RD AUDITORS

FUNDI FINANCE SMART CONTRACT, CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: FUNDI FINANCE
Prepared on: 25 June 2021
Platform: BSC
Language: Solidity

TABLE OF CONTENTS

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 Fundi Finance
Platform	BSC / Solidity
File 1	CRAOwner.sol
MD5 hash	B20746A915D591CE271D6CD99718D11C
SHA256 hash	3A707C305B9EEF09B682F3FEBF6F050F6E12AF7DF2C768241AA26D9A3013A31A
File 2	FundiATO.sol
MD5 hash	9B80389C1614A10E557CDE9E3F8C968D
SHA256 hash	AAD3DC5EE35D8273C0B79325F78D28EA3BC1C42C508FF13D26978638AB0ECA3A
File 3	FundiTeam.sol
MD5 hash	770D5CEEB5F6C62FDF82CCAD28B23D8A
SHA256 hash	27A514C50706C193C215EAD3539229F762D5B612F42D37FDBBBD8BA60ECCBE9A
File 4	ICRAOwner.sol
MD5 hash	E4B5CFD3BB2C5FDEFC6FDBB40163A200
SHA256 hash	D193C009F5E756481B05616287C5E03B8DB898F38924801ED000F52F33C69D46
File 5	ICRAToken.sol
MD5 hash	B9314434734DD49A40AD34868F5188B9

SHA256 hash	0A658CD3C13326CE51DB0E6B79EDEF39640FCBE69086D6A1F598A6942D2332F4
File 6	ICRAToken.sol
MD5 hash	B9314434734DD49A40AD34868F5188B9
SHA256 hash	0A658CD3C13326CE51DB0E6B79EDEF39640FCBE69086D6A1F598A6942D2332F4
File 6	IFFToken.sol
MD5 hash	C92CE923EB94575C54E916F5245364B3
SHA256 hash	11611DCED009B619210517E437A1A922061B77DDF8B0C81ED01051B47B957EB1
File 7	LPTokenFarm.sol
MD5 hash	9F81A7BE7D10AAD468AC6EA6077929AB
SHA256 hash	82D9DB255A963A45DECF2938131A07BDF02FDC5D397F72FCDA73205BB9E782DC
File 8	LPTokenWrapper.sol
MD5 hash	5E01D1E22B92170B347CDE7F9B6F83CC
SHA256 hash	31088B17F917AC3833BF91BDB6449D738573D8B51641B966A8E268841A7FB3B8
File 9	RewardPool.sol
MD5 hash	20FFF783B5B22ADB1283ED7C9D11EDEF
SHA256 hash	6EFE80B1A3066E1E59460E8570E973FEABE8BDA5BB3FCB7EA2B2044BB9A61CE3
File 10	SingleAssetFarm
MD5 hash	1CCA2B84B54892A89853F3410853F288
SHA256 hash	AE293A6152A39196500B78ADFEB84EEF64458469D460CFAB782F9CA9BECF88A7
Date	25/06/2021

Introduction

RD Auditors were contracted by Fundi Finance 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 20 - 25 June 2021.

This contract consists of ten files.

Project Scope

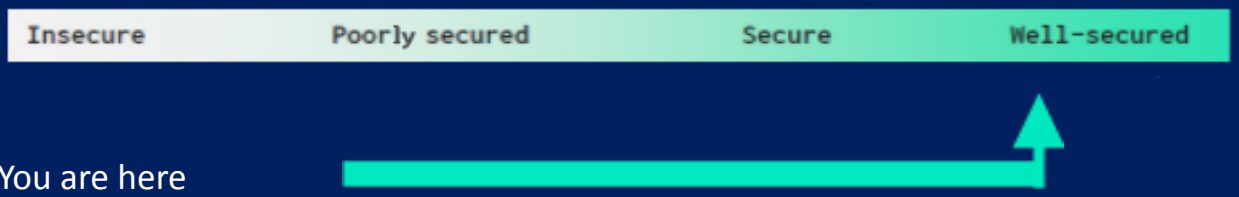
The scope of the project is a smart contract.

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

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 Fundi Finance team has provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is **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

We were given the Fundi Finance contract as a github link:

https://github.com/FundiFinance/fundi-protocol-beta/tree/master/contracts

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

Fundi Finance

File And Function Level Report

File 1: CRAOwner.sol

Contract: CRAOwner
Import: Ownable, AccessControl, SafeMath
Inherit: SafeMath, AccessControl, Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getToken	read	Passed	All Passed	No Issue	Passed
2	mint	write	Passed	All Passed	No Issue	Passed
3	addMinter	write	Passed	All Passed	No Issue	Passed
4	removeMinter	write	Passed	All Passed	No Issue	Passed
5	renounceMinter	write	Passed	All Passed	No Issue	Passed

File And Function Level Report

File 2: FundiATO.sol

Contract: FundiATO
Import: Ownable, IERC20, SafeERC20, IBondingCurve, FundiTeam, IFFToken
Inherit: SafeMath, Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	initialize	write	Passed	All Passed	No Issue	Passed
2	buy	write	Passed	All Passed	No Issue	Passed
3	buyWithRef	write	Passed	All Passed	No Issue	Passed
4	getEstimatedContinuousMintReward	read	Passed	All Passed	No Issue	Passed
5	withdrawBNB	write	Passed	All Passed	No Issue	Passed
6	_buy	write	Passed	All Passed	No Issue	Passed

File And Function Level Report

File 3: FundiTeam.sol

Contract: FundiTeam
Import: SafeMath, IERC20, SafeERC20
Inherit: SafeMath,IERC20
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	updateDonatelloAdd	write	Passed	All Passed	No Issue	Passed
2	updateRaphaelAdd	write	Passed	All Passed	No Issue	Passed
3	updateMichelangeloAdd	write	Passed	All Passed	No Issue	Passed
4	updateLeonardoAdd	write	Passed	All Passed	No Issue	Passed
5	withdrawFUNDI	write	Passed	All Passed	No Issue	Passed
6	updateFundiAdd	write	Passed	All Passed	No Issue	Passed
7	availableFundi	read	Passed	All Passed	No Issue	Passed

File And Function Level Report

File 4: ICRAOwner.sol

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

File And Function Level Report

File 5: ICRAToken.sol

Contract:	ICRAToken
Import:	IERC20
Observation:	Passed
Test Report:	Passed
Score:	Passed
Conclusion:	Passed

File And Function Level Report

File 6: IFFToken.sol

Contract: IFFToken
Import: IERC20
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

File And Function Level Report

File 7: LPTokenFarm.sol

Contract: LPTokenFarm
Import: Ownable, Math, SafeMath, IERC20, SafeERC20
Inherit: SafeMath,IERC20,Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	addPool	write	Passed	All Passed	No Issue	Passed
2	poolLength	read	Passed	All Passed	No Issue	Passed
3	totalSupply	read	Passed	All Passed	No Issue	Passed
4	balanceOf	read	Passed	All Passed	No Issue	Passed
5	stake	write	Passed	All Passed	No Issue	Passed
6	withdraw	write	Passed	All Passed	No Issue	Passed
7	lastTimeRewardApplicable	read	Passed	All Passed	No Issue	Passed

8	rewardPerToken	read	Passed	All Passed	No Issue	Passed
9	earned	read	Passed	All Passed	No Issue	Passed
10	exit	write	Passed	All Passed	No Issue	Passed
11	getReward	write	Passed	All Passed	No Issue	Passed
12	notifyRewardAmount	write	Passed	All Passed	No Issue	Passed
13	giveFFReward	write	Passed	All Passed	No Issue	Passed
14	setCore	write	Passed	All Passed	No Issue	Passed

File And Function Level Report

File 8: LPTokenWrapper.sol

Contract: LPTokenWrapper
Import: SafeMath, SafeERC20, IERC20
Inherit: SafeMath, IERC20
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	totalSupply	read	Passed	All Passed	No Issue	Passed
2	balanceOf	read	Passed	All Passed	No Issue	Passed
3	stake	write	Passed	All Passed	No Issue	Passed
4	withdraw	write	Passed	All Passed	No Issue	Passed

File And Function Level Report

File 9: RewardPool.sol

Contract: RewardPool
Import: SafeMath, SafeERC20, IERC20, Math, LPTokenWrapper
ICRAOwner
Inherit: SafeMath, LPTokenWrapper, Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	lastTimeRewardApplicable	read	Passed	All Passed	No Issue	Passed
2	rewardPerToken	read	Passed	All Passed	No Issue	Passed
3	earned	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	stake	write	Passed	All Passed	No Issue	Passed
7	withdraw	write	Passed	All Passed	No Issue	Passed
8	exit	write	Passed	All Passed	No Issue	Passed
9	getReward	write	Passed	All Passed	No Issue	Passed
10	getFFReward	read	Passed	All Passed	No Issue	Passed
11	notifyRewardAmount	write	Passed	All Passed	No Issue	Passed

File And Function Level Report

File 10: SingleAssetFarm.sol

Contract: SingleAssetFarm
Import: SafeMath, SafeERC20, IERC20, Math, EnumerableSet, ICRAOwner
Inherit: SafeMath,Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	addPool	write	Passed	All Passed	No Issue	Passed
2	setAllocPoint	write	Passed	All Passed	No Issue	Passed
3	setLockupDuration	write	Passed	All Passed	No Issue	Passed
4	startStaking	write	Passed	All Passed	No Issue	Passed
5	pendingRewards	read	Passed	All Passed	No Issue	Passed
6	updatePool	write	Passed	All Passed	No Issue	Passed
7	withdraw	write	Passed	All Passed	No Issue	Passed
8	deposit	write	Passed	All Passed	No Issue	Passed
9	withdraw	write	Passed	All Passed	No Issue	Passed
10	claim	write	Passed	All Passed	No Issue	Passed
11	safeRewardTransfer	read	Passed	All Passed	No Issue	Passed
12	setCRAPerBlock	write	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

Discussion

1) Check all loop iteration on its higher level.
2) The “require (from!=address(0))” should be used for the "stake" and "withdraw" functions.

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