



**RD
AUDITORS**

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Yearn Classic Finance
Prepared on: 15/04/2021
Platform: Binance Smart Chain
Language: Solidity

TABLE OF CONTENTS

Document	4
Introduction	4
Project Scope	4
Executive Summary	6
Code Quality	7
Documentation	8
Use of Dependencies	8
AS-IS Overview	8
Severity Definitions	11
Audit Findings	12
Conclusion	13
Our Methodology	14
Disclaimers	16

THIS DOCUMENT MAY CONTAIN CONFIDENTIAL INFORMATION ABOUT ITS SYSTEMS AND 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 DECISION OF CUSTOMER.

Document

Name	Smart Contract Code Review and Security Analysis Report for Yearn Classic Finance
Platform	BSC / Solidity
File	EarnX.sol
MD5 hash	34495574F16F54A0F6A006405AAD24BE
SHA256 hash	91E8679C42C6AFDA0CD270297D75C4DABCDD0C5E2C8AE E9FA1600B1F44B51FF5
Date	15/04/2021

Introduction

RD Auditors (Consultant) was contracted by Yearn Classic Finance (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contracts and its code review conducted between April 10, 2021 – April 15, 2021.

This contract consists of a single file.

Project Scope

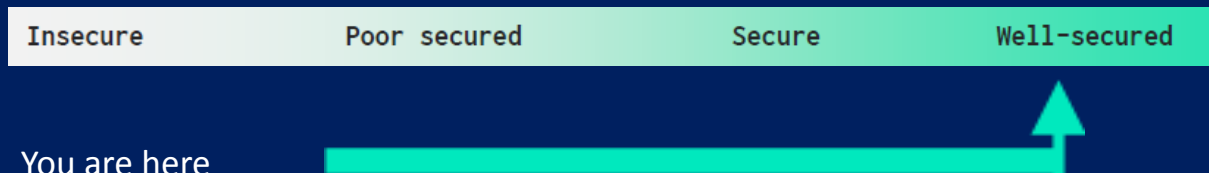
The scope of the project is a smart contract.

We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, the customer's solidity smart contract is **well secured**.



Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low and 0 very low level issues.

Code Quality

EarnX consists of a single smart contract file. This single file smart contract also contains safeMath and Address from the popular open source.

The libraries in EarnX are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Earnex.

EarnX has **not** provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is well commented. Commenting provides rich documentation for functions, return variables and more and also helps auditors to quickly cover the flow behind code logic. Use of Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given EarnX in the form of the link:

<https://www.bscscan.com/address/0x9C28E48Ea9170d9A9cB76d338655b8bd4124BC7A#code>

The hash of that file is mentioned in the table. As mentioned, It's well commented smart contract code, so anyone can quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects. And even core code blocks are written well and systematically.

AS-IS Overview

EarnX Overview

Earnx is a token contract which provides earning over investment in connection with uniswap protocol using pairing, liquidity etc.

File And Function Level Report

Contract: Ownable

Inherit: Context

Observation: Passed

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	owner	read	Passed	All Passed	No Issue	Passed
2	renounceOwnership	write	Passed	All Passed	No Issue	Passed
3	transferOwnership	write	Passed	All Passed	No Issue	Passed
4	geunlocktime	read	Passed	All Passed	No Issue	Passed
5	lock	write	Passed	All Passed	No Issue	Passed
6	unlock	write	Passed	All Passed	No Issue	Passed

Contract: EarnX

Inherit: Context , IBSC20, Ownable

Observation: Passed

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	read	Passed	All Passed	No Issue	Passed
2	symbol	read	Passed	All Passed	No Issue	Passed
3	decimals	read	Passed	All Passed	No Issue	Passed
4	totalSupply	read	Passed	All Passed	No Issue	Passed
5	balanceOf	read	Passed	All Passed	No Issue	Passed
6	transfer	write	Passed	All Passed	No Issue	Passed
7	allowance	read	Passed	All Passed	No Issue	Passed
8	approve	write	Passed	All Passed	No Issue	Passed
9	transferFrom	write	Passed	All Passed	No Issue	Passed
10	increaseAllowance	write	Passed	All Passed	No Issue	Passed
11	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
12	isExcludedFromReward	read	Passed	All Passed	No Issue	Passed
13	totalFees	read	Passed	All Passed	No Issue	Passed
14	deliver	write	Passed	All Passed	No Issue	Passed
15	reflectionFromToken	read	Passed	All Passed	No Issue	Passed
16	TokenFromReflection	read	Passed	All Passed	No Issue	Passed
17	excludeFromReward	write	Passed	All Passed	No Issue	Passed
18	includeInReward		Passed	All Passed	No Issue	Passed
19	_transferBothExcluded	write	Passed	All Passed	No Issue	Passed
20	excludeFromFee	write	Passed	All Passed	No Issue	Passed
21	includeInFee	write	Passed	All Passed	No Issue	Passed
22	setTaxFeePercent	write	Passed	All Passed	No Issue	Passed
23	SetNumTokensSellToAddToLiquidity	write	Passed	All Passed	No Issue	Passed
24	finalizeDXSale	write	Passed	All Passed	No Issue	Passed
25	finalizeListing	write	Passed	All Passed	No Issue	Passed
26	setLiquidityFeePercent	write	Passed	All Passed	No Issue	Passed
27	setMaxTxPercent	write	Passed	All Passed	No Issue	Passed
28	SetSwapAndLiquifyEnabled	write	Passed	All Passed	No Issue	Passed
29	_reflectFee	write	Passed	All Passed	No Issue	Passed
30	_getValues	read	Passed	All Passed	No Issue	Passed
31	_getTValues	read	Passed	All Passed	No Issue	Passed
32	_getRValues	read	Passed	All Passed	No Issue	Passed
33	_getRate	read	Passed	All Passed	No Issue	Passed
34	_getCurrentSupply	read	Passed	All Passed	No Issue	Passed
35	_takeLiquidity	read	Passed	All Passed	No Issue	Passed
36	CalculateTaxFee	read	Passed	All Passed	No Issue	Passed
37	CalculateLiquidityFee	read	Passed	All Passed	No Issue	Passed
38	removeAllFee	write	Passed	All Passed	No Issue	Passed
39	restoreAllFee	write	Passed	All Passed	No Issue	Passed
40	isExcludedFromFee	read	Passed	All Passed	No Issue	Passed
41	_approve	write	Passed	All Passed	No Issue	Passed
42	_transfer	write	Passed	All Passed	No Issue	Passed
43	swapAndLiquify	write	Passed	All Passed	No Issue	Passed
44	swaptokensforEth	write	Passed	All Passed	No Issue	Passed
45	addLiquidity	write	Passed	All Passed	No Issue	Passed
46	_tokenTransfer	write	Passed	All Passed	No Issue	Passed

47	transferStandard	write	Passed	All Passed	No Issue	Passed
48	transferToExcluded	write	Passed	All Passed	No Issue	Passed
49	transferFromExcluded	write	Passed	All Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have 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 tokens loss
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have 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 Findings

Critical

High

No high severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low

No very Low severity vulnerabilities were found.

Discussion

1. renounceOwnership will make all onlyowner function useless if it is part of plan then ok.
2. All useless functions for example "sendValue" and others should be removed from the library.
3. line - 193, 779 hard coded address should be double checked before deploying to production
4. "geUnlockTime" is like "getUnlockTime" practically no harm name can be any allowed character.

Conclusion

We were given a contract file and have used all possible tests based on the given object. The contract is written systematically. so **it is ready to go for production.**

Since possible test cases can be unlimited and developer level documentation (code flow diagram with function level description) not provided, for such extensive smart contract protocol, so we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is now “well secured”.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

RD Auditors 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).

Because the total number of test cases are unlimited, so 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 blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.



RD
AUDITORS