Protocol Audit Report

Version 1.0

Shishir S. Kakhandki

May 2, 2024

Protocol Audit Report

Shishir S. Kakhandki

May 2, 2024

Prepared by: Shishir S. Kakhandki Lead Auditors: - Shishir S. Kakhandki

Table of Contents

Protocol Summary

The protocol is looking to airdrop 100 USDC tokens on the zkSync era chain to 4 lucky addresses based on their activity on the Ethereum L1. The Ethereum addresses are:

```
0x20F41376c713072937eb02Be70ee1eD0D639966C
0x277D26a45Add5775F21256159F089769892CEa5B
0x0c8Ca207e27a1a8224D1b602bf856479b03319e7
0xf6dBa02C01AF48Cf926579F77C9f874Ca640D91D
```

Each address will recieve 25 USDC.

Disclaimer

The Shishir S. Kakhandki team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Scope

```
./src/
--> MerkleAirdrop.sol
./script/
--> Deploy.s.sol
```

Roles

Onwer - The one who can withdraw the fees earned by the `claim` function.

Issues found

Severity	Number of issues found
High	1
Medium	0
Low	0
Info	2
Gas	1
Total	4

Findings

High

[H-1] Incorrect merkle root generated, due to which users can claim more than they are elligible

Description The merkle root generated in line `bytes32 public s_merkleRoot = 0xf69aaa25bd4dd10deb2ccd8235266f7cc815f6e9d539e9f4d47cae16e0c36a05` in `Deploy.s.sol` is passed to the `MerkleAirdrop` contract for verifying merkle proofs. This merkle root is generated in `merkle.js`

In `makeMerkle.js`, in line `const amount = (25 * 1e18).toString()`, the variable `amount` is initialized with 18 decimal places while it should have been 6 for `USDC` causing incorrect merkle root generation

Impact More funds i.e. `25 * 1e18` wei of USDC can be claimed by each user while they are only eligible for `25 * 1e6`

Proof of Concepts

```
1  -        bytes32 public merkleRoot = 0
   x3b2e22da63ae414086bec9c9da6b685f790c6fab200c7918f2879f08793d77bd;
2  +        bytes32 public merkleRoot = 0
   xf69aaa25bd4dd10deb2ccd8235266f7cc815f6e9d539e9f4d47cae16e0c36a05;
3
4
5  -        uint256 amountToCollect = (25 * 1e6); // 25.000000
6  +        uint256 amountToCollect = (25 * 1e18); // 25.000000
7
8  -        uint256 amountToSend = amountToCollect * 4;
9  +        uint256 amountToSend = amountToCollect2 * 4;
```

```
10
11 -        address collectorOne = 0
       x20F41376c713072937eb02Be70ee1eD0D639966C;
12 +        address collectorOne = 0
       x20F41376c713072937eb02Be70ee1eD0D639966C;
13
14 -        bytes32 proofOne = 0
       x32cee63464b09930b5c3f59f955c86694a4c640a03aa57e6f743d8a3ca5c8838;
15 -        bytes32 proofTwo = 0
       x8ff683185668cbe035a18fccec4080d7a0331bb1bbc532324f40501de5e8ea5c;
16
17 +        bytes32 proofOne = 0
       x4fd31fee0e75780cd67704fbc43caee70fddcaa43631e2e1bc9fb233fada2394;
18 +        bytes32 proofTwo = 0
       xc88d18957ad6849229355580c1bde5de3ae3b78024db2e6c2a9ad674f7b59f84;
19
20 -        bytes32[] proof = [proofOne, proofTwo];
21 +        bytes32[] proof = [proofOne, proofTwo];
22
23         function testClaimMoreEthThanEligile() public {
24             uint newAmountToCollect = (25 * 1e18);
25             uint256 startingBalance = token.balanceOf(collectorOne);
26             vm.deal(collectorOne, airdrop.getFee());
27             vm.startPrank(collectorOne);
28             airdrop.claim{ value: airdrop.getFee() }(collectorOne,
                   newAmountToCollect, proof);
29             vm.stopPrank();
30             uint256 endingBalance = token.balanceOf(collectorOne);
31             assertEq(endingBalance - startingBalance,
                   newAmountToCollect);
32         }
```

The `merkleRoot` mentioned above is generated for address `0x20F41376c713072937eb02Be70ee1eD0D6399` with claim amount `25 * 1e18` and proof `0x4fd31fee0e75780cd67704fbc43caee70fddcaa43631e2e1b` and `0xc88d18957ad6849229355580c1bde5de3ae3b78024db2e6c2a9ad674f7b59f84` using the script mentioned in `makeMerkle.js`

Recommended mitigation Change line `const amount = (25 * 1e18).toString()` in `makeMerkle.js` to `const amount = (25 * 1e6).toString()`

Informational

[I-1] Move the if block which checks for invalid fee and invalid merkel proofs to a modifier, for cleaner code and better readability

Description In the `MerkleAirdrop.sol::claim`, the “if” blocks make the code cluttered and hard to read

```
function claim(address account, uint256 amount, bytes32[] calldata
    merkleProof) external payable {
        if (msg.value != FEE) {
            revert MerkleAirdrop__InvalidFeeAmount();
        }
        bytes32 leaf = keccak256(bytes.concat(keccak256(abi.encode(
            account, amount))));
        if (!MerkleProof.verify(merkleProof, i_merkleRoot, leaf)) {
            revert MerkleAirdrop__InvalidProof();
        }
        emit Claimed(account, amount);
        i_airdropToken.safeTransfer(account, amount);
    }
```

Impact Decreased readability and not a best practice

Recommended mitigation In the `MerkleAirdrop.sol::claim`, the "if" blocks which check for invalid fee and invalid merkel proofs can be moved to modifiers, like so:

```
+ modifier validFee() {
+         require(msg.value == FEE, "MerkleAirdrop__InvalidFeeAmount");
+         _;
+     }

+ modifier validMerkleProof(address account, uint256 amount, bytes32[]
    calldata merkleProof) {
+        bytes32 leaf = keccak256(abi.encode(account, amount));
+        require(MerkleProof.verify(merkleProof, i_merkleRoot, leaf), "
    MerkleAirdrop__InvalidProof");
+        _;
+    }

+ function claim(address account, uint256 amount, bytes32[] calldata
    merkleProof) external payable validFee validMerkleProof(account,
    amount, merkleProof) {
- function claim(address account, uint256 amount, bytes32[] calldata
    merkleProof) external payable {
-       if (msg.value != FEE) {
-           revert MerkleAirdrop__InvalidFeeAmount();
-       }
-       bytes32 leaf = keccak256(bytes.concat(keccak256(abi.encode(
    account, amount))));
-       if (!MerkleProof.verify(merkleProof, i_merkleRoot, leaf)) {
-           revert MerkleAirdrop__InvalidProof();
-       }
        emit Claimed(account, amount);
        i_airdropToken.safeTransfer(account, amount);
    }
```

[I-2] Using variable names instead of raw values is advisable for a cleaner and readable code

Description In `Deploy.s.sol::run` at line `IERC20(0x1d17CBcF0D6D143135aE902365D2E5e2A16538D4).transfer(address(airdrop), s_amountToAirdrop)`, the address `0x1d17CBcF0D6D143135aE902` is declared raw

Impact Not a best practice and considered messy code

Recommended mitigation Change the access modifier of `s_zkSyncUSDC` declared above to constant and use that instead, like so:

```
-     address public s_zkSyncUSDC = 0x1D17CbCf0D6d143135be902365d2e5E2a16538d4;
+    address public constant s_zkSyncUSDC = 0x1D17CbCf0D6d143135be902365d2e5E2a16538d4;
      function run() public {
         vm.startBroadcast();
         MerkleAirdrop airdrop = deployMerkleDropper(s_merkleRoot, IERC20(s_zkSyncUSDC));
         // Send USDC -> Merkle Air Dropper
         //@audit could have used the variable
-        IERC20(0x1d17CBcF0D6D143135aE902365D2E5e2A16538D4).transfer(address(airdrop), s_amountToAirdrop);
+        IERC20(s_zkSyncUSDC).transfer(address(airdrop), s_amountToAirdrop);
         vm.stopBroadcast();
     }
```

Gas

[G-1] State variables can be changed to constant to save gas

DescriptionIn `Deploy.s.sol` the following variables can be declared as `constant` to save on Gas

```
    address public s_zkSyncUSDC = 0x1D17CbCf0D6d143135be902365d2e5E2a16538d4;
    bytes32 public s_merkleRoot = 0xf69aaa25bd4dd10deb2ccd8235266f7cc815f6e9d539e9f4d47cae16e0c36a05;
    // 4 users, 25 USDC each
    uint256 public s_amountToAirdrop = 4 * (25 * 1e6);
```

Recommended mitigation Kindly use constant and immutable keywords wherever applicable to save some valuable gas