HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: TrustSwap
Date: April 28th, 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 TrustSwap.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	Staking contract
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://trustswap.com/
Timeline	14.04.2022 - 28.04.2022
Changelog	19.04.2022 - Initial Review 28.04.2022 - 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 TrustSwap (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/trustswap/locked-staking-contracts/tree/no-auto-comp
Commit:
3da39e219617d87ea8550cd8d152c836aa66e0be
Technical Documentation: Yes
(https://drive.google.com/file/d/1QDDsxA55K3mQ2NsOlMlyTpuyTW1gj0sm/view)
JS tests: Yes
(https://github.com/trustswap/locked-staking-contracts/tree/no-auto-comp/src/test)
Contracts:
File: ./src/contracts/LockedStaking.sol
SHA3: 04a7ddefd5af7b8846919556fe05a29248fae1b45125deab7ccb332414bbd1b5

Second review scope

Repository:
https://github.com/trustswap/locked-staking-contracts/tree/master
Commit:
cbfc31196d79f5dbff5a4fd70e66275c41671ab4
Technical Documentation: Yes
(https://drive.google.com/file/d/1QDDsxA55K3mQ2NsOlMlyTpuyTW1gj0sm/view)
JS tests: Yes
(https://github.com/trustswap/locked-staking-contracts/tree/no-auto-comp/src/test)
Contracts:
File: ./src/contracts/LockedStaking.sol
SHA3: cf685b6d9c0337e31c4c37eef9cbb8bab87ddb6c92004b278ca2d0d659402ea2

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

Documentation quality

The Customer provided superficial functional requirements and technical requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total Code Quality score is **10** out of **10**. The code follows official language style guides. Unit tests were provided.

Architecture quality

The architecture quality score is **10** out of **10**. The project has clear and clean architecture.

Security score

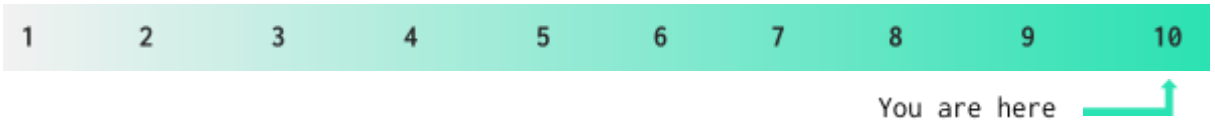
As a result of the audit, security engineers found **2** high, **2** medium, and **4** low severity issues. The security score is **0** out of **10**.

As a result of the second review, security engineers found no new issues. **2** high, **2** medium and **3** low issues from the previous revision were fixed. As a result, the code contains **1** low issue. 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**



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.	Not Relevant
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 destroyed until it has funds belonging to users.	Not Relevant
Reentrancy	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

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	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.	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.	Not Relevant
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.	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.	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.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with requirements provided by the Customer,	Passed
Repository Consistency	**Custom**	The repository 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. Tests coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed

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

System Overview

TrustSwap staking is a staking project with the following contract:

- Staking – a contract that rewards users for staking their tokens. APY depends on the duration of the lock period and the size of the reward pool.

Privileged roles

- The owner of the TrustSwap staking contract can add Swap tokens for rewards, update the reward period and transfer the ownership of the contract.

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. **Unexpected reward multiplier.**

   The contract has the function `getDurationMultiplier`, which calculates the reward multiplier based on the duration. The function has some conditional statements for specific duration values, which affect the calculated return value. The `duration` is an `uint256` argument that represents the duration in seconds. In some cases, the longer duration may lead to a lower multiplier. For example:

   1) if the duration is equal to `15552000` (180 days), the multiplier is `150`.
   2) if the duration is equal to `15638400` (181 days), the multiplier is `139`.

   Some users may get fewer tokens if they set a specific bigger `duration` value than other users.

   Contracts: LockedStaking.sol

   Function: getDurationMultiplier

   Recommendation: Remove the conditional statements for specific periods or specify the more profitable hardcoded periods in the public documentation.

   Status: Fixed (d7b62b2566208b228ecf19340b77e2706095e58c)

2. **Exposed private key.**

   The repository contains an exposed private key, which may be used during the contract deployment. If the repository is public, anybody will get access to the deployer account and intercept the ownership of the contract.

   If the private key is exposed, the contract ownership may be intercepted.

   File: deployDev.js

   Recommendation: Do not store private keys in the repository, all the keys should be stored in a special `.env` file.

   Status: Fixed (d7b62b2566208b228ecf19340b77e2706095e58c)

Medium

1. **Checks-Effects-Interactions pattern violation.**

   The state variables are updated after competition result data has been gathered from the oracle.

The state variables are updated after competition creation and configuration has made.

This can lead to unexpected behaviors in the function execution.

Contracts: LockedStaking.sol

Function: addReward, updateReward, addLock, updateLockAmount, updateLockDuration

Recommendation: Update state variables before making external calls.

Status: Fixed (a8d5f2b18816951a38fbbb102cf3822ac603e3c0)

2. **Wrong function argument.**

   The contract has the internal function `calculateUserClaimable`, which calculates the claimable amount of tokens for a specific `user` address. During the calculations the function operates the `userLastAccRewardsWeight[msg.sender]` value. This does not cause an error because, in the LockedStaking contract, the `calculateUserClaimable` function is only called when the `user` argument is equal to `msg.sender`, but this is dangerous for the contracts which may potentially inherit the LockedStaking.

   This may cause a claimable amount calculation error in the contracts, which will inherit the internal `calculateUserClaimable` function.

   Contracts: LockedStaking.sol

   Function: calculateUserClaimable

   Recommendation: Update the function to replace the `msg.sender` with a `user` address value.

   Status: Fixed (11c9c1d8703dd8356d82f8c68bd1a10650b837e3)

Low

1. **Implicit call to ERC-20 token.**

   The contract has the `unlock` function, which transfers the tokens to the user. The token instance variable is not wrapped with an `IERC20` explicitly.

   This may be confusing for developers during smart contract development.

   Contracts: LockedStaking.sol

   Function: unlock

   Recommendation: It is recommended to explicitly wrap the `swapToken` variable with an `IERC20` on `transfer` call.

   Status: Fixed (a8d5f2b18816951a38fbbb102cf3822ac603e3c0)

2. **Declaration of the popular math function.**

   The contract declares the popular pure math functions, such as `min`, `max`.

This may lead to a redundant use of gas during the contract deployment.

Contracts: LockedStaking.sol

Function: min, max

Recommendation: It is recommended to use the library for popular math functions.

Status: Reported

3. **Missing zero address validation.**

   Address parameters are being used without checking against the possibility of 0x0.

   This can lead to unwanted external calls to 0x0.

   Contracts: LockedStaking.sol

   Function: constructor

   Recommendation: Add a require or conditional statement to check for zero address.

   Status: Fixed (b0f0d1a76806af19fa046dd71a88ceb2dd9208ed)

4. **Floating pragma.**

   The project uses floating pragma ^0.8.0.

   Contracts: LockedStaking.sol

   Function: -

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

   Status: Fixed (d7b31ff4c9951f0121892fe560d4b50f18ba3f33)

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.