HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: FishCrypto
Date: February 10th, 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 FishCrypto.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 token; Transfer controller
Platform	Binance Smart Chain / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/fishcryptoio/smart-contract
Commit	4219441F1656BFF5FADC2823F03F2B29F3C0A383
Deployed contract	https://bscscan.com/address/0x29cabf2a1e5de6f0ebc39ca6fe83c687fe90fb6c
Technical Documentation	YES
JS tests	YES
Website	https://fishcrypto.io
Timeline	31 JANUARY 2022
Changelog	10 FEBRUARY 2022 – INITIAL AUDIT

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by FishCrypto (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 on February 10th, 2022.

Scope

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

Repository:
https://github.com/fishcryptoio/smart-contract
Commit:
4219441F1656BFF5FADC2823F03F2B29F3C0A383
Technical Documentation: Yes (https://whitepaper.fishcrypto.io)
JS tests: Yes (https://github.com/fishcryptoio/smart-contract/tree/master/test)
Contracts:
FICOERC20.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

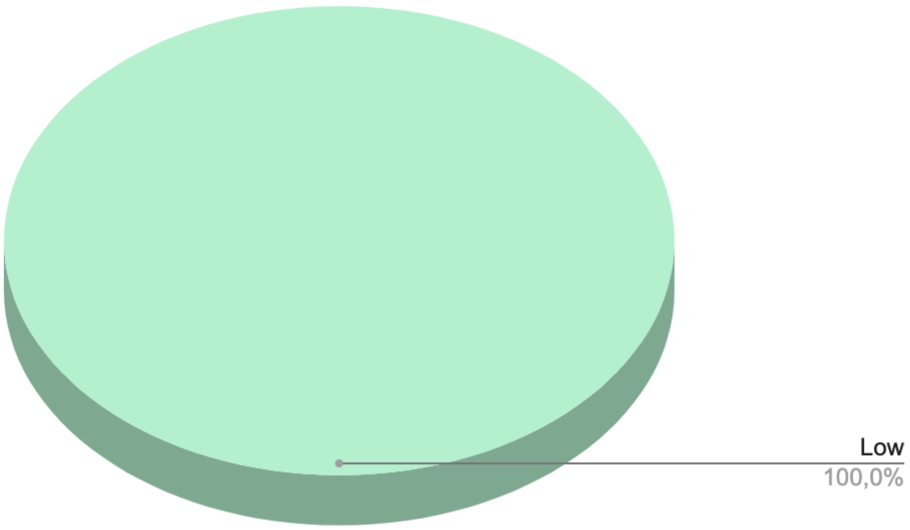
According to the assessment, the Customer's smart contracts are well-secured.

Insecure
Poor secured
Secured
Well-secured
You are here

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril, SmartCheck, Solgraph, 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 **4** low severity issue.

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

No medium severity issues were found.

🟦 Low

1. Files in “*utils*” folder (*Address.sol* and *Strings.sol*) are not used at all).

   Recommendation: delete them.

2. Unnecessary SafeMath usage.

   Solitidy >= 0.8.0 provides errors for buffer overflow and underflow. No need to use SafeMath anymore.

   Recommendation: Do not use SafeMath.

3. Duplicated variable names

   The contract has a variable called *owner* which represents the contract owner’s address. Besides the *owner* variable is used, for example, in *_approve* function and it means the funds owner but not contract owner. Also, *nonces* function has owner param which actually represents the *msg.sender*.

   Recommendation: Do not duplicate variable names. It leads to ambiguous meaning and complicates code understanding.

4. Variable names do not fit Solidity code style

Solidity recommends using *UPPER_CASE_WITH_UNDERSCORES* for constants and *mixedCase* for other variables.

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 **4** low severity issues.

Due to the fact that the contract is already deployed and issues are not severe enough - there it's not reasonable to fix them. You can take them into account for the future.

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.