SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Orakuru
Date: April 8th, 2021

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed - upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Orakuru - Second Review
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Vesting Token
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	Initial review https://github.com/orakurudata/orakuru-contracts 50a76b054d8d1588a6b05df1f2155e90efba772c Second review 30600f4a6ca717b07b05c8cf19f01d268af2d66f Third review 489953ef263110283cdadede68c9e845e7517e1d
Timeline	7 April 2021 – 8 April 2021
Changelog	7 April 2021 – Initial Audit 8 April 2021 - Second Review 8 April 2021 - Third Review

Table of contents

Introduction 4
Scope 4
Executive Summary 5
Severity Definitions 7
AS-IS overview 8
Conclusion 10
Disclaimers 11

Introduction

Hacken OÜ (Consultant) was contracted by Orakuru (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contract and its code review conducted on April 8th, 2021.

Scope

The scope of the project is two smart contracts (Orakuru and TokenVesting) provided in Git Repository:

https://github.com/orakurudata/orakuru-contracts
489953ef263110283cdadede68c9e845e7517e1d

/libs/TokenVesting.sol	In scope
/Orakuru.sol	In scope
/oracles/CakePriceOracle.sol	Out of scope
/libs/IBEP20.sol	Out of scope
/interfaces/IOrakuruFeedConnector.sol	Out of scope
/interfaces/IOrakuruCore.sol	Out of scope
/interfaces/IOrakuruFeedAggregator.sol	Out of scope
/interfaces/IOrakuruVault.sol	Out of scope
/interfaces/IOrakuruAggregator.sol	Out of scope
/interfaces/IOracle.sol	Out of scope
/OrakuruVault.sol	Out of scope
/presale/OrkKickPrivateRound.sol	Out of scope
/libs/BEP20.sol	Out of scope
/presale/DummyStakingPool.sol	Out of scope
/oracles/OrakuruFeedAggregator.sol	Out of scope
/oracles/OrakuruAggregator.sol	Out of scope
/OrakuruCore.sol	Out of scope
/oracles/OrakuruFeedConnector.sol	Out of scope
/libs/SafeBEP20.sol	Out of scope

We have scanned these smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops

	▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Asset's integrity ▪ User Balances manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are secured.

Insecure	Poor secured	Secured	Well-secured

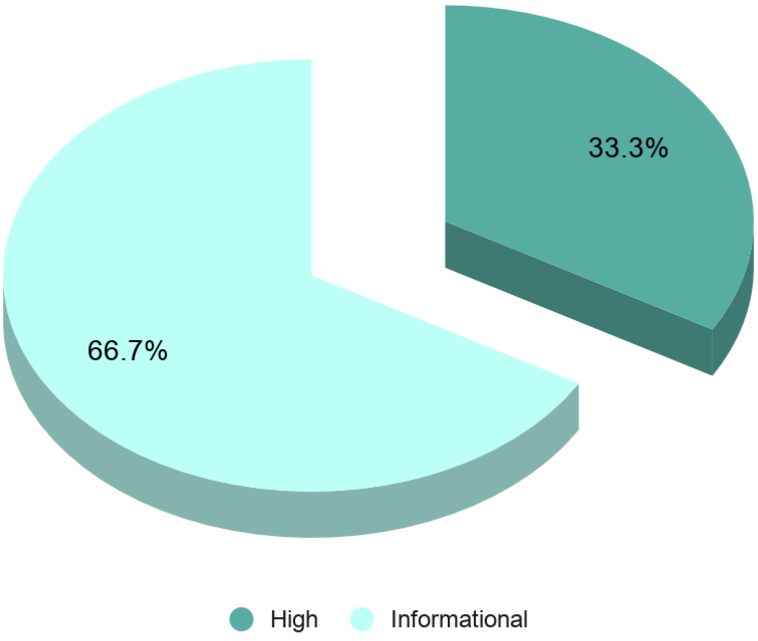
You are here

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section, and all found issues can be found in the Audit overview section.

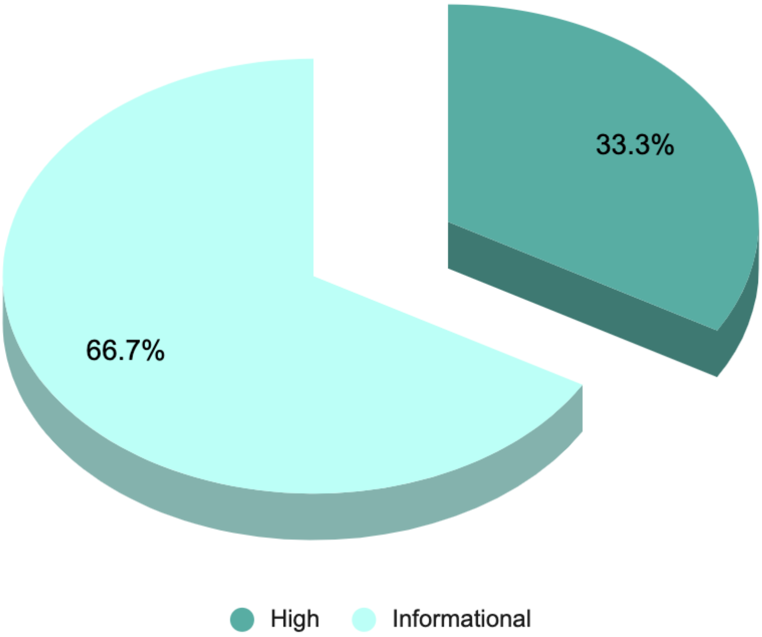
Security engineers found 1 high and 2 informational issues during the first review.

Security engineers found 1 high and 2 informational issues during the second review.

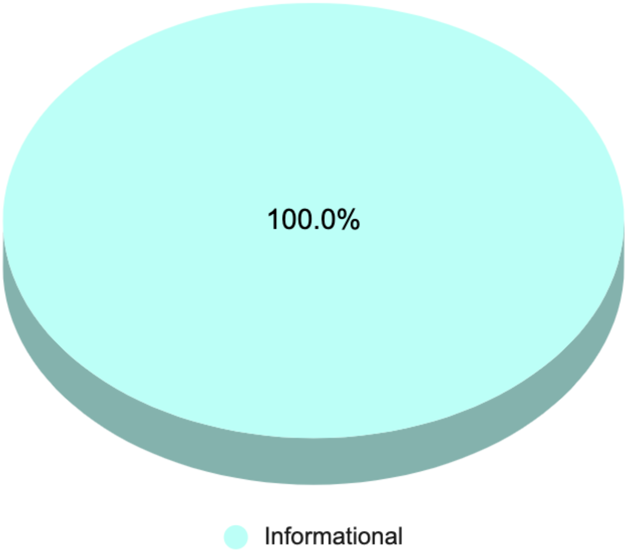
Graph 1. The distribution of vulnerabilities after the first review.



Graph 2. The distribution of vulnerabilities after the second review.



Graph 3. The distribution of vulnerabilities after the third review.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Audit overview

Critical

No Critical severity issues were found.

High

1. Vulnerability: The *mint* function of the *BEP20* contract which is inherited by the *Orakuru* contract allows the owner to mint an unrestricted amount of tokens at any time.

   Fixed before third review

Medium

No Medium severity issues were found.

Low

No Low severity issues were found.

Lowest / Code style / Best Practice

1. Vulnerability: Unnecessary inheritance
   Contract: TokenVesting

   TokenVesting inherits *Ownable* abstract contract, but never uses any of the methods or modifiers from it. Please consider removing unnecessary inheritance.

   In file:TokenVesting.sol:15

   ```
   contract TokenVesting is Ownable {
   ```

2. Vulnerability: Public function that could be declared external
   Contracts: TokenVesting, BEP20

   public functions that are never called by the contract should be declared external to save gas.

   Lines: libs/BEP20.sol#101

   ```
   function balanceOf(address account) public override view returns
   (uint256) {
   ```

   Lines: libs/BEP20.sol#113

```
function transfer(address recipient, uint256 amount) public override
returns (bool) {
```

Lines: libs/BEP20.sol#121

```
function allowance(address owner, address spender) public override view
returns (uint256) {
```

Lines: libs/BEP20.sol#132

```
function approve(address spender, uint256 amount) public override
returns (bool) {
```

Lines: libs/BEP20.sol#149

```
function transferFrom (address sender, address recipient, uint256
amount) public override returns (bool) {
```

Lines: libs/BEP20.sol#171

```
function increaseAllowance(address spender, uint256 addedValue) public
returns (bool) {
```

Lines: libs/BEP20.sol#190

```
function decreaseAllowance(address spender, uint256 subtractedValue)
public returns (bool) {
```

Lines: libs/BEP20.sol#203

```
function mint(uint256 amount) public onlyOwner returns (bool) {
```

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, high-level description of functionality was presented in As-Is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found 1 high and 2 informational issues during the first review.

Security engineers found 1 high and 2 informational issues during the second review.

Category	Check Items	Comments
➔ Code Review	➔ Style guide violation	➔ Unnecessary inheritance ➔ Public function that could be declared external

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.