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/swap-bonding-contract
Commit	219917c632c70585b69c6117b30aab56b791e329
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:

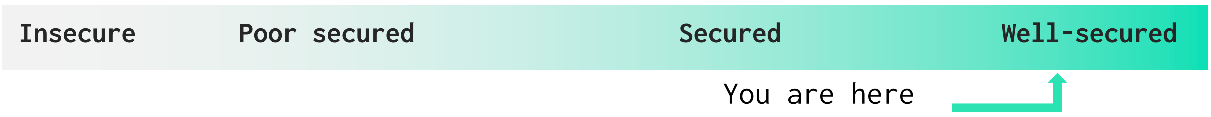
Repository:
https://github.com/trustswap/swap-bonding-contract
Commit:
219917c632c70585b69c6117b30aab56b791e329
Technical Documentation: No
JS tests: No
Contracts:
BondDepository.sol
library/FullMath.sol
TrustSwapAuthority.sol
interface/ITreasury.sol
BondingCalculator.sol
TetherToken.sol
library/TrustSwapAccessControlled.sol
interface/IUniswapV2Pair.sol
interface/IERC20.sol
interface/IERC20Permit.sol
interface/ITrustSwapAuthority.sol
external/ERC20.sol
external/ReentrancyGuard.sol
library/SafeMath.sol
interface/IBondingCalculator.sol
interface/IERC20Extended.sol
external/Context.sol
external/Ownable.sol
Treasury.sol
library/FixedPoint.sol
library/Address.sol
interface/IERC20Metadata.sol
interface/IUniswapV2ERC20.sol
SwapToken.sol
library/SafeERC20.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 well-secured.



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 **3** low severity issues.

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

No medium severity issues were found

🟦 Low

1. Block timestamp.

Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners within 15 minutes.

Contract: SwapBondDepository

Functions: initializeBondTerms, setAdjustment, deposit, adjust, bondPrice, percentVestedFor

Recommendation: Please consider relying on the block.number instead.

2. Boolean equality.

Boolean constants can be used directly and do not need to be compared to **true** or **false**.

Contract: TrustSwapTreasury

Functions: enable, queueTimelock, execute, disableTimelock, initialize

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.

3. State variables that could be declared constant.

Constant state variables should be declared constant to save gas.

Contract: TrustSwapTreasury

Variables: UNAUTHORIZED, notAccepted, notApproved, invalidToken, insufficientReserves

Recommendation: Add the **constant** attribute to state variables that never change.

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 **3** 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.