



# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**CUSTOMER:** FishCrypto

**DATE:** May 17th, 2022

## Disclaimers

### **DAudit Disclaimer**

The smart contracts submitted for audit were examined in accordance with best industry practices at the time of this report in terms of cybersecurity vulnerabilities and issues in smart contract source code, which are detailed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no claims or guarantees about the code's security. It also cannot be deemed an adequate appraisal of the code's utility and safety, bug-free status, or any other contractual assertions. While we did our best in completing the study and generating this report, it is crucial to emphasize that you should not rely only on this report; we advocate doing many independent audits and participating in a public bug bounty program to assure smart contract security.

### **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.

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

<b>Name</b>	Smart Contract Code Review and Sercurity Analysis Report for FishCrypto
<b>Approved by</b>	Hiep Le Quang - CTO DAudit
<b>Type</b>	ERC20 token; Transfer controller
<b>Platform</b>	Binance Smart Chain / Solidity
<b>Methods</b>	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
<b>Repository</b>	<a href="https://github.com/fishcryptoio/smart-contract">https://github.com/fishcryptoio/smart-contract</a>
<b>Comit</b>	4219441F1656BFF5FADC2823F03F2B29F3C0A383
<b>Deployed contract</b>	<a href="https://bscscan.com/address/0x29cabf2a1e5de6f0ebc39ca6fe83c687fe90fb6c">https://bscscan.com/address/0x29cabf2a1e5de6f0ebc39ca6fe83c687fe90fb6c</a>
<b>Technical Documentation</b>	Yes
<b>JS tests</b>	Yes
<b>Website</b>	<a href="https://fishcrypto.io">https://fishcrypto.io</a>
<b>Timeline</b>	10 MAY 2022
<b>Changelog</b>	17 MAY 2022 – INITIAL AUDIT

## Table of content

Disclaimers	2
Introduction	5
Scope	5
Executive Summary	6
Audit overview	9
Conclusion	10

## Introduction

DAudit.org (Consultant) was contracted by FishCrypto (Customer) to conduct a Smart Contract Code Review and Security Analysis. FishCrypto (Customer) hired DAudit.org (Consultant) to do a Smart Contract Code Review and Security Analysis. This report details the conclusions of the Customer's smart contract security assessment and code review, which took place on May 17th, 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 items
Code review	<ul style="list-style-type: none"><li>▪ Reentrancy</li><li>▪ Ownership Takeover</li><li>▪ Timestamp Dependence</li><li>▪ Gas Limit and Loops</li><li>▪ DoS with (Unexpected) Throw</li><li>▪ DoS with Block Gas Limit</li><li>▪ Transaction-Ordering Dependence</li></ul>

	<ul style="list-style-type: none"> <li>▪ Style guide violation</li> <li>▪ Costly Loop</li> <li>▪ ERC20 API violation</li> <li>▪ Unchecked external call</li> <li>▪ Unchecked math</li> <li>▪ Unsafe type inference</li> <li>▪ Implicit visibility level</li> <li>▪ Deployment Consistency</li> <li>▪ Repository Consistency</li> <li>▪ Data Consistency</li> </ul>
Functional review	<ul style="list-style-type: none"> <li>▪ Business Logics Review</li> <li>▪ Functionality Checks</li> <li>▪ Access Control &amp; Authorization</li> <li>▪ Escrow manipulation</li> <li>▪ Token Supply manipulation</li> <li>▪ Assets integrity</li> <li>▪ User Balances manipulation</li> <li>▪ Data Consistency manipulation</li> <li>▪ Kill-Switch Mechanism</li> <li>▪ Operation Trails &amp; Event Generation</li> </ul>

## Executive Summary

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

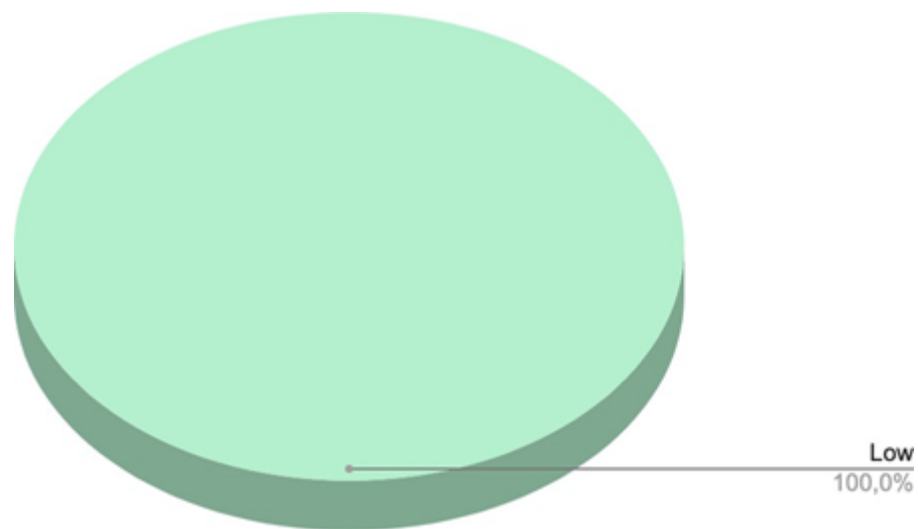


You are here 

With Mythril, SmartCheck, Solgraph, and Slither, DAudit did a code analysis, manual audit, and automated checks. All concerns discovered during automated analysis were carefully examined, and the Audit summary section contains critical vulnerabilities. The audit summary section contains a list of all problems discovered.

Security engineers discovered four low-severity problems as a result of the audit.

**Graph 1. The distribution of vulnerabilities after the audit.**



## Severity Definitions

Risk Level	Description
<b>Critical</b>	Critical vulnerabilities are usually straightforward to exploit and can lead to asset loss or data manipulations.
<b>High</b>	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
<b>Medium</b>	Medium-level vulnerabilities are important to fix; however, they can't lead to asset loss or data manipulations.
<b>Low</b>	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 critical issues were found.

### Medium

No critical 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.

Solidity  $\geq$  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 lows** 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.