SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: TrustSwap

Date: February 2nd, 2022

This document 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 containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed – upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for TrustSwap.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Vesting
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/trustswap/team-finance-contracts
Commit	8a18a0f7bc3df519236145b7375efe08d94fb192
Technical Documentation	NO
JS tests	NO
Website	https://trustswap.com/
Timeline	19 JANUARY 2022 - 02 FEBRUARY 2022
Changelog	02 FEBRUARY 2022 - INITIAL AUDIT

Table of contents

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 contract and its code review conducted between January 19th, 2022 - February 2nd, 2022.

Scope

The scope of the project is smart contracts in the repository:

Repository:
https://github.com/trustswap/team-finance-contracts
Commit:
8a18a0f7bc3df519236145b7375efe08d94fb192
Technical Documentation: No
JS tests: No
Contracts:
IERC20Extended.sol
IPriceEstimator.sol
LockToken.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are secured.

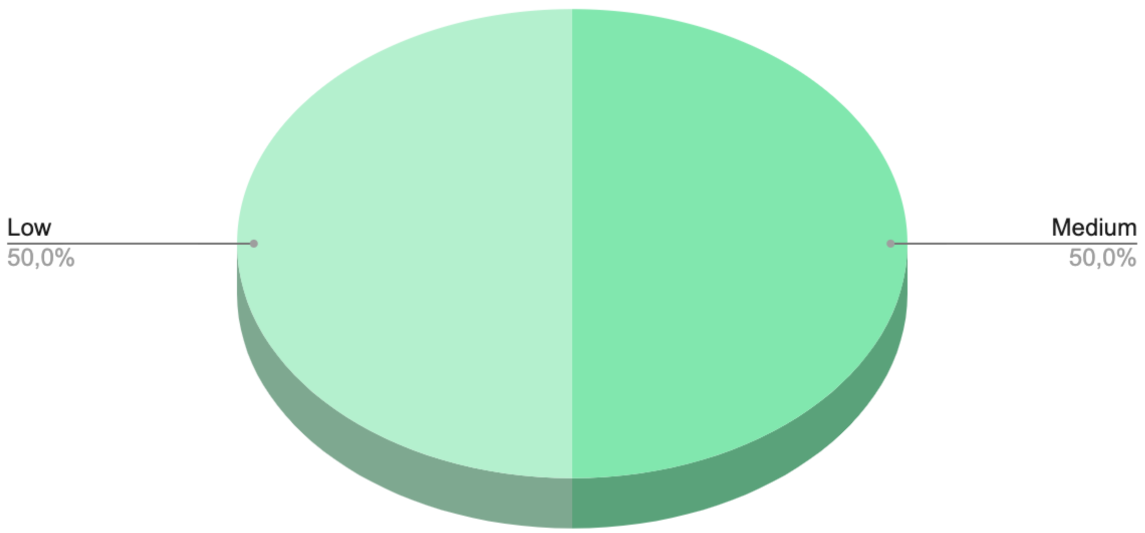
Insecure	Poor secured	Secured	Well-secured

You are here

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found **2** medium and **2** low severity issues.

Graph 1. The distribution of vulnerabilities after the audit.



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 can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution

Audit overview

🟥 🟥 🟥 🟥 Critical

No critical issues were found.

🟨 🟨 🟨 High

No high severity issues were found.

🟩 🟩 Medium

1. Calls inside the loop.

In the specified function there is a loop that continuously asks for a balance and transfers the same token from the sender to the contract address.

Contract: LockToken.sol

Functions: createMultipleLocks

Recommendation: It would be much more sufficient to get the balance once before the loop, then in the loop just sum all amounts and after the loop execute the only one transferFrom call. If you still need multiple transferFrom calls (i.e. for events emitting) please consider still having a balance as the local variable, not to call for it twice per loop.

2. Costly operations inside the loop.

In the specified function there is a loop that continuously updates state variables in the loop.

Contract: LockToken.sol

Functions: createMultipleLocks

Recommendation: It would be much more sufficient to get state variables into the memory local variables, update them in the loop and store them to the state after the loop.

🟦 Low

1. Unused variable.

Both functions are saving the result of the ETH refund to the local variable which is never used.

Contract: LockToken.sol

Functions: lockTokens, createMultipleLocks

Variable: refundSuccess

Recommendation: Remove unused variables.

2. Duplicate code.

Both functions are calculating ETH fees using the same code duplicated in both functions.

Contract: LockToken.sol

Functions: lockTokens, createMultipleLocks

Recommendation: To keep the code clean, readable, and to be sure both functions are calculating the same, please move the duplicated code to some private function and call it from both.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

As a result of the audit, security engineers found **2** medium and **2** low severity issues.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to 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 as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. 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 can't guarantee the explicit security of the audited smart contracts.