Cyberscope

Audit Report

V8Coin

December 2024

Repository https://github.com/dexolacom/V8COIN-smart-contract

Commit 5d9b274f9d56738d5a75270d471dab21b29504b7

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

Repository	https://github.com/dexolacom/V8COIN-smart-contract
Commit	5d9b274f9d56738d5a75270d471dab21b29504b7
Testing Deploy	https://testnet.bscscan.com/address/0x66503e186b51affcf5098e69fffeaa320f4171f4

Audit Updates

Initial Audit	23 Dec 2024

Source Files

Filename	SHA256
contracts/V8AirdropClaim.sol	477987c39e813b6d3d0209c09efe1430ae1e2fc433a06e447047d077e68d75a3

Overview

The V8AirdropClaim contract is designed to facilitate and manage token airdrop claims with a focus on security and efficiency. It allows eligible users to claim rewards by validating their claims through a Merkle-proof mechanism while enforcing strict role-based access controls for administrative functions. The contract also supports flexible management of airdrop sessions, claim fees, and rewards.

Claim Functionality

The claim functionality is the core of the contract, enabling users to securely receive airdrop rewards by providing a valid Merkle proof to prove their eligibility. The `claimReward` function ensures that each claim is valid by verifying the user's proof against the current Merkle root. Users are required to pay a predefined claim fee (`claimFeeAmount`), which is transferred to the designated `claimFeeReceiver` , ensuring proper fee handling. All users receive the same fixed reward amount of V8COIN tokens, which are transferred to the specified receiver address upon successful claim validation. To maintain fairness, the contract prevents duplicate claims by tracking addresses that have already claimed their rewards. Additionally, claims can only be made during active airdrop sessions, defined by the `claimSessionStart` and `claimSessionEnd` timestamps. This robust mechanism ensures fair and efficient reward distribution while safeguarding against invalid or duplicate claims.

Roles

Admin

The admin can interact with the following functions:

- `setRoot`
- `rescueERC20`
- `updateClaimFeeReceiver`
- `updateClaimFeeAmount`
- `updateRewardAmount`

AIRDROP_SESSION_OPENER_ROLE

The AIRDROP_SESSION_OPENER_ROLE can interact with the following functions:

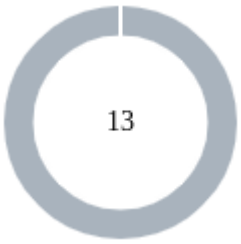
- `openAirdropSessionNow`
- `openAirdropSessionAt`

Users

The users can interact with the following functions:

- `claimReward`
- `isAirdropSessionOpen`
- `claimSessionStartHuman`
- `claimSessionEndHuman`

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	CFH	Claim Fee Handling	Unresolved
●	CFR	Claim Function Restriction	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	DPI	Decimals Precision Inconsistency	Unresolved
●	MPC	Merkle Proof Centralization	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PCRA	Potential Cross-Chain Replay Attack	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RCS	Redundant Conditional Statements	Unresolved
●	TUU	Time Units Usage	Unresolved
●	TC	TODO Comments	Unresolved
●	TSI	Tokens Sufficiency Insurance	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

CFH - Claim Fee Handling

Criticality	Minor / Informative
Location	contracts/V8AirdropClaim.sol#L150
Status	Unresolved

Description

The contract is designed to require users to pay a `claimFeeAmount` when invoking the `claimReward` function. However, it lacks a mechanism to refund any excess payment if the amount sent (`msg.value`) exceeds the required `claimFeeAmount` . This could lead to users unintentionally overpaying without receiving a refund, which may result in dissatisfaction or potential financial loss. Proper handling of excess payments is essential to ensure fairness and transparency in the claiming process.

```
    function claimReward(address payable _receiver, bytes32[]
memory _proof) payable external whenAirdropSessionIsOpen {
        if (msg.value < claimFeeAmount) {
            revert NotEnoughClaimFee();
        }

        if (_receiver == address(0)){
            revert ZeroAddress();
        }

        if (isClaimed[msg.sender]) {
            revert AlreadyClaimed();
        }

        if (!isProofValid(msg.sender, _proof)) {
            revert ProofNotValid();
        }

        isClaimed[msg.sender] = true;
        uint256 rewardAmountCache = rewardAmount;
        V8COIN.safeTransfer(_receiver, rewardAmountCache);
        totalClaimed += rewardAmountCache;
        _safeTransfer(claimFeeReceiver, msg.value);
        emit Claimed(msg.sender, _receiver, rewardAmountCache);
    }
```

Recommendation

It is recommended to evaluate the logic for handling the `claimFeeAmount` to ensure that users are only charged the exact required fee. If users send an amount exceeding `claimFeeAmount`, the contract should include a mechanism to refund the excess back to the sender. This would improve user trust and ensure equitable treatment, aligning the contract's behaviour with best practices for payment handling.

CFR - Claim Function Restriction

Criticality	Minor / Informative
Location	contracts/V8AirdropClaim.sol#L150
Status	Unresolved

Description

The contract is designed to prevent users from invoking the `claim` function more than once by enforcing a `require` check. While this ensures that users cannot make duplicate claims, it introduces an issue in scenarios where a user might legitimately need to claim again. For instance, if there is a change in the root, a user needs to claim on behalf of another recipient, or any other valid situation arises, the current logic permanently blocks the user from claiming again. This restriction reduces the contract's flexibility and could hinder its ability to meet its intended functionality.

```
    function claimReward(address payable _receiver, bytes32[]
memory _proof) payable external whenAirdropSessionIsOpen {
        ...

        if (isClaimed[msg.sender]) {
            revert AlreadyClaimed();
        }
        ...
        isClaimed[msg.sender] = true;
        ...
    }
```

Recommendation

It is advised to evaluate whether the same user might ever need to claim more than once under certain circumstances. If such a scenario is possible, the contract should ensure that valid users are able to make subsequent claims without compromising security. This could involve introducing mechanisms such as session-based claim validation, dynamic root updates with user resets, or a separate logic for handling valid re-claims.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	contracts/V8AirdropClaim.sol#L180,195,211,239,250,264
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.

```
    function openAirdropSessionNow(uint256
sessionDurationInMinutes) external
onlyRole(AIRDROP_SESSION_OPENER_ROLE) {
        ...
    }

    function openAirdropSessionAt(
        uint256 startYear,
        uint256 startMonth,
        uint256 startDay,
        uint256 startHours,
        uint256 startMinutes,
        uint256 sessionDurationInMinutes
    ) external onlyRole(AIRDROP_SESSION_OPENER_ROLE) {
        ...
    }

    function setRoot(bytes32 _root) external
onlyRole(DEFAULT_ADMIN_ROLE) {
        ...
    }

    function updateClaimFeeReceiver(address payable
_claimFeeReceiver) external onlyRole(DEFAULT_ADMIN_ROLE) {
        ...
    }

    function updateClaimFeeAmount(uint256 _claimFeeAmount)
external onlyRole(DEFAULT_ADMIN_ROLE) {
        ...
    }

    function updateRewardAmount(uint256 _rewardAmount) 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.

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	contracts/V8AirdropClaim.sol#L135
Status	Unresolved

Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals are set to `8`, it means that the smallest unit of the token is `0.00000001`, and if decimals are set to `18`, it means that the smallest unit of the token is `0.000000000000000001`.

However, there is an inconsistency in the way that the decimals field is handled in some ERC20 contracts. The ERC20 specification does not specify how the decimals field should be implemented, and as a result, some contracts use different precision numbers.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

```
rewardAmount = 10_000e18;
V8COIN = IERC20(_v8coin);
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to scale all the decimals to the greatest token's decimal. Hence, the contract will not lose precision in the calculations.

The following example depicts 3 tokens with different decimals precision.

ERC20	Decimals
Token 1	6
Token 2	9
Token 3	18

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

MPC - Merkle Proof Centralization

Criticality	Minor / Informative
Location	contracts/V8AirdropClaim.sol#L211,280
Status	Unresolved

Description

The contract uses a Merkle Proof mechanism in order to define many applicable addresses. The verification process is based on an off-chain configuration. The contract owner is responsible for updating the in-chain “Merkle Root” in order to validate correctly the provided message.

```
   function setRoot(bytes32 _root) external
onlyRole(DEFAULT_ADMIN_ROLE) {
        if (_root == 0x0) {
            revert ZeroRoot();
        }
        root = _root;
        emit SetRoot(_root);
    }

    function isProofValid(address _owner, bytes32[] memory
_proof) public view returns(bool) {
        bytes32 leaf =
keccak256(bytes.concat(keccak256(abi.encode(_owner))));
        return MerkleProof.verify(_proof, root, leaf);
    }
```

Recommendation

We state that the Merkle Proof algorithm is required for proper protocol operations and gas consumption decrease. Thus, we emphasize that the Merkle proof algorithm is based on an off-chain mechanism. Any off-chain mechanism could potentially be compromised and affect the on-chain state unexpectedly. The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/V8AirdropClaim.sol#L195
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function openAirdropSessionAt(
        uint256 startYear,
        uint256 startMonth,
        uint256 startDay,
        uint256 startHours,
        uint256 startMinutes,
        uint256 sessionDurationInMinutes
    ) external onlyRole(AIRDROP_SESSION_OPENER_ROLE) {
        ...
    }
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

PCRA - Potential Cross-Chain Replay Attack

Criticality	Minor / Informative
Location	contracts/V8AirdropClaim.sol#L280
Status	Unresolved

Description

The contract contains the `isProofValid` function which performs Merkle proof verification to validate claims based on a specific root and leaf. However, it does not incorporate any mechanism to prevent cross-chain replay attacks. Since the verification process only relies on the `_proof` , `root` , and `_owner` values, the same proof could potentially be reused on different chains where the same contract is deployed. This omission increases the risk of cross-chain replay attacks, where a valid proof on one chain could be reused on another.

```
    function isProofValid(address _owner, bytes32[] memory
_proof) public view returns(bool) {
        bytes32 leaf =
keccak256(bytes.concat(keccak256(abi.encode(_owner))));
        return MerkleProof.verify(_proof, root, leaf);
    }
```

Recommendation

To address the risk of cross-chain replay attacks, it is recommended to include the chain ID as part of the data used to compute the Merkle tree. By incorporating the chain ID into the hashed data for the `leaf` , the proof becomes bound to a specific blockchain, ensuring that it cannot be reused across chains. This approach enhances security by preventing proof reuse in multi-chain deployments.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/V8AirdropClaim.sol#L171,322
Status	Unresolved

Description

The contract sends funds to a `claimFeeReceiver` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
   function claimReward(address payable _receiver, bytes32[]
memory _proof) payable external whenAirdropSessionIsOpen {
        ...
        _safeTransfer(claimFeeReceiver, msg.value);
        emit Claimed(msg.sender, _receiver, rewardAmountCache);
    }

    function _safeTransfer(address payable _recipient, uint256
_amount) internal {
        (bool success, ) = _recipient.call{value: _amount}("");
        if (!success) {
            revert UnsuccessfulFeeTransfer();
        }
    }
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

RCS - Redundant Conditional Statements

Criticality	Minor / Informative
Location	contracts/V8AirdropClaim.sol#L287
Status	Unresolved

Description

The contract contains redundant conditional statements that can be simplified to improve code efficiency and performance. Conditional statements that merely return the result of an expression are unnecessary and lead to larger code size, increased memory usage, and slower execution times. By directly returning the result of the expression, the code can be made more concise and efficient, reducing gas costs and improving runtime performance. Such redundancies are common when simple comparisons or checks are performed within conditional statements, leading to redundant operations.

```
    function isAirdropSessionOpen() external view returns (bool)
{
        if (block.timestamp > claimSessionEnd ||
block.timestamp < claimSessionStart) {
            return false;
        } else {
            return true;
        }
    }
```

Recommendation

It is recommended to refactor conditional statements that return results by eliminating unnecessary code structures and directly returning the outcome of the expression. This practice minimizes the number of operations required, reduces the code footprint, and optimizes memory and gas usage. Simplifying such statements makes the code more readable and improves its overall performance.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	contracts/V8AirdropClaim.sol#L182,205
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint256 sessionEnd = block.timestamp + sessionDurationInMinutes
* 60;
....
claimSessionEnd = startTime + sessionDurationInMinutes * 60;
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds` , `minutes` , `hours` , `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

TC - TODO Comments

Criticality	Minor / Informative
Location	contracts/V8AirdropClaim.sol#L254
Status	Unresolved

Description

The contract contains `TODO` comments, which indicate incomplete or planned functionality. These comments suggest that certain aspects of the contract may still be under development or require further review and implementation. While the presence of `TODO` comments can be helpful during the development phase, their inclusion in deployed or production-ready contracts may raise concerns about the completeness and readiness of the contract for deployment.

```
if (_claimFeeAmount == 0) { //TODO: or not?
```

Recommendation

It is recommended to review and address all `TODO` comments in the contract. If the associated functionality is essential, ensure it is fully implemented, tested, and documented. If the `TODO` comments refer to non-critical or deferred features, consider removing them to avoid confusion and present the contract as a finished and polished product. Maintaining a clean codebase enhances clarity, professionalism, and confidence for users and auditors.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	contracts/V8AirdropClaim.sol#L168
Status	Unresolved

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.

```
uint256 rewardAmountCache = rewardAmount;
V8COIN.safeTransfer(_receiver, rewardAmountCache);
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/V8AirdropClaim.sol#L82,150,211,224,239,250,264,280
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.

```
IERC20 immutable public V8COIN
address payable _receiver
bytes32[] memory _proof
bytes32 _root
IERC20 _token
address payable _claimFeeReceiver
uint256 _claimFeeAmount
uint256 _rewardAmount
address _owner
```

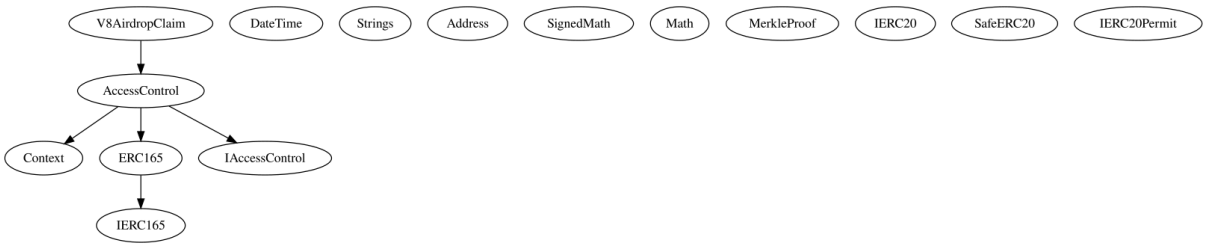
Recommendation

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.

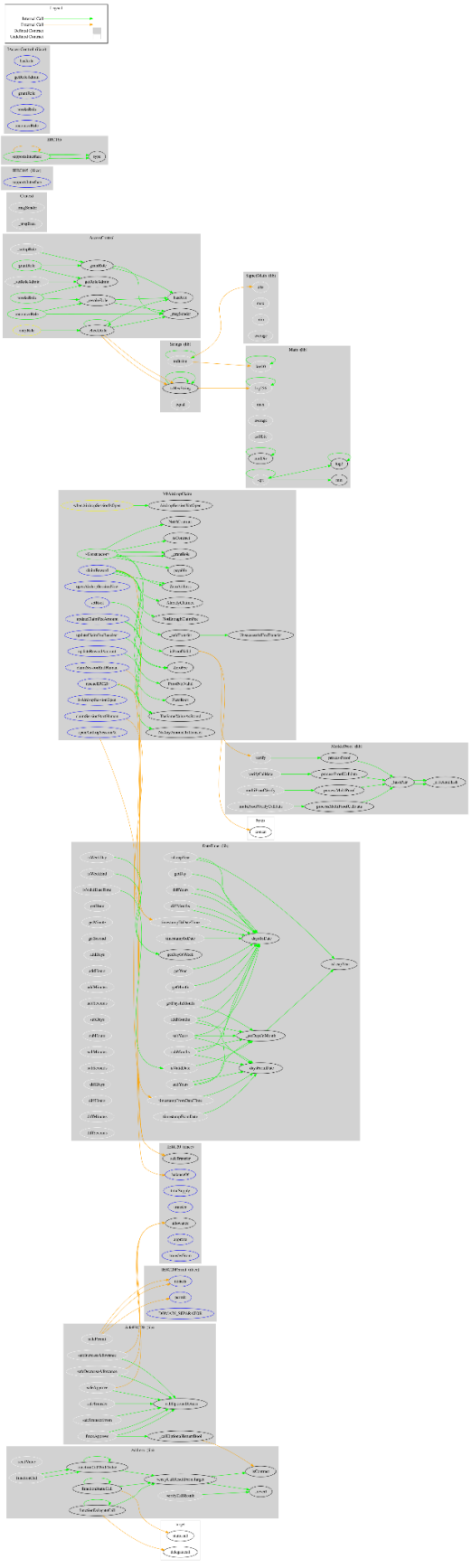
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
V8AirdropClaim	Implementation	AccessControl		
		Public	✓	-
	claimReward	External	Payable	whenAirdropSessionIsOpen
	openAirdropSessionNow	External	✓	onlyRole
	openAirdropSessionAt	External	✓	onlyRole
	setRoot	External	✓	onlyRole
	rescueERC20	External	✓	onlyRole
	updateClaimFeeReceiver	External	✓	onlyRole
	updateClaimFeeAmount	External	✓	onlyRole
	updateRewardAmount	External	✓	onlyRole
	isProofValid	Public		-
	isAirdropSessionOpen	External		-
	claimSessionStartHuman	External		-
	claimSessionEndHuman	External		-
	_safeTransfer	Internal	✓	
	_isContract	Internal		

Inheritance Graph



Flow Graph



Summary

The V8AirdropClaim contract implements a secure and efficient mechanism for managing token airdrop claims, incorporating features such as Merkle proof validation, role-based access control, and session-based claim management. This audit investigates potential security vulnerabilities, evaluates the correctness of business logic, and identifies areas for improvement to enhance the contract’s robustness and functionality.

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

cyberscope.io