# Phi
Security Review

NOVEMBER, 2024

https://x.com/ChDefendersEth

## Lead Auditors

PeterSR

0x539.eth

## Table of Contents



## Protocol Summary

`ExtraReward` is a smart contract that manages and executes ERC20 token airdrops in units called “seasons”. It uses Merkle trees to efficiently handle large-scale airdrops.

# Disclaimer

The ChainDefenders 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

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

# Audit Details

## Scope

| Id | Files in scope |
|---|---|
| 1 | ExtraReward.sol |

## Roles

| Id | Roles |
|---|---|
| 1 | Owner |
| 2 | User |

# Executive Summary

## Issues found

| Severity | Count | Description |
|---|---|---|
| High | 0 | Critical vulnerabilities |
| Medium | 1 | Significant risks |
| Low | 2 | Minor issues with low impact |
| Informational | 4 | Best practices or suggestions |
| Gas | 0 | Optimization opportunities |

# Findings

## Medium

### Mid 01 DoS Due To Claiming Swept Season

#### Description

Owner can update any season's end time, not only when the season is active. This will mean that the owner can update the end time even of a season which was already swept and make it active. Because sweeping does not update claimed amount it will mean that a user will be able to claim an already swept season with updated end time. This will cause double spending and will DoS other seasons using the same token due to insufficient token balance of the contract.

The vulnerable function:

```
function updateSeasonEndTime(uint256 season, uint256 newEndTime)
    external onlyOwner {
    if (season == 0 || season > currentSeason) {
        revert InvalidSeason();
    }

    if (newEndTime < block.timestamp) {
        revert InvalidEndTime();
    }

    Season storage seasonData = seasons[season];

    seasonData.endTime = newEndTime;
    emit SeasonEndTimeUpdated(season, newEndTime);
```

```
}
```

## Proof of Concept

**Add this file and run the test using forge:**

**ExtraRewardTest.sol:**

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.25;

import "forge-std/Test.sol";
import "./../src/ExtraReward.sol";
import "./mocks/MockERC20.sol";

contract ExtraRewardTest is Test {
    ExtraReward public extraReward;
    MockERC20 public mockToken;

    address public owner = address(0x123);
    address user = 0xB7e390864a90b7b923C9f9310C6F98aafE43F707;
    address public sweepDestination = address(0x789);

    bytes32[] public merkleProof;
    bytes32 merkleRoot = 0
    xf3d2fcbf18b4d03760d47567c4c19dbc9a9ac428c34fcd5665f6c110cd7f405b;

    function setUp() public {
        // Deploy mock token and ExtraReward contract
        mockToken = new MockERC20("MockToken", "MKT");
        extraReward = new ExtraReward(owner);

        // Mint tokens to owner and approve ExtraReward contract
        mockToken.mint(owner, 200);
        vm.startPrank(owner);
        mockToken.approve(address(extraReward), type(uint256).max);
        vm.stopPrank();
    }

    function testMultipleSeasonsAndSweep() public {
        vm.startPrank(owner);

        // Create first season at timestamp 0 with 20-second duration
        vm.warp(0);
        extraReward.createSeason(address(mockToken), merkleRoot, 20,
    100);

```

```
38          // Create second season at timestamp 10 with 20-second
    duration
39          vm.warp(10);
40          extraReward.createSeason(address(mockToken), merkleRoot, 20,
    100);
41
42          vm.stopPrank();
43
44          // Assert initial state of the seasons
45          (
46              address token1,
47              bytes32 root1,
48              uint256 startTime1,
49              uint256 endTime1,
50              uint256 total1,
51              uint256 claimed1,
52              bool swept1
53          ) = extraReward.getSeasonInfo(1);
54
55          assertEq(token1, address(mockToken));
56          assertEq(root1, merkleRoot);
57          assertEq(startTime1, 0);
58          assertEq(endTime1, 20);
59          assertEq(total1, 100);
60          assertEq(claimed1, 0);
61          assertFalse(swept1);
62
63          (
64              address token2,
65              bytes32 root2,
66              uint256 startTime2,
67              uint256 endTime2,
68              uint256 total2,
69              uint256 claimed2,
70              bool swept2
71          ) = extraReward.getSeasonInfo(2);
72
73          assertEq(token2, address(mockToken));
74          assertEq(root2, merkleRoot);
75          assertEq(startTime2, 10);
76          assertEq(endTime2, 30);
77          assertEq(total2, 100);
78          assertEq(claimed2, 0);
79          assertFalse(swept2);
80
81          // Warp to 20 seconds and sweep the first season
82          vm.warp(21);
```

```
        vm.startPrank(owner);
        extraReward.sweep(1, sweepDestination); // 100 out of 200 are send to sweepDestination
        vm.stopPrank();

        // Verify sweep results
        (,,,,,, bool sweptAfter) = extraReward.getSeasonInfo(1);
        assertTrue(sweptAfter);

        // Warp to 25 seconds and update the end time of the first season
        vm.warp(25);
        vm.startPrank(owner);
        extraReward.updateSeasonEndTime(1, 30);
        vm.stopPrank();

        vm.warp(25);

        vm.startPrank(user);
        extraReward.claim(1, 100, merkleProof); // 200 out of 200 are send to sweepDestination, no more tokens
        vm.stopPrank();

        vm.startPrank(user);
        extraReward.claim(2, 100, merkleProof); // revert due to the fact no more tokens in the contract
        vm.stopPrank();
    }
}
```

## Recommendation

Change the following things:

1. Make updating of end time possible only for active seasons:

```
if (block.timestamp < seasonData.startTime || block.timestamp > seasonData.endTime) {
    revert SeasonNotActive();
}
```

2. Change `claimedAmount` of a season to be equal to `totalRewardAmount` after sweeping:

```
function sweep(uint256 season, address destination) external onlyOwner {
    ...
+   seasonData.claimedAmount = seasonData.totalRewardAmount;
    ...
}
```

3. Disallow users to claim if a season was already swept:

```
function claim(uint256 season, uint256 amount, bytes32[] calldata merkleProof) external {
    ...
+   if (seasonData.swept) {
+       revert AlreadySwept();
+   }
    ...
}
```

### Status

**Fixed**

## Low

## Low 01 End Time Can Be Reduced

### Description

The `updateSeasonEndTime` function allows the contract owner to reduce a season's end time, making it less than initially planned. This can result in unexpected behaviour or disrupt users. For instance reducing the end time could invalidate or disadvantage users who planned their actions based on the original timeline. This is particularly problematic in cases where fairness or predictability is a core expectation of the system.

The vulnerable function:

```
function updateSeasonEndTime(uint256 season, uint256 newEndTime) external onlyOwner {
    if (season == 0 || season > currentSeason) {
        revert InvalidSeason();
    }
```

```
5
6       if (newEndTime < block.timestamp) {
7           revert InvalidEndTime();
8       }
9
10      Season storage seasonData = seasons[season];
11
12      seasonData.endTime = newEndTime;
13      emit SeasonEndTimeUpdated(season, newEndTime);
14  }
```

## Proof of Concept

Let's have the following scenario:

1. Assume `season 1` has an initial `endTime` of `1670000000`;
2. `block.timestamp` is equal to `1669000000`;
3. The owner calls `updateSeasonEndTime(1, 1669000000)`, reducing the end time;
4. Users who planned actions for the period between `1669000000` and `1670000000` are now unexpectedly excluded. What is more, these users will not be able to claim from the concrete season.

## Recommendation

Introduce a validation check to prevent the new end time from being earlier than the current end time of the season. This ensures that updates can only extend the season or keep the end time unchanged.

Example mitigation:

```
1  if (newEndTime < seasonData.endTime) {
2      revert InvalidEndTime();
3  }
```

## Status

Fixed

## Low 02 Pausable Tokens

### Description

Degen ERC20 token can be used when creating a season (here) However, the token is pausable. This would mean that the owner of the token can pause all transfers after a season is already created and DoS claiming of the given season.

### Recommendation

Either don't allow Degen ERC20 token or create a mechanism to change tokens of seasons.

### Status

Acknowledged

# Informational

## Info 01 PUSH0 Might Not Be Supported By All Chains

### Description

Current settings may produce incompatible bytecode with some of the chains supported by the protocol.

The `ExtraReward` contract supports and targets different chains (standard EVM chains), such as Ethereum, Base, etc.

The `ExtraReward` contract has the version pragma fixed to be compiled using Solidity 0.8.25. It uses the new `PUSH0` opcode introduced in the Shanghai hard fork. Some chains might not support this opcode.

An example for such a chain is Linea (here)

### Recommendation

Change the Solidity compiler version to 0.8.19 or define an EVM version, which is compatible across all of the intended chains to be supported by the protocol.

### Status

**Acknowledged**

## Info 02 ExtraReward Is Not Compatible With Rebasing And Fee On Transfer Tokens

### Description

`ExtraReward` is not compatible with rebasing and fee on transfer tokens, because `ExtraReward` :: `createSeason` will expect the amount that the contract will receive to be equal to `totalRewardAmount`, but this will not be true for these type of tokens. Smaller amount will be send, leading to wrong calculations during claiming.

### Recommendation

Refactor the code in the following way:

```
- newSeason.totalRewardAmount = totalRewardAmount;
  newSeason.swept = false;

+ uint256 balanceBefore = token.balanceOf(address(this));
  IERC20(token).safeTransferFrom(msg.sender, address(this),
   totalRewardAmount);
+ uint256 balanceAfter = token.balanceOf(address(this));
+ newSeason.totalRewardAmount = balanceAfter - balanceBefore;
```

### Status

**Fixed**

## Info 03 Errors Don't Give Any Relevant Information

### Description

Errors occurring within the `ExtraReward` contract are not providing sufficient data, which hinders effective debugging. Without detailed error information,

it becomes challenging to identify the root cause of failures and resolve issues efficiently.

### Recommendation

You can refactor the errors in the following way:

```
error SeasonNotActive(uint256 seasonId);
error SeasonNotEnded(uint256 seasonId);
error InvalidSeason(uint256 seasonId);
error InvalidEndTime(uint256 endTime);
error InvalidToken(address token);
```

### Status

Acknowledged

## Info 04 Wrong Error Naming

### Description

When sweeping owner can set a destination address to send unclaimed tokens to. There is a check whether the destination address is equal to address(0) and in such a case the transaction is reverted. However, the error is wrong and ambiguous:

```
if (destination == address(0)) {
    revert InvalidToken();
}
```

### Recommendation

Change `InvalidToken`'s name to `InvalidDestinationAddress`.

### Status

Fixed