RD AUDITORS

WASABIX FINANCE SMART CONTRACT, CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Wasabix Finance
Prepared on: 14 May 2021
Platform: Binance Smart Chain
Language: Solidity

TABLE OF CONTENTS

Document	4
Introduction	5
Project Scope	6
Executive Summary	7
Code Quality	8
Documentation	9
Use of Dependencies	10
AS-IS Overview	10
Severity Definitions	15
Audit Findings	16
Conclusion	17
Our Methodology	18
Disclaimers	20

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 of Wasabix Finance
Platform	BSC / Solidity
File 1	Alchemist.sol
MD5 hash	85176AE8A98C5849858FF68816489726
SHA256 hash	597BCA22BB5B74791D8E6B8206B973A5D64050378690412526 5A84B7DBA3AA24
File 2	CDP.sol
MD5 hash	DE5B55F86BFBD467C93FCE1530BE08D5
SHA256 hash	3BD5E5A1A512508A648FBC9BAA576E445DB1C1CE5F513DFCA43FC0C96B5CA402
File 3	CDPD8.sol
MD5 hash	16B108B0433F1CD8BFC9A657881BAC9A
SHA256 hash	E2BDF8CF97C6731EB6F95F377E68BA3D648BF36FD8116BDC29F944EA43649B7A
File 4	Vault.sol
MD5 hash	26EFBC8FBE3DB930D9DDB41A1831A141
SHA256 hash	499FB32BBF7F419269AD132E5219BC980E5FF4E85A641F6325D167BEC406233F
File 5	Vault V2.sol
MD5 hash	BC716DF46A98AB4E5771ADB12189A1DD
SHA256 hash	E6EFF9AE376E9E0EE956ABA41961D5F6296A1BDE2A60ACDB3977EF476CA16ADC
Date	14/05/2021

Introduction

RD Auditors (Consultant) was contracted by Wasabix Finance (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contracts and its code review conducted between 11 - 14 May 2021.

This contract consists of five files.

Project Scope

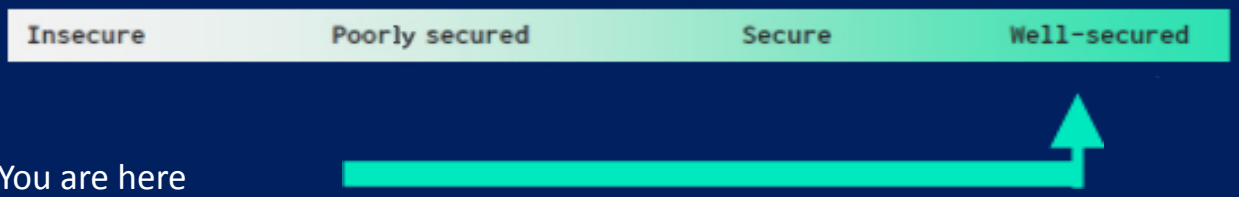
The scope of the project is a smart contract.

We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is 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

Please find a link that, within this report, contains SafeMath, Math and safeERC20 from the popular open source.

The libraries within this smart contract are part of a 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 to a specific address and its properties/methods can be reused many times by other contracts.

Wasabi Finance 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 quick cover the flow behind code logic. Use of Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given the contracts as a github link:

https://github.com/wasabi-swap-team/wasabix-yum/tree/main/contracts/libraries/alchemist

https://github.com/wasabi-swap-team/wasabix-yum/blob/main/contracts/Alchemist.sol

The hash of that file is mentioned in the table. As mentioned, it's well commented 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 that are written well and systematically.

AS-IS Overview

Wasabix Finance

The cross chain yield-backed synthetic asset and AMM DEX platform.

File And Function Level Report

File: Alchemist.sol

Contract: Alchemist
Import: math, reentrancyguard, Address, FixedPointMath, ITransmuter, IMintable, ERC20, IChainlink, IVaultAdapter, vault, console
Inherit: ReentrancyGuard
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	SetPendingGovernance	write	Passed	All Passed	No Issue	Passed
2	acceptGovernance	write	Passed	All Passed	No Issue	Passed
3	setSentinel	write	Passed	All Passed	No Issue	Passed
4	setTransmuter	write	Passed	All Passed	No Issue	Passed
5	setFlushActivator	write	Passed	All Passed	No Issue	Passed
6	SetRewards	write	Passed	All Passed	No Issue	Passed
7	setHarvestFee	write	Passed	All Passed	No Issue	Passed
8	setCollateralizationlimit	write	Passed	All Passed	No Issue	Passed
9	setOracleAddress	write	Passed	All Passed	No Issue	Passed
10	setEmergencyExit	write	Passed	All Passed	No Issue	Passed
11	collateralizationLimit	read	Passed	All Passed	No Issue	Passed
12	Initialize	write	Passed	All Passed	No Issue	Passed
13	migrate	write	Passed	All Passed	No Issue	Passed
14	harvest	write	Passed	All Passed	No Issue	Passed
15	recall	write	Passed	All Passed	No Issue	Passed
16	recallAll	write	Passed	All Passed	No Issue	Passed
17	FlushActiveVault	write	Passed	All Passed	No Issue	Passed
18	deposit	write	Passed	All Passed	No Issue	Passed
19	withdraw	write	Passed	All Passed	No Issue	Passed
20	repay	write	Passed	All Passed	No Issue	Passed

21	liquidate	write	Passed	All Passed	No Issue	Passed
22	mint	write	Passed	All Passed	No Issue	Passed
23	vaultCount	read	Passed	All Passed	No Issue	Passed
24	getVaultAdapter	read	Passed	All Passed	No Issue	Passed
25	getVaultTotalDeposited	read	Passed	All Passed	No Issue	Passed
26	getCdpTotalDeposited	read	Passed	All Passed	No Issue	Passed
27	getcdpTotalDept	read	Passed	All Passed	No Issue	Passed
28	getcdpLastDeposit	read	Passed	All Passed	No Issue	Passed
29	_distributeToTransmuter	write	Passed	All Passed	No Issue	Passed
30	_expectCaller	write	Passed	All Passed	No Issue	Passed
31	_UpdateActiveVault	write	Passed	All Passed	No Issue	Passed
32	_recallFunds	write	Passed	All Passed	No Issue	Passed
33	_withdrawFundsTo	write	Passed	All Passed	No Issue	Passed

File: CDP.sol

Contract: CDP
Import: Math, safeERC20, safeMath, FixedPointMath, IDetailedERC20, console
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Update	write	Passed	All Passed	No Issue	Passed
2	checkHealth	read	Passed	All Passed	No Issue	Passed
3	isHealthy	read	Passed	All Passed	No Issue	Passed
4	getUpdatedTotalDebt	read	Passed	All Passed	No Issue	Passed
5	getUpdatedTotalCredit	read	Passed	All Passed	No Issue	Passed
6	getEarnedYield	read	Passed	All Passed	No Issue	Passed
7	getCollateralizationRatio	read	Passed	All Passed	No Issue	Passed

File: CDPD8.sol

Contract: CDPD8
Import: Math, safeERC20, safeMath, FixedPointMathD8, IDetailedERC20, hardhat
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Update	write	Passed	All Passed	No Issue	Passed
2	checkHealth	read	Passed	All Passed	No Issue	Passed
3	isHealthy	read	Passed	All Passed	No Issue	Passed
4	getUpdatedTotalDebt	read	Passed	All Passed	No Issue	Passed
5	getUpdatedTotalCredit	read	Passed	All Passed	No Issue	Passed
6	getEarnedYield	read	Passed	All Passed	No Issue	Passed
7	getCollateralizationRatio	read	Passed	All Passed	No Issue	Passed

File: Vault.sol

Contract: Vault
Import: Math, safeERC20, safeMath, IVaultAdapter IDetailedERC20
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	totalValue	read	Passed	All Passed	No Issue	Passed
2	token	read	Passed	All Passed	No Issue	Passed
3	deposit	read	Passed	All Passed	No Issue	Passed
4	depositAll	write	Passed	All Passed	No Issue	Passed
5	withdraw	write	Passed	All Passed	No Issue	Passed
6	directWithdraw	write	Passed	All Passed	No Issue	Passed
7	withdrawAll	write	Passed	All Passed	No Issue	Passed
8	harvest	write	Passed	All Passed	No Issue	Passed
9	push	write	Passed	All Passed	No Issue	Passed
10	get	read	Passed	All Passed	No Issue	Passed
11	last	read	Passed	All Passed	No Issue	Passed
12	lastIndex	read	Passed	All Passed	No Issue	Passed
13	length	read	Passed	All Passed	No Issue	Passed

File: VaultV2.sol

Contract: VaultV2
Import: Math, safeERC20, safeMath, IvaultAdapterV2
IDetailedERC20, console
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	totalValue	read	Passed	All Passed	No Issue	Passed
2	token	read	Passed	All Passed	No Issue	Passed
3	deposit	write	Passed	All Passed	No Issue	Passed
4	depositAll	write	Passed	All Passed	No Issue	Passed
5	withdraw	write	Passed	All Passed	No Issue	Passed
6	directWithdraw	write	Passed	All Passed	No Issue	Passed

7	withdrawAll	write	Passed	All Passed	No Issue	Passed
8	harvest	write	Passed	All Passed	No Issue	Passed
9	push	write	Passed	All Passed	No Issue	Passed
10	get	read	Passed	All Passed	No Issue	Passed
11	last	read	Passed	All Passed	No Issue	Passed
12	lastIndex	read	Passed	All Passed	No Issue	Passed
13	length	read	Passed	All Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
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 cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical

No high severity vulnerabilities were found.

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.

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 an extensive smart contract protocol, 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 given repository link contains 57 solidity files and many external reference for library, interfaces and others and we are asked to audit only 5 files of them, which are centered around "Alchemist.sol" hence all our observations/findings are limited to these files only, we did not checked any connected impact with other files hence we are not sure as a whole the entire project is secured or not .

The security state of the reviewed contract is “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 analyse 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 scrutinised 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 analysed 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, 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, 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. 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

Email: info@rdauditors.com

Website: www.rdauditors.com