Dragon

Smart Contract Audit

Audited By: Iwaki Hiroto

April 2, 2024

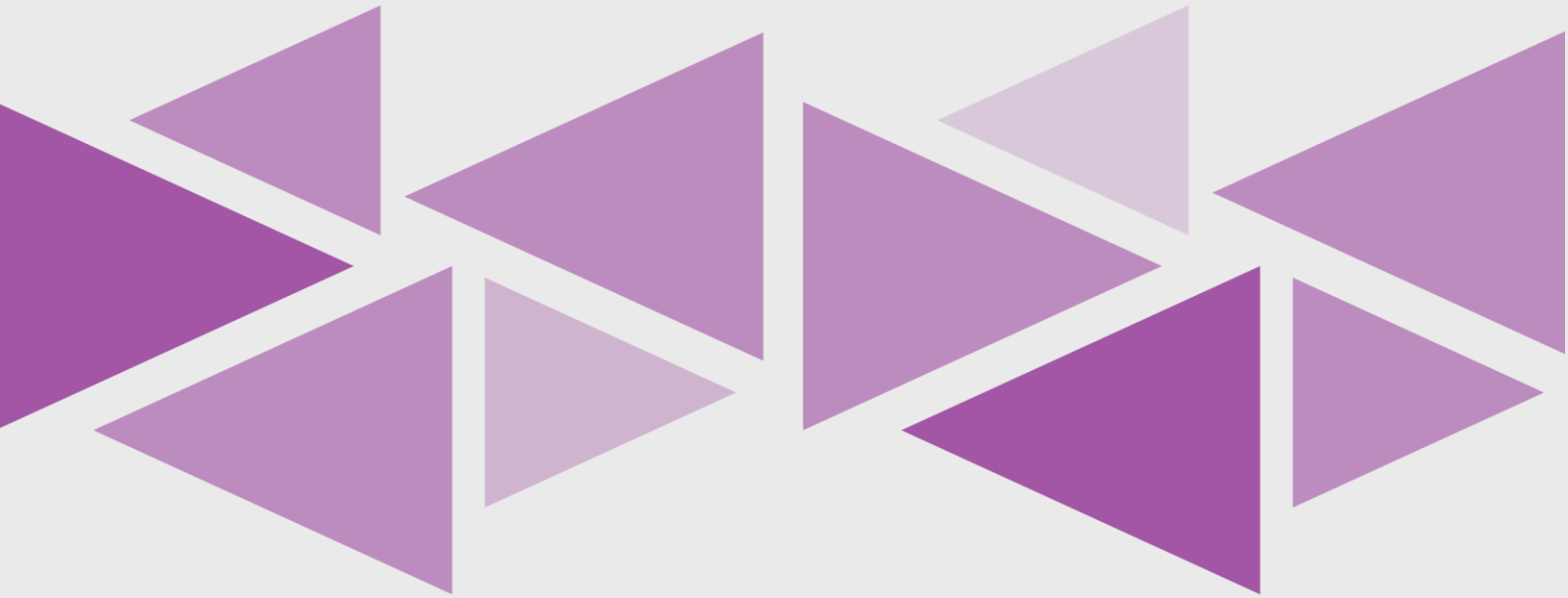

Table of Contents

Overview

This document describes the audit results for smart contract of dragon token. This smart contract is worked on Avalanche.

Since the past audits had been performed against centralization of privileges enough, this audit was performed focusing on finding the possibilities for loss of funds or other extra bugs, and optimizing the codebase.

Project	Dragon Token
Repository	DragonToken
Commit Hash	20b7278e9f464f832796eb292f175cc45cc40fb7
Network	Avalanche
Type of Project	ERC20
Audit Period	March 30 to April 2

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.

Info 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

High	0
Medium	0
Low	1
Info	1
Gas	4
Total	6

Finding Details

Low

[L-01] Use call instead of transfer on payable addresses.

Severity

Low

Affected Lines

DragonToken.sol#L868-L872

Finding Description

In Solidity, when transferring native token, `.transfer()` and `.send()` are commonly used. However, they have a limitation of gas, which isn't enough to execute any code in the recipient contract beyond a simple event emission. Thus, if the recipient is a contract, the transfer may fail unexpectedly.

```
    function withdrawAvaxTo(address payable to_, uint256 amount_) external
onlyOwner {
        require(to_ != address(0), "Cannot withdraw to 0 address");

@>      to_.transfer(amount_); // Can only send Avax from our contract, any
user's wallet is safe
        emit AvaxWithdraw(to_, amount_);
    }
```

To overcome this, Solidity introduced the `.call{value: _amount}("")` method, which forwards all available gas and can invoke more complex functionality. It's also safer in that it does not revert on failure but instead returns a `boolean` value to indicate success or failure. Therefore, it is generally a better choice to use `.call` when transferring native token to a payable address, with the necessary safety checks implemented to handle potential errors.

Info

[I-01] Consider using correct comparing for validation.

Severity

Informative

Affected Lines

DragonToken.sol#L756-L757

Finding Description

```
    function seedAndBurnCtLP(uint256 ctIndex_, address communityToken_,
uint256 amountDragon_, uint256 amountCt_) external {
        require(tradingPhase() != TOTAL_PHASES, "Phases are completed
already"); // This function is just to seed LP for IDO launch
        require(communityTokens[ctIndex_] == communityToken_, "Token not found
in communityTokens array at that index"); // Verify valid CT address
@>      require(amountDragon_ > 100000000000000000, "Must send at least 0.1
Dragon tokens");
@>      require(amountCt_ > 100000000000000000, "Must send at least 0.1
Community tokens");
        require(!swapping, "Already making CT LP, you have created a
reentrancy issue");
```

In above codebase, to be correct validation for `amountDragon_` and `amountCt_`, should use `>=`, not `>`.

Gas Optimization

[G-01] Multiple accesses of the same mapping/array key/index should be cached

Affected Lines

DragonToken.sol#L595-L599

Descriptions

Caching repeated accesses to the same mapping or array key/index in smart contracts can lead to significant gas savings. In Solidity, each read operation from storage (like accessing a value in a mapping or array using a key or index) costs gas. By storing the accessed value in a local variable and reusing it within the function, you avoid multiple expensive storage read operations.

```
    for (uint256 i = 0; i < ctLength_; i++){ // Loop for each CT
@>      uint256 initialCtBalance_ =
IERC20(communityTokens[i]).balanceOf(address(this
        swapAvaxForCT(avaxPerCt_, i);
@>      uint256 newCtBalance_ =
IERC20(communityTokens[i]).balanceOf(address(this)) - initialCtBalance_;
@>      addLiquidityCT(dragonPerCtLP_, newCtBalance_, communityTokens[i]);
@>      emit SwapAndLiquify(dragonPerCtLP_, newCtBalance_,
communityTokens[i]);
    }
```

In the above codebase, `communityTokens[i]` is called several times within same loop. To save gas effectively, it would be better to use cached variable before accessing the state repeatedly.

This practice is particularly beneficial in loops or functions with multiple reads of the same data. Implementing this caching approach enhances efficiency and reduces transaction costs, which is crucial for optimizing smart contract performance and user experience on the blockchain.

[G-02] `x + y` is more efficient than using `+=` for state variables (likewise for `-=`)

Affected Lines

DragonToken.sol#L562

Descriptions

In instances found where either `+=` or `-=` are used against state variables use `x = x + y` instead.

```
        require(allowlisted[to_] <= tradingPhase_, "Not allowlisted for
current phase
@>      totalPurchased[to_] += amount_; // Total amount user received in whale
limited
        require(totalPurchased[to_] <= maxWeiPerPhase[tradingPhase_],
"Receiving too much for current whale limited phase");
```

[G-03] Integer increments by one can be unchecked to save on gas fees

Affected Lines

DragonToken.sol#L336-L341

DragonToken.sol#L402-L408

DragonToken.sol#L490-L495

DragonToken.sol#L594-L600

DragonToken.sol#L683-L686

DragonToken.sol#L830-L834

DragonToken.sol#L853-L855

Descriptions

Using unchecked increments in Solidity can save on gas fees by bypassing built-in overflow checks, thus optimizing gas usage, but requires careful assessment of potential risks and edge cases to avoid unintended consequences.

```
@>      for (uint256 i = 0; i < length_; i++){
            uniswapV2Router_ = IUniswapV2Router02(ctRouters[i]);
            uniswapV2Routers.push(uniswapV2Router_);
            uniswapV2Pair_ =
IUniswapV2Factory(uniswapV2Routers[i].factory()).getPair(communityTokens[i],
WAVAX);
            require(uniswapV2Pair_ != address(0), "All CT/WAVAX LP Pairs must
be created first and have some LP already, to buy CT with AVAX");
        }
```

[G-04] Consider moving the ++/-- action to the left of the variable, especially in `for` loop definitions.

Affected Lines

DragonToken.sol#L336-L341

DragonToken.sol#L402-L408

DragonToken.sol#L490-L495

DragonToken.sol#L594-L600

DragonToken.sol#L683-L686

DragonToken.sol#L830-L834

DragonToken.sol#L853-L855

Descriptions

Move the `++`/`--` action to the left of the variable

```
        for (uint256 i = 0; i < length_; i++){
            uniswapV2Pair_ =
IUniswapV2Factory(uniswapV2Router.factory()).getPair(address(this),
communityTokens[i]);
            if (!dragonCtPairs[uniswapV2Pair_] && uniswapV2Pair_ !=
address(0)) {
                dragonCtPairs[uniswapV2Pair_] = true;
            }
        }
```

For example, to save gas, above codebase can be modified as follow.

```
-   for (uint256 i = 0; i < length_; i++){
+   for (uint256 i = 0; i < length_;){
        uniswapV2Pair_ =
IUniswapV2Factory(uniswapV2Router.factory()).getPair(address(this),
communityTokens[i]);
        if (!dragonCtPairs[uniswapV2Pair_] && uniswapV2Pair_ != address(0)) {
            dragonCtPairs[uniswapV2Pair_] = true;
        }
+       unchecked {
+           ++i;
+       }
    }
```

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 and well modified based on the past audit.