewardToken` and `ethUnwrapper` fall outside the scope of this particular audit. This means that while the provided contracts are thoroughly examined for security and functionality, any interactions, dependencies, or integrations with the aforementioned external contracts are not covered in this audit report. This limitation should be taken into consideration when interpreting the findings and conclusions of this audit.

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	2	0	0	0
Minor / Informative	19	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	SRV	Signature Replay Vulnerability	Unresolved
Medium	UVF	Uninitialized Variable Functionality	Unresolved
Minor / Informative	ISV	Insufficient Stake Validation	Unresolved
Minor / Informative	TSI	Tokens Sufficiency Insurance	Unresolved
Minor / Informative	DSM	Documentation Supply Mismatch	Unresolved
Minor / Informative	RTI	Reward Transfer Inconsistency	Unresolved
Minor / Informative	OEH	Optimize Error Handling	Unresolved
Minor / Informative	CCR	Contract Centralization Risk	Unresolved
Minor / Informative	RRC	Redundant Require Check	Unresolved
Minor / Informative	RCS	Redundant Code Segments	Unresolved
Minor / Informative	CR	Code Repetition	Unresolved
Minor / Informative	RFP	Redundant Function Parameter	Unresolved
Minor / Informative	MPV	Missing Parameter Validation	Unresolved
Minor / Informative	DTI	Data Type Inconsistency	Unresolved

●	RC	Repetitive Calculations	Unresolved
●	MU	Modifiers Usage	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved

SRV - Signature Replay Vulnerability

Criticality	Medium
Location	AipxGovernance.sol#L23
Status	Unresolved

Description

The contract, reveals a concern regarding the initialization of the `DOMAIN_SEPARATOR` value, which plays a pivotal role in the signature process. This value is intended to ensure that signatures are exclusive to a specific protocol, but in this case, it has not been initialized. As a result, this oversight allows for the possibility of signature replay attacks across different applications that utilize the same signature format. The exploit scenario demonstrates this vulnerability:

1. Alice signs a permit message to Bob in protocol X, which also employs a nonce system. However, the `DOMAIN_SEPARATOR` in protocol X is uninitialized and defaults to zero. Since this is Alice's first signed message in Protocol X, the nonce is zero. The `PERMIT_TYPEHASH` in protocol X is identical to that used in common ERC-20 functions.
2. Bob, aware that Alice holds LGO tokens, can exploit this vulnerability.
3. Bob replays Alice's signature by calling LevelGovernance.permit(), leveraging the identical PERMIT_TYPEHASH and the uninitialized DOMAIN_SEPARATOR.

```
bytes32 public DOMAIN_SEPARATOR;

 function permit(address owner, address spender, uint256 value, uint256
deadline, uint8 v, bytes32 r, bytes32 s)
        external
    {
        require(owner != address(0), "INVALID_OWNER");
        require(block.timestamp <= deadline, "INVALID_EXPIRATION");
        uint256 currentValidNonce = _nonces[owner];
        bytes32 digest = keccak256(
            abi.encodePacked(
                "\x19\x01",
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender,
value, currentValidNonce, deadline))
            )
        );

        require(owner == ECDSA.recover(digest, v, r, s),
"INVALID_SIGNATURE");
        _nonces[owner] = currentValidNonce + 1;
        _approve(owner, spender, value);
    }
```

Recommendation

It is recommended to initialize the `DOMAIN_SEPARATOR` in the contract's initializer to mitigate this vulnerability. This initialization will ensure that signatures are bound to the specific protocol, preventing their reuse in other applications. This upgrade should include the ability to set the initializer, thereby rectifying the oversight and enhancing the security of the signature process.

UVF - Uninitialized Variable Functionality

Criticality	Medium
Location	AIPXStake.sol#L50,52,113,221AipxStaking.sol#L94
Status	Unresolved

Description

The contract is currently including the `stakingTax` and `auctionTreasury` variables, which are initialized to zero and remain unmodified throughout the contract's code. This static state results in these variables always equating to zero, rendering their associated functionalities ineffective. For instance, the calculation `_taxAmount = _amount * stakingTax / STAKING_TAX_PRECISION` will always result in a zero value for `_taxAmount` due to `stakingTax` being zero. Similarly, the `auctionTreasury` address is never set, which could lead to issues in the `reserveAuctionFund` function. The presence of these unutilized variables not only adds unnecessary complexity but also can lead to misunderstandings about the contract's intended behavior.

```
address public auctionTreasury;

uint256 public stakingTax;

uint256 _taxAmount = _amount * stakingTax / STAKING_TAX_PRECISION;
...
function reserveAuctionFund(uint256 amount) external onlyOwner {
    AGO.safeTransfer(auctionTreasury, amount);
    emit AuctionFundReserved(amount);
}
```

```
uint256 _taxAmount = _amount * stakingTax / STAKING_TAX_PRECISION;
```

Recommendation

It is recommended to reconsider the usages of the `stakingTax` and `auctionTreasury` variables. If these variables are intended to play a significant role in the contract's functionality, appropriate mechanisms for setting and updating their values should be implemented. This could involve adding functions or conditions that allow for the dynamic modification of these variables. Alternatively, if these variables are not needed, removing them from the contract would simplify the code and reduce potential confusion.

ISV - Insufficient Stake Validation

Criticality	Minor / Informative
Location	AIPXStake.sol#L130AipxStaking.sol#L107AgoStaking.sol#L77
Status	Unresolved

Description

The contract is currently designed to handle the unstaking process in the `unstake` function. This function allows a user to unstake a specified amount of tokens. However, the contract lacks of validation to ensure that `msg.sender` actually has a staked balance before proceeding with the unstaking operation. As it stands, the function calculates the amount to unstake and the pending rewards, adjusts the user's staked amount and reward debt, and then transfers the unstaked tokens and any pending rewards to the specified address. The function emits the `Unstaked` event regardless of whether the user had an actual staked balance or not. This can lead to misleading event logs, as the `Unstaked` event could be emitted even when no tokens have been unstaked, potentially causing confusion and inaccuracies in tracking staked tokens.

```
    function unstake(address _to, uint256 _amount) external override {
        update();
        require(_amount != 0, "INVALID_AMOUNT");
        UserInfo storage user = userInfo[msg.sender];
        ...

        IERC20(AIPX).safeTransfer(_to, amountToUnstake);

        emit Unstaked(msg.sender, _to, amountToUnstake);
    }

    function unstake(address _to, uint256 _amount) external nonReentrant
{
        ...
        AIPX.safeTransfer(_to, _amount);
        emit Unstaked(sender, _to, _amount);
    }

    function unstake(address _to, uint256 _amount) external nonReentrant
{
        ...
        ago.safeTransfer(_to, _amount);
        emit Unstaked(sender, _to, _amount);
    }
```
`

Recommendation

It is recommended to introduce additional checks within the unstake function to verify that `msg.sender` has an actual stake balance before proceeding with the execution of the function. This can be achieved by adding a condition to confirm that the user's staked amount is greater than zero. Such a validation will prevent the function from executing if the user does not have any tokens staked, thereby ensuring that the `Unstaked` event is emitted only when an actual unstaking action has occurred. This enhancement will improve the accuracy of the contract's event logs and the overall integrity of the staking process.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	LyAipx.sol#L215
Status	Unresolved

Description

The `rewardToken` are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

```
    function claimRewards(uint256 _batchId, address _receiver) external {
        ...
        if (enableStaking) {
            rewardToken.safeIncreaseAllowance(address(aipxStake),
amount);
            aipxStake.stake(_receiver, amount);
        } else {
            rewardToken.safeTransfer(_receiver, amount);
        }
        emit Claimed(sender, _batchId, amount, _receiver);
    }
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the presale tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

DSM - Documentation Supply Mismatch

Criticality	Minor / Informative
Location	AipxToken.sol#L10
Status	Unresolved

Description

The contract is currently setting the maximum supply of the `AipxToken` to `50,000,000 ether`, as indicated by the `MAX_SUPPLY` constant in the contract code. However, this implementation is inconsistent with the project's official documentation (https://aipx.gitbook.io/aipx-documentation/tokenomics/aipx-utility-token), which states that the total supply of the AIPX Token should be `10,000,000`. Such discrepancies between the code and the documentation can lead to confusion and potentially undermine the trust of users and investors in the project. It is crucial for the code to accurately reflect the intended tokenomics as outlined in the project's documentation to maintain credibility and ensure that stakeholders have a clear understanding of the token's supply mechanics.

```
Total Supply: 10,000,000
...
uint256 public constant MAX_SUPPLY = 50_000_000 ether;
```

Recommendation

It is recommended to update the `MAX_SUPPLY` constant in the `AipxToken` contract to reflect the documented total supply of 10,000,000. This change will align the contract's implementation with the project's official documentation, thereby resolving the inconsistency. After making this modification, it is essential to conduct a thorough review and testing to ensure that the change does not adversely affect other aspects of the contract or the project's overall functionality. Additionally, communicating this change to stakeholders and updating any relevant materials or references to the token supply will be important to maintain transparency and trust.

RTI - Reward Transfer Inconsistency

Criticality	Minor / Informative
Location	AIPXStake.sol#L171,249
Status	Unresolved

Description

The contract is designed to manage the distribution of AGO rewards to users who stake AIPX tokens within the `AIPXStake` contract. It facilitates the transfer of rewards under three scenarios, staking, unstaking, and upon user request through the `claimRewards` function. A critical function in this process is the `AIPXStake._safeTransferAGO` , which ensures that the number of tokens transferred does not exceed the contract's balance. However, when the contract's balance is insufficient to cover the full amount of a user's reward, the user's `rewardDebt` is updated as though the entire reward amount was transferred, even though the actual transfer was less or did not occur due to the lack of sufficient funds.

This discrepancy between the recorded `rewardDebt` and the actual reward transferred can lead to inaccurate accounting and potential dissatisfaction among users. Their expected rewards may not match the actual rewards received. Furthermore, the `RewardsClaimed` event will emit the incorrect value, as it reflects the calculated reward rather than the actual amount transferred. This can mislead users and external observers about the true state of rewards distribution.

Additionally, the `pendingReward` function, which calculates the reward amount a user is entitled to, can also project an inflated reward. This happens if the balance of the contract is less than the calculated reward, leading to further discrepancies between expected and actual rewards.

```
    function claimRewards(address _to) external {
        update();
        UserInfo storage user = userInfo[msg.sender];

        uint256 accumulatedReward = uint256((user.amount *
accRewardPerShare) / ACC_REWARD_PRECISION);
        uint256 _pendingReward = uint256(accumulatedReward -
user.rewardDebt);

        user.rewardDebt = accumulatedReward;

        if (_pendingReward != 0) {
            _safeTransferAGO(_to, _pendingReward);
            emit RewardsClaimed(msg.sender, _to, _pendingReward);
        }
    }

    function _safeTransferAGO(address _to, uint256 _amount) internal {
        require(AGO != IERC20(address(0)), "AGO not set");
        uint256 agoBalance = AGO.balanceOf(address(this));
        if (_amount > agoBalance) {
            AGO.safeTransfer(_to, agoBalance);
        } else {
            AGO.safeTransfer(_to, _amount);
        }
    }

    function pendingReward(address _to) external view returns (uint256)
{
        UserInfo storage user = userInfo[_to];
        uint256 aipxSupply = AIPX.balanceOf(address(this));
        ...
        }
        return ((user.amount * _accRewardPerShare) /
ACC_REWARD_PRECISION) - user.rewardDebt;
    }
```

Recommendation

It is recommended to adjust the user's `rewardDebt` based on the actual amount of AGO tokens transferred, rather than the intended reward amount. This adjustment should be implemented within the `claimRewards` and `_safeTransferAGO` functions. By doing so, the rewardDebt will accurately reflect the rewards that have been physically transferred to the user. Additionally, the `RewardsClaimed` event should be modified to emit the actual amount of AGO tokens transferred. This change will enhance the transparency and fairness of the reward distribution process, aligning the recorded transactions with the actual state of the contract's balance and user rewards. It will also ensure that the `pendingReward` function provides a more accurate estimation of the rewards a user can expect to receive, based on the current contract balance.

OEH - Optimize Error Handling

Criticality	Minor / Informative
Location	Erc20Reserve.sol#L13
Status	Unresolved

Description

The contract is currently utilizing if statements followed by revert for error handling in its transfer function. This approach, while valid, is less efficient compared to the use of require statements. The `require` statement is a more concise and gas-efficient way to validate conditions and revert transactions with error messages. Using `if-revert` patterns can lead to increased complexity and higher gas costs, as each condition is checked separately. In contrast, require statements provide a cleaner, more readable way to assert conditions and automatically revert the transaction if the condition is not met, along with a specified error message.

```
    function transfer(IERC20 _token, address _to, uint256 _amount)
external onlyOwner {
        if (address(_token) == address(0)) {
            revert InvalidToken();
        }

        if (_to == address(0)) {
            revert InvalidAddress(_to);
        }

        if (_amount == 0) {
            revert InvalidAmount();
        }

        _token.safeTransfer(_to, _amount);
        emit Distributed(_to, _amount);
    }
```

Recommendation

It is recommended to replace the `if-revert` patterns in the transfer function with `require` statements. This change will streamline the function, making it more readable and potentially reducing gas costs due to more efficient error handling.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	AipxReferralController.sol#L189LyAipx.sol#L114
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion. Specifically, since the owner of the contracts is an EOA can arbitrarily change parameters that affect how the contract functionality. Additionally, the minter role address has the authority to mint new tokens in the `LyAipx` contract.

```
function setDistributor(address _distributor) external onlyOwner {
        require(_distributor != address(0),
"AipxReferralController::setDistributor: invalid address");
        distributor = _distributor;
        emit DistributorSet(distributor);
    }

    function setUpdater(address _updater) external onlyOwner {
        require(_updater != address(0),
"AipxReferralController::setUpdater: invalid address");
        updater = _updater;
        emit UpdaterSet(updater);
    }

    function setOracle(address _oracle) external onlyOwner {
        require(
            _oracle != address(0) && _oracle != address(oracle),
"AipxReferralController::setOracle: invalid address"
        );
        oracle = IAIPXOracle(_oracle);
        oracle.update();
        emit OracleSet(_oracle);
    }

    function setEpochDuration(uint256 _epochDuration) public onlyOwner {
        require(
            _epochDuration >= MIN_EPOCH_DURATION,
            "AipxReferralController::setEpochDuration: must >=
MIN_EPOCH_DURATION"
        );
        epochDuration = _epochDuration;
        emit EpochDurationSet(epochDuration);
    }

    function withdrawAIPX(address _to, uint256 _amount) external
onlyOwner {
        require(_to != address(0),
"AipxReferralController::withdrawAIPX: invalid address");
        AIPX.safeTransfer(_to, _amount);

        emit AIPXWithdrawn(_to, _amount);
    }

    function setEnableNextEpoch(bool _enable) external {
        ...
    }
```

```
function mint(address _to, uint256 _amount) external {
 require(_msgSender() == minter, "LyAipx: !minter");
 _mint(_to, _amount);
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

RRC - Redundant Require Check

Criticality	Minor / Informative
Location	AIPXStake.sol#L61,252
Status	Unresolved

Description

The contract is currently implementing a check in the `_safeTransferAGO` function. This function includes a `require` statement to ensure that the AGO address is not zero. However, this check is superfluous since the AGO address is already validated during the contract's initialization in the `initialize` function, where a similar `require` statement ensures that the `_ago` address is not zero. Given that the AGO address cannot be zero after the contract's initialization, the additional `require` statement in the `_safeTransferAGO` function is unnecessary and does not contribute to the contract's security or functionality.

```
    function initialize(address _aipx, address _ago, uint256
_rewardPerSecond) external initializer {
        __Ownable_init();
        require(_rewardPerSecond <= MAX_REWARD_PER_SECOND, ">
MAX_REWARD_PER_SECOND");
        require(_aipx != address(0), "Invalid AIPX address");
        require(_ago != address(0), "Invalid AGO address");
        AIPX = IBurnableERC20(_aipx);
        AGO = IERC20(_ago); //reward token pou den steleneteai sto
contract
        rewardPerSecond = _rewardPerSecond;
    }

    function _safeTransferAGO(address _to, uint256 _amount) internal
{
        require(AGO != IERC20(address(0)), "AGO not set");
        uint256 agoBalance = AGO.balanceOf(address(this));
        if (_amount > agoBalance) {
            AGO.safeTransfer(_to, agoBalance);
        } else {
            AGO.safeTransfer(_to, _amount);
        }
    }
```

`

Recommendation

It is recommended to remove the redundant `require` statement within the `_safeTransferAGO` function. This simplification will streamline the function by eliminating an unnecessary check, thereby reducing the contract's complexity and gas costs associated with this function's execution.

RCS - Redundant Code Segments

Criticality	Minor / Informative
Location	AIPXStake.sol#L155
Status	Unresolved

Description

The contract is currently containing code segments, specifically the `cooldown` and `deactivateCooldown` functions, that do not provide any actual functionality. These functions are essentially placeholders with comments indicating "doing nothing." Such redundant code segments can lead to confusion and misinterpretation of the contract's purpose and functionality. Moreover, they contribute to unnecessary bloat in the contract, potentially impacting its efficiency and clarity.

```
 function cooldown() external override {
        // doing nothing
    }

    function deactivateCooldown() external override {
        // doing nothing
    }
```

`

Recommendation

It is recommended to remove these redundant code segments, namely the `cooldown` and `deactivateCooldown` functions, from the contract. Eliminating these non-functional parts will streamline the contract, making it more efficient and easier to comprehend. This action will also reduce the potential for confusion among users and developers who interact with or audit the contract. After removing these segments, a thorough review and testing should be conducted to ensure that their removal does not inadvertently affect other parts of the contract's functionality.

CR - Code Repetition

Criticality	Minor / Informative
Location	AipxMaster.sol#L147,172
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

Specifically the `pendingReward` and `updatePool` and also the `withdraw` and `_deposit` functions share similar code segments.

```
    function pendingReward(uint256 _pid, address _user) external view
returns (uint256 pending) {
        PoolInfo memory pool = poolInfo[_pid];
        ...
        if (block.timestamp > pool.lastRewardTime && lpSupply != 0)
{
            uint256 time = block.timestamp - pool.lastRewardTime;
            accRewardPerShare = accRewardPerShare
                + (time * rewardPerSecond * pool.allocPoint *
ACC_REWARD_PRECISION / totalAllocPoint / lpSupply);
        }
        ...
    }

    function updatePool(uint256 pid) public returns (PoolInfo memory
pool) {
        pool = poolInfo[pid];
        if (block.timestamp > pool.lastRewardTime) {
            uint256 lpSupply =
lpToken[pid].balanceOf(address(this));
            if (lpSupply != 0) {
                uint256 time = block.timestamp -
pool.lastRewardTime;
                pool.accRewardPerShare = pool.accRewardPerShare
                    + uint128(time * rewardPerSecond *
pool.allocPoint * ACC_REWARD_PRECISION / totalAllocPoint /
lpSupply);
            }
            ...
        }
    }
```

```
    function withdraw(uint256 pid, uint256 amount, address to)
public {
        ...
        // Effects
        user.rewardDebt = user.rewardDebt - int256(amount *
pool.accRewardPerShare / ACC_REWARD_PRECISION);
        user.amount = user.amount - amount;

        // Interactions
        IRewarder _rewarder = rewarder[pid];
        if (address(_rewarder) != address(0)) {
            _rewarder.onReward(pid, msg.sender, to, 0,
user.amount);
        }

        lpToken[pid].safeTransfer(to, amount);

        ...
    }

    function _deposit(uint256 pid, uint256 amount, address to)
internal {
        ...
        // Effects
        user.amount = user.amount + amount;
        user.rewardDebt = user.rewardDebt + int256(amount *
pool.accRewardPerShare / ACC_REWARD_PRECISION);

        // Interactions
        IRewarder _rewarder = rewarder[pid];
        if (address(_rewarder) != address(0)) {
            _rewarder.onReward(pid, msg.sender, to, 0,
user.amount);
        }
        ...
    }
```
`

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

RFP - Redundant Function Parameter

Criticality	Minor / Informative
Location	AipxMaster.sol#L121
Status	Unresolved

Description

The contract is currently using the `overwrite` parameter in the `set` function to determine whether the `_rewarder` should be set. This approach, while functional, introduces unnecessary complexity and an additional parameter that could be avoided. A more streamlined and efficient method would be to check if the `_rewarder` address is non-zero, which inherently indicates the intent to update the `rewarder` variable. This change would simplify the function's logic, making it more intuitive and less prone to errors, especially for users unfamiliar with the contract's intricacies.

```
  function set(uint256 _pid, uint256 _allocPoint, bool _staking,
IRewarder _rewarder, bool overwrite)
        public
        onlyOwner
    {
        require(_allocPoint <= MAX_ALLOCT_POINT, "Alloc point too
high");
        totalAllocPoint = totalAllocPoint - poolInfo[_pid].allocPoint
+ _allocPoint;
        poolInfo[_pid].allocPoint = uint64(_allocPoint);
        poolInfo[_pid].staking = _staking;
        if (overwrite) {
            rewarder[_pid] = _rewarder;
        }
        emit LogSetPool(_pid, _allocPoint, overwrite ? _rewarder :
rewarder[_pid], overwrite);
    }
```

Recommendation

It is recommended to refactor the `set` function by removing the `overwrite` parameter. Instead, the function should check if the `_rewarder` address is different from the zero address (indicating a valid address) to decide whether to update the `rewarder[_pid]`. This approach reduces the function's complexity and the potential for mistakes in passing the correct boolean value for overwriting. It also aligns with common smart contract practices of using address checks to infer intent.

MPV - Missing Parameter Validation

Criticality	Minor / Informative
Location	AipxMaster.sol#L121,147,172
Status	Unresolved

Description

The contract is currently lacking a crucial check in its set function to verify the existence of the `_pid` value within the `poolInfo` struct. This omission can lead to potential risks, such as referencing an uninitialized or non-existent pool, which might result in unexpected behavior in the contract's logic. Ensuring that all referenced elements within a contract are valid and initialized is essential for maintaining the integrity and security of smart contract operations, especially when dealing with complex data structures and state changes.

```
 function set(uint256 _pid, uint256 _allocPoint, bool _staking,
IRewarder _rewarder, bool overwrite)
        public
        onlyOwner
    {
        require(_allocPoint <= MAX_ALLOCT_POINT, "Alloc point too
high");
        totalAllocPoint = totalAllocPoint - poolInfo[_pid].allocPoint +
_allocPoint;
        poolInfo[_pid].allocPoint = uint64(_allocPoint);
        poolInfo[_pid].staking = _staking;
        if (overwrite) {
            rewarder[_pid] = _rewarder;
        }
        emit LogSetPool(_pid, _allocPoint, overwrite ? _rewarder :
rewarder[_pid], overwrite);
    }

    function pendingReward(uint256 _pid, address _user) external view
returns (uint256 pending) {
    ...
    }

    function updatePool(uint256 pid) public returns (PoolInfo memory
pool) {
    ...
    }
    ...
```

Recommendation

It is recommended to implement a validation check in the set function to confirm that the `_pid` value passed as a parameter corresponds to an existing entry in the `poolInfo` struct. This can be achieved by adding a condition to verify the existence of `_pid` before proceeding with any modifications or state changes. Such a check will safeguard against unintended interactions with uninitialized or invalid pool entries, thereby enhancing the contract's robustness and reliability.

DTI - Data Type Inconsistency

Criticality	Minor / Informative
Location	AipxMaster.sol#L38,96
Status	Unresolved

Description

The contract is currently utilizing different data types for the `allocPoint` parameter in various parts of its code. Specifically, `allocPoint` is declared as a `uint64` data type. However, in the `add` function, `allocPoint` is used as a `uint256` data type. This inconsistency in data types can lead to potential issues such as data overflow or underflow, which might result in unexpected behavior or vulnerabilities in the contract's execution. Ensuring data type consistency is crucial for the robustness and security of smart contracts, especially in the context of handling numerical values and calculations.

```
    uint64 allocPoint;

    function add(uint256 allocPoint, IERC20 _lpToken, bool _staking,
IRewarder _rewarder) public onlyOwner {
        ...
    }
```

Recommendation

It is recommended to maintain consistency in the data types used throughout the contract. In this case, aligning the data type of the `allocPoint` parameter in the `add` function with its declaration as `uint64` would be advisable. This change will help prevent potential data type-related issues and enhance the overall integrity and security of the contract. Additionally, thorough testing should be conducted to ensure that this modification does not introduce any new issues or affect the contract's intended functionality.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	AgoStaking.sol#L69,82,92,110
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

```
_userInfo.rewardDebt += int256((_amount * accRewardPerShare) /
ACC_REWARD_PRECISION);
...
_userInfo.rewardDebt -= int256((_amount * accRewardPerShare) /
ACC_REWARD_PRECISION);
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

MU - Modifiers Usage

Criticality	Minor / Informative
Location	AgoStaking.sol#L64,76
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(_amount > 0, "AgoStaking::stake: amount > 0");
require(_amount > 0, "AgoStaking::unstake: amount > 0");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	AIPXOracle.sol#L21oracle/AIPXOracle.sol#L21AipxMaster.sol#L54
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
updater
rewardToken
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	LyAipx.sol#L29,30,31,46AipxStaking.sol#L44AIPXStake.sol#L50,52AipxGovernance.sol#L23
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
address public distributor
uint256 public nextBatchAmount
uint256 public nextBatchTimestamp
IAIPXStake public aipxStake
uint256 public stakingTax
address public auctionTreasury
bytes32 public DOMAIN_SEPARATOR
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	LyAipx.sol#L35,36,37,57,69,73,79,83,96,102,113,198,215,231,237,243,251interfaces/IAIPXStake.sol#L23,25,27,29Erc20Reserve.sol#L13AIpRewardDistributor.sol#L39,57,63,80,96,108,114,120,127,134,142AipxStaking.sol#L29,30,31,50,76,90,107,119,134,163,169,177,184,197,209AIPXStake.sol#L37,38,61,79,100,130,171,191,199,209AipxReferralRegistry.sol#L24,37AipxReferralController.sol#L39,63,97,123,130,146,175,187,193,199,208,217,224,230AipxMaster.sol#L96,121,137,147AipxGovernance.sol#L17,19,21,23,30,32,33,35AgoStaking.sol#L28,42,57,63,75,88,102,131
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
mapping(uint256 => mapping(address => uint256)) public
_balances
mapping(address => mapping(address => uint256)) public
_allowances
mapping(uint256 => mapping(address => uint256)) public _rewards
address _rewardToken
address _account
address _to
uint256 _amount
address _spender
address _owner
uint256 _addedValue
uint256 _subtractedValue
uint256 _batchId
address _receiver
uint256 _duration

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	AIPXStake.sol#L18AIPXOracle.sol#L9AipxGovernance.sol#L13,25
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
uint8 constant VERSION = 3
uint256 private constant PRECISION = 1e6
uint8 constant VERSION = 1

bytes32 internal constant EIP712_DOMAIN =
        keccak256("EIP712Domain(string name,string
version,uint256 chainId,address verifyingContract)")
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	AipxMaster.sol#L97
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(addedTokens[address(_lpToken)] == false, "Token already
added")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
LyAipx	Implementation	Initializable, OwnableUpgradeable, IERC20		
		Public	✓	-
	initialize	External	✓	initializer
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	mint	External	✓	-
	burn	Public	✓	-
	burnFrom	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	

	_spendAllowance	Internal	✓	
	getNextBatch	Public		-
	claimable	Public		-
	claimRewards	External	✓	-
	setBatchVestingDuration	External	✓	onlyOwner
	setMinter	External	✓	onlyOwner
	setEpoch	Public	✓	onlyOwner
	enableRewardStaking	External	✓	onlyOwner
	allocate	External	✓	-
GovernancePowerDelegationERC20	Implementation	ERC20Upgradeable, IGovernancePowerDelegationToken		
	delegateByType	External	✓	-
	delegate	External	✓	-
	getDelegateeByType	External		-
	getPowerCurrent	External		-
	getPowerAtBlock	External		-
	totalSupplyAt	External		-
	_delegateByType	Internal	✓	
	_moveDelegatesByType	Internal	✓	
	_searchByBlockNumber	Internal		
	_getDelegationDataByType	Internal		
	_writeSnapshot	Internal	✓	

	_getDelegatee	Internal		
Erc20Reserve	Implementation	Ownable		
	transfer	External	✓	onlyOwner
AlpRewardDistributor	Implementation	Initializable, OwnableUpgradeable		
	initialize	External	✓	initializer
	transferRewards	External	✓	onlyRequester
	transferRewardsToSingleToken	External	✓	onlyRequester
	swap	External	✓	onlyController
	convertToAlp	External	✓	onlyController
	setRewardsPerSecond	External	✓	onlyController
	setRequester	External	✓	onlyOwner
	recoverFund	External	✓	onlyOwner
	recoverETH	External	✓	onlyOwner
	setTokenWithdrawable	External	✓	onlyOwner
	setController	External	✓	onlyOwner
	_safeTransferToken	Internal	✓	
	_safeUnwrapETH	Internal	✓	
AipxToken	Implementation	ERC20Burnable		
		Public	✓	ERC20

AipxStaking	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
		Public	✓	-
	initialize	External	✓	initializer
	pendingRewards	External		-
	stake	External	✓	nonReentrant
	unstake	External	✓	nonReentrant
	claimRewards	External	✓	nonReentrant
	claimRewardsToSingleToken	External	✓	nonReentrant
	update	Public	✓	-
	setController	External	✓	onlyOwner
	setTokenWithdrawable	External	✓	onlyOwner
	setRewardsPerSecond	External	✓	onlyController
	swap	External	✓	onlyController
	convert	External	✓	onlyController
	recoverFund	External	✓	onlyOwner
	_swapRewardsToToken	Internal	✓	
	_safeTransferToken	Internal	✓	
	_safeUnwrapETH	Internal	✓	
AipxReferralRegistry	Implementation	Initializable, OwnableUpgradeable		
		Public	✓	-

	initialize	External	✓	initializer
	setReferrer	External	✓	-
	setController	External	✓	onlyOwner
AipxReferralController	Implementation	Initializable, OwnableUpgradeable		
		Public	✓	-
	initialize	External	✓	initializer
	getNextEpoch	Public		-
	claimable	Public		-
	setReferrer	External	✓	-
	updatePoint	External	✓	-
	claim	External	✓	-
	nextEpoch	External	✓	-
	start	External	✓	-
	setDistributor	External	✓	onlyOwner
	setUpdater	External	✓	onlyOwner
	setOracle	External	✓	onlyOwner
	setEpochDuration	Public	✓	onlyOwner
	withdrawAIPX	External	✓	onlyOwner
	setEnableNextEpoch	External	✓	-
	setEpochVestingDuration	External	✓	onlyOwner
	_updateTier	Internal	✓	

AipxMaster	Implementation	Ownable, ReentrancyGuard		
		Public	✓	-
	poolLength	Public		-
	add	Public	✓	onlyOwner
	set	Public	✓	onlyOwner
	setRewardPerSecond	Public	✓	onlyOwner
	pendingReward	External		-
	massUpdatePools	External	✓	-
	updatePool	Public	✓	-
	deposit	Public	✓	-
	withdraw	Public	✓	-
	harvest	Public	✓	-
	harvestAll	External	✓	-
	withdrawAndHarvest	Public	✓	-
	addLiquidity	External	✓	nonReentrant
	addLiquidityETH	External	Payable	nonReentrant
	removeLiquidity	External	✓	nonReentrant
	removeLiquidityETH	External	✓	nonReentrant
	emergencyWithdraw	Public	✓	-
	_deposit	Internal	✓	
	_withdrawAndHarvest	Internal	✓	
	_safeTransferETH	Internal	✓	
	_transferReward	Internal	✓	

		External	Payable	-
AipxGovernance	Implementation	Initializable, GovernancePowerDelegationERC20		
		Public	✓	-
	initialize	External	✓	initializer
	permit	External	✓	-
	_beforeTokenTransfer	Internal	✓	
	_getDelegationDataByType	Internal		
	delegateByTypeBySig	Public	✓	-
	delegateBySig	Public	✓	-
AgoStaking	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	External	✓	initializer
	getRewardToken	External		-
	pendingRewards	External		-
	getRewardsPerSecond	External		-
	withdrawableTokens	External		-
	stake	External	✓	nonReentrant
	unstake	External	✓	nonReentrant
	claimRewards	External	✓	nonReentrant
	claimRewardsToSingleToken	External	✓	nonReentrant

	updateRewards	Public	✓	-
	setAlpRewardDistributor	External	✓	onlyOwner
AIPXStake	Implementation	Initializable, OwnableUpgradeable, IAIPXStake		
		Public	✓	-
	initialize	External	✓	initializer
	pendingReward	External		-
	stake	External	✓	-
	unstake	External	✓	-
	cooldown	External	✓	-
	deactivateCooldown	External	✓	-
	claimRewards	External	✓	-
	setBooster	Public	✓	onlyOwner
	setRewardPerSecond	Public	✓	onlyOwner
	setBoostedReward	Public	✓	-
	reserveAuctionFund	External	✓	onlyOwner
	getBoostedReward	Internal		
	update	Public	✓	-
	_safeTransferAGO	Internal	✓	
AIPXOracle	Implementation			
		Public	✓	-

	getCurrentTWAP	Public		-
	update	External	✓	-
LyAipx	Implementation	Initializable, OwnableUpgradeable, IERC20		
		Public	✓	-
	initialize	External	✓	initializer
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	mint	External	✓	-
	burn	Public	✓	-
	burnFrom	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	

	getNextBatch	Public		-
	claimable	Public		-
	claimRewards	External	✓	-
	setBatchVestingDuration	External	✓	onlyOwner
	setMinter	External	✓	onlyOwner
	setEpoch	Public	✓	onlyOwner
	enableRewardStaking	External	✓	onlyOwner
	allocate	External	✓	-
AipxToken	Implementation	ERC20Burnable		
		Public	✓	ERC20
AipxGovernance	Implementation	Initializable, GovernancePowerDelegationERC20		
		Public	✓	-
	initialize	External	✓	initializer
	permit	External	✓	-
	_beforeTokenTransfer	Internal	✓	
	_getDelegationDataByType	Internal		
	delegateByTypeBySig	Public	✓	-
	delegateBySig	Public	✓	-
AipxReferralRegistry	Implementation	Initializable, OwnableUpgradeable		

		Public	✓	-
	initialize	External	✓	initializer
	setReferrer	External	✓	-
	setController	External	✓	onlyOwner
AipxReferralController	Implementation	Initializable, OwnableUpgradeable		
		Public	✓	-
	initialize	External	✓	initializer
	getNextEpoch	Public		-
	claimable	Public		-
	setReferrer	External	✓	-
	updatePoint	External	✓	-
	claim	External	✓	-
	nextEpoch	External	✓	-
	start	External	✓	-
	setDistributor	External	✓	onlyOwner
	setUpdater	External	✓	onlyOwner
	setOracle	External	✓	onlyOwner
	setEpochDuration	Public	✓	onlyOwner
	withdrawAIPX	External	✓	onlyOwner
	setEnableNextEpoch	External	✓	-
	setEpochVestingDuration	External	✓	onlyOwner
	_updateTier	Internal	✓	

AIPXOracle	Implementation			
		Public	✓	-
	getCurrentTWAP	Public		-
	update	External	✓	-
GovernancePowerDelegationERC20	Implementation	ERC20Upgradeable, IGovernancePowerDelegationToken		
	delegateByType	External	✓	-
	delegate	External	✓	-
	getDelegateeByType	External		-
	getPowerCurrent	External		-
	getPowerAtBlock	External		-
	totalSupplyAt	External		-
	_delegateByType	Internal	✓	
	_moveDelegatesByType	Internal	✓	
	_searchByBlockNumber	Internal		
	_getDelegationDataByType	Internal		
	_writeSnapshot	Internal	✓	
	_getDelegatee	Internal		
Erc20Reserve	Implementation	Ownable		
	transfer	External	✓	onlyOwner

AlpRewardDistributor	Implementation	Initializable, OwnableUpgradeable		
	initialize	External	✓	initializer
	transferRewards	External	✓	onlyRequester
	transferRewardsToSingleToken	External	✓	onlyRequester
	swap	External	✓	onlyController
	convertToAlp	External	✓	onlyController
	setRewardsPerSecond	External	✓	onlyController
	setRequester	External	✓	onlyOwner
	recoverFund	External	✓	onlyOwner
	recoverETH	External	✓	onlyOwner
	setTokenWithdrawable	External	✓	onlyOwner
	setController	External	✓	onlyOwner
	_safeTransferToken	Internal	✓	
	_safeUnwrapETH	Internal	✓	
AipxStaking	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
		Public	✓	-
	initialize	External	✓	initializer
	pendingRewards	External		-
	stake	External	✓	nonReentrant

	unstake	External	✓	nonReentrant
	claimRewards	External	✓	nonReentrant
	claimRewardsToSingleToken	External	✓	nonReentrant
	update	Public	✓	-
	setController	External	✓	onlyOwner
	setTokenWithdrawable	External	✓	onlyOwner
	setRewardsPerSecond	External	✓	onlyController
	swap	External	✓	onlyController
	convert	External	✓	onlyController
	recoverFund	External	✓	onlyOwner
	_swapRewardsToToken	Internal	✓	
	_safeTransferToken	Internal	✓	
	_safeUnwrapETH	Internal	✓	
AipxMaster	Implementation	Ownable, ReentrancyGuard		
		Public	✓	-
	poolLength	Public		-
	add	Public	✓	onlyOwner
	set	Public	✓	onlyOwner
	setRewardPerSecond	Public	✓	onlyOwner
	pendingReward	External		-
	massUpdatePools	External	✓	-
	updatePool	Public	✓	-

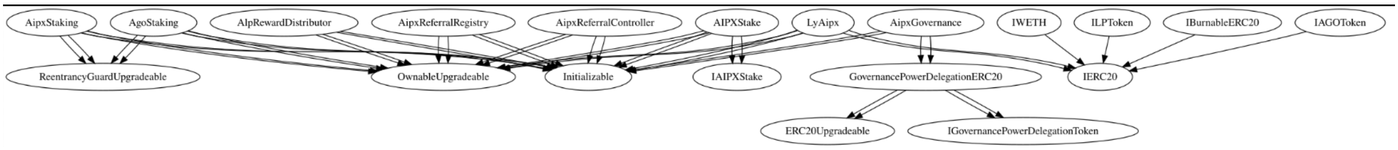
	deposit	Public	✓	-
	withdraw	Public	✓	-
	harvest	Public	✓	-
	harvestAll	External	✓	-
	withdrawAndHarvest	Public	✓	-
	addLiquidity	External	✓	nonReentrant
	addLiquidityETH	External	Payable	nonReentrant
	removeLiquidity	External	✓	nonReentrant
	removeLiquidityETH	External	✓	nonReentrant
	emergencyWithdraw	Public	✓	-
	_deposit	Internal	✓	
	_withdrawAndHarvest	Internal	✓	
	_safeTransferETH	Internal	✓	
	_transferReward	Internal	✓	
		External	Payable	-
AgoStaking	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	External	✓	initializer
	getRewardToken	External		-
	pendingRewards	External		-
	getRewardsPerSecond	External		-
	withdrawableTokens	External		-

	stake	External	✓	nonReentrant
	unstake	External	✓	nonReentrant
	claimRewards	External	✓	nonReentrant
	claimRewardsToSingleToken	External	✓	nonReentrant
	updateRewards	Public	✓	-
	setAlpRewardDistributor	External	✓	onlyOwner
AIPXStake	Implementation	Initializable, OwnableUpgradeable, IAIPXStake		
		Public	✓	-
	initialize	External	✓	initializer
	pendingReward	External		-
	stake	External	✓	-
	unstake	External	✓	-
	cooldown	External	✓	-
	deactivateCooldown	External	✓	-
	claimRewards	External	✓	-
	setBooster	Public	✓	onlyOwner
	setRewardPerSecond	Public	✓	onlyOwner
	setBoostedReward	Public	✓	-
	reserveAuctionFund	External	✓	onlyOwner
	getBoostedReward	Internal		
	update	Public	✓	-

	_safeTransferAGO	Internal	✓	

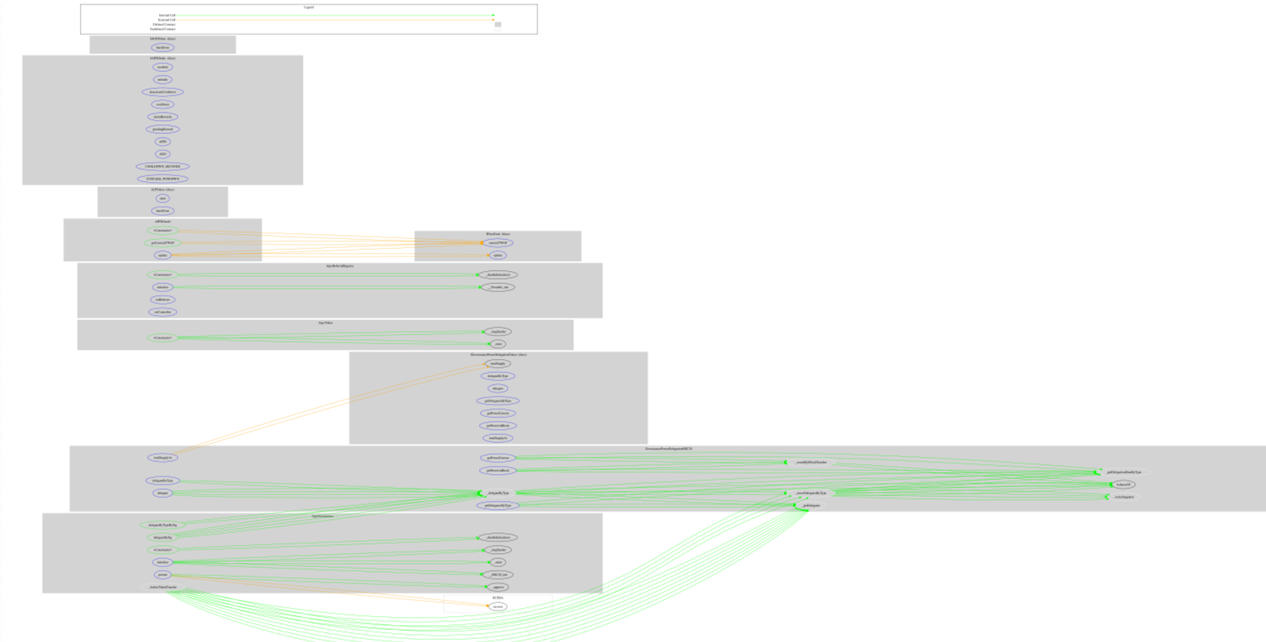
Inheritance Graph

See the detailed image in the github repository.



Flow Graph

See the detailed image in the github repository.



Summary

aiPX contract implements a decentralized application. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io