Smart Contract Audit Report

for

Defina.Finance

Version 1.0

Trustlook Blockchain Labs

Email: bd@trustlook.com

TRUSTLOOK

Project Overview

Project Name	Defina.Finance
Contract codebase	N/A
Platform	BSC
Language	Solidity
Submission Time	2021.10.22

Report Overview

Report ID	TBL_20211022_00
Version	1.0
Reviewer	Trustlook Blockchain Labs
Starting Time	2021.10.22
Finished Time	2021.11.05

@ Copyright 2021 Trustlook - All rights reserved

Disclaimer

Trustlook audit reports do not provide any warranties or guarantees on the vulnerability-free nature of the given smart contracts, nor do they provide any indication of legal compliance. The Trustlook audit process is aiming to reduce the high level risks possibly implemented in the smart contracts before the issuance of audit reports. Trustlook audit reports can be used to improve the code quality of smart contracts and are not able to detect any security issues of smart contracts that will occur in the future. Trustlook audit reports should not be considered as financial investment advice.

@ Copyright 2021 Trustlook - All rights reserved

About Trustlook Blockchain Labs

Trustlook Blockchain Labs is a leading blockchain security team with a goal of security and vulnerability research on current blockchain ecosystems by offering industry-leading smart contracts auditing services. Please contact us for more information at (https://www.trustlook.com/services/smart.html) or Email (bd@trustlook.com)

The Trustlook blockchain laboratory has established a complete system test environment and methods.

Black-box Testing	The tester has no knowledge of the system being attacked. The goal is to simulate an external hacking or cyber warfare attack.
White-box Testing	Based on the level of the source code, test the control flow, data flow, nodes, SDK etc. Try to find out the vulnerabilities and bugs.
Gray-box Testing	Use Trustlook customized script tools to do the security testing of code modules, search for the defects if any due to improper structure or improper usage of applications.

@ Copyright 2021 Trustlook - All rights reserved

Introduction

By reviewing the implementation of Defina.Finance's smart contracts, this audit report has been prepared to discover potential issues and vulnerabilities of their source code. We outline in the report about our approach to evaluate the potential security risks. Advice to further improve the quality of security or performance is also given in the report.

About Defina.Finance

Defina.Finance is a data aggregator specially designed for DeFi and NFT, providing customisable smart contracts to simplify the investment process of DeFi and NFT for users of all levels.

About Methodology

To evaluate the potential vulnerabilities or issues, we go through a checklist of well-known smart contracts related security issues using automatic verification tools and manual review. To discover potential logic weaknesses or project specific implementations, we thoroughly discussed with the team to understand the business model and reduce the risk of unknown vulnerabilities. For any discovered issue, we might test it on our private network to reproduce the issue to prove our findings.

The checklist of items is shown in the following table:

Category	Type ID	Name	Description
Coding Specification	CS-01	ERC standards	The contract is using ERC standards.
	CS-02	Compiler Version	The compiler version should be specified.
	CS-03	Constructor Mismatch	The constructor syntax is changed with Solidity versions. Need extra attention to make the constructor function right.
	CS-04	Return standard	Following the ERC20 specification, the transfer and approve

@ Copyright 2021 Trustlook - All rights reserved

			functions should return a bool value, and a return value code needs to be added.
	CS-05	Address(0) validation	It is recommended to add the verification of require(_to!=address(0)) to effectively avoid unnecessary loss caused by user misuse or unknown errors.
	CS-06	Unused Variable	Unused variables should be removed.
	CS-07	Untrusted Libraries	The contract should avoid using untrusted libraries, or the libraries need to be thoroughly audited too.
	CS-08	Event Standard	Define and use Event appropriately
	CS-09	Safe Transfer	Using transfer to send funds instead of send.
	CS-10	Gas consumption	Optimize the code for better gas consumption.
	CS-11	Deprecated uses	Avoid using deprecated functions.
	CS-12	Sanity Checks	Sanity checks when setting key parameters in the system
Coding Security	SE-01	Integer overflows	Integer overflow or underflow issues.
	SE-02	Reentrancy	Avoid using calls to trade in smart contracts to avoid reentrancy vulnerability.
	SE-03	Transaction Ordering Dependence	Avoid transaction ordering dependence vulnerability.
	SE-04	Tx.origin usage	Avoid using tx.origin for authentication.
	SE-05	Fake recharge	The judgment of the balance and the transfer amount needs to use the “require function”.
	SE-06	Replay	If the contract involves the demands for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks.
	SE-07	External call checks	For external contracts, pull instead of push is preferred.
	SE-08	Weak random	The method of generating random numbers on smart contracts requires more considerations.
Additional Security	AS-01	Access control	Well defined access control for functions.
	AS-02	Authentication management	The authentication management is well defined.
	AS-03	Semantic Consistency	Semantics are consistent.
	AS-04	Functionality checks	The functionality is well implemented.

@ Copyright 2021 Trustlook - All rights reserved

	AS-05	Business logic review	The business model logic is implemented correctly.

The severity level of the issues are described in the following table:

Severity	**Description**
Critical	The issue will result in asset loss or data manipulations.
High	The issue will seriously affect the correctness of the business model.
Medium	The issue is still important to fix but not practical to exploit.
Low	The issue is mostly related to outedate, unused code snippets.
Informational	This issue is mostly related to code style, informational statements and is not mandatory to be fixed.

@ Copyright 2021 Trustlook - All rights reserved

Audit Results

Here are the audit results of the smart contracts. The new release of the smart contracts adds more features to restrict the privilege of the owner and therefore reduce the risk of private key loss or hacking events.

Scope

Following files have been scanned by our internal audit tool and manually reviewed and tested by our team:

File names	Sha1
FinaFarming.sol	9f8937af44d52a59c4f310b134498cbb49872cb4
FinaLockFarming.sol	2b5d897998a58072953691ce6976fb9301ca8bfd

Summary

Issue ID	Severity	Location	Type ID	Status
TBL_SCA_001	Info	FinaFarming.sol:81 FinaFarming.sol:88	CS-10	Fixed
TBL_SCA_002	Info	FinaFarming.sol:67 FinaFarming.sol:68	CS-12	Fixed
TBL_SCA_003	Info	FinaFarming.sol:133 FinaFarming.sol:146	CS-12	Fixed
TBL_SCA_004	Info	FinaFarming.sol:73	CS-12	Closed

@ Copyright 2021 Trustlook - All rights reserved

		FinaFarming.sol:84 FinaFarming.sol:175 FinaFarming.sol:182 FinaFarming.sol:187 FinaFarming.sol:192		
TBL_SCA_005	Medium	FinaFarming.sol:200	AS-04	Fixed
TBL_SCA_006	Info	FinaLockFarming.sol:91	CS-10	Closed
TBL_SCA_007	Low	FinaLockFarming.sol:112 FinaLockFarming.sol:117	AS-05	Closed
TBL_SCA_008	Info	FinaLockFarming.sol:70	CS-10	Closed

@ Copyright 2021 Trustlook - All rights reserved

Details

• ID: TBL_SCA-001

• Severity: Info

• Type: CS-10 (Gas consumption)

• Description:

Function *updateLPAllocPoint()* is called to update the value of *totalLPAllocPoint*. The new value of *totalLPAllocPoint* actually can be adjusted by applying the difference brought by *allocPoint_*. Therefore, the calculation loop in *updateLPAllocPoint()* is not needed.

• Remediation:

The team has fixed the issue in commit 4fd56b40abd115f7efd2b3137373703387733fba

@ Copyright 2021 Trustlook - All rights reserved

• ID: TBL_SCA-002

• Severity: Informational

• Type: CS-12 (Sanity Checks)

• Description:

For key parameters in the system, it is recommended to add some sanity checks on update.

It is recommended to validate parameters *devAddr_* and *rewardPerBlock_* to be non-zero values before the assignment. In particular, *rewardPerBlock* will be used as a divisor in the function *depositLP()*.

• Remediation:

The team has fixed the issue in commit 4fd56b40abd115f7efd2b3137373703387733fba

@ Copyright 2021 Trustlook - All rights reserved

• ID: TBL_SCA-003

• Severity: Informational

• Type: CS-12 (Sanity Checks)

• Description:

It is recommended to validate *totalLPAllocPoint* to be non-zero before using it as a divisor.

• Remediation:

The team has fixed the issue in commit 4fd56b40abd115f7efd2b3137373703387733fba

@ Copyright 2021 Trustlook - All rights reserved

• ID: TBL_SCA-004

• Severity: Informational

• Type: CS-08 (Event Standard)

• Description:

Functions setRewardPerBlock, setFinaAddress, setSecondTokenAddress, setDevAddress, addLPPool, and resetLPPool did not emit an event for the operation. Recommend emitting events for all the essential state variable updates.

• Remediation:

The team has decided to leave it as is.

@ Copyright 2021 Trustlook - All rights reserved

• ID: TBL_SCA-005

• Severity: Medium

• Type: AS-04 (Functionality checks)

• Description:

Function pullFunds() can transfer any token (including Staked LP tokens) to the Owner. The safety of the Owner's private key would bring a security concern to staked LP tokens for users.

• Remediation:

The team has fixed the issue in commit 4fd56b40abd115f7efd2b3137373703387733fba

@ Copyright 2021 Trustlook - All rights reserved

• ID: TBL_SCA-006

• Severity: Info

• Type: CS-10 (Gas consumption)

• Description:

Function *pendingLPReward()* has a duplicated implementation of *updateLPPool()*. Therefore, it is not necessary to call the function followed by the execution of *updateLPPool()*.

It is recommended to use the following code to get the value of pending rewards:

```
uint pending = user.amount * lpPool.accRewardPerShare / 1e12 - user.rewardDebt;
uint pendingExtra = pending * secondRewardPerBlock / rewardPerBlock;
```

• Remediation:

The team has decided to leave it as is.

@ Copyright 2021 Trustlook - All rights reserved

• ID: TBL_SCA-007

• Severity: Low

• Type: AS-05 (Business logic review)

• Description:

A user’s averageDepositedTime records the average deposit timestamp with the consideration of the deposit amount. Logically, it should not be affected by withdrawal actions except that the whole deposit was taken.

It is recommended to not update the averageDepositedTime when users only withdraw a part of their deposit or claim the rewards.

• Remediation:

The team has decided to leave it as is.

@ Copyright 2021 Trustlook - All rights reserved

• ID: TBL_SCA-008

• Severity: Info

• Type: CS-10 (Gas consumption)

• Description:

The return value of function *pendingLPReward()* in function *currentUserTier()* is used for the later IF statement. The validation of the pending value in the IF statement can be substituted to check the value of *userLPInfo[_pid][_user].averageDepositedTime* to be non-zero or not.

• Remediation:

The team has decided to leave it as is.

@ Copyright 2021 Trustlook - All rights reserved