Smart Contract Audit Report

for

TOR

Version 1.0

Trustlook Blockchain Labs

Email: bd@trustlook.com

Project Overview

Project Name	TOR
Contract codebase	N/A
Platform	BSC
Language	Solidity
Submission Time	2021.05.13

Report Overview

Report ID	TBL_20210513_00
Version	v1.0
Reviewer	Trustlook Blockchain Labs
Starting Time	2021.05.13
Finished Time	2021.05.26

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 TOR'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 TOR

The TOR is a data aggregator specially designed for the next generation of payments infrastructure designed for the privacy VPN market.

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	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 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
LPRewardTor.sol	29b0a1ecbadb34aa51332351956d799a791c2247
RingBase.sol	5afaff82cc9e69574a29d1c0f104cc9499113a1e
RingFactory.sol	b0a4cfcf004762191f6ce772610cb5320f2e33b3
TeamTimelock.sol	109511af65126d46a2203eafdcdef247b3b39e22
TorCollectibles.sol	6ac264152eff21112c02297d45478182484916f0
TorToken.sol	c2e8f57da7e9b8b90c135050f6e67f4de0280f90

Summary

Issue ID	Severity	Location	Type ID	Status
TBL_SCA_001	Low	RingBase.sol:100	AS-04	fixed
TBL_SCA_002	Info	RingBase.sol:121	CS-10	fixed
TBL_SCA_003	Info	LPRewardTor.sol:50	CS-10	closed

TBL_SCA_004	Info	LPRewardTor.sol:111	CS-10	fixed

Details

• ID: TBL_SCA-001

• Severity: Low

• Type: AS-04 (Functionality checks)

• Description:

In the implementation of function _beforeTokenTransfer(), the token index was tried to be extracted from the _allRingsIndex table. However, the item for the tokenId is not set yet (it was set in the function _addRingToAllRingsEnumeration() which is called later). Therefore, the index will always be 0 here.

• Remediation:

The TOR team has updated the function with the correct execution sequence in the new release.

• ID: TBL_SCA-002

• Severity: Informational

• Type: CS-10 (Gas consumption)

• Description:

Considering the ringIndex is rarely equal with lastRingIndex, the *if* statement can be removed for gas efficiency. In that case the swap operation updates nothing to the variables.

• Remediation:

The TOR team has removed the *if* statement in the new release.

• ID: TBL_SCA-003

• Severity: Informational

• Type: CS-10 (Gas consumption)

• Description:

A variable poolInfo is defined as type List in the contract LPRewardTor.sol. However, only one Pool was appended into the List in Constructor function. There is no other function which is defined to append more items into the List. Therefore, it is not necessary to define the variable as it. Also, function poolLength() is not needed too since it always returns 1.

We advise to

1. re-define the variable just as follows:

   PoolInfo public poolInfo;

2. Remove function poolLength();
3. Remove function massUpdatePools() and update function updatePool() without arguments;
4. Update all usage of poolInfo[0] to be poolInfo;

• Remediation:

The TOR team understands the redundancy of the implementation. However, the team decided not to update the code.

• ID: TBL_SCA-004

• Severity: Informational

• Type: CS-10 (Gas consumption)

• Description:

The require() function is redundant since the sub() function in SafeMath library will do the same validation.

• Remediation:

The TOR team has removed the require() function in the new release.