Security Audit Report

Fluity Airdrop

SECBIT

May 14, 2021

1. Introduction

Fluity Airdrop is a smart contract for claiming airdrop rewards. SECBIT Labs conducted an audit from May 13th to May 14th, 2021, including an analysis of the contract in 3 areas: **code bugs**, **logic flaws**, and **risk assessment**. The assessment shows that the Fluity Airdrop contract has no critical security risks. The SECBIT team has some tips on logical implementation, potential risks, and code revising(see part 4 for details).

Type	Description	Level	Status
Gas Optimization	4.3.1 Using `external` function instead of `public` function to save gas.	Info	Discussed

2. Contract Information

This part describes the basic contract information and code structure.

2.1 Basic Information

The basic information about the Bridge Warm Wallet contract is shown below:

- Smart contract code
  - https://github.com/fluity-finance/lqty-airdrop/blob/main/contracts/MerkleAirdropper.sol
  - commit: d359dba

2.2 Contract List

The following content shows the contracts included in the Bridge Warm Wallet project:

Name	Lines	Description
MerkleAirdropper.sol	68	Claim airdrop rewards

3. Contract Analysis

This part describes code assessment details, including two items: "role classification" and "functional analysis".

3.1 Role Classification

There are two key roles in the protocol, namely Authority Account and Reward Claimer Account.

- Authority Account
    - Description Contract administrator
    - Authority
        - Update merkle root
        - Update authority address
    - Method of Authorization The creator of the contract, or authorized by the transferring of governance account
- Reward Claimer Account
    - Description This account can claim airdrop rewards
    - Authority
        - Receive rewards
    - Method of Authorization Anyone qualified

3.2 Functional Analysis

Fluity Airdrop is a smart contract for claiming airdrop rewards. We can divide the critical functions of the auction contract into several parts:

- proposewMerkleRoot

Update Merkle root and start a new reward cycle.

- claim

  Qualified users claim airdrop rewards.

4. Audit Detail

This part describes the process, and the detailed results of the audit also demonstrate the problems and potential risks.

4.1 Audit Process

The audit strictly followed the audit specification of SECBIT Lab. We analyzed the project from code bug, logical implementation, and potential risks. The process consists of four steps:

- Fully analysis of contract code line by line.
- Evaluation of vulnerabilities and potential risks revealed in the contract code.
- Communication on assessment and confirmation.
- Audit report writing.

4.2 Audit Result

After scanning with adelaide, sf-checker, and badmsg.sender (internal version) developed by SECBIT Labs and open source tools including Mythril, Slither, SmartCheck, and Securify, the auditing team performed a manual assessment. The team inspected the contract line by line, and the result could be categorized into two types:

Number	Classification	Result
1	Normal functioning of features defined by the contract	✓
2	No obvious bug (e.g., overflow, underflow)	✓
3	Pass Solidity compiler check with no potential error	✓

4	Pass common tools check with no obvious vulnerability	✓
5	No obvious gas-consuming operation	✓
6	Meet with ERC20	✓
7	No risk in low-level call (call, delegatecall, callcode) and in-line assembly	✓
8	No deprecated or outdated usage	✓
9	Explicit implementation, visibility, variable type, and Solidity version number	✓
10	No redundant code	✓
11	No potential risk manipulated by timestamp and network environment	✓
12	Explicit business logic	✓
13	Implementation consistent with annotation and other info	✓
14	No hidden code about any logic that is not mentioned in design	✓
15	No ambiguous logic	✓
16	No risk threatening the developing team	✓
17	No risk threatening exchanges, wallets, and DApps	✓
18	No risk threatening token holders	✓
19	No privilege on managing others' balances	✓
20	No minting method	✓

21	Correct managing hierarchy	✓

4.3 Issues

4.3.1 Using `external` function instead of `public` function to save gas.

Risk Type	Risk Level	Impact	Status
Gas Optimization	Info	More gas consumption	Unfixed

Description

The declaration of `public` functions that are never called by the contract should be declared `external` to save gas.

```
function setAuthority(address _authority) public authorityOnly
{
        authority = _authority;
}

// Each week, the authority calls to submit the merkle root
for a new airdrop.
function proposewMerkleRoot(bytes32 _merkleRoot) public
authorityOnly {
        ......
}
```

Suggestion

Use the `external` attribute for functions never called from the contract.

5. Conclusion

After auditing and analyzing the Fluity Airdrop contract, SECBIT Labs found some issues to optimize and proposed corresponding suggestions, which have been shown above. The team has developed a multi-round airdrop contract that is both lean and efficient. SECBIT Labs holds the view that the Fluity Airdrop smart contract has high code quality.

Disclaimer

SECBIT smart contract audit service assesses the contract's correctness, security, and performability in code quality, logic design, and potential risks. The report is provided "as is", without any warranties about the code practicability, business model, management system's applicability, and anything related to the contract adaptation. This audit report is not to be taken as an endorsement of the platform, team, company, or investment.

APPENDIX

Vulnerability/Risk Level Classification

Level	Description
High	Severely damage the contract's integrity and allow attackers to steal ethers and tokens, or lock ethers inside the contract.
Medium	Damage contract's security under given conditions and cause impairment of benefit for stakeholders.
Low	Cause no actual impairment to contract.
Info	Relevant to practice or rationality of the smart contract, could possibly bring risks.

SECBIT Lab is devoted to construct a common-consensus, reliable and ordered blockchain economic entity.

http://www.secbit.io

audit@secbit.io

@secbit_io