HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: HederaPad
Date: April 06th, 2022

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 are fixed — upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for HederaPad.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	Staking; Pools; Proxy; Whitelist
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://www.hederastarter.fi
Timeline	21.03.2022 - 06.04.2022
Changelog	31.03.2022 - Initial Review 06.04.2022 - Revise

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by HederaPad (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:

Repository:
    https://github.com/HederaStarter/hedera-contracts
Commit:
    6621e739ece1b66bfa05a7b89407cb59be1ad8ed
Technical Documentation: Yes
(https://github.com/HederaStarter/hedera-contracts/blob/main/README.md)
JS tests: Yes
(https://github.com/HederaStarter/hedera-contracts/tree/main/test)
Contracts:
    Pools/Base.sol
    Pools/State.sol
    Pools/FixedPricePool.sol
    Whitelist.sol
    UpgradeabilityProxy.sol
    StakingImp.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 ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ EIP standards violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency

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

Executive Summary

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

Documentation quality

The Customer provided the description of functions, events, and states. However, neither functional requirements nor technical documentation (flows, sequences, diagrams, tech specs) was provided. The total Documentation Quality score is **4** out of **10**.

Code quality

The total CodeQuality score is **6** out of **10**. Code duplications. Unit tests were provided. No NatSpecs. No comments through the code. Not following code style guidelines.

Architecture quality

The architecture quality score is **6** out of **10**. The logic is split by files. Functions are overwhelmed with template code that could be moved to separate functions and be reused.

Security score

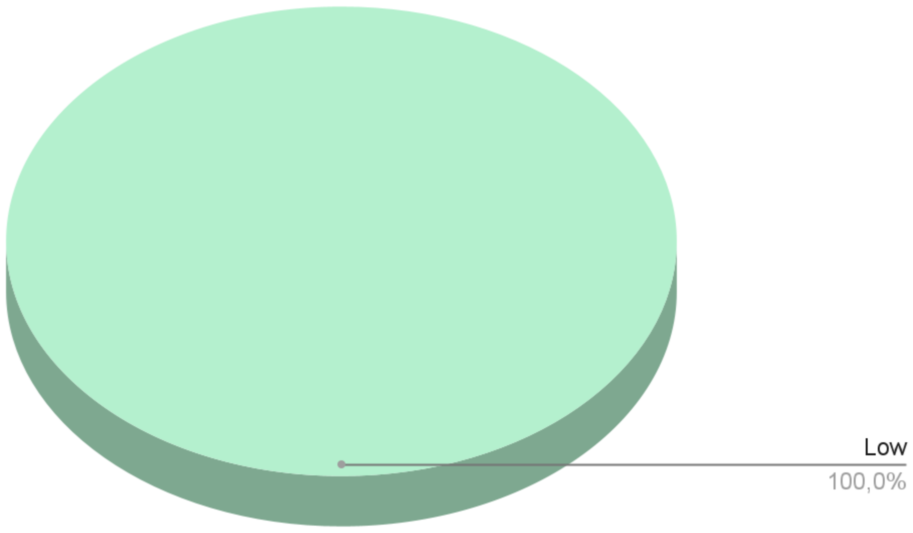
As a result of the audit, security engineers found **1** low severity issue. The security score is **10** out of **10**. All found issues are displayed in the “Issues overview” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **8.6**

1
2
3
4
5
6
7
8
9
10
You are here

Graph 1. The distribution of vulnerabilities after the audit.



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

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟨🟨🟨 High

No high severity issues were found.

🟩🟩 Medium

Rewards could be collected by the owner.

A contract owner could take all rewards by calling the `withdrawFee` function.

Contracts: StakingImp.sol

Functions: withdrawFee

Recommendation: we recommend incrementing some state variable (ie. collectedFees) when fees are collected and then allowing the owner to withdraw only collected fees, not rewards.

Status: Fixed (Revised Commit: 6621e73)

🟦 Low

1. **SPDX license identifier not provided in a source file.**

   Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.

   Contracts: StakingImp.sol, Whitelist.sol

   Recommendation: add SPDX-license identifiers.

   Status: Fixed (Revised Commit: 6621e73)

2. **Solidity compiler version.**

   An old solidity compiler version is used. It is always recommended to use the latest stable version of the compiler. Using old compiler forces to use the outdated openzeppelin libraries, which do not include the latest updates.

   Contract: all

   Function: _validateInputData

   Recommendation: use the latest compilers version (i.e.: 0.8.11)

   Status: Not changed (Revised Commit: 6621e73)

3. **Floating solidity version.**

It is recommended to specify the exact solidity version in the contracts.

Contracts: UpgradeabilityProxy.sol, StakingImp.sol, State.sol, FixedPricePool.sol, Base.sol

Recommendation: please specify the exact solidity version (ex. pragma solidity 0.7.6 instead of pragma solidity ^0.7.0).

Status: Fixed (Revised Commit: 6621e73)

4. **No event was emitted.**

   Functions that change sensitive the contract state should emit events.

   Contracts: Base.sol, StakingImp.sol, FixedPricePool.sol

   Functions: Base.changeTokenAmountToSell, StakingImp.setFeeBurn, FixedPricePool.changePrice, StakingImp.setNewUnstakingFeeRatio, StakingImp.changeUnstakingFeeRatio, Base.changeToken, Base.changeTokenAmountToSell, Base.emergencyPause, Base.emergencyUnpause, Base.emergencyCancel

   Recommendation: emit events on changing the sensitive contract state.

   Status: Fixed (Revised Commit: 6621e73)

5. **Boolean equality.**

   Boolean constants can be used directly and do not need to be compared to **true** or **false**.

   Contracts: Whitelist.sol, StakingImp.sol

   Functions: Whitelist.initialize, StakingImp.initialize

   Recommendation: Remove the equality to the boolean constant.

   Status: Fixed (Revised Commit: 6621e73)

6. **Reading state in the loop.**

   Reading the \`length\` attribute of the array in the loop is gas insufficient.

   Contracts: Base.sol

   Functions: _checkMaximumAllocations, _checkUnlockParameters

   Recommendation: store the \`length\` value into a local memory variable and use it in the for-loop.

   Status: Fixed (Revised Commit: 6621e73)

7. **A public function that could be declared external.**

   Public functions that are never called by the contract should be declared **external**.

Contracts: Base.sol, FixedPricePool.sol, StakingImp.sol, UpgradeabilityProxy.sol, Whitelist.sol

Functions: Base.changeToken, Base.changeTime, Base.changeTokenAmountToSell, Base.changeMinimumFillPercentage, Base.changeMinimumOrderSize, Base.changeClaimLockDuration, Base.changeMaximumAllocations, Base.changeStakingContract, Base.changeWhitelistContract, Base.setStakingTokenReward, Base.setPoolOngoing, Base.setPoolFinish, Base.withdrawFund, Base.withdrawTokenAfterFail, Base.withdrawRemainingTokens, Base.withdrawRemainingStakingToken, Base.withdrawStakingTokenAfterFail, Base.refund, Base.withdrawReservedTokens, Base.emergencyChangeClaimAvailable, Base.changeUnlockCheckpointsAndPercentages, Base.emergencyPause, Base.emergencyUnpause, Base.emergencyCancel, FixedPricePool.changePrice, FixedPricePool.reserve, StakingImp.initialize, StakingImp.updateSchedule, StakingImp.getCheckPoints, StakingImp.getRewardPerSecond, StakingImp.setFeeBurn, StakingImp.transferStake, StakingImp.getRewardThenStake, StakingImp.withdrawFee, StakingImp.setNewUnstakingFeeRatio, StakingImp.changeUnstakingFeeRatio, StakingImp.showPendingReward, UpgradeabilityProxy.setNewProxyOwner, UpgradeabilityProxy.transferProxyOwnership, UpgradeabilityProxy.setNewImplementation, UpgradeabilityProxy.transferImplementation, Whitelist.initialize, Whitelist.changeTier, Whitelist.changeTierBatch, Whitelist.getTier

Recommendation: use the external attribute for functions never called from the contract.

Status: Fixed (Revised Commit: 6621e73)

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.