4/6/2024

Dragon Presale Report

Iwaki Hiroto

Table of Contents

Overview

This document describes the result of auditing for the smart contract of dragon presale.

Project	Dragon Presale
Repository	Dragon Presale
Commit Hash	6259e643aae3b5255065ebf8a76f0da0c6c950b1
Network	Avalanche
Type of Project	ICO
Audit Period	April 4 to April 6

Scope

Smart Contracts	Sloc	Review Status
DragonToken.sol	924	✔
dragonPresale.sol	456	✔

Risk Classification

- High: Assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).
- Medium: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.
- Low: Assets are not at risk: state handling, function incorrect as to spec, issues with comments.
- QA: Code style, clarity, syntax, versioning, off-chain monitoring events, etc.
- Gas: Re-writing Solidity code to accomplish the same business logic while consuming fewer gas.

Summary of Findings

Severity	Count
High	0
Medium	0
Low	0
QA	1
Gas	2
Total	3

Finding Details

Gas Optimization

[G-01] Cache the length of the presaleBuyers array in a variable to avoid recalculating the length on each iteration.

Severity

Gas

Affected Lines

dragonPresale.sol#L233-L238

Descriptions

Reading the array length repeatedly is an expensive operation in terms of gas due to the interaction with the underlying storage.

Every time you access the array length, Solidity has to retrieve it from storage, which includes multiple steps – loading the slot, decoding the value, and converting it to the requested type.

Retrieving the length value from storage takes considerably more gas than reading from memory. Memory reads benefit from spatial locality, whereas storage reads suffer from poor spatial locality.

```
 function _airdrop() private {

    uint256 limitCount_ = airdropIndex + 100; //Max amount of addresses to
airdrop to per call is 100 addresses

    address buyer_;

    uint256 amount_;


    while (

@>        airdropIndex < presaleBuyers.length && airdropIndex < limitCount_

    ) {

        buyer_ = presaleBuyers[airdropIndex];
```

```
        amount_ =
            (totalSent[buyer_] * Presalers_Dragon_Supply_Wei) /
            totalAvaxPresale; //Calculate amount of Dragon tokens to send to
buyer as ratio of AVAX sent
        require(
            dragonInterface.transfer(buyer_, amount_),
            "Transfer failed"
        );
        airdropIndex++;
    }

    if (airdropIndex == presaleBuyers.length) {
        airdropCompleted = true;
    }

    emit AirdropSent(msg.sender, airdropIndex);
}
```

Recommendation

To optimize the gas usage for the presaleBuyers array, you can cache the array length in a variable, like so:

```
uint256 buyersLength = presaleBuyers.length;
for (uint256 i = airdropIndex; i < buyersLength && i < limitCount_;) {
    buyer_ = presaleBuyers[i];
    amount_ = (totalSent[buyer_] * Presalers_Dragon_Supply_Wei) /
totalAvaxPresale;
    require(dragonInterface.transfer(buyer_, amount_), "Transfer failed");
    unchecked {
        ++i;
    }
```

```
}
```

Now, the loop uses the buyersLength variable instead of presaleBuyers.length. This change will save gas by avoiding the repeated storage read operations, ultimately reducing the overall gas cost of the function.

Moreover, note that limitCount_ should also be replaced with a variable initialized to airdropIndex + 100. This replacement prevents recalculating the expression on every iteration, offering a minor gas cost reduction.

```
uint256 limitCount_ = airdropIndex + 100;
for (uint256 i = airdropIndex; i < buyersLength && i < limitCount_; i++) {
    // ...
}    }
```

These modifications together will help reduce gas costs and improve the efficiency of the _airdrop function.

[G-02] Set the deadline and slippage in swapAvaxForCT.

Severity

Gas

Affected Lines

dragonPresale.sol#L248-264

Descriptions

The swapExactAVAXForTokensSupportingFeeOnTransferTokens function accepts a deadline argument, which defaults to the current timestamp if not provided. In the present implementation, the current timestamp is passed as the deadline. Consider passing a more distant deadline, e.g., block.timestamp + 1 hours, to allow more flexibility for completing the transaction.

Also, setting the slippage tolerance to a finite value instead of 100 (effectively infinite) can help account for potential losses incurred during the swapping process. You can determine the optimal slippage tolerance based on your risk appetite and desired safety margin.

```
    function swapAvaxForCT(uint256 avaxAmount_, uint256 index_) private {
```

```
        address[] memory path = new address[](2); //Our swap path WAVAX/CT
        path[0] = WAVAX; //WAVAX
        path[1] = Community_Tokens[index_]; //CT

        try
            IUniswapV2Router02(CT_Routers[index_])
                .swapExactAVAXForTokensSupportingFeeOnTransferTokens{ //AVAX to
CT, swap on dex router with most LP
                value: avaxAmount_
            }(
@>              100, //Infinite slippage basically since it's in Wei
                path,
                address(this), //Send CT tokens received to this contract
@>              block.timestamp
            )
        {} catch {
            revert(string("swapAvaxForCT failed"));
        }
    }
```

Recommendation

Consider passing a more distant deadline, e.g., block.timestamp + 1 hours, to allow more flexibility for completing the transaction.

Ideally, the deadlines should be checked to ensure they aren't too far apart, which may impact the success probability of the transaction.

Setting the slippage tolerance to a finite value instead of 100 (effectively infinite) can help account for potential losses incurred during the swapping process. You can determine the optimal slippage tolerance based on your risk appetite and desired safety margin.

```
    function swapAvaxForCT(uint256 avaxAmount_, uint256 index_) private {
        address[] memory path = new address[](2);
        path[0] = WAVAX;
        path[1] = Community_Tokens[index_];

        uint256 deadline = block.timestamp + 1 hours;
        try
            IUniswapV2Router02(CT_Routers[index_]).swapExactAVAXForTokensSupporti
ngFeeOnTransferTokens{
                value: avaxAmount_
            }(
                100, // Finite slippage tolerance, e.g., 10%
                path,
                address(this),
                deadline
```

```
            )
        {} catch {
            revert(string("swapAvaxForCT failed"));
        }
    }
```

QA

[Q-01] The final conditional check.

Severity

QA

Affected Lines

dragonPresale.sol#L240

Descriptions

Currently, the code sets airdropCompleted to true when airdropIndex equals the length of the presaleBuyers array. Due to floating-point math imprecision, comparing integers for strict equality can sometimes produce incorrect results. Therefore, it is safer to compare if airdropIndex is greater than or equal to the length of the presaleBuyers array.

```
-   if (airdropIndex == presaleBuyers.length) {
+   if (airdropIndex >= presaleBuyers.length) {
        airdropCompleted = true;
    }
```

Conclusion

The audit was performed focusing on finding the possibilities for loss of funds or other extra bugs, and optimizing the code. As a result of auditing, the contract is healthy for overall codes.