verichains

SECURITY AUDIT OF

DRAGON KART TOKEN AND TOKENSVESTING CONTRACTS

Public Report

Nov 11, 2021

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Nov 11, 2021. We would like to thank the Dragon Kart for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Dragon Kart Token and TokensVesting Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified one vulnerable issue in the application, along with one recommendation.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Dragon Kart Token and TokensVesting Contracts

Dragon Kart is the first 3D Skill-Based Battle Racing Game between characters taken from 'Pikalong Series' by arties Thang Fly.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of Dragon Kart game.

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The initial review was conducted on Oct 20, 2021 and a total effort of 5 working days was dedicated to identifying and documenting security issues in the code base of the Dragon Kart Token and TokensVesting Contracts.

The audited contracts are the Dragon Kart Token and TokensVesting Contracts that deployed on Binance Smart Chain Mainnet.

The link of the deployed smart contracts are listed in the below table:

Contract Name	Deploy link
DragonKart	https://bscscan.com/address/0x8BDd8DBcBDf0C066cA5f3286d33673aA7A553C10
TokensVesting	https://bscscan.com/address/0x2da09af644fd517ad7dcf659e40172d70a94d01f

Table 2. The deployed smart contract links

2.2. Findings

During the audit process, the audit team found one vulnerability in the given version of Dragon Kart Token and TokensVesting Contracts.

2.2.1. Incorrect calculation of _vestedAmount function MEDIUM

In _vestedAmount function, this function doesn't check gaps > totalGaps. Therefore, _vestedAmount function can return with the value greater than totalAmount in some cases.

```
749  function _vestedAmount(
750          uint256 totalAmount_,
751          uint256 tgeAmount_,
752          uint256 cliff_,
753          uint256 duration_,
754          uint256 basis_
755      ) private view returns (uint256) {
756          require(
757              totalAmount_ >= tgeAmount_,
758              "TokensVesting::_vestedAmount: Bad params!"
759          );
```

```
760
761         if (block.timestamp < genesisTimestamp) {
762             return 0;
763         }
764
765         uint256 timeLeftAfterStart = block.timestamp - genesisTimest…
    amp;
766
767         if (timeLeftAfterStart < cliff_) {
768             return tgeAmount_;
769         }
770
771         uint256 linearVestingAmount = totalAmount_ - tgeAmount_;
772         if (timeLeftAfterStart >= cliff_ + duration_) {
773             return linearVestingAmount + tgeAmount_;
774         }
775
776         uint256 gaps = (timeLeftAfterStart - cliff_) / basis_ + 1;
777         uint256 totalGaps = duration_ / basis_;
778         return (linearVestingAmount / totalGaps) * gaps + tgeAmount_;
779     }
```

Snippet 1. TokensVesting.sol incorrect calculation of _vestedAmount function

For instance with a testcase,

basic=3; timeLeftAfterStart=9; cliff_=2; duration_=8.

After calculating at line 776 and 777, the value of gaps is 3 while the value of totalGaps is 2.

Therefore, the return value at line 778 will be greater than totalAmount_.

RECOMMENDATION

Adding a if statement to check the return value. If the return value is greater than totalAmount_, the function will return totalAmount_.

UPDATES

- *2021-10-21*: This issue has been acknowledged and fixed by the Dragon Kart team.

2.3. Additional notes and recommendations

2.3.1. Unnecessary check owner in pause and unpause functions INFORMATIVE

The contract inherits ERC20Pausable to pause and unpause contract by a specific address which has PAUSER_ROLE. But in the contract, to pause or unpause the specific address must have both PAUSER_ROLE and owner role. It will be an inconvenience if the contract changes another specific address to pause or unpause.

```
74  function pause() public onlyOwner {
75          require(
76              hasRole(PAUSER_ROLE, _msgSender()),
77              "Token: must have pauser role to pause"
78          );
79          _pause();
80      }
```

Snippet 2. Token.sol unnecessary check owner in pause function

```
91  function unpause() public onlyOwner {
92          require(
93              hasRole(PAUSER_ROLE, _msgSender()),
94              "Token: must have pauser role to unpause"
95          );
96          _unpause();
97      }
```

Snippet 3. Token.sol unnecessary check owner in unpause function

RECOMMENDATION

We suggest removing onlyOwner modifier in the functions which are mentioned above for gas saving.

UPDATES

- *2021-10-21*: This recommendation has been acknowledged and fixed by the Dragon Kart team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 20,2021*	Private Report	Verichains Lab
1.1	*Oct 21,2021*	Public Report	Verichains Lab
1.2	*Oct 22,2021*	Public Report	Verichains Lab
1.3	*Nov 04,2021*	Public Report	Verichains Lab
1.4	*Nov 08, 2021*	Public Report	Verichains Lab
1.5	*Nov 11,2021*	Public Report	Verichains Lab

Table 3. Report versions history