Smart Contract Security Audit

(LooksRareToken Staking)

Deliverable: Smart Contract Audit Report

Security Report

April 2022

Disclaimer

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

Report Summary

Title	Smart Contract Security Audit		
Project Owner	Blockchain Support		
Type	Private		
Reviewed by	Blockchain Guru	Revision date	06/04/2022
		Approval date	06/04/2022
		Nº Pages	19

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant effect on the security of the project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

The Full List of Check Items:

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	MONEY-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks

	Business Logics Review
	Functionality Checks
Advanced Scrutiny	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

Common Weakness Enumeration (CWE) Classifications Used in This Audit:

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.

Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an ex pilotable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

Background

The purpose of the audit was to achieve the following:

● Ensure that the smart contract functions as intended.

● Identify potential security issues with the smart contract.

The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified

Nice Features

The contract provides a good suite of functionality that will be useful for the entire contract

1. Modifiers to control
2. Beautiful coding styles
3. Correct function visibility
4. Readable code
5. Enough comments

Findings

Summary

Here is a summary of the findings after analyzing the LooksRareToken Staking Contract Review. During the first phase of the audit, I studied the smart contract source code and ran my in-house static code analyzer through the Specific tool and also manually. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by tool. I further manually review business logics, examine system operations, and place aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	No. of Issues
Critical	0
High	1
Medium	0
Low	1
Weak	2
Total	4

We have so far identified that there are potential issues with severity of 0 Critical, 1 High, 0 Medium, 1 Low and 1 Weak. Overall, these smart contracts are well-designed and engineered.

Functional Overview

($) = payable function	[Pub] public
# = non-constant function	[Ext] external
	[Prv] private
	[Int] internal

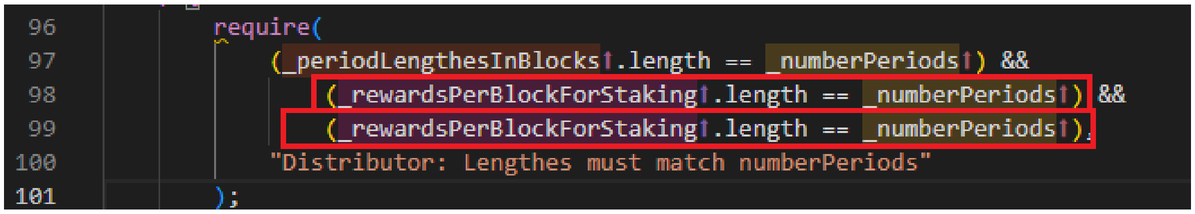
```
+ [Int] ILooksRareToken(IERC20)
  - [Ext] SUPPLY_CAP
  - [Ext] mint#
+ [Int] IUniswapV3SwapCallback
  - [Ext] uniswapV3SwapCallback #
+ [Int] ISwapRouter(IUniswapV3SwapCallback)
  - [Ext] exactInputSingle ($)
  - [Ext] exactInput ($)
  - [Ext] exactOutputSingle ($)
  - [Ext] exactOutput ($)
+ LooksRareToken(ERC20, Ownable, ILooksRareToken  )
  - [ ] <Constructor> #
  - [Ext] mint #
   - modifiers: onlyOwner
  - [Ext] SUPPLY_CAP
+   TokenSplitter(Ownable, ReentrancyGuard)
  - [ ] <Constructor> #
  - [Pub] TOTAL_SHARES
  - [Pub] looksRareToken
  - [Pub] TOTAL_SHARES
  - [Pub] totalTokensDistributed
  - [Pub] accountInfo
  - [Ext] releaseTokens #
   - modifiers: nonReentrant
  - [Ext] updateSharesOwner #
   - modifiers: onlyOwner
  - [Ext] calculatePendingRewards
+ TokenDistributor (ReentrancyGuard)
  - [ ] <Constructor> #
  - [Pub] PRECISION_FACTOR
  - [Pub] looksRareToken
  - [Pub] TOTAL_SHARES
  - [Pub] NUMBER_PERIODS
  - [Pub] START_BLOCK
  - [Pub] accTokenPerShare
  - [Pub] currentPhase
  - [Pub] endBlock
  - [Pub] lastRewardBlock
  - [Pub] rewardPerBlockForOthers
  - [Pub] rewardPerBlockForStaking
  - [Pub] stakingPeriod
  - [Pub] userInfo
  - [Ext] deposit #
    - modifiers: nonReentrant
  - [Ext] harvestAndCompound #
    - modifiers: nonReentrant
  - [Ext] updatePool #
    - modifiers: nonReentrant
  - [Ext] withdraw #
    - modifiers: nonReentrant
   - [Ext] withdrawAll #
    - modifiers: nonReentrant
    - [Ext] calculatePendingRewards
    - [ Int] _updatePool #
    - [ Int] _updateRewardsPerBlock #
    - [ Int] _getMultiplier
+ FeeSharingSystem(ReentrancyGuard, Ownable)
   - [  ] <Constructor> #
   - [Pub] PRECISION_FACTOR
   - [Pub] looksRareToken
   - [Pub] rewardToken
   - [Pub] tokenDistributor
   - [Pub] currentRewardPerBlock
   - [Pub] lastRewardAdjustment
   - [Pub] lastUpdateBlock
   - [Pub] periodEndBlock
   - [Pub] rewardPerTokenStored
   - [Pub] totalShares
   - [Pub] userInfo
   - [Ext] deposit #
    - modifiers: nonReentrant
   - [Ext] harvest #
    - modifiers: nonReentrant
   - [Ext] withdraw #
    - modifiers: nonReentrant
   - [Ext] withdrawAll #
    - modifiers: nonReentrant
   - [Ext] updateRewards #
    - modifiers: onlyOwner
   - [Ext] calculatePendingRewards
   - [Ext] calculateSharesValueInLOOKS
   - [Ext] calculateSharePriceInLOOKS
   - [Ext] lastRewardBlock
   - [ Int] _calculatePendingRewards
   - [Int] _checkAndAdjustLOOKSTokenAllowanceIfRequired #
   - [Int] _lastRewardBlock
   - [Int] _rewardPerToken
   - [Int] _updateReward #
   - [Int] _withdraw #
```

```
+ AggregatorFeeSharingWithUniswapV3 (Ownable, Pausable, ReentrancyGuard)
  - [ ] <Constructor> #
  - [Pub] MAXIMUM_HARVEST_BUFFER_BLOCKS
  - [Pub] feeSharingSystem
  - [Pub] uniswapRouter
  - [Pub] MINIMUM_DEPOSIT_LOOKS
  - [Pub] looksRareToken
  - [Pub] rewardToken
  - [Pub] canHarvest
  - [Pub] harvestBufferBlocks
  - [Pub] lastHarvestBlock
  - [Pub] maxPriceLOOKSInWETH
  - [Pub] thresholdAmount
  - [Pub] totalShares
  - [Pub] userInfo
  - [Ext] deposit #
   - modifiers: nonReentrant, whenNotPaused
  - [Ext] withdraw #
   - modifiers: nonReentrant
  - [Ext] withdrawal #
   - modifiers: nonReentrant
  - [Ext] harvestAndSellAndCompound #
   - modifiers: nonReentrant, onlyOwner
  - [Ext] checkAndAdjustLOOKSTokenAllowanceIfRequired #
   - modifiers: onlyOwner
  - [Ext] checkAndAdjustRewardTokenAllowanceIfRequired #
   - modifiers: onlyOwner
  - [Ext] updateHarvestBufferBlocks #
   - modifiers: onlyOwner
  - [Ext] startHarvest #
   - modifiers: onlyOwner
  - [Ext] stopHarvest #
   - modifiers: onlyOwner
  - [Ext] updateMaxPriceOfLOOKSInWETH #
   - modifiers: onlyOwner
  - [Ext] updateTradingFeeUniswapV3 #
   - modifiers: onlyOwner
  - [Ext] updateThresholdAmount #
   - modifiers: onlyOwner
  - [Ext] pause #
   - modifiers: onlyOwner, whenNotPaused
  - [Ext] unpause #
   - modifiers: onlyOwner, whenNotPaused
  - [Ext] calculateSharePriceInLOOKS
  - [Ext] calculateSharePriceInPrimeShare
  - [Ext] calculateSharesValueInLOOKS
  - [ Int] _harvestAndSellAndCompound #
  - [ Int] _sellRewardTokenToLOOKS #
  - [ Int] _withdraw #
```

Detailed Results

Issues Checking Status

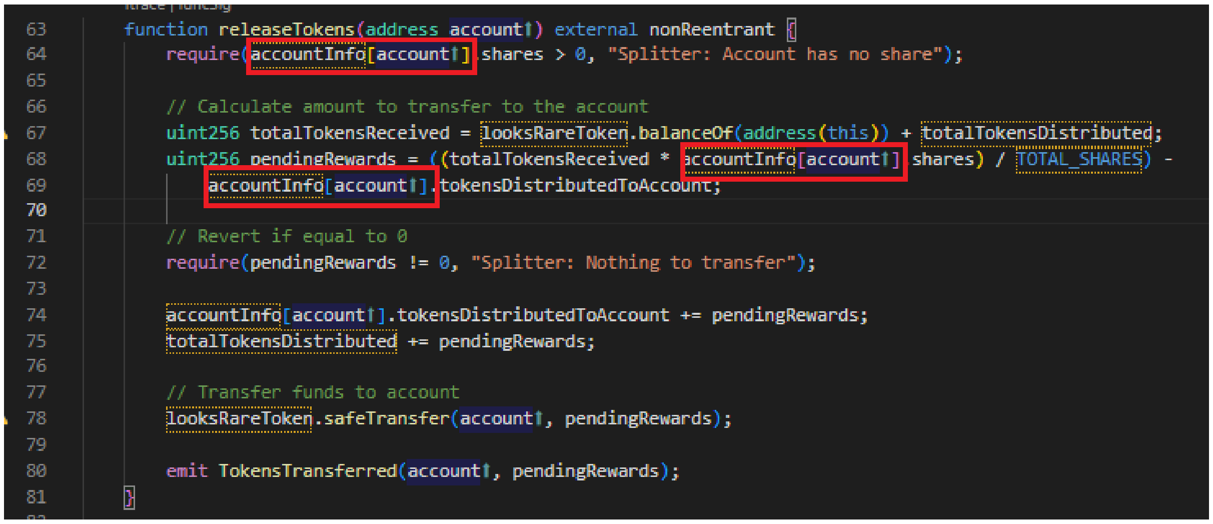
1. Arguments misselection error
   - Severity: High
   - Location:
     TokenDistributor.sol#99



```
96          require(
97              (_periodLengthesInBlocks.length == _numberPeriods) &&
98                  (_rewardsPerBlockForStaking.length == _numberPeriods) &&
99                  (_rewardsPerBlockForStaking.length == _numberPeriods),
100             "Distributor: Lengthes must match numberPeriods"
101         );
```

   - Description: Arguments misselection error occurred in line 99 of TokenDistributor.sol. It is overlapped with line 98 of TokenDistributor.sol, By this error, owner can enter the parameters wrongly and deploy the smart contracts having wrong arguments. It will affect the action of the contract and will lead to fatal error.
   - Remediations: Replace the line with this.
     (_periodLengthesInBlocks.length == _numberPeriods),

2. Not enough gas fee optimization
   - Severity: Low
   - Location:
     TokenSplitter.sol#64, 68, 69



```
63  function releaseTokens(address account) external nonReentrant {
64      require(accountInfo[account].shares > 0, "Splitter: Account has no share");
65
66      // Calculate amount to transfer to the account
67      uint256 totalTokensReceived = looksRareToken.balanceOf(address(this)) + totalTokensDistributed;
68      uint256 pendingRewards = ((totalTokensReceived * accountInfo[account].shares) / TOTAL_SHARES) -
69          accountInfo[account].tokensDistributedToAccount;
70
71      // Revert if equal to 0
72      require(pendingRewards != 0, "Splitter: Nothing to transfer");
73
74      accountInfo[account].tokensDistributedToAccount += pendingRewards;
75      totalTokensDistributed += pendingRewards;
76
77      // Transfer funds to account
78      looksRareToken.safeTransfer(account, pendingRewards);
79
80      emit TokensTransferred(account, pendingRewards);
81  }
```

   - Description: The unnecessary access to storage parameters should be as less as possible because it increases gas fee.
   - Remediations: You should use local variable.
     AccountInfo memory accountInfo_ = accountInfo[_account];

     require(accountInfo_.shares > 0, "Splitter: Account has no share");

You should replace all the unnecessary access to something like this.

There are a lot of unnecessary accesses in the source code and this will occur gas fee increasing.

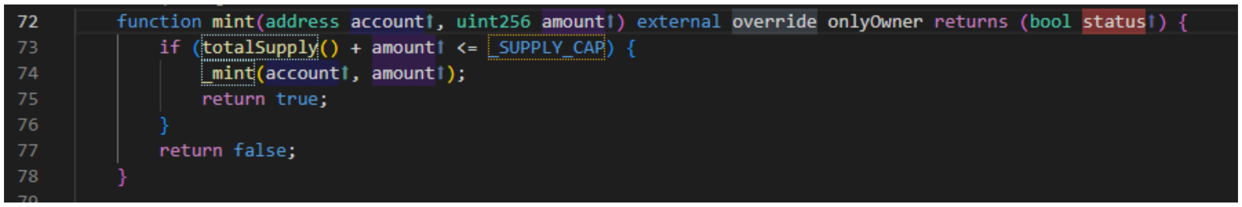
TokenSplitter.sol#64, #68, #69, #89, #90, #109, #114, #115

TokenDistributor.sol#270, #271, #279, #293, #294, #315, #320, #325

FeeSharingSystem.sol#184, #192, #193, #185, #188, #193, #195

AggregatorFeeSharingWithUniswapV3.sol#108, #111, #121, #364, #367, #377, #380, #383

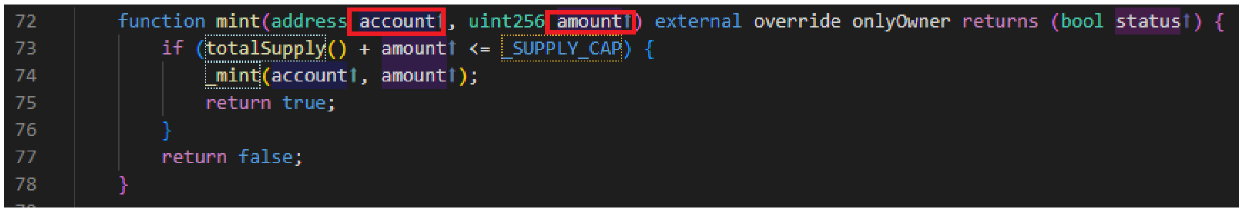
3. Parameter mistyping error
   - Severity: Weak
   - Location:
     LooksRareToken.sol#72



```
72  function mint(address account, uint256 amount) external override onlyOwner returns (bool status) {
73      if (totalSupply() + amount <= _SUPPLY_CAP) {
74          _mint(account, amount);
75          return true;
76      }
77      return false;
78  }
```

   - Description: The parameter status is declared but never used in the function body.
   - Remediations: The status parameter should be removed here

4. Unconventional coding
   - Severity: Weak
   - Location:
     LooksRareToken.sol#72



```
72  function mint(address account, uint256 amount) external override onlyOwner returns (bool status) {
73      if (totalSupply() + amount <= _SUPPLY_CAP) {
74          _mint(account, amount);
75          return true;
76      }
77      return false;
78  }
```

   - Description: The input parameter of the function should start with underscore(_).
     It is important convention in coding style.
     By keeping this, developers can see that it is the input parameter of the function easily and reduce fatal error.
   - Remediations:
     The account and amount should be replaced with _account and _amount.
     There are a lot of wrong conventions in this coding style.
     LooksRareToken.sol#72
     TokenSplitter.sol#63, #108

TokenDistributor.sol#142, #210, #269, #387

FeeSharingSystem.sol#80, #161, #174, #182, #202, #210, #245, #302

AggregatorFeeSharingWithUniswapV3.sol#100, #130, #292

Basic Coding Bugs

No.	Name	Description	Severity	Result
1.	Constructor Mismatch	Whether the contract name and its constructor are not identical to each other.	Critical	PASSED
2.	Ownership Takeover	Whether the set owner function is not protected.	Critical	PASSED
3.	Redundant Fallback Function	Whether the contract has a redundant fallback function.	Critical	PASSED
4.	Overflows & Underflows	Whether the contract has general overflow or underflow vulnerabilities	Critical	PASSED
5.	Reentrancy	Reentrancy is an issue when code can call back into your contract and change state, such as withdrawing ETHs	Critical	PASSED
6.	MONEY-Giving Bug	Whether the contract returns funds to an arbitrary address	High	PASSED

7.	Blackhole	Whether the contract locks ETH indefinitely: merely in without out	High	PASSED
8.	Unauthorized Self-Destruct	Whether the contract can be killed by any arbitrary address	Medium	PASSED
9.	Revert DoS	Whether the contract is vulnerable to DoS attack because of unexpected revert	Medium	PASSED
10.	Unchecked External Call	Whether the contract has any external call without checking the return value	Medium	PASSED
11.	Gasless Send	Whether the contract is vulnerable to gasless send	Medium	PASSED
12.	Send Instead of Transfer	Whether the contract uses send instead of transfer	Medium	PASSED
13.	Costly Loop	Whether the contract has any costly loop which may lead to Out-Of-Gas exception	Medium	PASSED
14.	(Unsafe) Use of Untrusted Libraries	Whether the contract use any suspicious libraries	Medium	PASSED

15.	(Unsafe) Use of Predictable Variables	Whether the contract contains any randomness variable, but its value can be predicated	Medium	PASSED
16.	Transaction Ordering Dependence	Whether the final state of the contract depends on the order of the transactions	Medium	PASSED
17.	Deprecated Uses	Whether the contract use the deprecated tx.origin to perform the authorization	Medium	PASSED
18.	Semantic Consistency Checks	Whether the semantic of the white paper is different from the implementation of the contract	Critical	PASSED

Conclusion

Overall, the code is well organized and clear on what it's supposed to do for each function. The mechanism to bet and distribute rewards is simple so it shouldn't bring major issues.

My final recommendation would be to pay more attention to gas fee optimization and arguments error of the functions since it's quite important to define who's supposed to executed the functions and to follow best practices regarding the use of them.

This is a secure contract that will store safely the funds while it's working.