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 Times	2021.11.10, 2021.11.25

Report Overview

Report ID	TBL_20211110_00
Version	1.0
Reviewer	Trustlook Blockchain Labs
Starting Time	2021.11.10
Finished Time	2021.12.01

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.

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.

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

			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.

	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.

TRUSTLOOK

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
BlindBoxV2.sol	8fbac24fa808d7cb29df392029d98520759b03a0
FinaMaster.sol	4103efc68a7baa8f5ed55269628e1bb573553852
DefinaNFTMaster.sol	31d677f7ae470d3f273d21de0c53f94b2b27be0e
GameAccountRegister.sol	28040fe10525f67a1634bf8756ae78bc1bb925f0
DefinaCardAdapter.sol	aeb0a81ce5a2e8fc217c6e4f6aa652becd7b7846
FocLockFarming.sol	e1655a63fea5122f04dd977b8273db30ea1f8019

Summary

Issue ID	Severity	Location	Type ID	Status
TBL_SCA_001	Info	BlindBoxV2.sol:61	CS-08	closed

TBL_SCA_002	Info	BlindBoxV2.sol:138 BlindBoxV2.sol:143 BlindBoxV2.sol:148 BlindBoxV2.sol:153	CS-08	closed
TBL_SCA_003	Info	BlindBoxV2.sol:173	SE-08	closed
TBL_SCA_004	Info	GameAccountRegister.sol:17 GameAccountRegister.sol:18 GameAccountRegister.sol:19 GameAccountRegister.sol:20	CS-08	closed
TBL_SCA_005	Medium	GameAccountRegister.sol:64	AS-04	closed
TBL_SCA_006	High	FocLockFarming.sol:101	AS-05	closed
TBL_SCA_007	low	DefinaCardAdapter.sol:97	CS-06	closed
TBL_SCA_008	low	DefinaCardAdapter.sol:138	CS-06	closed

Details

- ID: TBL_SCA-001
- Severity: Info
- Type: CS-08 (Event Standard)
- Description:

  Event SetMaxMintAmount() is defined but not used.

- Remediation:

  The issue has been fixed in commit 3087909.

• ID: TBL_SCA-002

• Severity: Informational

• Type: Type: CS-08 (Event Standard)

• Description:

Functions setNftPrice(), setErc20TokenReceiveAddress(), setCurrToken(), transferAdmin() 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.

• ID: TBL_SCA-003

• Severity: Informational

• Type: SE-08 (Weak random)

• Description:

The random number generation algorithm implemented in function _randModulus() is weak. It should be aware that the weakness should not affect the correctness of the business model.

• Remediation:

The team has decided to leave it as is.

• ID: TBL_SCA-004

• Severity: Informational

• Type: CS-08 (Event Standard)

• Description:

When defining an Event with address parameters, it is recommended to add *indexed* keywords for them for better query operations.

We advise to update these Events as following:

event Binding(address indexed user, bytes32 account);
event UnBinding(address indexed user);
event Delegated(address indexed user, bytes32 delegatedAccount);
event UnDelegated(address indexed user, bytes32 delegatedAccount);

• Remediation:

The issue has been fixed in commit 3087909.

• ID: TBL_SCA-005

• Severity: Medium

• Type: AS-04 (Functionality checks)

• Description:

Function *undelegateAccount()* validates whether the *emailHash* exists in the *delegatedEmails*. However, it does not validate if the *emailHash* is equal to *delegatedAccounts[_msgSender()]* or not. Therefore, the function caller can remove the *emailHash* in *delegatedEmails* belonging to other users. That will result in the victim can't successfully execute *undelegateAccount()* in future.

It is recommended to add following validation:

require(emailHash == delegatedAccounts[_msgSender()], "The email was not registered as the delegated account");

• Remediation:

The issue has been fixed in commit 3087909.

• ID: TBL_SCA-006

• Severity: High

• Type: AS-05 (Business Logic Review)

• Description:

The function *depositLP()* does not send the pending reward to the user. However, the *user.rewardDebt* is adjusted with the new deposited amount. It will make the user lose the pending reward from previous deposits.

It is recommended to send the pending reward first before adjusting the *user.rewardDebt* value.

• Remediation:

The issue has been fixed in commit 6a3607a.

• ID: TBL_SCA-007

• Severity: Low

• Type: Type: CS-06 (Unused Variable)

• Description:

The argument *cardId_* is not used in the function *mint()* at all.

It is recommended to remove the argument in the *mint()* function.

• Remediation:

The team has confirmed the argument exists due to backward compatibility reasons.

• ID: TBL_SCA-008

• Severity: Low

• Type: Type: CS-06 (Unused Variable)

• Description:

The argument *from* is not used in the function *safeTransferFrom()* at all.

It is recommended to remove the argument in the *safeTransferFrom()* function.

• Remediation:

The team has confirmed the argument exists due to backward compatibility reasons.