HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: KICK

Date: March 9th, 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 KICK.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Token
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	
Commit	
Deployed contract	
Timeline	8 MAR 2021 – 9 MAR 2021
Changelog	9 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 KICK (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 8th, 2021 – March 9th, 2021.

Scope

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

Contract deployment address:
Repository
File:
KickPad.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 contract is secure but does not match the whitepaper.

Insecure	Poor secured	Secured	Well-secured

You are 

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.

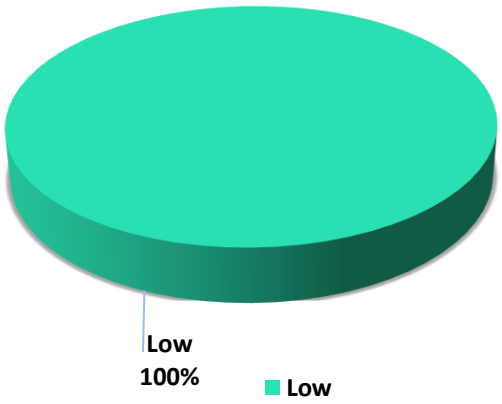
Security engineers found **2** low severity issues during the audit.

Notices:

1. The code implements a simple ERC20 token. OpenZeppelin version of the token may be used instead.
2. Reviewed code is a token that mints all initial supply to a deployer immediately after deployment. No vesting functionality described in the whitepaper is implemented.

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

KickPad.sol

Description

KickPad is token contract.

Imports

KickPad contract has following imports:

- ERC20 – an interface from the OpenZeppelin.
- SafeMath – a library from the OpenZeppelin.

Inheritance

KickPad is ERC20.

Usages

KickPad contract has following usages:

- SafeMath for uint256;

Structs

KickPad contract has no data structures.

Enums

KickPad contract has no enums.

Events

KickPad contract has following events:

- event Transfer(address indexed from, address indexed to, uint256 value);
- event Approval(address indexed owner, address indexed spender, uint256 value);

Modifiers

KickPad contract has no modifiers.

Fields

KickPad contract has following fields and constants:

- address private deployer;
- string public name = "KickPad";
- string public symbol = "KPAD";
- uint8 public constant decimals = 18;
- uint256 private constant decimalFactor = 10 ** uint256(decimals);
- uint256 public constant totalSupply = 203768315 * decimalFactor;
- bool public minted = false;
- mapping (address => uint256) balances;
- mapping (address => mapping (address => uint256)) internal allowed;

Public Functions

KickPad contract has following public functions:

- ***constructor***

  Description

  Initializes the contract.

  Visibility

  public

  Input parameters

  None

  Constraints

  None

  Events emit

  - Transfer(address(0), msg.sender, totalSupply)

  Output

None

- ***owner***

  Description

  Used to get the owner address.

  Visibility

  public view

  Input parameters

  None

  Constraints

  None

  Events emit

  None

  Output

  - address deployer
- ***balanceOf***

  Description

  Used to get the balance of the address.

  Visibility

  public view

  Input parameters

  - address _owner

  Constraints

  None

Events emit

None

Output

- uint256 balance

- ***allowance***

  Description

  Used to get the allowed amount that can be spent.

  Visibility

  public view

  Input parameters

  - address _owner
  - address _spender

  Constraints

  None

  Events emit

  None

  Output

  - uint256

- ***transfer***

  Description

  Used to transfer tokens.

  Visibility

  public

Input parameters

- address _to
- uint256 _value

Constraints

- _to address can not be zero.
- _value should be less than or equal to balance of msg.sender.

Events emit

- Transfer(msg.sender, _to, _value)

Output

- bool

- ***transferFrom***

Description

Used to transfer tokens.

Visibility

public

Input parameters

- address _from
- address _to
- uint256 _value

Constraints

- _to address can not be zero.
- _value should be less than or equal to balance _from.
- _value should be less than or equal to allowed balance.

Events emit

- Transfer(_from, _to, _value)

Output

- bool

- ***approve***

  Description

  Used to set amount that can be spent.

  Visibility

  public

  Input parameters

  - address _spender
  - uint256 _value

  Constraints

  None

  Events emit

  - Approval(msg.sender, _spender, _value)

  Output

  - bool

- ***increaseApproval***

  Description

  Used to increase amount that can be spent.

  Visibility

  public

  Input parameters

  - address _spender
  - uint _addedValue

Constraints

None

Events emit

- Approval(msg.sender, _spender, allowed[msg.sender][_spender])

Output

- bool

- ***decreaseApproval***

Description

Used to decrease amount that can be spent.

Visibility

public

Input parameters

- address _spender
- uint _subtractedValue

Constraints

None

Events emit

- Approval(msg.sender, _spender, allowed[msg.sender][_spender])

Output

- bool

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

No medium severity issues were found.

Low

1. The *minted* global variable is declared but never used.
2. The *deployer* global variable is never used internally and does not store any useful information.

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.

Security engineers found **2** low severity issues during the audit.

Notices:

1. The code implements a simple ERC20 token. OpenZeppelin version of the token may be used instead.
2. Reviewed code is a token that mints all initial supply to a deployer immediately after deployment. No vesting functionality described in the whitepaper is implemented.

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

Category	Check Item	Comments
Code review	▪ Business Logics Review	▪ The code does not match the whitepaper.
	▪ Style guide violation	▪ The code contains unused fields.

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.