Smart Contract Audit Report

for

TheForce.Trade

Version 1.0

Trustlook Blockchain Labs

Email: bd@trustlook.com

Project Overview

Project Name	TheForce.Trade
Contract codebase	N/A
Platform	Binance Smart Chain
Language	Solidity
Submission Time	2021.04.09

Report Overview

Report ID	TBL_20210411_00
Version	v1.0
Reviewer	Trustlook Blockchain Labs
Starting Time	2021.04.09
Finished Time	2021.04.18

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. 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)

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 TheForce.Trade'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 TheForce.Trade

The Force.Trade 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 show in 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.
	CV-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	Follow the ERC20 specification and require the transfer and approve functions to trigger the corresponding event.
	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.
Coding Security	SE-01	Integer overflows	Using safeMath library to avoid integer overflows.
	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 is implemented logically correct.

The severity level the issues are described as 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.

Audit Results

Here are the audit results of the smart contracts.

Scope

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

File names	Sha1
forcestrategy.sol	3166e1f8956d7fc8c6893fc77c01a8edbb030709
forcevault.sol	b5f5c95a602bd19f34f29e7be46f9ada9fdc4c36
cakevault.sol	789661110968672849340 2aa2d4442ebba3a5125
cakestrategy.sol	cf3e9d55a1dcb52494c7810c2863bf7aae1ad2e5
cakeLPstrategy.sol	4195afc1f1bb88c862ab20e0c8495756e1a3fc34
cakeLPvault.sol	97dcc48a9d217da46dc4fe0f92d7ad0019061583
ForceTreasury.sol	66eeb4da4f8e48fcb8f6b1b85c4377fe9a5696b3
ForceToken.sol	1219079cbdfefc317d5d04eb698d5d6af7b7d1ff
ForceLPToken.sol	69e04b909a2e51efcd689207cea0697b7dadabb9
JediMaster.sol	17c6b9d0c5ebfb1a179e19bd2f4cd49dfcd499c0

Summary

Issue ID	Severity	Location	Type ID	Status
TBL_SCA_001	High	ForceToken.sol:873	AS-03	fixed
TBL_SCA_002	High	ForceLPToken.sol:1106	AS-03	fixed
TBL_SCA_003	High	JediMaster.sol:1203	AS-03	fixed
TBL_SCA_004	Info	cakestrategy.sol:1194	AS-04	closed
TBL_SCA_005	Info	cakeLPstrategy.sol:1216	AS-04	closed
TBL_SCA_006	Info	forcestrategy.sol:1188	AS-04	closed
TBL_SCA_007	Info	cakestrategy.sol:1238	SE-04	closed
TBL_SCA_008	Info	cakeLPstrategy.sol:1266	SE-04	closed
TBL_SCA_009	Info	forcestrategy.sol:1232	SE-04	closed
TBL_SCA_010	Info	cakestrategy.sol:1253	AS-04	fixed
TBL_SCA_011	Info	cakeLPstrategy.sol:1283	AS-04	fixed
TBL_SCA_012	Info	forcestrategy.sol:1247	AS-04	fixed
TBL_SCA_013	Info	cakestrategy.sol:1287	CS-10	fixed
TBL_SCA_014	Info	cakeLPstrategy.sol:1348	CS-10	fixed
TBL_SCA_015	Info	forcestrategy.sol:1277	CS-10	fixed

Details

• ID: TBL_SCA-001 - TBL_SCA-003

• Severity: High

• Type: AS-03 (Semantic Consistency)

• Description:

During the implementation of function burn(), the delegates are supposed to be decreased from the delegator. However, the logic is mistakenly to still increase the delegate as the mint() function.

We recommend to change the line in burn() to be as follows:

“_moveDelegates(_delegates[_from], address(0), _amount);”

• Remediation:

The issue has been fixed.

• ID: TBL_SCA-004 - TBL_SCA-006

• Severity: Informational

• Type: AS-04 (Functionality checks)

• Description:

The functions call safeApprove with uint(-1), which makes the target address have the right to transfer an unlimited amount of balance. Though the target addresses are the administrative contract addresses.

We recommend only use the necessary amount for the safeApprove() in the later deposit() functions.

• Remediation:

TheForce.Trade understands the risk and keeps the contract as is.

• ID: TBL_SCA-007 - TBL_SCA-009

• Severity: Informational

• Type: SE-04 (tx.origin usage)

• Description:

Theoretically, if the contract owner calls a malicious contract and results in calling the affected withdraw() functions. It will bypass the validation of the ownership. The potential risk should be avoided given the fact that the contacts are under control of TheForce.Trade.

We just advise to avoid using tx.origin to any form of authentication if possible.

• Remediation:

TheForce.Trade team understands the risk and it is controllable by avoiding calling untrusted contracts.

• ID: TBL_SCA-010 - TBL_SCA-012

• Severity: Informational

• Type: AS-04 (Functionality checks)

• Description:

1. isContract function can not guarantee the caller is a non-contract user, since the constructor of a contract can also make EXTCODESIZE returns 0.

2. The information shown in line “!contract” is misleading the meaning of the code.

For 1. We just inform the project to be aware of this situation.

For 2. We recommend updating the information as “contract”.

• Remediation:

TheForce.Trade understands the situation of issue1. Issue 2 was fixed with better readable words.

• ID: TBL_SCA-013 - TBL_SCA-015

• Severity: Informational

• Type: CS-10 (Gas consumption)

• Description:

Function balanceOf() is not called by any of the functions inside the contract.

We recommend using external instead of public for lower gas cost.

• Remediation:

The issue was fixed.