Cyberscope

Audit Report

Tea-Fi Airdrop

October 2025

Network : BSC

Address : 0x5fF04BE69c1bd92C91EF0259bdC1C6E601371E90

Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Explorer	https://bscscan.com/address/0x5ff04be69c1bd92c91ef0259bdc1c6e601371e90

Audit Updates

Initial Audit	27 Oct 2025
Corrected Phase 2	29 Oct 2025
Corrected Phase 3	31 Oct 2025

Source Files

Filename	SHA256
AirdropStaking.sol	1faffbc5341decbd2078252df22af8dabf704851be4dd8bb1db2ef72acb7d882
interfaces/airdrop/IAirdropStruct.sol	6edf6701bbb0c83ab65e008331e54aa7bde3d2894c2ceabcc0465d3d4d34ee26
interfaces/airdrop/IAirdropStaking.sol	d4a8088e42d54162eca2ddb1e1fe834fd7f2bcc29c96242d678c445a0ca9aba3

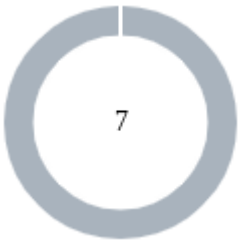
Overview

The `AirdropStaking` contract is a token management system designed to handle airdrop claims, staking, and reward distribution in a secure and controlled manner. It allows users to participate in staking and claiming operations, authorized by an off-chain administrator, and processed according to predefined time windows. The contract leverages OpenZeppelin upgradeable modules for access control, pausability, and security, while using EIP-712 signatures and nonces to validate user actions and prevent replay attacks.

Core Functionality

Users can submit operator-signed staking parameters to claim tokens and record stakes in a single transaction. Claimable tokens are transferred from the treasury to the users, while the staked amounts are tracked internally in the contract until a locking period ends. Upon withdrawal, users receive the staked amounts incremented by rewards calculated proportionally to their stake relative to the total staking pool and the available reward pool. The contract also provides administrative controls for updating the treasury, adjusting reward pools, and pausing or unpausing operations, while enforcing strict role-based access and balance checks for security and transparency.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	0	7	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	CCR	Contract Centralization Risk	Acknowledged
●	MAC	Missing Access Control	Acknowledged
●	MC	Missing Check	Acknowledged
●	PIC	Post Initialization Configuration	Acknowledged
●	TSI	Tokens Sufficiency Insurance	Acknowledged
●	L04	Conformance to Solidity Naming Conventions	Acknowledged
●	L17	Usage of Solidity Assembly	Acknowledged

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	AirdropStaking.sol#L193,206,215,223,229
Status	Acknowledged

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.

```shell
function setTreasury(address newTreasury) external
onlyRole(DEFAULT_ADMIN_ROLE) {...}
function setCurrentRewardPool(uint256 newRewardPool) external
onlyRole(DEFAULT_ADMIN_ROLE) {...}
function setClaimRoot(bytes32 newRoot) external
onlyRole(DEFAULT_ADMIN_ROLE) {...}
function pause() external onlyRole(DEFAULT_ADMIN_ROLE) {...}
function unpause() external onlyRole(DEFAULT_ADMIN_ROLE) {...}
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

The reliance on external configuration is intentional and necessary for the airdrop-staking logic, which requires backend-driven flexibility (e.g., campaign parameters, reward logic, eligibility updates). Migrating this logic fully on-chain would reduce adaptability and increase operational complexity. Access to external configuration is strictly controlled and monitored, ensuring a balanced trade-off between flexibility and decentralization.

MAC - Missing Access Control

Criticality	Minor / Informative
Location	AirdropStaking.sol#L80
Status	Acknowledged

Description

The `initialize` functions can be frontrun during deployment, allowing administrative roles to be transferred to third parties not associated with the team. Such third parties would gain access to all the functions of the contract.

Shell

```
function initialize(
address initialAdmin,
uint256 initialRewardPool,
address _treasury,
address _token,
uint256 airdropStartDate
) public initializer {
if (_token == address(0)) {
revert NoZeroAddress();
}

if (initialRewardPool == 0) {
revert NoZeroInitialRewardPool();
}

AirdropStakingStorage storage $ = _getAirdropStakingStorage();
$.token = _token;
$.airdropStartDate = airdropStartDate;
$.stakingStartDate = airdropStartDate + CLAIM_PERIOD;
if (_treasury == address(0)) revert NoZeroAddress();
$.treasury = _treasury;
__AccessControlEnumerable_init_unchained();
__Pausable_init_unchained();
__EIP712_init_unchained("AirdropStaking", "1");
__Nonces_init_unchained();
__ReentrancyGuard_init_unchained();
```

```
if (initialAdmin == address(0)) {
revert NoZeroAddress();
}
_grantRole(DEFAULT_ADMIN_ROLE, initialAdmin);
$.currentRewardPool = initialRewardPool;
$.totalClaimedAmount = 0;

}
```

Recommendation

The team is advised to implement proper access controls to ensure that only authorized team members can call these functions.

Team Update

The team has acknowledged that this is not a security issue and states:

This finding is not applicable in our deployment setup. The initialize function cannot be frontrun, as it is executed atomically during proxy deployment via the _data parameter in the constructor (ERC1967Proxy(_logic, _data)), which encodes and calls the initialization within the same transaction. Therefore, external access before initialization is not possible.

MC - Missing Check

Criticality	Minor / Informative
Location	AirdropStaking.sol#L98,215
Status	Acknowledged

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

Specifically, the contract does not validate that `airdropStartDate` is a non-zero, future value and that the merkle root is a non-zero value.

```
Shell
$.airdropStartDate = airdropStartDate;
```

```
Shell
function setClaimRoot(bytes32 newRoot) external
onlyRole(DEFAULT_ADMIN_ROLE) {
AirdropStakingStorage storage $ = _getAirdropStakingStorage();
$.claimRoot = newRoot;
emit ClaimRootSet(newRoot);
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

PIC - Post Initialization Configuration

Criticality	Minor / Informative
Location	AirdropStaking.sol#L193,206
Status	Acknowledged

Description

The contract allows the `DEFAULT_ADMIN_ROLE` to modify critical state variables, including `treasury` and `currentRewardPool`, even after the system has been initialized for normal operations. These changes are not integrated into the main execution flow, which can lead to inconsistencies in the distribution mechanism for users who are subjected to different configurations of the contract.

Shell
```
function setTreasury(address newTreasury) external
onlyRole(DEFAULT_ADMIN_ROLE) {
if (newTreasury == address(0)) {
revert NoZeroAddress();
}
AirdropStakingStorage storage $ = _getAirdropStakingStorage();
$.treasury = newTreasury;
emit TreasurySet(newTreasury)
}

function setCurrentRewardPool(uint256 newRewardPool) external
onlyRole(DEFAULT_ADMIN_ROLE) {
AirdropStakingStorage storage $ = _getAirdropStakingStorage();
$.currentRewardPool += newRewardPool;
emit UpdatedRewardPool(newRewardPool);}
```

Recommendation

The team is advised to prevent any modifications to the contract's execution flow once it has been initialized. This approach enhances decentralization and strengthens user trust.

Team Update

The team has acknowledged that this is not a security issue and states:

This behavior is intentional and required for our architecture. The setTreasury and setCurrentRewardPool functions provide necessary flexibility, as multiple treasuries may be used and the reward pool can be increased after staking begins. Additionally, setCurrentRewardPool only allows increasing the value, preventing any reduction or misuse.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	AirdropStaking.sol#L126,166,186
Status	Acknowledged

Description

The tokens 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. Such dependencies may result in tokens not being available for claim cycles and reward distributions.

Furthermore, the contract introduces restrictions based on the available balance of the treasury. At the same time, the contract maintains internal variables to monitor the amounts available for distribution. The contract however does not always update the internal variables in accordance to the dynamic balance of the treasury. As a result a discrepancy may occur in the sufficiency of the tokens for successful operations.

```shell
if (params.amountToClaim > 0) {
$.userInfo[params.user].claimedAmount += params.amountToClaim;
$.totalClaimedAmount += params.amountToClaim;
emit Claimed(params.user, params.amountToClaim,
$.userInfo[params.user].claimedAmount);
if (IERC20($.token).balanceOf($.treasury) <
params.amountToClaim) revert InsufficientTreasuryBalance();
IERC20($.token).safeTransferFrom($.treasury, params.user,
params.amountToClaim);}
```

```shell
if (IERC20($.token).balanceOf($.treasury) < totalAmount)
revert InsufficientTreasuryBalance();
IERC20($.token).safeTransferFrom($.treasury, _msgSender(),
totalAmount);

emit Unstaked(_msgSender(), totalAmount);
```

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 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.

Team Update

The team has acknowledged that this is not a security issue and states:

This design is intentional to provide operational flexibility. Tokens are managed externally to avoid the need for additional transfers and on-chain storage within the contract, while internal checks ensure sufficient balances in the treasury before any transfers.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	AirdropStaking.sol#L50,83,84
Status	Acknowledged

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.

```shell
bytes32 private constant AirdropStakingStorageLocation =
0xdca611c1f1f24ec24f9ba8bbb98c28ae11043ccda1d20d4df1b4f58c19a4
1000
address _treasury
address _token
```

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/stable/style-guide.html#naming-conventions.

Team Update

The team has acknowledged that this is not a security issue and states:

Conformance to Solidity Naming Conventions Variable names were chosen intentionally to follow our internal coding style guidelines. While they differ slightly from the Solidity style guide, they maintain consistency and readability within our codebase.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	AirdropStaking.sol#L293
Status	Acknowledged

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```shell
assembly {
$.slot := AirdropStakingStorageLocation
}
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

Team Update

The team has acknowledged that this is not a security issue and states:

The usage of assembly in this context is intentional and follows a standard, widely adopted pattern from OpenZeppelin v5 for efficient storage slot management. This approach has been thoroughly tested and is considered safe.

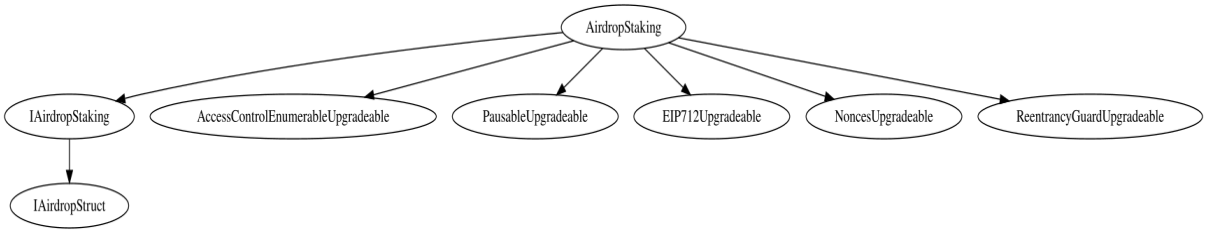
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
AirdropStaking	Implementation	IAirdropStaking, AccessControlEnumerableUpgradeable, PausableUpgradeable, EIP712Upgradeable, NoncesUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	claimAndStake	External	✓	whenNotPaused nonReentrant
	withdraw	External	✓	whenNotPaused nonReentrant
	setTreasury	External	✓	onlyRole
	setCurrentRewardPool	External	✓	onlyRole
	setClaimRoot	External	✓	onlyRole
	pause	External	✓	onlyRole
	unpause	External	✓	onlyRole
	calculateUserReward	Public		-
	currentRewardPool	External		-
	currentStakingPool	External		-
	currentTotalClaimedAmount	External		-

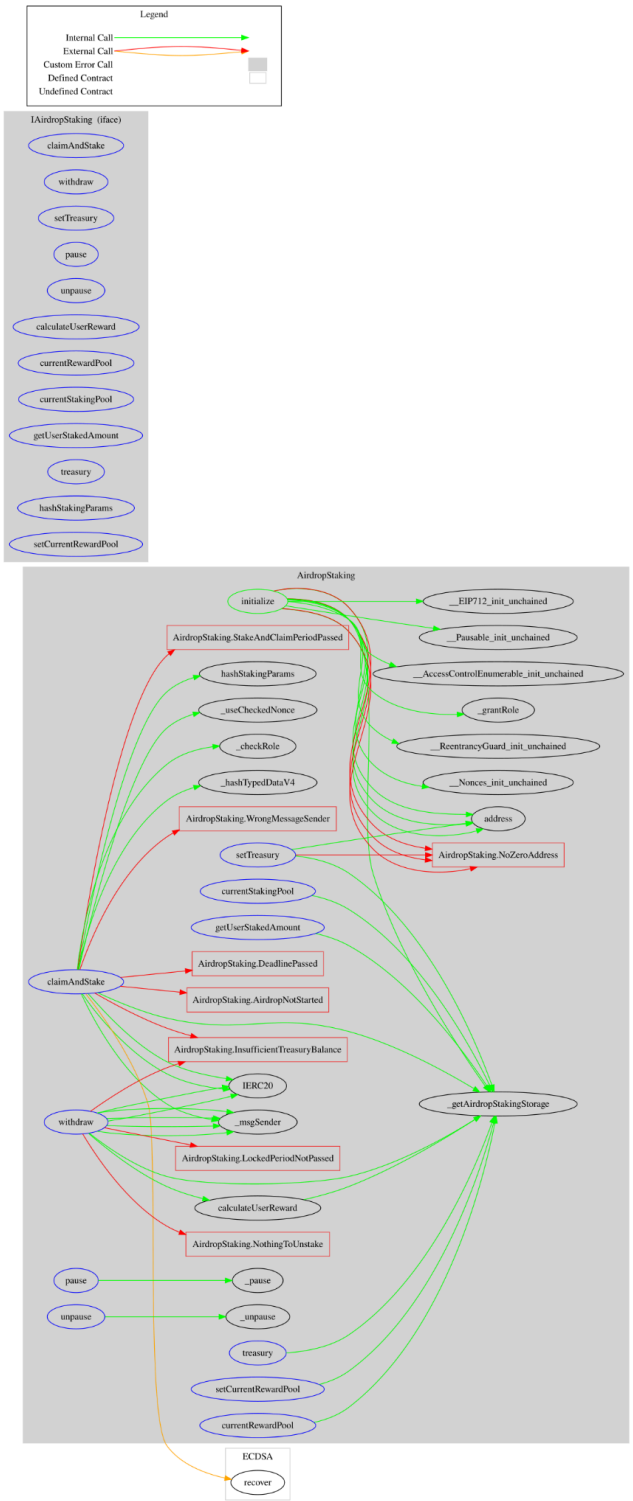
	getUserStakedAmount	External		-
	getUserClaimedAmount	External		-
	treasury	External		-
	claimRoot	External		-
	_getAirdropStakingStorage	Private		
	hashStakingParams	Public		-
IAirdropStruct	Interface			
IAirdropStaking	Interface	IAirdropStruct		
	claimAndStake	External	✓	-
	withdraw	External	✓	-
	setTreasury	External	✓	-
	pause	External	✓	-
	unpause	External	✓	-
	calculateUserReward	External		-
	currentRewardPool	External		-
	currentStakingPool	External		-
	currentTotalClaimedAmount	External		-
	getUserStakedAmount	External		-
	getUserClaimedAmount	External		-
	treasury	External		-

	hashStakingParams	External		-
	setCurrentRewardPool	External	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
IAirdropStaking (iface)
claimAndStake
withdraw
setTreasury
pause
unpause
calculateUserReward
currentRewardPool
currentStakingPool
getUserStakedAmount
treasury
hashStakingParams
setCurrentRewardPool
AirdropStaking
initialize
__EIP712_init_unchained
__Pausable_init_unchained
__AccessControlEnumerable_init_unchained
_grantRole
__ReentrancyGuard_init_unchained
__Nonces_init_unchained
address
AirdropStaking.NoZeroAddress
AirdropStaking.StakeAndClaimPeriodPassed
hashStakingParams
_useCheckedNonce
_checkRole
_hashTypedDataV4
AirdropStaking.WrongMessageSender
setTreasury
currentStakingPool
getUserStakedAmount
AirdropStaking.DeadlinePassed
AirdropStaking.AirdropNotStarted
AirdropStaking.InsufficientTreasuryBalance
IERC20
_msgSender
AirdropStaking.LockedPeriodNotPassed
calculateUserReward
AirdropStaking.NothingToUnstake
_getAirdropStakingStorage
pause
_pause
unpause
_unpause
treasury
setCurrentRewardPool
currentRewardPool
ECDSA
recover

Summary

Tea-Fi contract implements an airdrop mechanism. 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