HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: uStaking
Date: April 03, 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 uStaking
Approved By	Yevheniy Bezuhlyi | SC Audits Head at Hacken OU
Type	BEP20 token; Staking
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://ustaking.io/
Changelog	01.03.2023 - Initial Review 14.03.2023 - 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 uStaking (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 review and security analysis of smart contracts in the repository:

Initial review scope

Repository	https://github.com/Ustaking/Contracts
Commit	c0c2f7daba519c0797d5ba1e7676a55cc77873a1
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/UStaking.sol SHA3: 2d4a1c1ded72975684819c017b4ab2a22716c3ec316ffbc95ea3a3e15d85fa7f File: ./contracts/UStakingToken.sol SHA3: c8c409bf4c5f3c7bf5c5c7ea3788794a73ef82dbff1e30e07f9fa73c1a7baaf0

Second review scope

Repository	https://github.com/Ustaking/Contracts
Commit	5d17ba8ca4e450490f25a4cb73b249eb8a61ebba
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/UStaking.sol SHA3: ae62be26ba4b136ebaab7eaa2cde7fbdab2d395c5cf30c2a9121de524994a254 File: ./contracts/UStakingToken.sol SHA3: 985c24adf2967117ce440d7a51a7a0b1f4bbaec415707b7a25aaf633a1966516

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

Third review scope

Repository	https://github.com/Ustaking/Contracts
Commit	31b93f58e86d056557992117897b460a688d1762
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/UStaking.sol SHA3: 3bddf3bc5f06b76f8f8aa5d3d28090beb2db35c39b4a4e7d1f7ce9552f0b67fe File: ./contracts/UStakingToken.sol SHA3: 985c24adf2967117ce440d7a51a7a0b1f4bbaec415707b7a25aaf633a1966516

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

- Functional requirements are provided.
- Technical description is provided.
- NatSpec for UStaking and UStaking is provided.

Code quality

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

- No Solidity Style Guide violations are present.
- Code follows best practices.
- Development environment is provided.

Test coverage

Code coverage of the project is **100%** (branch coverage).

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: **10.0**.

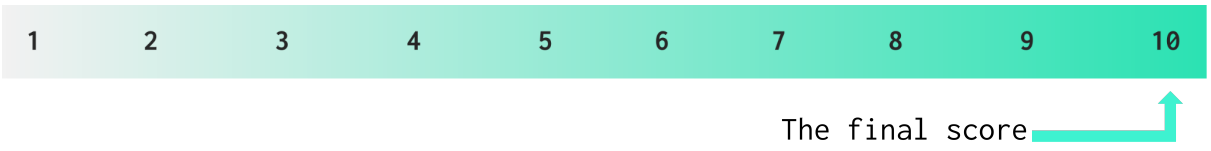

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
28 February 2023	8	5	1	0
14 March 2023	3	0	0	0
03 April 2023	0	0	0	0

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

System Overview

uStaking is a mixed-purpose system with the following contracts:

- *UStakingToken* — simple ERC-20 token that mints all initial supply to a deployer. Additional minting is allowed to an address with a MINTER_ROLE. There is an additional administrative role called PAUSER_ROLE, addresses with this role can pause or unpause the transfer of tokens. There is a set token max supply (2000000000).
- *UStaking* — a contract that rewards users for staking their tokens. Users can choose the duration of staking, and a longer duration equals a bigger profit per day.

Privileged roles

- PAUSER_ROLE is the only role with a permission to pause and unpause transfers of the uStakingToken.
- MINTER_ROLE is the only role with permission to mint new uStakingToken.
- DEFAULT_ADMIN_ROLE is the only role with permission to grant and revoke other roles to users of uStakingToken.

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.	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.	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.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless 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.	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.	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

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

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 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 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.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

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

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Funds Lock

Native coins and tokens should have mechanisms for their withdrawal if they are accepted by the contract. The contract has a *receive() function* but there isn't a function to withdraw native tokens.

Path: ./contracts/UStaking.sol: *

Recommendation: Delete the *receive()* function or create a mechanism for the withdrawal of native tokens.

Found in: c0c2f7da

Status: Fixed (5d17ba8)

■■ Medium

M01. Same checks in functions

The same check *require(isContains(msg.sender,stakeId), "invalid stake id");* is used in several functions, overwhelming the code and making further development difficult.

Path: ./contracts/UStaking.sol : claim(),

./contracts/UStaking.sol : cashBack(),

./contracts/UStaking.sol : withdraw(),

Recommendation: Move the check to a special modifier. Delete *isContains()* and move the logic to the newly created modifier.

Found in: c0c2f7da

Status: Fixed (5d17ba8)

M02. SafeMath in ^0.8.0

Starting with Solidity ^0.8.0, SafeMath functions are built-in. This makes the library redundant.

Path: ./contracts/UStaking.sol

Recommendation: Remove the redundant functionality.

Found in: c0c2f7da

Status: Fixed (ada2b59)

M03. Requirements Compliance

The whitepaper states that the initial circulation will be 150 million uStakingTokens, but in the constructor of uStakingToken only 79 million are minted.

Path: ./contracts/UStakingToken.sol : function constructor()

Recommendation: UStakingToken should mint 150 million in its constructor.

Found in: c0c2f7da

Status: Fixed (5d17ba8)

M04. Copypasting Of Template Contracts.

The contract contains copies of OZ contracts that can be imported instead.

Path: ./contracts/UStaking.sol

Recommendation: import templates and libraries instead of copying them.

Found in: c0c2f7da

Status: Fixed (5d17ba8)

M05. Checks-Effects-Interactions Pattern Violation

The code violates the Checks-Effects-Interactions pattern. This lowers code quality and can lead to reentrancy attacks.

Path: ./contracts/UStaking.sol : claim(),

./contracts/UStaking.sol : stake(),

./contracts/UStaking.sol : withdraw(),

Recommendation: Refactor the code to fit the Checks-Effects-Interactions pattern.

Found in: c0c2f7da

Status: Fixed (5d17ba8)

Low

L01. Floating Pragma

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.

Path: ./contracts/UStaking.sol,

./contracts/UStakingToken.sol

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

Found in: c0c2f7da

Status: Fixed (5d17ba8)

L02. Missing Zero Address Validation

Address parameters are used without checking against the possibility of 0x0. This can lead to unwanted external calls to 0x0.

Path: ./contracts/UStaking.sol : constructor(),

./contracts/UStaking.sol : refWalletUpdate()

Recommendation: Implement zero address checks.

Found in: c0c2f7da

Status: Fixed (5d17ba8)

L03. Address instead of interface

When a function takes a contract address as an argument, it is better to pass an interface or contract type rather than a raw address. If the function is called elsewhere within the source code, the compiler will provide additional type safety guarantees.

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

Recommendation: Change the address argument to an IBEP20 type in the function argument declaration.

Found in: c0c2f7da

Status: Fixed (5d17ba8)

L04. Style Guide Violation - Naming Conventions

Contract code should follow the official guidelines.

Structs should be named using the CapWords style.

Events should be named using the CapWords style.

Local Variable Names should be named using the mixedCase style.

Path: ./contracts/UStaking.sol : struct user{},

./contracts/UStaking.sol : struct viewStore{},

./contracts/UStaking.sol : event stakeEvent(),

./contracts/UStaking.sol : event withdrawEvent(),

./contracts/UStaking.sol : event claimEvent(),

./contracts/UStakingToken.sol : MaxSupply,

Recommendation: Transform code to fit Solidity Naming Conventions.

Found in: c0c2f7da

Status: Fixed (5d17ba8)

L05. State variables can be declared immutable

Compared to regular state variables, the gas costs of constant and immutable variables are much lower. Immutable variables are evaluated once at construction time and their value is copied to all the places in the code where they are accessed.

Path: ./contracts/UStaking.sol : token,

Recommendation: Declare mentioned variables as immutable.

Found in: c0c2f7da

Status: Fixed (5d17ba8)

L06. State variables can be declared constant

State variables that do not change their value should be declared constant to save Gas.

Path: ./contracts/UStaking.sol : cashBackReward,

./contracts/UStaking.sol : refWalletReward,

./contracts/UStakingToken.sol : MaxSupply,

Recommendation: Declare the above-mentioned variables as constants.

Found in: c0c2f7da

Status: Fixed (5d17ba8)

L07. Style Guide Violation - Order of Layout

Contract code should follow the official guidelines.

Contracts implementation should be below libraries.

Structs (type declaration) should be in first place inside the contract.

The Receive function should be below constructor implementation.

External and public functions should be grouped and external functions should be put before public functions.

Path: ./contracts/UStaking.sol : contract Ownable {},

./contracts/UStaking.sol : struct user{},

./contracts/UStaking.sol : struct viewStore{},

./contracts/UStaking.sol : receive(),

Recommendation: Declare the above-mentioned variables as constants.

Found in: c0c2f7da

Status: Fixed (5d17ba8)

L08. Functions that can be declared external

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

Path: ./contracts/UStaking.sol : refWalletUpdate(),

./contracts/UStaking.sol : cashBack(),

./contracts/UStaking.sol : stakeTotalIds(),

./contracts/UStaking.sol : getUserData(),

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

Found in: c0c2f7da

Status: Fixed (5d17ba8)

L09. Style Guide Violation - Naming Conventions

Contract code should follow the official guidelines.

Constants should be named using the UPPER_CASE_WITH_UNDERSCORES style.

Path: ./contracts/UStaking.sol : cashBackReward,

./contracts/UStaking.sol : refWalletReward,

Recommendation: Change a name of constant values to fit the UPPER_CASE_WITH_UNDERSCORES style.

Found in: 5d17ba8

Status: Fixed (31b93f5)

L10. State variables default visibility

The visibility of the variable *token* is not specified. Specifying state variables visibility helps to catch incorrect assumptions about who can access the variable.

This makes the contract`s code quality and readability higher.

Path: ./contracts/UStaking.sol : token

Recommendation: Specify variables as public, internal, or private. Explicitly define visibility for all state variables.

Found in: 5d17ba8

Status: Fixed (31b93f5)

L11. Missing event

Events for critical state changes should be emitted for tracking things off-chain.

Path: ./contracts/UStaking.sol : cashBack()

Recommendation: Create and emit related events.

Found in: 5d17ba8

Status: Fixed (31b93f5)

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 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.