OpenZeppelin | security

Forta Network Airdrop Audit

Forta

June 9, 2022

Table of Contents

Summary

Type	DeFi
Timeline	From 2022-00-00 To 2022-00-00
Languages	Solidity

Total Issues	21 (0 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (0 resolved)
Medium Severity Issues	4 (0 resolved)
Low Severity Issues	8 (0 resolved)
Notes & Additional Information	8 (0 resolved)

Scope

We audited the "Forta Network/Airdrop" repository at the d69a0cccd8db3259ee6ae02080bb890529156521 commit.

Only the `Airdrop.sol` file was in scope.

During the audit we assumed:

- the OpenZeppelin dependency contracts work as documented.
- the token to be distributed is a standard ERC20 token without additional features.
- the contract is preloaded with sufficient tokens to ensure all valid claimants can receive their share.

System overview, privileged roles and trust assumptions

The `Airdrop` contract is intended to manage token distributions using one of three mechanism:

- it is initialized with a Merkle root that represents a collection of addresses and their claimable balance.
    - we expect the entire contents of the Merkle tree will be published so that all users can identify their own record and ensure there are no `index` conflicts with other users (which could prevent valid claims)
- it is initialized with a list of *external tiers*, which represent additional tokens that can be claimed. Each tier defines a maximum number of claimants that receive a predefined amount. The claimants themselves are chosen by an *External Verifier* account and do not need to be defined at deployment time. If the External Verifier authorizes too many claims in a tier, the available tokens will be distributed on a first-come-first-serve basis until the tier is depleted.
- there is an *Additional Claim Manager* role that can arbitrarily assign new claims, but must also provide the tokens to fulfill those claims.

There is also an *Airdrop Manager* role, which can:

- enable or disable the airdrop. Claimants can only receive tokens (using any of the three mechanisms) while the airdrop is enabled. On Ethereum mainnet, the airdrop must last for at least 90 days. There is no minimum duration on any other chain.
- block any chosen token addresses from receiving tokens.
- retrieve any funds left in the contract after the airdrop is complete, including funds provided by the Additional Claim Manager.

There is an *Upgrader* role, which can upgrade the contract logic arbitrarily at any point.

Although there are distinct roles with individual powers, they are (at least initially) managed by a single administrator address.

Lastly, it should be noted that there is nothing in this contract that limits the total supply of the distributed token. There could be other addresses that received tokens independently.

Findings

Here we present our findings.

High Severity

H-01 The `inputAdditionalClaimer` function can be front-run

If an account with the `ADDITIONAL_CLAIM_MANAGER` role calls the `inputAdditionalClaimer` function to modify the amount of tokens that an account can additionally claim, the claimer can front-run the call to claim the initial amount, and then claim the new amount, leading to a double-claiming issue.

For example:

1. An account with the `ADDITIONAL_CLAIM_MANAGER` role sets the additional claim amount for Alice as X tokens.
2. The additional claim manager calls the `inputAdditionalClaimer` function to update Alice's additional claim from X to Y.
3. Alice front-runs this transaction, and claims the original X amount of tokens by calling the `additionalClaim` function, and Alice's additional claim amount goes to 0.
4. The `inputAdditionalClaimer` function call is executed afterwards, setting Alice's additional claim amount to Y.
5. Alice back-runs this function by calling the `additionalClaim` function again, and claims Y tokens, claiming X + Y tokens in total.

Consider specifying the increase or decrease to the claimer's additional claim amount instead of overwriting it.

Medium Severity

M-01 Access control on redemptions

The `claim` and `externalClaim` functions have no access control, allowing anyone to process a token claim on behalf of any address, thereby rearranging the claimant's asset holdings. Consider ensuring that only the recipient can claim tokens. Alternatively, if this behavior is expected, consider renaming the functions to `distribute` and `distributeClaim` or similar.

M-02 Inaccurate airdrop duration

The `Airdrop` contract includes a minimum duration when deployed on the Ethereum mainnet. However, the airdrop completion time is calculated from the time of contract deployment rather than when the airdrop commences. Moreover, it does not account for the possibility that an airdrop manager temporarily disables the airdrop. Since the timeline is indifferent to whether the contract can actually process the claims and distribute tokens, users cannot rely on the minimum duration guarantee.

Consider setting the completion time based on when the airdrop is enabled, and extending or resetting it whenever the airdrop is disabled.

M-03 Inconsistent duplicate filtering

The `externalClaim` function ensures that neither the nonce nor the destination address have been used in a previous external claim. Although holders of the `EXTERNAL_VERIFIER_ROLE` have complete control over the claim specifications, we suspect they must be trusted to avoid creating multiple valid claims that share either a nonce or destination address. Otherwise, this would introduce a race condition for which claim is redeemable. If this assumption is accurate, consider removing the `claimedAddresses` mapping and rely purely on the `nonce` to prevent duplicate claims.

M-04 Sparse Comments

The codebase contains minimal code comments and no function comments. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, the only code comment identifies the behavior (that some roles are not initialized) but does not provide a justification.

Docstrings and code comments improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Consider thoroughly documenting all functions (and their parameters) that are part of the contracts' public API. Functions implementing sensitive or non-self-explanatory functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Low Severity

L-01 Deadline timestamp is not inclusive

External claimers have to claim tokens strictly before the deadline (as long as the end time has not been reached), while regular claimers can claim tokens on the end time. In the case where deadline equals the end time, external claimers will not be able to claim their tokens.

Consider checking that the external claim timestamp is less than or equal to the deadline.

L-02 Disable implementation contract

The `Airdrop` contract does not disable initializers in the constructor, as recommended by the imported `Initializable` contract. This means that anyone can call the initializer on the implementation contract to set the contract variables and assign the roles. To reduce the potential attack surface, consider calling `_disableInitializers` in the constructor.

L-03 Inaccurate error messages

The `claim` function of the `Airdrop` contract will revert if it cannot verify that the claim corresponds to the Merkle tree. However, the associated error message only identifies one of several possible reasons for failure.

Additionally, there are multiple instances where the phrase "valid address" implicitly means "non-zero address".

Consider using more precise error messages to better convey the reason for failure.

L-04 Lack of event emission after sensitive actions

The `setTreasury` function does not emit an event after changing the value of the `treasury` state variable. Consider emitting events after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

L-05 Missing validation

The `initialize` function validates that three of the parameters are non-zero, but does not perform the same validation for the `_token` parameter. Consider ensuring this parameter is non-zero as well.

L-06 Redundant validations

The `claim` and `externalClaim` functions redundantly check the airdrop status, since they are both protected by the `ifAirdropEnabled` modifier. Additionally, the `externalClaim` function ensures the `deadline` is non-zero even though this is implied by the second condition, where the deadline is greater than the block timestamp.

Consider removing the redundant validations.

L-07 Renaming suggestions

Some functions and variables could benefit from renaming. These are our suggestions:

- the `ifAirdropEnabled` modifier could be renamed to `ifAirdropActive`, since the `enabled` status is only one of the conditions.
- the `enableAirdrop` function could be renamed to `setAirdropStatus`, since it could be used to disable the airdrop.
- the `inputAdditionalClaimer` function could be renamed to `addAdditionalClaimer`.
- the `additionalClaim` function could be renamed to `claimAdditionalTokens`.
- the `setAddressBlock` function could be renamed to `setAddressStatus`.
- the `blockList` variable could be renamed to `denyList`.

L-08 Wrong event emitted

The `additionalClaim` function emits the `TokensReclaimed` event , which is meant to be used only in the `recoverTokens` function. Consider removing it and adding the `TokensReleased` event instead.

Notes & Additional Information

N-01 Inconsistent coding style

Deviations from the Solidity Style Guide were identified throughout the codebase. Taking into consideration how much value a consistent coding style adds to the project's readability, enforcing a standard coding style with help of linter tools such as Solhint is recommended.

N-02 Inconsistent parameter naming

There is no clear convention used for function parameter naming. Some functions such as `claim` and `initialize` define parameters using a prepended underscore, for example, `_index`, `_account`, and `_amount`. However, other functions such as `externalClaim` and `inputAdditionalClaimer` define parameters without using underscores, for example `claimer` and `amount`.

To improve readability, consider defining a naming convention for parameters and variables, and following it consistently throughout the codebase.

N-03 Incorrect function visibility

The `initialize` function is defined as `public` but is not used internally in the `Airdrop` contract. Consider changing its visibility to `external`.

N-04 Misleading comment

The `Airdrop` contract contains a comment that appears unrelated to the surrounding code. Consider clarifying the comment or removing it.

N-05 Outdated Solidity version in use

An outdated Solidity version, `0.8.0`, is currently in use. As Solidity is now under a fast release cycle, consider using the latest version of the compiler at the time of deployment (presently `0.8.14`).

N-06 Unnecessary storage gap

The `Airdrop` contract contains a `__gap` contract variable that is supposed to ensure the contract uses exactly 50 storage slots. This can be useful during upgrades when its descendant contracts also usage storage. However:

- it incorrectly causes the contract to use 51 storage slots, and
- it is not intended to be inherited.

Consider removing the unnecessary `__gap` variable.

N-07 Unused event

Line 49 of the `Airdrop` contract declares a `ExternalAllocationsReduced` event. As it is never emitted, consider removing the declaration or emitting the event appropriately. If the event is emitted, consider indexing the `tier` parameter to avoid hindering the task of off-chain services searching and filtering for specific events.

N-08 Using deprecated `_setupRole` AccessControl function

In the `initialize` function, the `_setupRole` function from the `AccessControlUpgradeable` contract is being used to initialize the `DEFAULT_ADMIN_ROLE`, `AIRDROP_MANAGER_ROLE`, and `UPGRADER_ROLE` roles, which is deprecated in the version of openzeppelin-contracts-upgradeable in use. Consider using the `_grantRole` function instead.

Conclusions

One high-severity issue was found. Some changes were proposed to follow best practices and reduce the potential attack surface.