SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Paribus
Date: December 17th, 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 Paribus.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Staking
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/Paribus/staking-contracts
Commit	cb06e063f1887a53b5bc981c7d2119a72f0234b5
Technical Documentation	YES
JS tests	YES
Website	paribus.io
Timeline	06 DECEMBER 2021 - 17 DECEMBER 2021
Changelog	10 DECEMBER 2021 - Initial Audit 17 DECEMBER 2021 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Paribus (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contract and its code review conducted between December 6th, 2021 - December 10th, 2021.

Second review conducted on December 17th, 2021.

Scope

The scope of the project is smart contracts in the repository:

Repository:
https://github.com/Paribus/staking-contracts
Commit:
cb06e063f1887a53b5bc981c7d2119a72f0234b5
Technical Documentation: Yes (WP: https://paribus.io/documents/PARIBUS-Litepaper-V1.0.pdf)
JS tests: Yes (https://github.com/Paribus/staking-contracts/tree/cb06e063f1887a53b5bc981c7d2119a72f0234b5/test)
Contracts:
ParibusStakeContractV3.sol
ParibusStakeManager.sol

We have scanned this smart contract 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 ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are well-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. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found **1** high and **2** low severity issues.

After the second review security engineers found that **all issues were addressed.**

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

Audit overview

🟥 🟥 🟥 🟥 Critical

No critical issues were found.

🟨 🟨 🟨 High

Possible DoS attack.

While using iteration over mapOfUserToUserShare it is possible for an attacker to send dozens of calls to the “stakeFor” function providing the same “to” address which could create “mapOfUserToUserShare[_to]” array of very big size.

After the attack function “_getUserRewardInformationFor” would not be accessible for the user used by attacker as “to”.

Contract: ParibusStakeContractV3.sol

Functions: _stakeFor, _getUserRewardInformationFor

Recommendation: Please use maths to calculate shares instead of for-loop calculations.

Status: Fixed

🟩 🟩 Medium

No medium severity issues were found.

🟦 Low

1. Reading state variable in the loop.

Accessing a state variable in the loop is causing excess gas expenses.

Contract: ParibusStakeManager.sol

Function: removeContract

Recommendation: Please store “contracts.length” value into the local variable to save some gas.

Status: Fixed

2. Iteration over unpredictable array length.

It is not recommended to iterate over an array of unpredictable length.

Contract: ParibusStakeManager.sol

Functions: removeContract, hasContract

Recommendation: Please store and index of added contract in the mapping state variable. It will allow to skip the for-loop and save lots of gas.

Status: Fixed

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

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

As a result of the audit, security engineers found **1** high and **2** low severity issues.

After the second review security engineers found that **all issues were addressed.**

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 the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free 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 the security of smart contracts.

Technical Disclaimer

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