

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT



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 - Initial Audit
Approved by	Andrew Matiukhin CTO Hacken OÜ
Type	ERC20 token with factory
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Deployed smart contract	https://ropsten.etherscan.io/address/0xf94413cf315cb461637be61c2b9cf2a4b457a466#code
Timeline	12 MAY 2021 - 13 MAY 2021
Changelog	13 MAY 2021 - INITIAL AUDIT



Table of contents

Introduction	4
Scope	4
Executive Summary	6
Severity Definitions	7
Audit overview	8
Conclusion	9
Disclaimers	10

Introduction

Hacken OÜ (Consultant) was contracted by TeamToken (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contract and its code review conducted on May 13th, 2021.

Scope

The scope of the project is the smart contracts deployed in the ropsten test net:

<https://ropsten.etherscan.io/address/0xf94413cf315cb461637be61c2b9cf2a4b457a466#code>

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

Category	Check Item
Code review	<ul style="list-style-type: none">▪ 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

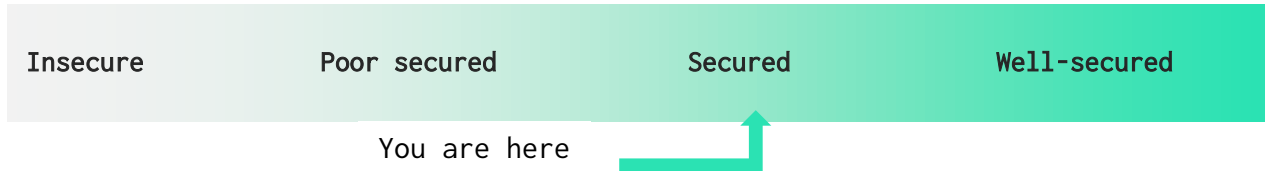


Functional review

- Business Logics Review
- Functionality Checks
- Access Control & Authorization
- Escrow manipulation
- Token Supply manipulation
- Asset's integrity
- User Balances manipulation
- Kill-Switch Mechanism
- Operation Trails & Event Generation

Executive Summary

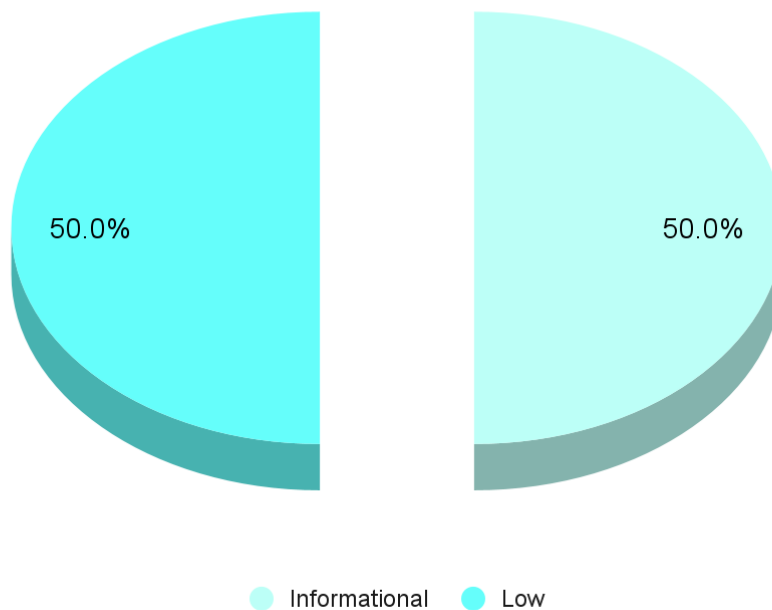
According to the assessment, the Customer's smart contracts are secured



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.

Security engineers found 1 low and 1 informational issue during the first review.

Graph 1. The distribution of vulnerabilities after the first review.



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
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Audit overview

■ ■ ■ ■ Critical

No Critical severity issues were found.

■ ■ ■ High

No High severity issues were found.

■ ■ Medium

No Medium severity issues were found.

■ Low

1. Vulnerability: Missing zero address validation

Lack of the zero-address validation in the `setFeeWallet` method could potentially lead to the issue when you're minting 0.01 tokens to zero-address

■ Lowest / Code style / Best Practice

1. Vulnerability: Public function that could be declared external

`public` functions that are never called by the contract should be declared `external` to save gas.

Lines: TeamTokenFactory.sol#16

```
function setFeeWallet(address _feeWallet) public onlyOwner{
```




Conclusion

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

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

Security engineers found 1 low and 1 informational issue during the first review.

Category	Check Items	Comments
Code Review	Style guide violation	→ public function that could be declared external → lack of zero-address validation



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 security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree 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 security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.