Audit of The DuckDao Contracts

a report of findings by

Van Cam Pham, PhD

innovative fortuna iuvat

December 15th, 2020

Table of Contents

Document Info

Client	DuckDao
Title	Smart Contract Audit of DuckDao Contracts
Auditors	Van Cam Pham, PhD
Edited By	Joel Farris
Approved By	Rasikh Morani

Contact

For more information on this report, contact The Arcadia Media Group Inc.

Rasikh Morani
(972) 543-3886
rasikh@arcadiamgroup.com
https://t.me/thearcadiagroup

Executive Summary

A Representative Party of DuckDAO ("DuckDAO") engaged The Arcadia Group ("Arcadia"), a software development, research, and security company, to conduct a review of the following DuckDAO smart contracts on the DuckDAO repo at Commit #7b5cba2e3145580a352ccacce040db5428bda5b8.

DuckToken.sol
Pool.sol
PoolController.sol
TokenWrapper.sol

After a first review and report, and discussion of findings with the DuckDAO team, they fixed all reported issues. Arcadia then performed a second review of the code at commit #05414a401452b7c684946f16e229f8c6a613de89 specifically regarding only those remediated issues.

Arcadia completed the reviews using various methods primarily consisting of dynamic and static analysis. This process included a line by line analysis of the in-scope contracts, optimization analysis, analysis of key functionalities and limiters, and reference against intended functionality.

There were 09 issues found, 01 of which were deemed to be ‘critical’, and 03 of which were rated as ‘high’.

Severity Rating	Number Of Original Occurrences	Number Of Remaining Occurrences
Critical	01	00
High	03	00
Medium	02	00
Low	01	00
Notice	01	01
Informational	01	01

Findings

1. Duck token cannot be traded on Uniswap

- DLC-1
- Severity: Critical
- Likelihood: High
- Impact: High
- Target: DucToken.sol, PoolControlleer.sol
- Category: Token Untradeable
- Finding Type: Dynamic

The `DuckToken` contract is implemented in order to create the following farming feature:

- A user farms `DLC` in the farming pools by depositing liquidity tokens (`LP` token) generated by providing liquidity to uniswap pairs.
- When the user decides to exit the farming pool and remove liquidity from uniswap, the `DLC` token amount should be burned and the user should only receive the other token in the pair. This is what's called "one-sided burn farming".

The implementation is done in the `DLC DuckToken` contract as follows:

- The `DuckToken` contract keeps a list of liquidity pools (Uniswap pairs) addresses. When there is transfer from one of the pairs, the transferred token will be burned.
- The expected behavior is that when there is liquidity removal from Uniswap, DLC token will be sent from the Uniswap pairs, thus burning the transferred token.

This is, however, buggy and causes tokens to be untradeable on Uniswap as follows:

- There are two cases in which DLC token is transferred from the liquidity Uniswap pairs:
    - Liquidity Removal of DLC/X pair where X is the other token in the pair. In this case, DLC is transferred from the corresponding pair to user address.
    - DLC market-buy in Uniswap: X will be transferred from the buyer's wallet to the uniswap pair while DLC is transferred from back from the pair to the buyer's wallet. In the current implementation, DLC will be burned immediately before transferring to the buyer's wallet. This is not the expected behavior as it causes DLC to be untradable because the buyer will receive no DLC.
- If a user only adds liquidity to the Uniswap pair DLC/X without farming into the farming pools, the user should be able to withdraw liquidity from Uniswap without having their DLC tokens burned. However, the current implementation always burns DLC tokens whenever there is liquidity removal, regardless of whether the user has farmed or not.

Action Recommended: As one-side burning is only applied to users who farm in the pool, the burn mechanism should be intuitively implemented at the pool level in order to differentiate farmers from liquidity providers only.

The farming pool should have a withdraw function which works as follows:

- Burn the expected withdrawn liquidity token amount in order to remove liquidity from Uniswap to the farming pool.
- Burn the DLC token removed from Uniswap.
- Send the other withdrawn token to the user.

Due to this issue, the logic for function `newPool` of the `PoolController` contract becomes invalid, as liquidity pairs should not be added to the `DLC` token contract.

Review of the remediation performed at commit #05414a401452b7c684946f16e229f8c6a613de89:

- Liquidity pairs have been removed from the DuckToken contract
- The Pool contract now has a function that burns either DLC or DDIM token when a user decides to withdraw her/his liquidity token from the farming pool.
- The issue has been resolved by the development team**.**

2. The owner of the PoolController contract should be transferred to governance

- DLC-2
- Severity: Medium
- Impact: Medium
- Target: PoolController.sol
- Category: Ownership
- Finding Type: Dynamic

The ownership of the `PoolController` contract should be transferred to a governance-powered contract in order to avoid centralization when adding a new pool.

```
    function newPool(address lpToken, uint startingBlock, uint[] memory blocks, uint[] memory
farmingSupplies) public onlyOwner {
        Pool pool = new Pool(lpToken, startingBlock, blocks, farmingSupplies);
        pools.push(pool);

        canMint[address(pool)] = true;
        duck.addLiquidityPool(lpToken);

        emit NewPool(address(pool), lpToken);
    }
```

Action Recommended: Transfer the ownership of the `PoolController` contract to a governance-powered contract in order to stay decentralized.

Review of the remediation performed at commit #05414a401452b7c684946f16e229f8c6a613de89:

- As discussed with the DuckDAO team: the team will operate as a company and the team believes having control over the governance is good for the company.
- We recommend that whenever the project becomes fully decentralized, the ownership should be transferred to a community-driven governance contract.

3. Function `addPeriod` should check for `startingBlock` value

- DLC-3
- Severity: Low
- Likelihood: Low
- Impact: Low

- Target: Pool.sol
- Category: Informational
- Finding Type: Static
- Lines 123-140

In the contract `Pool`, the function `addPeriod` should check that the input parameter `startingBlock` should be greater than the current block so that rewards will only pay in the future.

```
function addPeriod(uint startingBlock, uint blocks, uint farmingSupply) public onlyController {
   if(periods.length > 0) {
     require(startingBlock >
periods[periods.length-1].startingBlock.add(periods[periods.length-1].blocks), "two periods in the same
time");
   }
```

Action Recommended: Add a check for `startingBlock` compared to the current block number:

```
function addPeriod(uint startingBlock, uint blocks, uint farmingSupply) public onlyController {
   require(startingBlock >= block.number);
   if(periods.length > 0) {
     require(startingBlock >
periods[periods.length-1].startingBlock.add(periods[periods.length-1].blocks), "two periods in the same
time");
   }
```

Review of the remediation performed at commit #05414a401452b7c684946f16e229f8c6a613de89:

- The issue was resolved by the development team.

4. Gas consumption can become too high

- DLC-4
- Severity: Informational
- Likelihood: Medium
- Impact: Medium

- Target: Pool.sol
- Category: Gas
- Finding Type: Dynamic
- Lines: 227-230, 294-300

In the `Pool` contract, there are several for-loops that loop over the list of `periods` and `revenues`. Those for-loops always start from the beginning till the end of the list. This can cost high gas if there are many `periods` and `revenues` in the pool.

```
for(uint i = 0; i < periods.length; i++) {
    if(block.number < periods[i].startingBlock) {
      break;
    }
    ...
}

for(uint i = 0; i < revenues.length; i++) {
    if(!revenuesClaimed[userAddress][i]) {
      revenuesClaimed[userAddress][i] = true;
      uint userRevenue = revenues[i].amount.mul(user.amount).div(revenues[i].totalSupply);

      safeRevenueTransfer(revenues[i].tokenAddress, userAddress, userRevenue);
    }
  }
```

Action Recommended: Use storage variables to:

- Save the last checked `period` so that the for-loop only needs to start from the last checked period
- Save the last checked `revenue` per user so that the for-loop only needs to start from the last checked `revenue`.

By updating the code to use last saved variables, functions `getCurrentPeriodIndex` and `getUserLastRevenue` can also be optimized (even those are just view functions).

Review of the remediation performed at commit #05414a401452b7c684946f16e229f8c6a613de89:

- Limitation for the number of `revenues` has been put into the contract. There can be a maximum of 50 `revenues`. This can be done with the current block gas limit of ethereum. However, for user gas consumption optimization for transactions, we still recommend having a storage field to track the revenue last paid index for each user.
- The same recommendation is applied to storage field `periods` in the contract if there is a high max limitation of the number of periods.

5. Function `calculateDuckTokensForMint` causes `updatePool` to mint more `DLC` than expected

- DLC-5
- Severity: High
- Likelihood: High
- Impact: High

- Target: Pool.sol
- Category: DLC Mint Computation
- Finding Type: Dynamic
- Lines: 120-142, 223-248

5.1 In the `Pool` contract, the function `calculateDuckTokensForMint` calculates the amount of DLC needed to mint at the current block. The function has a bug that results in more DLC than expected will be minted when `updatePool` is called. The bug is caused by not updating `lastRewardBlock` after each iteration in the for-loop of the function `calculateDuckTokensForMint`. n example of this would be:

- There are 3 periods in the pools with (startingBlock, blocks) as: (10, 5), (18, 6), (25, 4) with 1 DLC minted per block.
- `updatePool` is called at block 11, thus `calculateDuckTokensForMint` is called and returns 1 as line 241 is executed (see an extended issue below). At this point, `lastRewardBlock` updated to 11.
- Next, `updatePool` is called at block 26. Function `calculateDuckTokensForMint` executes three iterations:
    - First iteration: Line 235 executed, which results in totalTokens = 4.
    - Second iteration: Line 235 executed, but totalTokens = 4 + (24 - 11) = 17.
    - Third iteration: Line 239 executed, totalTokens = 17 + (26 - 25) = 18.
- A total of 32 DLC would be minted in this simple example while the total for all three periods at block 26 should be 5 + 6 + 1 = 12 DLC.

The reasons are:

- `lastRewardBlock` is not updated after each iteration.
- Line 235 should check if `lastRewardBlock < periods[i].startingBlock`

5.2. In function `calculateDuckTokensForMint` , the local variable `overflown` should be set to false initially to avoid solidity compilation that could result in true as default value for `overflown`.

5.3 An issue related to line 241, as in the example above, at block 11, the number of blocks to pay rewards should be 2 (blocks 10 and 11), but only 1 DLC is minted at block 11. Is this the code issue or an intended design?

```
function calculateDuckTokensForMint() public view returns(uint) {
    uint totalTokens;
    bool overflown;
```

```
    for(uint i = 0; i < periods.length; i++) {
      if(block.number < periods[i].startingBlock) {
        break;
      }

      uint buf = periods[i].startingBlock.add(periods[i].blocks);

      if(block.number > buf && buf > lastRewardBlock) {
        totalTokens += buf.sub(lastRewardBlock).mul(periods[i].tokensPerBlock);
        overflown = true;
      } else {
        if(overflown) {
          totalTokens += block.number.sub(periods[i].startingBlock).mul(periods[i].tokensPerBlock);
        } else {
          totalTokens += block.number.sub(lastRewardBlock).mul(periods[i].tokensPerBlock);
        }

        break;
      }
    }

    return totalTokens;
  }
```

Action Recommended:

- Review function `calculateDuckTokensForMint`
- Use a local variable in the function to store lastRewardBlock, and update the local variable after each iteration
- Line 235: check if `_lastRewardBlock < periods[i].startingBlock`, the code at line 235 should become:

```
function calculateDuckTokensForMint() public view returns(uint) {
    uint totalTokens;
    bool overflown;

    for(uint i = 0; i < periods.length; i++) {
      if(block.number < periods[i].startingBlock) {
        break;
      }

      uint buf = periods[i].startingBlock.add(periods[i].blocks);

      if(block.number > buf && buf > _lastRewardBlock) {
        if (lastRewardBlock < periods[i].startingBlock) {
          totalTokens += buf.sub(periods[i].startingBlock).mul(periods[i].tokensPerBlock);
```

```
      } else {
        totalTokens += buf.sub(lastRewardBlock).mul(periods[i].tokensPerBlock);
      }
      overflown = true;
    } else {
      if(overflown) {
        totalTokens += block.number.sub(periods[i].startingBlock).mul(periods[i].tokensPerBlock);
      } else {
        totalTokens += block.number.sub(lastRewardBlock).mul(periods[i].tokensPerBlock);
      }

      break;
    }
  }

  return totalTokens;
}
```

Review of the remediation performed at commit #05414a401452b7c684946f16e229f8c6a613de89: The issue has been resolved by the development team.

6. Function `addRevenue` has to transfer tokens from the revenue source to the contract

- DLC-6
- Severity: High
- Likelihood: High
- Impact: High

- Target: Pool.sol
- Category: Revenue Transfer
- Finding Type: Dynamic
- Lines: 266-275

In the Pool contract, function `addRevenue` must call `transferFrom` to transfer the revenue tokens to the pool contract address from the revenue source. Without transferring enough revenue tokens to the pool, the following can happen:

- The pool does not have enough tokens if the pool does not receive enough revenue tokens from the revenue source.
- This causes any function that calls `claimRevenue` function to fail. Those functions are: `withdraw` and `deposit`, thus users cannot deposit or withdraw from the pool without waiting for the revenue to come to the pool.

```
function addRevenue(address _tokenAddress, uint _amount) public onlyController {

   Revenue memory revenue = Revenue({
     tokenAddress: _tokenAddress,
     totalSupply: lpToken.balanceOf(address(this)),
     amount: _amount
   });

   revenues.push(revenue);
 }
```

Action Recommended: As a common practice, when revenue is added to the pool, the pool should call transferFrom function in order to receive revenue tokens in the pool. The following is a suggested fix:

```
function addRevenue(address _tokenAddress, uint _amount, address _revenueSource) public onlyController
{
   uint revenueBefore = IERC20(_tokenAddress).balanceOf(address(this));
   IERC20(_tokenAddress).transferFrom(_revenueSource, address(this), _amount);
   uint revenueAfter = IERC20(_tokenAddress).balanceOf(address(this));
   _amount = revenueAfter.sub(revenueBefore);
   Revenue memory revenue = Revenue({
     tokenAddress: _tokenAddress,
     totalSupply: lpToken.balanceOf(address(this)),
     amount: _amount
```

```
  });

  revenues.push(revenue);
}
```

Review of the issue at commit #05414a401452b7c684946f16e229f8c6a613de89: The issue has been resolved by the development team.

7. Unit Tests are insufficient

- DLC-7
- Severity: High
- Likelihood: High
- Impact: High
- Target: Pool.sol
- Category: Unit tests
- Finding Type: Dynamic

The code lacks unit tests for deposits and withdrawals. Unit tests are important, especially for functions that do math in order to ensure the contracts function correctly and follow the design.

Review of the remediation performed at commit #05414a401452b7c684946f16e229f8c6a613de89: Some unit tests for deposits have been added, but unit tests for withdrawals are missing. We recommend having unit tests on those two important functions before deploying to Mainnet.

8. The owner of the `DuckToken` contract should be transferred to `PoolController`

- DLC-8
- Severity: Medium
- Impact: Medium
- Target: DuckToken.sol
- Category: Ownership
- Finding Type: Dynamic

The ownership of the `DuckToken` contract should be transferred to the `PoolController` contract immediately after the two contracts are deployed in order to ensure that after the Team and Presale allocation, DLC can only be minted through farming.

This is more of a recommendation for a proper deployment rather than an issue.

9. Function `claimRevenue` can optimize gas

- DLC-8
- Severity: Notice
- Impact: Medium
- Target: DuckToken.sol
- Category: Gas
- Finding Type: Dynamic

Function `claimRevenue` can be optimized by:

- Compute the total revenue that can be transferred to user by using the for-loop
- Use a single `safeRevenueTransfer` function call to transfer the total revenue for the user. This will also reduce the number of events emitted by the `claimRevenue` function to 1. This is because if there are 50 revenues, 50 transfer events will be emitted, which is hard to trace later.

Conclusion

Arcadia identified issues that occurred at hash #7b5cba2e3145580a352ccacce040db5428bda5b8 that were confirmed to be patched as of #b53876b56b01fcbf9cea2fe289cf47cafb5385d5. Due to the size of the subsequent modifications, no analysis was done of any potentially introduced issues after the originally identified and addressed issues were remediated.

Disclaimer

While best efforts and precautions have been taken in the preparation of this document, The Arcadia Group and the Authors assume no responsibility for errors, omissions, or damages resulting from the use of the provided information. Additionally, Arcadia would like to emphasize that the use of Arcadia's services does not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One audit on its own is not enough for a project to be considered secure; that categorization can only be earned through extensive peer review and battle testing over an extended period.