AirDropper Security Audit Report by MK (12th DEC 2024)

High Risk Findings

[H-1] Incorrect USDC Address in `Deploy.s.sol` Causes Claim Failure

Description:

The s_zkSyncUSDC address in Deploy.s.sol is set incorrectly, causing the claiming process to fail. This issue locks funds in the MerkleAirdrop contract because the token address is immutable and cannot be changed after deployment.

Impact:

The incorrect s_zkSyncUSDC address prevents users from claiming their airdropped funds, leading to a poor user experience and potential loss of trust. Additionally, the immutability of the token address results in funds being permanently stuck in the MerkleAirdrop contract, requiring a redeployment and redistribution effort to resolve the issue. This impacts both operational efficiency and contract reliability.

Proof of Concept:

1. The address at line 8 is not same as in line 18.

```
8   address public s_zkSyncUSDC = 0x1D17CbCf0D6d143135be902365d2e5E2a16538d4;

18  IERC20(0x1d17CBcF0D6D143135aE902365D2E5e2A16538D4).transfer(address(airdrop),
s_amountToAirdrop);
```

Recommended Mitigation:

1. Always use variables in your code and use a valid USDC address. The following changes will fix the bug.

```
-  address public s_zkSyncUSDC = 0x1D17CbCf0D6d143135be902365d2e5E2a16538d4;
//remove this line
+  address public s_zkSyncUSDC = 0x1d17CBcF0D6D143135aE902365D2E5e2A16538D4;
//modified to this
-  IERC20(0x1d17CBcF0D6D143135aE902365D2E5e2A16538D4).transfer(address(airdrop),
s_amountToAirdrop); //removed this line
+  IERC20(s_zkSyncUSDC).transfer(address(airdrop), s_amountToAirdrop);
// modified and used variable
```

[H-2] Winners can withdraw multiple times

Description:

An eligible airdrop user can verify their inclusion in the Merkle tree and claim their allocated amount. However, there is no mechanism to track users who have already claimed their airdrop, leaving the Merkle tree unchanged. This flaw allows users to repeatedly call the MerkleAirdrop::claim() function, potentially draining the contract's funds.

Impact:

A malicious user can repeatedly call the MerkleAirdrop::claim() function, draining the contract of all its funds and preventing other eligible users from claiming their airdrop amounts.

Proof of Concept:

1. Write a unit test to check for multiple withdrawal by a single user.

```
function testUsersCanClaimMultipleTimes() public {
        uint256 startingBalance = token.balanceOf(collectorOne);
        vm.deal(collectorOne, airdrop.getFee() * 4);

        vm.startPrank(collectorOne);
        airdrop.claim{value: airdrop.getFee()}(
            collectorOne,
            amountToCollect,
            proof
        );
        console2.log("Now collector 1 has :", token.balanceOf(collectorOne));
        airdrop.claim{value: airdrop.getFee()}(
            collectorOne,
            amountToCollect,
            proof
        );
        console2.log("Now collector 1 has :", token.balanceOf(collectorOne));
        airdrop.claim{value: airdrop.getFee()}(
            collectorOne,
            amountToCollect,
            proof
        );
        console2.log("Now collector 1 has :", token.balanceOf(collectorOne));
        airdrop.claim{value: airdrop.getFee()}(
            collectorOne,
            amountToCollect,
            proof
        );
        vm.stopPrank();

        uint256 endingBalance = token.balanceOf(collectorOne);
        console2.log("Now collector 1 has :", token.balanceOf(collectorOne));
        assertEq(endingBalance - startingBalance, amountToCollect * 4);
    }
```

2. Console log:

```
[PASS] testUsersCanClaimMultipleTimes() (gas: 123761)
Logs:
  Now collector 1 has : 25000000
  Now collector 1 has : 50000000
  Now collector 1 has : 75000000
  Now collector 1 has : 100000000

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.79ms (543.70µs CPU
time)
```

Recommended Mitigation:

Implement a mapping to track the addresses that have already claimed their airdrop amounts. Each time a user attempts to claim, check this mapping to ensure they haven't already received their allocation. If they haven't, update the mapping and allow the claim; otherwise, revert the transaction to prevent duplicate claims. This ensures that each eligible user can only claim once.

[H-3] MerkleRoot value is wrong

Description:

The value specified in the Deploy::s_merkleRoot contract does not correspond to the root of the Merkle tree used for the airdrop. This mismatch invalidates the Merkle proofs, preventing eligible users from successfully claiming their airdropped tokens.

Impact:

Every claim attempt by the winners will revert because the "amount" values in the Merkle tree (18 decimals) do not match the actual token amount in the contract (6 decimals). As a result, the verification function returns false for every call.

Notably, even if someone attempted to exploit this by claiming with an amount in 18 decimals, hoping to withdraw more than allocated, the contract would still revert. This is because the available USDC balance in the contract is limited to 100 tokens with 6 decimals, insufficient to fulfill such a claim.

Proof of Concept:

1. In the `makeMerkle.js` file we have the following snippet of code:

```
const amount = (25 * 1e18).toString();
const values = [
  ["0x20F41376c713072937eb02Be70ee1eD0D639966C", amount],
  ["0x277D26a45Add5775F21256159F089769892CEa5B", amount],
  ["0x0c8Ca207e27a1a8224D1b602bf856479b03319e7", amount],
  ["0xf6dBa02C01AF48Cf926579F77C9f874Ca640D91D", amount],
];
const tree = StandardMerkleTree.of(values, ["address", "uint256"]);
console.log("Merkle Root:", tree.root);
```

2. If we execute

```
make merkle
```

3. we obtain that the Root value is the same as the one wrote in `Deploy::s_merkleRoot` :

```
Merkle Root: 0xf69aaa25bd4dd10deb2ccd8235266f7cc815f6e9d539e9f4d47cae16e0c36a05
```

4. However, the tree that has that root value assumes that the token that we´re airdropping has 18 decimals!

```
const amount = (25 * 1e18).toString();
```

5. USDC has 6 decimals. So, the correct amount would be:

```
const amount = (25 * 1e6).toString();
```

6. Doing the previous change, the output of

```
make merkle
```

7. we get the following merkle

```
Merkle Root: 0x3b2e22da63ae414086bec9c9da6b685f790c6fab200c7918f2879f08793d77bd
```

Recommended Mitigation:

1. Modify the value of the MerkleRoot in the Deploy contract to the correct value found in the `MerkleAirdropTest` file : 0x3b2e22da63ae414086bec9c9da6b685f790c6fab200c7918f2879f08793d77bd

[H-4] Account Abstraction Wallets Cause Claim Issues Due to Cross-Chain Address Differences

Description:

Users with account abstraction wallets have different addresses across chains. The MerkleAirdrop::claim function verifies eligibility using Ethereum addresses in the Merkle root, but on zkSync, account abstraction wallets have different addresses. As a result, affected users cannot claim their tokens.

Impact:

Users with account abstraction wallets have different addresses on the zkSync Era chain compared to Ethereum. As a result, they cannot claim their USDC tokens because the Merkle root verification requires their Ethereum address.

Proof of Concept:

1. In the docs, it is mentioned as follows":

```
"Our team 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"
```

Recommended Mitigation: Ensure that the addresses in the makeMerkle file for the lucky users correspond to their zkSync Era chain addresses.

Low Risk Findings

[L-1] `MerkleAirdrop::FEE` Makes Claiming Fees Economically Impractical

Description:

The low MerkleAirdrop::FEE of 1 Gwei makes it economically impractical for the owner to claim fees, even on the zkSync chain with its low gas costs. With a gas cost of 30,479 units for the claimFees function and a zkSync gas price of 0.02 Gwei, the total cost is about 0.000000609 Ether. To make claiming fees economically sensible, at least 609 successful airdrop claims would be required, which is unlikely with the current number of addresses in the Merkle tree. The fee should be increased or removed to address this issue.

Proof of Concept:

1. This contract is designed to airdrop 25 USDC to 4 lucky winners, with a claiming fee of only 1 Gwei (1e9). The expected fee balance would be 4 Gwei, but calling claimFees to transfer this amount to the contract owner would incur a cost of 21,000 Gwei, even with a gas price of 1 Gwei.

Recommended Mitigation:

Either removing the fee for claiming the airdrop altogether or increasing it to a more practical value. By eliminating the fee, we would ensure that the claiming process is economically feasible for the owner, especially given the low transaction costs on the zkSync chain. Alternatively, adjusting the fee to a higher value could make it more cost-effective to claim the funds, aligning the fee structure with the actual cost of the transaction.