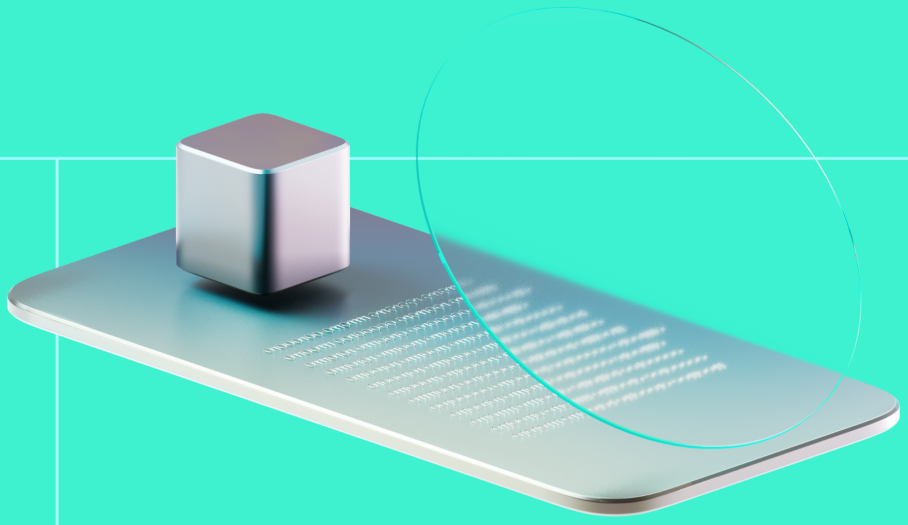
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Justfarming

Date: 07 Dec, 2023

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

We thank Justfarming for allowing us to conduct a Smart Contract Security Assessment. This document outlines our methodology, limitations, and results of the security assessment.

Platform: EVM

Timeline: 10.11.2023 - 07.12.2023

Language: Solidity

Methodology: Link

Tags: Governance, Staking

Last review scope

Repository	https://github.com/justfarming/contracts
Initial Commit	03d77111
Remediation Commit	1364644

View full scope

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Audit Summary

10/10	10/10	80.65%	10/10
Security score	Code quality score	Test coverage	Documentation quality score

Total: 9.3/10

The system users should acknowledge all the risks summed up in the risks section of the report.

1	0	0	1
Total Findings	Resolved	Acknowledged	Mitigated

Findings by severity	Findings Number	Resolved	Mitigated	Acknowledged
Critical	0	0	0	0
High	1	0	1	0
Medium	0	0	0	0
Low	0	0	0	0

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

This report 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 can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Justfarming
Approved By	Paul Fomichov | SC Audits Expert at Hacken OÜ
Audited By	Turgay Arda Usman | SC Lead Auditor at Hacken OÜ Kornel Światłowski | SC Auditor at Hacken OÜ
Website	https://www.justfarming.xyz/
Changelog	14.11.2023 – Preliminary Report 06.12.2023 – Remediation Report

Introduction

Hacken OÜ (Consultant) was contracted by Justfarming (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.

System Overview

Justfarming is a staking protocol with the following contracts:

- BatchDeposit - contract enables the batch deposit of validators to the Ethereum 2.0 deposit contract. Validators available for deposit are registered by the contract owner.
- StakingRewards - contract facilitates the division of staking rewards between a rewards recipient and a fee recipient. The reward split affects only accumulated rewards, not any returned stake. The rewards recipient is responsible for registering (activating) and deregistering (exiting) any associated validators. The contract operates on a pull payment model, holding payments until the release() function is used.

Privileged roles

The BatchDeposit contract uses the Ownable library from OpenZeppelin to restrict access to key functions. The owner can register new validators.

The StakingRewards contract uses AccessControl to define 3 roles and grant them special privileges.

- DEPOSITOR_ROLE - role assigned to the BatchDeposit contract. Addresses possessing this role have the ability to activate new validators.
- FEE_RECIPIENT_ROLE - role associated with the Client. Addresses holding these roles possess the capability to release rewards designated for the fee recipient.
- REWARDS_RECIPIENT_ROLE - address associated with the user. Addresses holding these roles possess the capability to release rewards designated for the user.

Executive Summary

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

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are detailed.
- Technical description is robust.

Code quality

The total Code Quality score is **10** out of **10**.

- The code follows the best practices and guidelines

Test coverage

Code coverage of the project is **80.65%** (branch coverage).

- Basic functionality is tested.
- Deployment instructions are provided.

Security score

As a result of the audit, the code contains **no** 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: **9.3**. The system users should acknowledge all the risks summed up in the risks section of the report.

Risks

- The BatchDeposit contract interacts with the DepositContract, which falls outside the audit scope. For each validator, a transfer of 32 ETH is initiated to this external, out-of-scope contract.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Incorrect Calculation Due to Missing Update

The *releasable()* function calculates the amount of releasable ETH for the caller. This calculation is made for either the fee or the receiver of the reward. The amount of exited stake, *_exitedStake*, plays an important role in this calculation. It is used to calculate the total reward/fee amount. This function, *releasable(),* is called in the *release()* function, where withdrawal of the accumulated rewards or fees to the respective recipient occurs. The *_exitedStake* variable is updated in a function other than the *release()* function, *exitValidators()*, which is only callable by the reward recipient. This means that, for calculations to be accurate, the *exitValidators()* has to be called.

Since withdrawals to the withdrawal address will simply be increased to the account's ETH balance that does NOT trigger any EVM execution, the funds will be transferred to the StakingRewards contract and the *exitValidators()* function won't be called. This will lead to a miscalculation of the releasable funds.

```
    function releasable(address account⬆) public view returns (uint256) {
        uint256 totalBalance = address(this).balance + _totalReleased;
        uint256 billableRewards = 0;

        if (totalBalance >= _exitedStake) {
            billableRewards = totalBalance - _exitedStake;
        }

        if (account⬆ == _feeRecipient && billableRewards > 0) {
            uint256 totalFees = (billableRewards * _feeBasisPoints) / 10 ** 4;
            uint256 releasedFees = released(_feeRecipient);
            return totalFees - releasedFees;
        } else if (account⬆ == _rewardsRecipient) {
            uint256 totalRewards = (billableRewards *
                (MAX_BASIS_POINTS - _feeBasisPoints)) / 10 ** 4;
            uint256 releasedRewards = released(_rewardsRecipient);
            return totalRewards + _exitedStake - releasedRewards;
        } else {
            return 0;
        }
    }
```

```
    function exitValidator(
        bytes calldata pubkey⬆
    ) external virtual onlyRole(REWARDS_RECIPIENT_ROLE) {
        require(
            pubkey⬆.length == PUBKEY_LENGTH,
            "public key must be 48 bytes long"
        );
        require(_isActiveValidator[pubkey⬆], "validator is not active");
        _isActiveValidator[pubkey⬆] = false;
        _numberOfActiveValidators = _numberOfActiveValidators - 1;
        _exitedStake = _exitedStake + STAKE_PER_VALIDATOR;
        emit ValidatorExited(pubkey⬆);
    }
```

Paths: ./contracts/lib/StakingRewards.sol : releasable ()

Recommendation: Make sure that the *_exitedStake* variable is being updated after each release.

Found in: 03d77111

Status: Mitigated. (Due to the nature of the Ethereum infrastructure, it is currently not possible to update the aforementioned variable. Thus it must be updated manually via the related function by the clients. This necessity is included in the related documentation of the protocol.) (Commit: 90c75cb576666c2df357d5791d1b4395e0f404c7)

Medium

No medium severity issues were found.

Low

No low severity issues were found.

Informational

I01. Public Functions That Should Be External

Functions that are meant to be exclusively invoked from external sources should be designated as "external" rather than "public." This is essential to enhance both the Gas efficiency and the overall security of the contract.

Paths: ./contracts/lib/StakingRewards.sol : numberOfActiveValidators(), feeBasisPoints(), totalReleased(), activateValidators(),

./contracts/lib/BatchDeposit.sol : isValidatorAvailable(),

Recommendation: Transition the relevant functions, which are exclusively utilized by external entities, from their current "public" visibility setting to the "external" visibility setting.

Found in: 03d77111

Status: Fixed (Revised commit: 1364644)

Remediation: Mentioned function visibility is now set to “external”.

I02. Use Custom Errors Instead Of Error Strings To Save Gas

Custom errors were introduced in Solidity version 0.8.4, and they offer several advantages over traditional error handling mechanisms:

Gas Efficiency: Custom errors can save approximately 50 Gas each time they are hit because they avoid the need to allocate and store revert strings. This efficiency can result in cost savings, especially when working with complex contracts and transactions.

Deployment Gas Savings: By not defining revert strings, deploying contracts becomes more Gas-efficient. This can be particularly beneficial when deploying contracts to reduce deployment costs.

Versatility: Custom errors can be used both inside and outside of contracts, including interfaces and libraries. This flexibility allows for consistent error handling across different parts of the codebase, promoting code clarity and maintainability.

Paths: ./contracts/lib/StakingRewards.sol,

./contracts/lib/BatchDeposit.sol,

Recommendation: To save Gas, it is recommended to use custom errors.

Found in: 03d77111

Status: Fixed (Revised commit: a533a2a)

Remediation: Custom errors are now used.

I03. Redundant Payable Modifier In StakingRewards Constructor

The *StakingReward* *constructor()* includes a redundant *payable* modifier. Neither Natspec nor the documentation provides any justification or explanation for the inclusion of a *payable* modifier in the *constructor()*.

Path: ./contracts/lib/StakingRewards.sol : constructor(),

Recommendation: Remove the redundant *payable* modifier, or provide a rationale for the necessity of the *payable* modifier.

Found in: 03d77111

Status: Fixed (Revised commit: d4c6be3)

Remediation: Redundant “payable” modifier is now removed.

I04. Redundant Check Of FeeBasisPoints

In the *StakingReward* *constructor()*, a redundant check is identified within one of its require conditions. Specifically, the condition checks if *newFeeBasisPoints* is both greater than or equal to 0 and lower than or equal to *MAX_BASIS_POINTS*. The second check inherently encompasses the first, rendering the initial check superfluous and eligible for removal.

```
require(
    newFeeBasisPoints >= 0 && newFeeBasisPoints <= MAX_BASIS_POINTS,
    "fees must be between 0% and 100%"
);
```

Path: ./contracts/lib/StakingRewards.sol : constructor(),

Recommendation: Remove the redundant check in the mentioned *require*.

Found in: 03d77111

Status: Fixed (Revised commit: a533a2a)

Remediation: The check is replaced with the following if condition:

```
if (newFeeBasisPoints > MAX_BASIS_POINTS)
    revert InvalidFeeBasisPoints(newFeeBasisPoints);
```

I06. Misleading Error Message

In the *BatchDeposit* *batchDeposit()* function, there is a check for the *Registered* status of the supplied validator public key. However, the error message (*validator is not available*) in case of a failure is misleading and lacks a valid revert reason, thus hindering effective understanding of the issue.

Path: ./contracts/lib/BatchDeposit.sol : batchDeposit(),

Recommendation: It is recommended that the error message accurately reflects the reason for the revert and provides sufficient information for users to identify and address the issue.

Found in: 03d77111

Status: Fixed (Revised commit: e8c872d)

Remediation: A custom error message has been implemented.

I07. Floating Pragma

The *IDepositContract.sol* uses floating pragmas ^0.8.0.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version, which may include bugs that affect the system negatively.

Path: ./contracts/interfaces/IDepositContract.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Found in: 03d77111

Status: Fixed (Revised commit: e8c872d)

Remediation: Pragma is now set to 0.8.21.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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 Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope details

Repository	https://github.com/justfarming/contracts/tree/main
Initial Commit	03d77111
Remediation Commit	1364644
Whitepaper	Not provided
Requirements	Link
Technical Requirements	Link

Contracts in Scope

./contracts/lib/BatchDeposit.sol
./contracts/lib/StakingRewards.sol
./contracts/interfaces/IDepositContract.sol

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia