Endless Clouds Security Review

Pashov Audit Group

Conducted by: pontifex, ast3ros, Shaka

August 1st 2024 - August 2nd 2024

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **coffeexcoin/endless-clouds** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Endless Clouds

The EndlessCloudsClaimer smart contract allows eligible recipients to claim tokens based on predefined vesting schedules, verified through Merkle proofs to ensure authenticity.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - e9ef869920abb7c3c7372d08b39d2c5fb3b0f9a1

fixes review commit hash - be156b94b42facebe18c7ac11716799cff524321

Scope

The following smart contracts were in scope of the audit:

- `EndlessCloudsClaimer`
- `EndlessClouds`

7. Executive Summary

Over the course of the security review, pontifex, ast3ros, Shaka engaged with Endless Clouds to review Endless Clouds. In this period of time a total of **4** issues were uncovered.

Protocol Summary

Protocol Name	Endless Clouds
Repository	https://github.com/coffeexcoin/endless-clouds
Date	August 1st 2024 - August 2nd 2024
Protocol Type	Claim contract

Findings Count

Severity	**Amount**
Low	4
Total Findings	**4**

Summary of Findings

ID	Title	Severity	Status
[L-01]	Inconsistent behavior when the claim is paused	Low	Resolved
[L-02]	Missing period end check	Low	Acknowledged
[L-03]	getClaimableAmount function can return incorrect information	Low	Acknowledged
[L-04]	Owner is wrongly set in case of deploying the contract with a multi-sig wallet	Low	Resolved

8. Findings

8.1. Low Findings

[L-01] Inconsistent behavior when the claim is paused

The EndlessCloudsClaimer contract has inconsistent behavior when claims are paused (`claimEnabled` = false). The issue stems from the `_computeClaimableAmount` function, which returns 0 when `claimEnabled` is false:

```
function _computeClaimableAmount
  (ClaimData calldata claimData) internal view returns (uint256) {
    if (!claimEnabled) {
        return 0;
    }
    // ... rest of the function
}
```

This design creates two distinct scenarios in the claim function:

- For users who have not claimed any tokens, the claim transaction succeeds but no tokens are transferred (claimAmount = 0).
- For users who have partially claimed their tokens, the claim transaction reverts due to underflow in the calculation below. When `totalClaimableVested` is 0 and `claimedAmount` > 0, this operation will underflow.

```
uint256 claimAmount = totalClaimableVested - claimedAmount;
```

```
function claim(
      ClaimDatacalldataclaimData,
      addressrecipient,
      bytes32[]calldataproof
    ) external {
        ...
        uint256 totalClaimableVested = _computeClaimableAmount(claimData);
        uint256 claimAmount = totalClaimableVested - claimedAmount; // @audit
        // when claimEnabled = false, totalClaimableVested = 0 and claim tx reverts

        ...
    }
```

The inconsistency in the behavior of the claim tx can be confusing for the users. It's recommended to return the error message when the claim is paused by reverting.

```
function _computeClaimableAmount
      (ClaimData calldata claimData) internal view returns (uint256) {
        if (!claimEnabled) {
-           return 0;
+           revert ClaimUnabled();
        }
```

[L-02] Missing period end check

The `withdrawTokens()` and `enableClaim()` allow the owner to withdraw any token from the contract or withdraw EndlessClouds token after the claim period ends. However, there is no actual enforcement of this condition in the code.

```
/// @notice Withdraw tokens from the contract. To be used in case of
    // accidental deposits,
    /// or to withdraw any remaining tokens if the claim period is ended.
    /// @param token The token to withdraw
>>  function withdrawTokens(address token) external onlyOwner {
        SafeTransferLib.safeTransferAll(token, msg.sender);
    }

    /// @notice Enable the token claim functionality
>>  function enableClaim(bool enabled) external onlyOwner {
        claimEnabled = enabled;
    }
```

It is recommended to check if the token is EndlessClouds and if the claim period has ended before allowing the owner to withdraw the token.

[L-03] `getClaimableAmount` function can return incorrect information

The `getClaimableAmount` view function does not check the existence of a leaf with given `claimData` and `recipient`. Users can receive answers even for incorrect data. Consider checking leaves existence.

```
function getClaimableAmount(
      ClaimDatacalldataclaimData,
      addressrecipient
    ) external view returns (uint256
        return _computeClaimableAmount(claimData) - totalClaimed[keccak256
          (abi.encode(recipient, claimData))];
    }
<...>
    function _computeClaimableAmount
      (ClaimData calldata claimData) internal view returns (uint256) {
        if (!claimEnabled) {
            return 0;
        }

        uint256 periodsElapsed;
        if (block.timestamp >= claimData.vestStart) {
            uint256 elapsedTime = block.timestamp - claimData.vestStart;
            periodsElapsed = elapsedTime / VESTING_PERIOD_LENGTH;

            if (periodsElapsed >= claimData.vestingPeriods) {
                periodsElapsed = claimData.vestingPeriods;
            }
        }


                return claimData.immediateClaim + periodsElapsed * claimData.amountPe
    }
```

[L-04] Owner is wrongly set in case of deploying the contract with a multi-sig wallet

The owner of the EndlessCloudsClaimer is given significant power to withdraw all tokens, set a new Merkle root, and enable/disable the claim process. Therefore it's reasonable to expect the owner to be a multi-sig wallet.

However, if the EndlessClouds is deployed through a multi-sig wallet (e.g., Gnosis Safe), the owner becomes the EOA (`tx.origin`) that initiated the transaction, not the multi-sig address as intended. This allows the `tx.orgin` address to drain all tokens.

```
constructor(address _endlessClouds, bytes32 _merkleRoot) {
>>>     _initializeOwner(tx.origin);
        ENDLESS_CLOUDS = _endlessClouds;
        merkleRoot = _merkleRoot;
    }
```

Specify the owner of the contract as a parameter in the constructor.

```
-       constructor(address _endlessClouds, bytes32 _merkleRoot) {
+       constructor
+ (address _endlessClouds, bytes32 _merkleRoot, address _owner) {
-           _initializeOwner(tx.origin);
+           _initializeOwner(_owner);
            ENDLESS_CLOUDS = _endlessClouds;
            merkleRoot = _merkleRoot;
        }
```