HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: RichQuack
Date: April 26th, 2022

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

Document

Name	Smart Contract Code Review and Security Analysis Report for RichQuack.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	Staking
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://www.richquack.com/
Timeline	20.04.2022 - 26.04.2022
Changelog	26.04.2022 - Initial Review

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 RichQuack (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 contracts.

Scope

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

Initial review scope
Repository:
https://github.com/Richquack/Launchpad/tree/develop
Commit:
502bf137100af1c2417828a0e6a04f641aedaeae
Deployed Contract Address:
https://bscscan.com/address/0x24E1FB7a781d255EdC40e80C89d9289dC61925F2#code
Technical Documentation: Yes
(https://docs.google.com/document/d/17_0R2ofAMVmUupnG3w2MgK-vq4GJAL_QpAT4C3HdUqI/edit?usp=sharing)
JS tests: Yes
Contracts:
Staking.sol
SHA3: b8ef7bfc3f900c9ff6d4840fc5101d5a8ba2d1c049fb163ea663cf8394c60d67

IStaking.sol
SHA3: 68db8e0ad697769cde9a4bb50a9ada78fe6f94e90fb812911f19d6faffe3eec4

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 cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

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

Executive Summary

The score measurements details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided functional requirements and technical requirements. However, they have inconsistent information. The total Documentation Quality score is **8** out of **10**.

Code quality

The total CodeQuality score is **10** out of **10**. Code follows official language style guides. Unit tests were provided.

Architecture quality

The architecture quality score is **10** out of **10**. The architecture is clear.

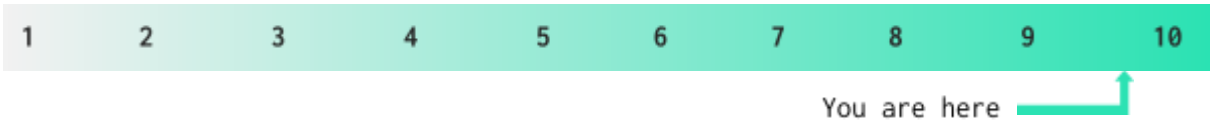
Security score

As a result of the audit, security engineers found **4** low severity issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.8**



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

Checked Items

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

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Failed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be destroyed until it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed

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

Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Failed
Signature Unique Id	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Failed
EIP standards violation	EIP	EIP standards should not be violated.	Not Relevant
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Not Relevant

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block gas limit.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with requirements provided by the Customer,	Passed
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Tests coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed

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

System Overview

Staking is an ERC-20 staking project with the following contracts:

- Staking — a contract that rewards users for staking their tokens. APY depends on the lock period (7 days - 0% APY, 14 days - 8% APY, 30 days - 13% APY, 28 days - 0% APY). The staking has 9 levels depending on the amount of tokens staked. Each lock period has allowed staking levels. Staking allows re-stakings (adding 5 more days to each of 7, 14, 30 days stakings) that users can add with the “PRESALE” role. ¾ of stalking can be withdrawn if the lock period has not finished. **Special Notice:** if the contract does not have enough rewards balance to send during deposit upgrade, execution will fail.
- IStaking — an interface that defines some staking functions.

Privileged roles

- The user with the “DEFAULT_ADMIN_ROLE” role of the staking contract can start and finish the staking pool, withdraw collected fees, grant the “FABRIC” and the “PRESALE” roles to addresses.
- Users with the “FABRIC” role can grant the “PRESALE” roles to addresses.
- Users with the “PRESALE” role can add re-lock (add 5 days to each of 7, 14, 30 days stakings of the user) anytime.

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

Findings

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

1. **Incorrect level eligibility.**

   According to the functional requirements, level eligibility for 7 and 14 days locks is 1-7 levels for both, but in the technical requirements and code they are 1-3 levels for the 7 days lock and 1-6 for the 14 days lock.

   Stacking rules must match those described in the documentation so that users have the correct information.

   Contracts:Staking.sol

   Function: deposit

   Recommendation: change the code to meet the functional requirements or modify the documentation.

   Status: Fixed

Low

1. **Unused variable.**

   Field “relockOw” is never used.

   Contracts: Staking.sol

   Function: -

   Recommendation: remove unused variable.

   Status: New

2. **Unlocked pragma**

   Contracts with unlocked pragmas may be deployed by the latest compiler, which may have higher risks of undiscovered bugs.

   Contracts: Staking.sol, IStaking.sol

   Function: -

   Recommendation: lock pragmas to a specific compiler version.

   Status: New

3. **Using block numbers for time calculations**

The contract uses block.timestamp for time calculations. It is not precise and safe.

Contracts: Staking.sol

Functions: startPool, endPool, deposit, _calcFee, _timestamp

Recommendation: it is recommended to avoid using block.timestamp. Alternatively, it is safe to use oracles.

Status: New

4. **No events on state variables changings.**

   It is recommended to emit events on important state changes.

   Contracts: Staking.sol

   Functions: startPool, endPool, deposit, emergencyWithdraw, withdraw, _updateLevel

   Recommendation: emit events on important state changes.

   Status: New

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 by the best industry practices at the date of this report, with 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 a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. 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 cannot guarantee the explicit security of the audited smart contracts.