AirDropper Audit Report

Version 1.0

FalseGenius

AirDropper Audit Report

FalseGenius

May 02, 2024

Prepared by: FalseGenius Lead Auditors: - None

Table of Contents

Protocol Summary

This project airdrops 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.

The codebase was designed to be deployed to the zkSync era chain and therefore uses the zksync-foundry fork of foundry.

Risk Classification

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

Audit Details

The findings described in this document correspond to the following commit hash: Latest branch

Scope

```
./src/
#-- MerkleAirdrop.sol

```

```
./script/
#-- Deploy.s.sol
```

Roles

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

Executive Summary

This was part of First-flights on CodeHawks. The deadline was within 7 days. I used Foundry tests, Slither, Aderyn and Manual Review for the Audit

Issues found

Severity	Number of Issues Found
High	3
Medium	0
Low	1
Gas	0
Informational	2
Total	6

Findings

High

[H-01] Missing checks in `MerkleAirdrop::claim()` for accounts that have already claimed USDC tokens, allowing users to claim token multiple times.

Description: The `claim()` function currently lacks checks to prevent accounts from claiming more than once, allowing winners to receive a larger share of the drop than intended. Lucky winners can unfairly deplete the pool for other winners, leading to an improper distribution of tokens, eroding trust in the contract.

Impact: Without a check for duplicate claims, users can repeatedly claim USDC tokens, causing financial harm to the contract. This undermines fairness and integrity of the airdrop distribution process.

Proof of Concept: Have a collector i.e., collectorOne trigger claim function multiple times,

Logs: - MerkleAirdrop token balance before: 100000000 - MerkleAirdrop token balance before: 0

PoC

```
1
2     function testUsersCanWithdrawSeveralTimes() public {
3         uint256 startingBalance = token.balanceOf(collectorOne);
4         console.log("MerkleAirdrop balance before: %s", token.balanceOf
              (address(airdrop)));
5
6         vm.deal(collectorOne, airdrop.getFee()*4);
7         vm.startPrank(collectorOne);
8
9         airdrop.claim{ value: airdrop.getFee() }(collectorOne,
              amountToCollect, proof);
10        airdrop.claim{ value: airdrop.getFee() }(collectorOne,
              amountToCollect, proof);
11        airdrop.claim{ value: airdrop.getFee() }(collectorOne,
              amountToCollect, proof);
12        airdrop.claim{ value: airdrop.getFee() }(collectorOne,
              amountToCollect, proof);
13        console.log("Contract balance after: %s", token.balanceOf(
              address(airdrop)));
14        vm.stopPrank();
15        uint256 endingBalance = token.balanceOf(collectorOne);
16        assertEq(endingBalance - startingBalance, amountToCollect*4);
17    }
```

Recommended Mitigation: Consider introducing an claimlist that stores accounts that have already claimed. Before processing a claim, verify whether the account has already claimed tokens and reject duplicate claims accordingly.

```
1
2 +   mapping(address => bool) private claimed;
3
4     function claim(address account, uint256 amount, bytes32[] calldata
          merkleProof) external payable {
5 +   if (claimed[account]) revert
      MerkleAirdrop__AccountHasAlreadyClaimedTokens();
6
7     // Rest of the code
8
9
10 +   claimed[account] = true;
11    emit Claimed(account, amount);
12    i_airdropToken.safeTransfer(account, amount);
```

```
13
14      }
```

[H-02] Incorrect `s_merkleRoot` in `Deploy.s.sol`

Description: `s_merkleRoot` value is generated using an amount with 18 decimals while actual USDC has 6 decimals. This mismatch results in an invalid Merkle root. Consequently, passing this invalid root to MerkleAirdrop contract would render `claim` function useless as the amount passed to it needs to have 6 decimals.

Impact: `MerkleAirdrop::claim` function would consistently revert if the merkle root passed in the constructor is invalid. Consequently,users would be unable to withdraw USDC tokens, severely disrupting functionality of the protocol.

Proof of Concept: Pass the amount `(25 * 1e6)` for the lucky addresses in `makeMerkle.js`. It results in the following Merkle root,

- 0x3b2e22da63ae414086bec9c9da6b685f790c6fab200c7918f2879f08793d77bd

The one hardcoded in deploy contract is the result when root is generated with addresses and amount of 18 decimals,

- 0xf69aaa25bd4dd10deb2ccd8235266f7cc815f6e9d539e9f4d47cae16e0c36a05

```
1
2       function testRootsAreDifferentForDifferentAmountDecimals() public
            pure {
3           // This is s_merkleRoot in deploy
4           bytes32 eighteenDecimalsRoot = 0
                xf69aaa25bd4dd10deb2ccd8235266f7cc815f6e9d539e9f4d47cae16e0c36a05
                ;
5           bytes32 sixDecimalsRoot = 0
                x3b2e22da63ae414086bec9c9da6b685f790c6fab200c7918f2879f08793d77bd
                ;
6
7           assertTrue(eighteenDecimalsRoot != sixDecimalsRoot);
8       }
```

Recommended Mitigation: Prior to passing the `s_merkleRoot` to the MerkleAirdrop contract, ensure that its value is generated using correct decimal precision.

[H-03] Inconsistent USDC Token address and Ownership issue in `Deploy` script.

Description: The USDC token address `0x1d17CBcF0D6D143135aE902365D2E5e2A16538D4` hardcoded in `Deploy::run()` to transfer tokens differs from the `s_zkSyncUSDC` address which

is passed down as airdropToken when deploying `MerkleAirdrop` contract. This directly affects `MerkleAirdrop::claim` where the it's used to transfer tokens, thereby directly distrupting the functionality of the protocol if this address is not the correct token address of USDC token.

```
1 @>  address public s_zkSyncUSDC = 0
    x1D17CbCf0D6d143135be902365d2e5E2a16538d4;
2     function run() public {
3         vm.startBroadcast();
4 @>       MerkleAirdrop airdrop = deployMerkleDropper(s_merkleRoot,
    IERC20(s_zkSyncUSDC));
5
6 @>      IERC20(0x1d17CBcF0D6D143135aE902365D2E5e2A16538D4).transfer(
    address(airdrop), s_amountToAirdrop);
7         vm.stopBroadcast();
8     }
```

Additionally, for transfering `s_amountToAirdrop` USDC to `MerkleAirdrop`, the account executing the transaction needs to have sufficient balance of USDC tokens on Mainnet/Testnet, depending on the network. The way deploy script is set up, the owner is the Deploy contract which does not possess any USDC tokens, so the transfer function would revert and deployment would fail.

Impact: The inconsistency in the USDC token address and the ownership issue significantly disrupts the functionality of the MerkleAirdrop protocol. Users may experience failed token transfers, resulting in a loss of trust and confidence in the protocol. The deployment failures hinder the successful initialization of the contract, delaying or preventing the airdrop distribution process altogether.

Proof of Concept: The address assigned to s_zkSyncUSDC is not equal to the one used for transfer of token,

```
1     function testInconsistentUSDCTokenAddressInDeploy() public pure {
2         address s_zkSyncUSDC = 0
              x1D17CbCf0D6d143135be902365d2e5E2a16538d4;
3         address transferAddress = 0
              x1d17CBcF0D6D143135aE902365D2E5e2A16538D4;
4
5         assertTrue(s_zkSyncUSDC != transferAddress);
6     }
```

Run `make deploy` script of the Makefile. The script would revert with error routing to the transfer token function,

```
1 @>   IERC20(0x1d17CBcF0D6D143135aE902365D2E5e2A16538D4).transfer(
    address(airdrop), s_amountToAirdrop);
```

Recommended Mitigation

Either use `transferFrom` to transfer USDC from another account (with USDC possession) to the airdrop, or ensure the caller of the run() function has enough USDC to transfer. Consider using the

`s_zkSyncUSDC` for the IERC20 transfer instead of hardcoded address, and ensure that the USDC token address is correct.

```
-   address zksyncusdc = 0x1D17CbCf0D6d143135be902365d2e5E2a16538d4;
    // The correct zksyncUSDC mainnet address
+   address zksyncusdc = 0x1d17CBcF0D6D143135aE902365D2E5e2A16538D4;

    function run() public {
-       vm.startBroadcast();
        // Start broadcast with an account with ownership of USDC
+       vm.startBroadcast(ACCOUNT_A);
        MerkleAirdrop airdrop = deployMerkleDropper(s_merkleRoot,
            IERC20(s_zkSyncUSDC));

-       IERC20(0x1d17CBcF0D6D143135aE902365D2E5e2A16538D4).transfer(
    address(airdrop), s_amountToAirdrop);
+       IERC20(zksyncusdc).approve(address(airdrop), s_amountToAirdrop)
    ;
+       IERC20(zksyncusdc).transfer(address(airdrop), s_amountToAirdrop
    );
        vm.stopBroadcast();
    }
```

Low

[L-01] `Deploy::run()` doesn't explicitly set the owner of the deployed `MerkleAirdrop` contract.

Description: The owner address is not clear, and this could lead to problems claiming fees later as only the owner can withdraw contract fees.

```
    // Deploy contract is the owner...
@>  vm.startBroadcast();
    MerkleAirdrop airdrop = deployMerkleDropper(s_merkleRoot, IERC20(
        s_zkSyncUSDC));
```

Impact: Fee deposited by users into the contract would be locked forever if the owner is not explicitly set.

Recommended Mitigation: Consider checking your test setup to determine who becomes owner of the `MerkleAirdrop` contract before it is deployed.

Informational/Non-Crit

[I-01] Event `MerkleRootUpdated()` is not used anywhere

Description: Unused events/functions clutter up the code. It's generally good practice to remove unused code for better maintainability.

```
1 -  event MerkleRootUpdated(bytes32 newMerkleRoot);
```

[I-02] Event is missing `indexed` field

Description: Index event fields make the field more quickly accessible to off-chain tools that parse events.

```
1 @>  event Claimed(address account, uint256 amount);
```