RD AUDITORS

GLOBAL MONEY SMART CONTRACT, CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Global Money
Prepared on: 19 Dec 2021
Platform: Binance
Blockchain**Language**: Solidity

TABLE OF CONTENTS

Document	4
Introduction	5
Project Scope	6
Executive Summary	7
Code Quality	8
Documentation	9
Use of Dependencies	9
AS-IS Overview	10
Audit Findings	17
Conclusion	18
Our Methodology	19
Disclaimers	21

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 THE DECISION OF THE CUSTOMER.

u

Document

Name	Smart Contract Code Review and Security Analysis Report of Global Money
Platform	Binance Blockchain
File 1	PrivateFarming.sol
MD5 hash	E1412751153F8FE80E3E1945234DFB62
SHA256 hash	4E2EFF69B30010801CC8D43AE46ADE6D1BB4BEA62F18D18D55B58DB1DD1B4B69
File 2	Global Money.sol
MD5 hash	926CDF642826EC86A8E271331522B2C9
SHA256 hash	E3976675DAD2B57D48CA1B33D72E72EFC06233D968A59E3B18E65CA344BBE5FD
File 3	StableRateFeeder.sol
MD5 hash	6520994763D018C22D30E38860D683B1
SHA256 hash	9297B70EBE68E6A2991BFB5AAC7AE19F69E2C27D89CD27E75DFFF14C1FC53DB1
File 4	SwaplessConversionPool.sol
MD5 hash	21F9CAF2B7D17E882CA3557C2FF3ECE0
SHA256 hash	C8AC753D0D6FD27A44418095F5F39198705FD5B4AC4820F96359A1B797120138
File 5	SingleLevelRateFeeder.sol

MD5 hash	4DCD50E7D0FF3CA5452F92264B7665C3
SHA256 hash	633B7F587089D295DABE515DF7BF9D326005E6321E865605A1D6388DB2ACC29B
File 6	UpdateStableRateFeeder.sol
MD5 hash	41258A7A4D0C4D739900035060BDDBE8
SHA256 hash	5990C314780C276FBE239BD0A21CB95E76C93948C367A23B0EEB8F761EAB19D0
File 7	VUSTExchangeRateFeeder.sol
MD5 hash	B8582CC175647F2B972340F9DA48DB46
SHA256 hash	1DDA52EC608477846351595B7CC335C4379F61DC561C5FACBB5B764BB040FF57
Date	13/08/2021

Introduction

RD Auditors (Consultant) were contracted by Global Money (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 2 - 19 Dec 2021.

This contract consists of seven files.

Project Scope

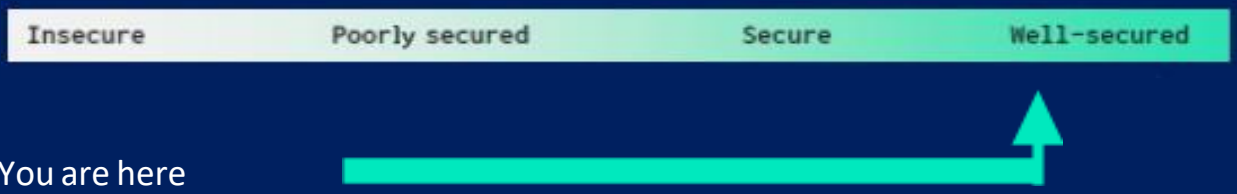
The scope of the project is a smart contract.

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 safeMath,BEP-20, ownable taken 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.

The Global Money team 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 almost not commented. Commenting can provide rich documentation for functions, return variables and more. Use of Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

The hash of that file is mentioned in the table. As mentioned above, It's recommended to write comments in the 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 that are written well and systematically.

AS-IS Overview

Global Money

File And Function Level Report

File: PrivateFarming.sol

Contract: PrivateFarming
Import: BEP-20, Ownable
Inherit: Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	addToken	write	Passed	All Passed	No Issue	Passed
2	checkToken	read	Passed	All Passed	No Issue	Passed
3	updateToken	write	Passed	All Passed	No Issue	Passed
4	removeToken	write	Passed	All Passed	No Issue	Passed
5	isValidToken	read	Passed	All Passed	No Issue	Passed
6	getTokenInfo	read	Passed	All Passed	No Issue	Passed
7	convert_atokens_to_tokens	read	Passed	All Passed	No Issue	Passed
8	convert_tokens_to_atokens	read	Passed	All Passed	No Issue	Passed
9	convert_tokens_to_universal_stable_coins	read	Passed	All Passed	No Issue	Passed
10	withdrawAnchored	write	Passed	All Passed	No Issue	Passed
11	withdrawLocal	write	Passed	All Passed	No Issue	Passed
12	Withdraw	write	Passed	All Passed	No Issue	Passed
13	finalizeWithdrawUpToUser	write	Passed	All Passed	No Issue	Passed
14	getActiveWithdrawOperationsCount	read	Passed	All Passed	No Issue	Passed

15	getWithdrawOperationsAbleToProcess	read	Passed	All Passed	No Issue	Passed
16	getWithdrawOperationByIndex	read	Passed	All Passed	No Issue	Passed
17	getActiveWithdrawOperation	read	Passed	All Passed	No Issue	Passed
18	getDepositMinLimit	read	Passed	All Passed	No Issue	Passed
19	getDepositMaxLimit	read	Passed	All Passed	No Issue	Passed
20	getFarmingLimitDate	read	Passed	All Passed	No Issue	Passed
21	getCombinedFarmedTokensOnDate	read	Passed	All Passed	No Issue	Passed
22	getFarmedTokensOnDate	read	Passed	All Passed	No Issue	Passed
23	update_balances_after_deposit	write	Passed	All Passed	No Issue	Passed
24	nextVestingDate	read	Passed	All Passed	No Issue	Passed
25	updateToken	write	Passed	All Passed	No Issue	Passed
26	removeToken	write	Passed	All Passed	No Issue	Passed
27	getTokenInfo	read	Passed	All Passed	No Issue	Passed
28	setDefaultSlippageTolerance	write	Passed	All Passed	No Issue	Passed
29	setEarlyWithdrawParams	write	Passed	All Passed	No Issue	Passed
30	setLimits	write	Passed	All Passed	No Issue	Passed
31	getLimits	read	Passed	All Passed	No Issue	Passed
32	getFarmingParams	read	Passed	All Passed	No Issue	Passed
33	getFarmingLimitDate	read	Passed	All Passed	No Issue	Passed
34	getCombinedFarmedTokensOnDate	read	Passed	All Passed	No Issue	Passed
35	getFarmedTokensOnDate	read	Passed	All Passed	No Issue	Passed
36	getFarmedTokens	read	Passed	All Passed	No Issue	Passed
37	getTimeTruncToVestingPeriod		Passed	All Passed	No Issue	Passed
38	fixFarmedTokens	write	Passed	All Passed	No Issue	Passed
39	update_balances_after_deposit	write	Passed	All Passed	No Issue	Passed
40	depositLocal	write	Passed	All Passed	No Issue	Passed
41	checkCanDeposit	read	Passed	All Passed	No Issue	Passed
42	Deposit	write	Passed	All Passed	No Issue	Passed
43	depositAnchored	write	Passed	All Passed	No Issue	Passed
44	withdrawAnchored	write	Passed	All Passed	No Issue	Passed
45	update_balances_after_withdraw	write	Passed	All Passed	No Issue	Passed
46	withdrawLocal	write	Passed	All Passed	No Issue	Passed
47	Withdraw	write	Passed	All Passed	No Issue	Passed
48	transferDeposit	write	Passed	All Passed	No Issue	Passed
49	takeAnchoredProfit	write	Passed	All Passed	No Issue	Passed
50	takeEarlyWithdrawFee	write	Passed	All Passed	No Issue	Passed

51	takeUnusedOrionTokens	write	Passed	All Passed	No Issue	Passed
52	registerInvestors	write	Passed	All Passed	No Issue	Passed
53	restoreInvestorAccess	write	Passed	All Passed	No Issue	Passed
54	getVestingInfo	write	Passed	All Passed	No Issue	Passed
55	getVestedTokensOnDate	write	Passed	All Passed	No Issue	Passed
56	getTokensAvailableToClaimOnDate	read	Passed	All Passed	No Issue	Passed
57	claimVestedTokens	write	Passed	All Passed	No Issue	Passed

File: GlobalMoney.sol

Contract: GlobalMoney
Import: OwnableUpgradeable, IDepositable
Inherit: Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	addToken	write	Passed	All Passed	No Issue	Passed
2	updateToken	write	Passed	All Passed	No Issue	Passed
3	removeToken	write	Passed	All Passed	No Issue	Passed
4	setLocalLimits	write	Passed	All Passed	No Issue	Passed
5	setLimits	read	Passed	All Passed	No Issue	Passed
6	getLimits	read	Passed	All Passed	No Issue	Passed
7	setDefaultSlippageTolerance	write	Passed	All Passed	No Issue	Passed
8	getTokenInfo	read	Passed	All Passed	No Issue	Passed
9	addToWhiteList	write	Passed	All Passed	No Issue	Passed
10	balanceOf	read	Passed	All Passed	No Issue	Passed
11	convert_atokens_to_tokens	read	Passed	All Passed	No Issue	Passed
12	convert_tokens_to_atokens	read	Passed	All Passed	No Issue	Passed
13	convert_tokens_to_orioned_amount	read	Passed	All Passed	No Issue	Passed
14	convert_orioned_amount_to_tokens	read	Passed	All Passed	No Issue	Passed
15	getFreeAnchoredAmount	read	Passed	All Passed	No Issue	Passed
16	takeAnchoredProfit	write	Passed	All Passed	No Issue	Passed
17	getDepositableAmount	read	Passed	All Passed	No Issue	Passed

18	takeProfit	write	Passed	All Passed	No Issue	Passed
19	depositFreeFunds	write	Passed	All Passed	No Issue	Passed
20	withdrawFreeFunds	write	Passed	All Passed	No Issue	Passed
21	canDepositLocal	read	Passed	All Passed	No Issue	Passed
22	depositLocal	write	Passed	All Passed	No Issue	Passed
23	deposit	write	Passed	All Passed	No Issue	Passed
24	depositAnchored	write	Passed	All Passed	No Issue	Passed
25	deposit	write	Passed	All Passed	No Issue	Passed
26	canWithdrawLocal	read	Passed	All Passed	No Issue	Passed
27	withdrawLocal	write	Passed	All Passed	No Issue	Passed
28	withdraw	write	Passed	All Passed	No Issue	Passed
29	transferDeposit	write	Passed	All Passed	No Issue	Passed
30	finalizeWithdrawUpToUser	write	Passed	All Passed	No Issue	Passed
31	getWithdrawOperationsAbleToProcess	read	Passed	All Passed	No Issue	Passed
32	getWithdrawOperationByIndex	read	Passed	All Passed	No Issue	Passed
33	getActiveWithdrawOperation	read	Passed	All Passed	No Issue	Passed
34	hasActiveWithdrawOperation	read	Passed	All Passed	No Issue	Passed
35	getTotalPendingWithdrawAmount	read	Passed	All Passed	No Issue	Passed

File: StableRateFeeder.sol

Contract: Context
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	powerValueBy	read	Passed	All Passed	No Issue	Passed
2	multiplyByRate	read	Passed	All Passed	No Issue	Passed
3	multiplyByCurrentRate	read	Passed	All Passed	No Issue	Passed

File: SwalessConversionPool.sol

Contract: Ownable
Inherit: Context
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	setOperationRouter	write	Passed	All Passed	No Issue	Passed
2	setExchangeRateFeeder	write	Passed	All Passed	No Issue	Passed
3	setDepositAllowance	write	Passed	All Passed	No Issue	Passed
4	setRedemptionAllowance	write	Passed	All Passed	No Issue	Passed
5	migrate	write	Passed	All Passed	No Issue	Passed
6	provideReserve	write	Passed	All Passed	No Issue	Passed
7	removeReserve	write	Passed	All Passed	No Issue	Passed
8	getFeederRate	read	Passed	All Passed	No Issue	Passed
9	deposit	write	Passed	All Passed	No Issue	Passed
10	redeem	write	Passed	All Passed	No Issue	Passed
11	profitAmount	read	Passed	All Passed	No Issue	Passed
12	takeProfit	write	Passed	All Passed	No Issue	Passed

File: SingleLevelRateFeeder.sol

Contract: SingleLevelRateFeeder
Inherit: Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	setBaseRateStartValue	read	Passed	All Passed	No Issue	Passed
2	updateBaseRate	write	Passed	All Passed	No Issue	Passed
3	updateDiscount	write	Passed	All Passed	No Issue	Passed
4	multiplyByRate	read	Passed	All Passed	No Issue	Passed
5	multiplyByDiscountedRate	read	Passed	All Passed	No Issue	Passed
6	powerRateBy	read	Passed	All Passed	No Issue	Passed
7	getBaseAPY	read	Passed	All Passed	No Issue	Passed
8	getDiscountedAPY	write	Passed	All Passed	No Issue	Passed

File: UpdatableStableRateFeeder.sol

Contract: StableRateFeeder, Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	updateRate	write	Passed	All Passed	No Issue	Passed

File: VUSTExchangeRateFeeder.sol

Contract: IExchangeRateFeeder
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	exchangeRateOf	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 critical 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.

Discussion

1. Functions might be reverted if the loops are too high.
2. Static values should be checked before deployment.

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 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 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