HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Blocksquare
Date: July 06th, 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 Blocksquare
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC20 token; Staking
Platform	EVM
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	https://blocksquare.io/
Timeline	07.06.2022 - 05.07.2022
Changelog	14.06.2022 - Initial Review 06.07.2022 - Second 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/oceanpoint-contracts
Commit: 457ce1d
Technical Documentation:
Type: Project documentation (partial functional requirements provided)
Link

Type: Smart contract description
Link

Integration and Unit Tests: Yes (in “test” directory)
Contracts:
File: ./contracts/BSPTStaking.sol
SHA3: 6e9331022968a64df2bc427f4cc341b2fe04d3d7c853751174c7a76471c78d95

Second review scope

Repository:
https://github.com/blocksquare/oceanpoint-contracts
Commit: 7f6cc66
Technical Documentation:

Type: Technical description
Code comments

Type: Functional requirements
Link

Integration and Unit Tests: Yes (in “test” directory)
Contracts:
File: ./contracts/BSPTStaking.sol
SHA3: 88e2d59345a4ea373bf678aca03fa31bfe87ef73ad9a27971fb5f53d5ef3953f

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

Documentation quality

The total Documentation Quality score is **8** out of **10**. Superficial functional requirements are provided in project documentation and in a brief contract overview. Code commentaries violate NatSpec.

Code quality

The total CodeQuality score is **9** out of **10**. Deployment and basic user interactions are covered with tests. Code violates the order of functions defined in the style guide.

Architecture quality

The architecture quality score is **10** out of **10**. Contract use best practices.

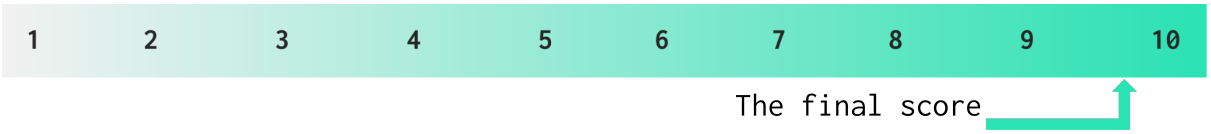
Security score

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: **9.7**.



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.	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.	Passed
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
Race	SWC-114	Race Conditions and Transactions Order	Passed

Conditions		Dependency should not be possible.	
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	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 or be predictable.	Passed
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.	Not Relevant
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.	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.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There	Passed

		should not be any cases when execution fails due to the block Gas limit.	
Style guide violation	**Custom**	Style guides and best practices should be followed.	Failed
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
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

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

System Overview

BSPTStaking is an ERC20 and staking smart contract that incentivizes users to hold tokens for rewards.

Privileged roles

The *owner* of *BSPTStaking* contract has control over functions of smart contract and is able to change lock period, reward fee, OceanPoint, DataProxy, PropertyRegistry, and VestingReward contract addresses.

Risks

- In case of an admin keys leak, an attacker can change contract addresses to malicious, resulting in a loss of funds.
- Users may not receive rewards if the owner changes the fee.
- The *property* token validation is essential to contract security.

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

Findings

Critical

No critical severity issues were found.

High

1. Highly permissive owner access

The owner can change the lock period and rewards fee. The lock period in documentation is set to 6 months, but in code can be changed by the owner. The reward fee amount is not specified in the documentation and can be changed by the owner.

This can lead to users` funds manipulation.

Contracts: BSPTStaking.sol

Functions: changeRewardFee, changeLockPeriod

Recommendation: Add highly permissive functionality to the documentation.

Status: Fixed (documentation link)

Medium

1. Unchecked call return value

The return value of a message call is not checked. Execution will resume even if the called contract throws an exception. If the call fails accidentally or an attacker forces the call to fail, this may cause unexpected behavior in the subsequent program logic.

This can lead to a loss of funds.

Contract: BSTPStaking.sol

Functions: addReward:148

Recommendation: By choosing low-level call methods, make sure to handle the possibility that the call will fail by checking the return value.

Status: Fixed (revised commit: 7f6cc66)

Low

1. Floating Pragma

Contracts files use floating pragma `>=0.8.0<0.9.0;`

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Contracts: BSPTStaking.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed (revised commit: 7f6cc66)

2. **Gas optimization**

   Using inefficient code makes running the transaction more expensive.

   Contracts: BSPTStaking.sol

   Functions: _updateBeforeStakestart:68, deposit:104, 106-109

   Recommendation: Precompute the divisor

   Status: Fixed (revised commit: 7f6cc66)

3. **Gas optimization**

   The difference between variables balanceBeforeTransfer and balanceAfterTransfer is equal to the function argument *amount.*

   This leads to rudimentary code and higher Gas expenses.

   Contracts: BSPTStaking.sol

   Functions: deposit:201-208

   Recommendation: Use *amount* variable instead of loading *beforeTransfer* and *afterTransfer* balances into memory and subtracting them.

   Status: Mitigated (with customer notice)

4. **Functions can be declared as external**

   To save Gas, public functions that are never called in the contract should be declared as external.

   Contracts: BSPTStaking.sol

   Functions: changeOceanPointContract, changeVestingRewardContract

   Recommendation: Declare mentioned functions as external.

   Status: Fixed (revised commit: 7f6cc66)

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.