HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: BlockSquare
Date: March 8, 2023

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 BlockSquare
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token; Staking
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://blocksquare.io/
Changelog	13.01.2023 - Initial Review 02.02.2023 - Second Review 09.02.2023 - Third Review 08.03.2023 - Fourth 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 BlockSquare (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/blocksquare/LPStaking-Audit
Commit	08f6760a66c8e490a550202a8d504d0adf1f21cc
Functional Requirements	https://github.com/blocksquare/LPStaking-Audit/blob/main/README.md
Technical Requirements	https://github.com/blocksquare/LPStaking-Audit/blob/main/README.md
Contracts	File: ./contracts/LPStaking.sol SHA3: ba3dc2589ec504078fd4cca73d027dd5447dbcc7d6d2af3ae7a8f32f231e4064

Second review scope

Repository	https://github.com/blocksquare/LPStaking-Audit
Commit	78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c
Functional Requirements	https://github.com/blocksquare/LPStaking-Audit/blob/main/README.md
Technical Requirements	https://github.com/blocksquare/LPStaking-Audit/blob/main/README.md
Contracts	File: ./contracts/LPStaking.sol SHA3: 6b3beb5e8635a1176bb092a507fa088f17cf98ff35c2e02ee672298c0e9a63b2

Third review scope

Repository	https://github.com/blocksquare/LPStaking-Audit
Commit	ec432b10e47e69377cad4421db93ed429af79492
Functional Requirements	https://github.com/blocksquare/LPStaking-Audit/blob/main/README.md
Technical Requirements	https://github.com/blocksquare/LPStaking-Audit/blob/main/README.md
Contracts	File: ./contracts/LPStaking.sol SHA3: f189785f5a63606b9a255057612d2473d6df1185f291b763f8ea130d73a7ecaa

Fourth review scope

Repository	https://github.com/blocksquare/LPStaking-Audit
Commit	8c02c0c9484b73faa5b4f642ab229809c8597ee7
Functional Requirements	https://github.com/blocksquare/LPStaking-Audit/blob/main/README.md
Technical Requirements	https://github.com/blocksquare/LPStaking-Audit/blob/main/README.md
Contracts	File: ./contracts/LPStaking.sol SHA3: 9ae6274a50111e6809c16ccb0eb1b6197391fa49bf161a826d1df7560790f816

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 the loss of user funds or contract state manipulation by external or internal actors.
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 by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

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

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 **8** out of **10**.

- Functional requirements are partially missed.
- Technical description is provided.
- The code is extensively documented using NatSpec.

Code quality

The total Code Quality score is **8** out of **10**.

- Test coverage is insufficient.
- CEI pattern violations are masked by nonReentrant.

Test coverage

Code coverage of the project is **69.23%** (branch coverage).

- Not all negative scenarios are covered.

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: **9.4**.

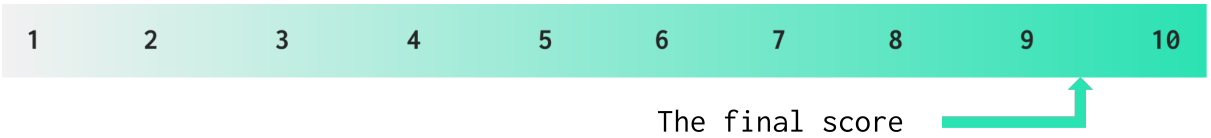

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
13 January 2023	13	4	3	1
02 February 2023	3	0	0	0
09 February 2023	0	0	0	0
08 March 2023	0	0	0	0

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

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items 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.	Not Relevant
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.	Passed
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 self-destructible while 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
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
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 required.	Passed
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.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	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 or be predictable.	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.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Not Relevant
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Not Relevant
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.	Not Relevant
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

LPStaking is a staking system allowing to stake ERC20 tokens (“LP”) for the reward in the form of another token (“Reward Token”).

Privileged roles

- The Smart Contract has an Owner role, but its usage is limited to serving as a parameter for indexing events.

Risks

- No substantial risks were identified.

Findings

■■■■ Critical

C01. Requirements Violation

The execution logic of `getBoost()` does not conform with the Technical Requirements: “If user lock below minimum they get boost of 1 (or no boost)”, while the code returns 100.

This can lead to incorrect calculations in the `_updateBeforeStakeStart()` function.

Path:
./contracts/LPStaking.sol : getBoost()

Recommendation: The `getBoost()` function should be finalized.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

■■■ High

H01. Non-Finalized Code

The code contains TODO comments. It means that the code is not finalized, and additional changes will be introduced in the future.

This can lead to incorrect implementation.

Path:
./contracts/LPStaking.sol : _updateAtStakeEnd()

Recommendation: The code should be finalized, and all TODO comments should be addressed.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

H02. Denial of Service Vulnerability

The `withdraw()` function calls the _updateAtStakeEnd() private function, which in turn makes unchecked subtractions potentially leading to Denial of Service through underflow error.

Path:
./contracts/LPStaking.sol : _updateAtStakeEnd()

Recommendation: Implement a check that the state variables in `_updateAtStakeEnd()` never get smaller than the subtraction operand.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

H03. Use of Hard-Coded Values

It is assumed that when the user makes a deposit, the `_minDays` state variable will be used to check if the `numberOfDays` parameter

is not too small; however, a hardcoded value `2` is used instead. This may possibly be a coding mistake.

Path:
./contracts/LPStaking.sol : deposit()

Recommendation: The state variable `_minDays` should be used instead of the hardcoded value, or a new constant should be declared.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

Medium

M01. Unscalable Functionality - Shadowing State Variables

State variables should not be shadowed in order to keep abstraction levels clear. In the `initialize()` function, one of the parameters is named `owner`. The parent Smart Contract `OwnableUpgradeable.sol` already has a member `owner` in the form of a function. Functions in Solidity can be addressed without the parentheses, so this can lead to confusion. The same issue is detected with the `name` and `symbol` parameters, under which name also exists a function inside the parent contract `ERC20Upgradeable.sol`.

Path:
./contracts/LPStaking.sol : initialize()

Recommendation: Rename the `owner`, `name` and `symbol` parameters.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

M02. Best Practice Violation - Unchecked Transfer

It is considered following best practices to avoid unchecked transfer functions, which can potentially lead to DoS vulnerabilities.

Path:
./contracts/LPStaking.sol : addReward(), withdraw(), deposit()

Recommendation: Follow common best practice: use the OpenZeppelin's `SafeERC20.sol` library and replace `transfer()` calls with `safeTransfer()` and `transferFrom()` to `safeTransferFrom()`.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

M03. Unscalable Functionality - Bad Variable Naming

Undocumented usage of variables named `tLP`, `sLP` and `_tLPBalanceThis` overwhelm code and makes further development difficult.

Path:
./contracts/LPStaking.sol

Recommendation: Provide variable names according to their purposes.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

M04. Contradiction - Missing Validation

According to documentation, there should be a check if `numberOfDays >= maxDays` which must always return `maxBoost` in the `getBoost()` function. This can lead to incorrect values when calling the function directly. In the code, this is mitigated by checking in the `deposit()` function.

Path:
./contracts/LPStaking.sol : getBoost()

Recommendation: Provide an additional check for condition `numberOfDays >= maxDays`.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

Low

L01. Unfinished NatSpec

It is recommended that the code should be kept clean and properly documented with NatSpec. There are multiple functions, structs, and public storage variables that are missing proper NatSpec documentation.

Path:
./contracts/LPStaking.sol

Recommendation: NatSpec documentation best practices should be followed. For reference:

https://docs.soliditylang.org/en/v0.8.17/natspec-format.html#documentation-example

https://dev.to/perelynsama/natspec-the-right-way-to-comment-ethereum-smart-contracts-1b0c

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

L02. Ignoring Solidity Time Unit Suffixes

When dealing with converting time units (exception: years) to seconds, it is generally recommended to use built-in suffixes.

Path:
./contracts/LPStaking.sol

Recommendation: Replace the usage of `_DAY_TO_SECONDS` constant to inline `days` suffix. For reference:

https://docs.soliditylang.org/en/v0.8.17/units-and-global-variables.html?highlight=days#time-units

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

L03. Style Guide Violation

Function order is incorrect, external function can not go after external view function.

Path:
./contracts/LPStaking.sol

Recommendation: Move `external` function before `external view` function. Reference:

https://docs.soliditylang.org/en/v0.8.17/style-guide.html#order-of-functions

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

L04. Unused Function Parameters

The code contains an overridden function with unused, yet named parameters.

Path:
./contracts/LPStaking.sol : _transfer()

Recommendation: Remove the names of the parameters to show explicitly that the parameters are not going to be used and are only intended to override an inherited function. For reference:

https://docs.soliditylang.org/en/v0.8.17/contracts.html#function-parameters

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

L05. NatSpec Comments Contradiction

The NatSpec comments of the function deposit() contradict the name of the parameter: the comments say `Number of weeks you want to stake` while the name of the parameter is `numberOfDays`.

Path:
./contracts/LPStaking.sol : deposit()

Recommendation: Change comments to represent the real meaning of the parameter.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

L06. Unused Variables

The constant `_EIGHTEEN_DECIMALS` is never accessed, the state variable `_rewardVesting` is never used.

Path:
./contracts/LPStaking.sol

Recommendation: Remove the state variables `_EIGHTEEN_DECIMALS` and `_rewardVesting`.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

L07. Typo in the Error Message

The error message mentions "LPStaking: Couldn't transfer LI". The term “LI” is not described in the technical docs and never appears again in the code. This is likely a typo.

Path:
./contracts/LPStaking.sol : deposit(),

Recommendation: Correct spelling of “LI” to “LP” or provide an explanation in NatSpec.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

L08. Unindexed Events

The event `LPStakingInit` has no indexed fields.

Path:
./contracts/LPStaking.sol

Recommendation: Having indexed parameters in the events makes it easier to search for these events using indexed parameters as filters.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

L09. Functions that Can Be Declared External

“public” functions that are never called by the contract should be declared “external” to save Gas.

Path:
./contracts/LPStaking.sol : getUnclaimedReward()

Recommendation: Declare the mentioned function as “external”.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

L10. Code Inconsistency - Use of Libraries

ReentrancyGuardUpgradeable should be used with upgradable contracts. With ReentrancyGuard the _status will be 0. So the first nonReentrant check will be 20k Gas more expensive.

Path:
./contracts/LPStaking.sol

Recommendation: ReentrancyGuardUpgradeable should be used instead of ReentrancyGuard.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

L11. Code Inconsistency - Use of Modifiers

According to the OpenZeppelin library, `_disableInitializers()` should be used in the constructor and not the `initializer` modifier.

Path:
./contracts/LPStaking.sol : constructor()

Recommendation: Remove `initializer`, replace it with `_disableInitializers()`.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

L12. 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.

Path:
./contracts/LPStaking.sol

Recommendation: Add SPDX License Identifier.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

L13. Excessive State Variable Access

It is not recommended to read the state at each code line. It would be much more Gas effective to store the state value into the local memory variable and use it for reading. Emitting the `LPStakingInit` event in the initializer can be done using function parameters instead of reading from the storage.

Path:
./contracts/LPStaking.sol : initializer()

Recommendation: Use function parameters `minDays`, `maxDays`, `minBoost`, `maxBoost` when emitting the `LPStakingInit` event in the initializer.

Status: Fixed
(revised commit: 78ee4cd280c3e4c1bc4f43d6f0900c586b6b715c)

L14. Functions that Can Be Declared External

“public” functions that are never called by the contract should be declared “external” to save Gas.

Path:
./contracts/LPStaking.sol : getLockedUntil(), getConfiguration()

Recommendation: Declare the mentioned function as “external”.

Status: Fixed
(revised commit: ec432b10e47e69377cad4421db93ed429af79492)

L15. Typo in Code

The constant `_MINIUM_LOCK_DAYS` contains a typo.

Path:
./contracts/LPStaking.sol : _MINIUM_LOCK_DAYS

Recommendation: Rename the constant to `_MINIMUM_LOCK_DAYS`.

Status: Fixed
(revised commit: ec432b10e47e69377cad4421db93ed429af79492)

L16. Style Guide Violation

The code violates the following style guide rule: “External functions can not go after a private function”.

Path:
./contracts/LPStaking.sol

Recommendation: Follow the official style guide:

https://docs.soliditylang.org/en/v0.8.17/style-guide.html#order-of-functions

Move “external” functions before “public” functions and “public” before “private”. Within a grouping, “view” and “pure” functions should come last.

Status: Fixed
(revised commit: ec432b10e47e69377cad4421db93ed429af79492)

L17. Style Guide Violation

The source file contains non-standard formatting of SPDX License Identifier. The license identifier should follow the standard in order to be machine readable and be supplied to bytecode metadata after deployment.

Path:
./contracts/LPStaking.sol

Recommendation: Follow the official style guide:

https://docs.soliditylang.org/en/v0.8.18/layout-of-source-files.html#spdx-license-identifier

Use the following format of SPDX License Identifier:
// SPDX-License-Identifier: UNLICENSED

Status: Fixed
(revised commit: 8c02c0c9484b73faa5b4f642ab229809c8597ee7)

L18. Style Guide Violation

The code violates the following style guide rule: “internal pure function can not go after private function”.

Path:
./contracts/LPStaking.sol

Recommendation: Follow the official style guide:

https://docs.soliditylang.org/en/v0.8.18/style-guide.html#order-of-functions

Move “internal” functions before “private” functions. Within a grouping, “view” and “pure” functions should come last.

Status: Fixed
(revised commit: 8c02c0c9484b73faa5b4f642ab229809c8597ee7)

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.