

# Flare Network

## Smart Contract Audit



# Flare Airdrop Update

## Smart Contract Audit

---

V221109

Prepared for Flare • November 2022

1. Executive Summary

2. Assessment and Scope

3. Summary of Findings

4. Detailed Findings

FLR-23 Failed transfers funds' will be irreversibly locked in InitialAirdrop

FLR-24 Accounting mismatch on failed transfers

FLR-25 Airdrop contract can not be funded with native transfers

5. Disclaimer

# 1. Executive Summary

In October 2022, Flare engaged [Coinspect](#) to perform a source code review of an update to the initial airdrop smart contract to be used during the TDE of the **Flare Network**. The objective of the project was to evaluate the security of the smart contract modifications.

The following issues were identified during the initial assessment:

High Risk	Medium Risk	Low Risk
Open <b>0</b>	Open <b>0</b>	Open <b>0</b>
Fixed <b>0</b>	Fixed <b>1</b>	Fixed <b>1</b>
Reported <b>0</b>	Reported <b>1</b>	Reported <b>1</b>

Coinspect found one medium risk issue related to failed airdrop transfers resulting in these funds being locked in the contract. A low risk issue is reported as well, related to the incorrect accounting of failed airdrop transfers.

The new functionality added to the initial airdrop contract sends funds to a distribution account in some scenarios. The redistribution of those funds to the originally intended addresses was not in scope for this audit.

## 2. Assessment and Scope

The audit started on **October 27, 2022** and was conducted on the `initial_airdrop_update` branch of the git repository at <https://gitlab.com/flarenetwork/flare-smart-contracts> as of commit `bab470c18fe01cd6228f029c1b3aa139630839b6` of **October 25, 2022**.

The audited files have the following sha256sum hash:

```
51e8b658586b46b61ee50ad76aac0087abb6dc4b9ab7039df1f4bb51d3fd4d36
genesis/implementation/InitialAirdrop.sol
eeb90a716cbbc317f02c9a80c188d2543d33a25f4f210317bb28efe1f436722a  utils/implementation/AddressSet.sol
```

This audit focused on the changes introduced by the following merge request to the smart contracts:

- [https://gitlab.com/flarenetwork/flare-smart-contracts/-/merge\\_requests/551](https://gitlab.com/flarenetwork/flare-smart-contracts/-/merge_requests/551)

The `InitialAirdrop` contract manages how the tokens will be distributed on different airdrop stages. The actions performed by this contract are mainly governance-controlled by the `onlyGovernance` modifier.

These are the new changes introduced in this update:

- 1) The `Initial Airdrop` contract does not inherit from `GovernedAtGenesis` anymore. Instead, `Governed` is now used.
- 2) A new `removeAirdropAccount` function was added. Which allows the governance to send funds to certain accounts before the token distribution event (TDE) and rearranges the `airdropAccounts` array.

The newly added `removeAirdropAccount` accepts a flag to determine if the funds are sent back to a distribution account or directly to the removed account. The previously reviewed code releases did not have that functionality. This is meant to be used to fund exchanges before the airdrop begins so they can support the market once the airdrop takes place. Besides that, the exact handling of the funds destined to the distribution account is uncertain. **Coinspect did not review during this audit the process that should redistribute those funds sent from the `InitialAirdrop` contract to the distribution account back to their intended owners.**

Also, the external `transferAirdrop` function processes the transfer of funds once the airdrop begins. As it loops over the target accounts, unsuccessful transfers do not halt the execution of the loop emitting a `AirdropTransferFailure` event. However, if an airdropped address is a contract where the transfer of value fails, **the allocated tokens for that user will remain locked** as no recovery function is implemented (FLA-23). Also, the internal accounting does not contemplate failure scenarios, adding to the total amount sent the amount corresponding to failed transactions (FLA-24).

Currently the `InitialAirdrop` contract lacks a payable fallback which allows its funding with native tokens (FLA-25). It is advised to document how it will be funded, for example: by a regular transfer or self-destructing a foreign contract. **If the first option is chosen, a payable fallback must be added.**

It is worth mentioning that the codebase declares an outdated pragma version (0.7.6) which has breaking differences regarding the current commonly used compiler versions (0.8.x).

### 3. Summary of Findings

Id	Title	Total Risk	Fixed
<a href="#">FLR-23</a>	Failed transfers funds' will be irreversibly locked in InitialAirdrop	Medium	✓
<a href="#">FLR-24</a>	Accounting mismatch on failed transfers	Low	✓
<a href="#">FLR-25</a>	Airdrop contract can not be funded with native transfers	Info	✓

## 4. Detailed Findings

**FLR-23**

Failed transfers funds' will be irreversibly locked in InitialAirdrop

Total Risk  
**Medium**

Impact  
Medium

Location  
InitialAirdrop.sol

Fixed



Likelihood  
Medium

### Description

There is no way to recover funds sent to the initial airdrop contract. All the funds belonging to failed transfers will be locked in the contract forever.

The airdrop will require having in advance the funds to be sent in the InitialAirdrop. Once the airdrop begins, the transferAirdrop function sends the tokens by looping over batches of at most 50 addresses. Receivers might be contracts that do not implement a payable fallback or spend more than the 21,000 hardcoded gas amount on receipt, as a consequence in order not to halt the whole loop an event is emitted without reverting.

However, funds allocated for those accounts whose airdrop reverted for the reasons mentioned before will remain locked on the contract as there are no recovery methods implemented once the airdrop began.

### Recommendation

Include a function that allows recovering locked funds after the airdrop is finished only callable by the Governance.

### Status

Fixed.

A `withdrawUndistributedFunds` function with the following properties was added:

- It is access controlled, only callable by `governance()`.

- It could only be called if `transferAirdrop` was previously called enough times to loop over all the airdrop accounts. Has a strict equal to check that property. Coinspect did not observe ways that invariants could be broken, however, in case of exceeding the checked amount, funds will still remain locked.
- Performs a low level call transferring to an **arbitrary recipient** the remaining balance of the contract.



**FLR-24****Accounting mismatch on failed transfers**

Total Risk	Impact	Location
<b>Low</b>	Low	InitialAirdrop.sol
Fixed	Likelihood	
✓	Low	

## Description

The `totalTransferredAirdropWei` state variable does not reflect the total actually transferred if any of the transfers fails. Any code addition or off-chain components relying on this value could be misguided.

While calling `transferAirdrop()`, the `totalTransferredAirdropWei` is added to the `weiAmount` that is assumed to be sent on the last step of the loop. However if the transfer is not successful, this amount is not subtracted from the accumulator. Consequently, a mismatch between the real amount sent and the accounted one will exist.

The internal accountability while transferring airdrop is made as follows:

```
function transferAirdrop() external airdropStarted mustBalance nonReentrant {
    uint256 upperBound = Math.min(nextAirdropAccountIndexToTransfer + 50, airdropAccounts.length);
    uint256 totalTransferredAirdropWeiTemp = 0;
    for (uint256 i = nextAirdropAccountIndexToTransfer; i < upperBound; i++) {
        // Get the account and amount
        address account = airdropAccounts[i];
        uint256 amountWei = airdropAmountsWei[account];
        // update state
        delete airdropAmountsWei[account];
        delete airdropAccountsIndex[account];
        delete airdropAccounts[i];
        // Update total transferred
        totalTransferredAirdropWeiTemp = totalTransferredAirdropWeiTemp.add(amountWei);
        // Send
        /* solhint-disable avoid-low-level-calls */
        //slither-disable-next-line arbitrary-send-eth
        (bool success, ) = account.call{ value: amountWei, gas: 21000 }("");
        /* solhint-enable avoid-low-level-calls */
        if (!success) {
            emit AirdropTransferFailure(account, amountWei);
        }
    }
    // Update grand total transferred
    totalTransferredAirdropWei =
        totalTransferredAirdropWei.add(totalTransferredAirdropWeiTemp);

    // Update current position
    nextAirdropAccountIndexToTransfer = upperBound;
}
```

For each step, the `totalTransferredAirdropWeiTemp` accumulates the `amountWei` of each transfer. If the low level call performed one step later fails (e.g. if the gas spent is greater than 21,000 or if its a contract that does not implement a payable fallback), the non transferred `amountWei` will be later added to `totalTransferredAirdropWei`.

*It is worth noting that the event emitted on transfer failure could be used to account for the transfers.*

No further impact was identified during the time allotted for this assessment.

## Recommendation

If the call fails, subtract the `amountWei` from the temporary accumulator.

Alternatively, clearly document that the `totalTransferredAirdropWei` variable includes attempted transfers that were not successful.


Consider modifying the `totalTransferredAirdropWei` variable name in order to reflect what is actually represented.

## Status

Fixed.

Now the `transferAirdrop` function distinguishes with two conditional branches successful (incrementing the accumulator) from failed calls (as before, emitting `AirdropTransferFailure`).

**FLR-25****Airdrop contract can not be funded with native transfers**

Total Risk	Impact	Location
<b>Info</b>	-	InitialAirdrop.sol
Fixed	Likelihood	
	-	

## Description

The `InitialAirdrop` contract lacks a payable fallback function that allows receiving native tokens prior to the airdrop. Regular transfers as a funding method won't be supported.

Considered an informational issue, as Coinspect is not aware of how the funding will be conducted and in case of deploying the contract as-is, funds could be sent by self destructing a contract.

## Recommendation

Document how the `InitialAirdrop` contract will be funded before the airdrop begins. If it is going to be funded by regular transfers, add a payable fallback.

## Status

Fixed.

The `InitialAirdrop` contract now allows only governance native token transfers via `receive payable`.

## 5. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.