HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: PeakDeFi

Date: March 3rd, 2021

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 fixed - upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for PeakDeFi.
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/PeakDeFi/liquidy_mining
Commit	
Deployed contract	
Timeline	01 MAR 2021 – 03 MAR 2021
Changelog	03 MAR 2021 – INITIAL AUDIT

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by PeakDeFi (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 between March 01st, 2021 – March 3rd, 2021.

Scope

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

Contract deployment address:
Repository
File:
PeakStakingRewards.sol
SafeERC20.sol
RewardsDistributionRecipient.sol
ReentrancyGuard.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

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

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

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.

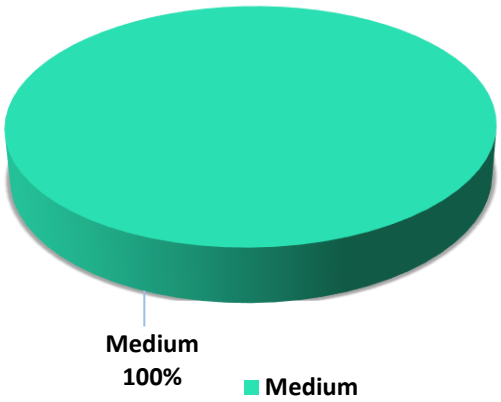
Notice:

1. The code is not tested. Existing tests does not validate any conditions.
2. The audit scope includes only contracts from the Scope section of this report and its security rating may not be extrapolated to another contracts of the PeakDeFi project.

Security engineers found **2** medium issues during the audit.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

AS-IS overview

PeakStakingRewards.sol

Description

PeakStakingRewards is a staking contract.

Imports

PeakStakingRewards has following imports:

- @openzeppelin/contracts/math/Math.sol
- ./interfaces/IStakingRewards.sol"
- ./interfaces/IUniswapV2ERC20.so
- ./libraries/ReentrancyGuard.sol
- ./libraries/SafeERC20.sol
- ./libraries/RewardsDistributionRecipient.sol

Inheritance

PeakStakingRewards is IStakingRewards, RewardsDistributionRecipient, ReentrancyGuard.

Usages

PeakStakingRewards contract has following usages:

- SafeMath for uint256
- SafeERC20 for IERC20

Structs

PeakStakingRewards contract has no data structures.

Enums

PeakStakingRewards contract has no enums.

Events

PeakStakingRewards contract has following events:

- event RewardAdded(uint256 reward)
- event Staked(address indexed user, uint256 amount)
- event Withdrawn(address indexed user, uint256 amount)

- event RewardPaid(address indexed user, uint256 reward)

Modifiers
PeakStakingRewards has following modifiers:

- updateReward – updates global and users reward params.

Fields

PeakStakingRewards contract has following fields and constants:

- IERC20 public rewardsToken
- IERC20 public stakingToken
- uint256 public periodFinish = 0
- uint256 public rewardRate = 0
- uint256 public rewardsDuration =30 days
- uint256 public lastUpdateTime
- uint256 public rewardPerTokenStored
- mapping(address => uint256) public userRewardPerTokenPaid
- mapping(address => uint256) public rewards
- uint256 private _totalSupply
- mapping(address => uint256) private _balances

Functions
PeakStakingRewards has following public functions:

- ***constructor***
  Description
  Inits the contract and sets default parameters.
  Visibility
  public
  Input parameters
  - address _rewardsDistribution
  - address _rewardsToken
  - address _stakingToken

  Constraints
  None
  Events emit
  None
  Output
  None

- ***totalSupply, balanceOf, lastTimeRewardApplicable, rewardPerToken, earned, getRewardForDuration***
  Description
  View functions that calculates corresponding values.
- ***stakeWithPermit***
  Description
  Stakes an *amount* of tokens. Requires permit.
  Visibility
  external
  Input parameters
  - uint256 amount
  - uint deadline
  - uint8 v
  - bytes32 r
  - bytes32 s

  Constraints
  - amount should be greater than 0.

  Events emit
  Emits Staked event.
  Output
  None
- ***stake***
  Description
  Stakes an *amount* of tokens.
  Visibility
  external
  Input parameters
  - uint256 amount

  Constraints
  - amount should be greater than 0.

  Events emit
  Emits Staked event.
  Output
  None
- ***withdraw***
  Description
  Withdraws an amount of tokens.
  Visibility
  public
  Input parameters

- uint256 amount

Constraints

- amount should be greater than 0.

Events emit

Emits Withdrawn event.

Output

None

- ***getReward***

Description

Withdraws all rewards received by a user.

Visibility

public

Input parameters

None

Constraints

None

Events emit

Emits RewardPaid event.

Output

None

- ***exit***

Description

Withdraws all tokens and rewards.

Visibility

public

Input parameters

external

Constraints

None

Events emit

Emits Withdrawn and RewardPaid events.

Output

None

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Audit overview

Critical

No critical issues were found.

High

No high severity issues were found.

Medium

1. Rewards calculations relies on 18 decimals. Tokens with another number of decimals may not be used.
   We recommend fetching token decimals from the token itself in the constructor and using this value afterwards.
2. safeApprove, safeIncreaseAllowance and safeDecreaseAllowance of the SafeERC20 contract is never used.
   We recommend removing unused functions.

Low

No low severity issues were found.

Informational / Code style / Best Practice

No informational issues were found.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.

Notice:

1. The code is not tested. Existing tests does not validate any conditions.
2. The audit scope includes only contracts from the Scope section of this report and its security rating may not be extrapolated to another contracts of the PeakDeFi project.

Security engineers found **2** medium issues during the audit.

Violations in the following categories were found and addressed to Customer:

Category	Check Item	Comments
Code review	▪ Repository consistency	▪ The code is not covered with unit tests.
	▪ Unused code	▪ Unused code were found.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.