RD AUDITORS

IoTex Shiba Smart Contract, Code Review and Security Analysis Report

Customer: IoTex Shiba
Prepared on: 20th Nov 2021
Platform: Binance Smart Chain
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer

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 the decision of the customer.

Document

Name	Smart Contract Code Review and Security Analysis Report of IoTex Shiba
Platform	BSC / Solidity
File	IoTexShibaReflection
MD5 hash	EEF5BC95D2B814DC8A3ED94E321BA446
SHA256 hash	AD27060B00925530485C344FBD039BF769F9C461940F4C3DA7EB9F1D56320964
Date	20/11/2021

Introduction

RD Auditors (Consultant) were contracted by IoTex Shiba (Customer) 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 16th - 20th November 2021.

This contract consists of one file.

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 the following;

Total Issues	0
Critical	0
High	0
Medium	0
Low	0
Very Low	5

Code Quality

Please find a link that, within this report Address, safeMath, Ownable and context taken from the popular open source.

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 IoTex Shiba team has not provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way

.

Overall, the code is almost not commented. Commenting can provide rich documentation for functions, return variables and more. Use of the Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

The hash of that file is mentioned in the table. As mentioned above, It's recommended to write comments on 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

IoTexShibaReflection

File And Function Level Report

Interface: IERC20

Observation: Passed

Test Report: Passed

Score: Passed

Conclusion Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	totalSupply	external	Passed	All Passed	No Issue	Passed
2	balanceOf	external	Passed	All Passed	No Issue	Passed
3	transfer	external	Passed	All Passed	No Issue	Passed
4	allowance	external	Passed	All Passed	No Issue	Passed
5	approve	external	Passed	All Passed	No Issue	Passed
6	transferFrom	external	Passed	All Passed	No Issue	Passed

Contract: Ownable

inherit: Context

Observation: Passed

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Owner	read	Passed	All Passed	No Issue	Passed
2	renounceOwnership	onlyowner	Passed	All Passed	No Issue	Passed
3	transferOwnership	onlyowner	Passed	All Passed	No Issue	Passed
4	getUnlocktime	read	Passed	All Passed	No Issue	Passed
5	lock	onlyOwner	Passed	All Passed	No Issue	Passed
6	Unlock	write	Passed	All Passed	No Issue	Passed

Interface: IUniswapV2Factory
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	FeeTo	external	Passed	All Passed	No Issue	Passed
2	FeeToSetter	external	Passed	All Passed	No Issue	Passed
3	getPair	external	Passed	All Passed	No Issue	Passed
4	allPairs	external	Passed	All Passed	No Issue	Passed
5	allPairsLength	external	Passed	All Passed	No Issue	Passed
6	CreatePair	external	Passed	All Passed	No Issue	Passed
7	SetFeeTo	external	Passed	All Passed	No Issue	Passed
8	SetFeeToSetter	external	Passed	All Passed	No Issue	Passed

Contract: IUniswapv2pair
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	external	Passed	All Passed	No Issue	Passed
2	symbol	external	Passed	All Passed	No Issue	Passed
3	decimals	external	Passed	All Passed	No Issue	Passed
4	totalSupply	external	Passed	All Passed	No Issue	Passed
5	balanceOf	external	Passed	All Passed	No Issue	Passed
6	allowance	external	Passed	All Passed	No Issue	Passed
7	approve	external	Passed	All Passed	No Issue	Passed
8	transfer	external	Passed	All Passed	No Issue	Passed
9	transferFrom	external	Passed	All Passed	No Issue	Passed
10	DOMAIN_SEPARATOR	external	Passed	All Passed	No Issue	Passed
11	PERMIT_TYPEHASH	external	Passed	All Passed	No Issue	Passed
12	nonces	external	Passed	All Passed	No Issue	Passed
13	permit	external	Passed	All Passed	No Issue	Passed
14	MINIMUM_LIQUIDITY	external	Passed	All Passed	No Issue	Passed
15	token0	external	Passed	All Passed	No Issue	Passed
16	token1	external	Passed	All Passed	No Issue	Passed
17	getReserves	external	Passed	All Passed	No Issue	Passed
18	Price0CumulativeLast	external	Passed	All Passed	No Issue	Passed
19	Price1CumulativeLast	external	Passed	All Passed	No Issue	Passed

20	KLast	external	Passed	All Passed	No Issue	Passed
21	mint	external	Passed	All Passed	No Issue	Passed
22	burn	external	Passed	All Passed	No Issue	Passed
23	swap	external	Passed	All Passed	No Issue	Passed
24	skim	external	Passed	All Passed	No Issue	Passed
25	sync	external	Passed	All Passed	No Issue	Passed
26	initialize	external	Passed	All Passed	No Issue	Passed

Contract: IUniSwapV2Router01

Observation: Passed

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Factory	external	Passed	All Passed	No Issue	Passed
2	WETH	external	Passed	All Passed	No Issue	Passed
3	addLiquidityETH	external	Passed	All Passed	No Issue	Passed
4	removeLiquidityETH	external	Passed	All Passed	No Issue	Passed
5	removeLiquidityWithPermit	external	Passed	All Passed	No Issue	Passed
6	removeLiquidityETHWithPermit	external	Passed	All Passed	No Issue	Passed

7	SwapExactTokensForTokens	external	Passed	All Passed	No Issue	Passed
8	SwapExactTokensForETH	external	Passed	All Passed	No Issue	Passed
9	SwapETHForExactTokens	external	Passed	All Passed	No Issue	Passed
10	quote	external	Passed	All Passed	No Issue	Passed
11	getAmountOut	external	Passed	All Passed	No Issue	Passed
12	getAmountIn	external	Passed	All Passed	No Issue	Passed
13	getAmountOut	external	Passed	All Passed	No Issue	Passed
14	getAmountIn	external	Passed	All Passed	No Issue	Passed

Contract: IUniSwapV2Router02
Inherit: IUniSwapV2Router01
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	removeLiquidityETHSupportingFeeonTransferTokens	external	Passed	All Passed	No Issue	Passed
2	removeLiquidityETHWithPermitSupportingFeeOn	external	Passed	All Passed	No Issue	Passed
3	TransferTokens	external	Passed	All Passed	No Issue	Passed

4	SwapExactTokensForTokenSupportingFeeOnTransferTokens	external	Passed	All Passed	No Issue	Passed
5	SwapExactETHForTokenSupportingFeeOnTransferTokens	external	Passed	All Passed	No Issue	Passed
6	SwapExactTokensForETHsupportingFeeOnTransferTokens	external	Passed	All Passed	No Issue	Passed

Contract: IoTexShiba
Inherit: Context, IERC20, Ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	read	Passed	All Passed	No Issue	Passed
2	Symbol	read	Passed	All Passed	No Issue	Passed
3	decimals	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	transfer	write	Passed	All Passed	No Issue	Passed
7	allowance	read	Passed	All Passed	No Issue	Passed
8	approve	write	Passed	All Passed	No Issue	Passed
9	transferFrom	write	Passed	All Passed	No Issue	Passed
10	increaseAllowance	write	Passed	All Passed	No Issue	Passed
11	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
12	isExcludedFromReward	read	Passed	All Passed	No Issue	Passed
13	totalFees	read	Passed	All Passed	No Issue	Passed
14	deliver	write	Passed	All Passed	No Issue	Passed
15	reflectionFromToken	read	Passed	All Passed	No Issue	Passed
16	tokenFromReflection	read	Passed	All Passed	No Issue	Passed
17	excludeFromReward	OnlyOwner	Passed	All Passed	No Issue	Passed
18	IncludeInReward	OnlyOwner	Infinite loop possibility	Passed with client consent	Owner must exclude limited wallets	Passed with client consent
19	_transferBothExcluded	write	Passed	All Passed	No Issue	Passed
20	excludeFromFee	OnlyOwner	Passed	All Passed	No Issue	Passed
21	includeInFee	OnlyOwner	Passed	All Passed	No Issue	Passed

22	SetTaxFeePercent	OnlyOwner	Passed	All Passed	No Issue	Passed
23	SetLiquidityFeePercent	OnlyOwner	Passed	All Passed	No Issue	Passed
24	setMaxTxPercent	OnlyOwner	Passed	All Passed	No Issue	Passed
25	setSwapAndLiquifyEnabled	OnlyOwner	Passed	All Passed	No Issue	Passed
26	_reflectFee	write	Passed	All Passed	No Issue	Passed
27	_getValues	read	Passed	All Passed	No Issue	Passed
28	_getTValues	read	Passed	All Passed	No Issue	Passed
29	_getRValues	read	Passed	All Passed	No Issue	Passed
30	_getRate	read	Passed	All Passed	No Issue	Passed
31	_getCurrentSupply	read	Passed	All Passed	No Issue	Passed
32	_takeLiquidity	write	Passed	All Passed	No Issue	Passed
33	CalculateTaxFee	read	Passed	All Passed	No Issue	Passed
34	CalculateLiquidityFee	read	Passed	All Passed	No Issue	Passed
35	removeAll fee	read	Passed	All Passed	No Issue	Passed
36	isExcludedFromFee	read	Passed	All Passed	No Issue	Passed
37	_approve	write	Passed	All Passed	No Issue	Passed
38	_transfer	write	Passed	All Passed	No Issue	Passed
39	swapAndLiquify	read	Passed	All Passed	No Issue	Passed

40	swapTokensForETH	read	Passed	All Passed	No Issue	Passed
41	addLiquidity	read	Passed	All Passed	No Issue	Passed
42	_tokenTransfer	read	Passed	All Passed	No Issue	Passed
43	_transferStandard	read	Passed	All Passed	No Issue	Passed
44	_transferToExcluded	read	Passed	All Passed	No Issue	Passed
45	_transferFromExcluded	read	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:

1) Use the latest solidity version

Using the latest solidity will prevent any compiler-level bugs.

```
10
11   pragma solidity ^0.6.12;
```

Solution

Please use > 0.8.0 which is the latest version.

2) Infinite loop possibility

```
855     function includeInReward(address account) external onlyOwner() {
856         require(_isExcluded[account], "Account is already excluded");
857         for (uint256 i = 0; i < _excluded.length; i++) {
858             if (_excluded[i] == account) {
859                 _excluded[i] = _excluded[_excluded.length - 1];
860                 _tOwned[account] = 0;
861                 _isExcluded[account] = false;
862                 _excluded.pop();
863                 break;
864             }
865         }
```

```
939     function _getCurrentSupply() private view returns(uint256, uint256) {
940         uint256 rSupply = _rTotal;
941         uint256 tSupply = _tTotal;
942         for (uint256 i = 0; i < _excluded.length; i++) {
943             if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
944             rSupply = rSupply.sub(_rOwned[_excluded[i]]);
945             tSupply = tSupply.sub(_tOwned[_excluded[i]]);
946         }
947         if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
948         return (rSupply, tSupply);
949     }
```

If the _excluded array length is increased, then it might hit the gas limit.

Solution

Please keep this excluded wallet as minimum as possible. Ideally under 100 wallets.

3) Address library was not used anywhere; so kindly remove it.

4) Ownership transfer function.

It is good practice to implement an acceptOwnership style to prevent ownership sent to invalid addresses by human error. Code flow similar to below:

```
function transferOwnership(address _newOwner) public onlyOwner {
    newOwner = _newOwner;
}

//the reason for this flow is to protect owners from sending ownership to unintended address due to human error
function acceptOwnership() public {
    require(msg.sender == newOwner);
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
    newOwner = address(0);
}
```

5) Hardcoded addresses should be checked carefully.

```
742     constructor () public {
743         _rOwned[_msgSender()] = _rTotal;
744
745         IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(0x95cB18889B968AbABb9104f30aF5b310bD007Fd8);
```

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”

Note For contract User

Technical auditing does not guarantee the project's ethical side.

- transferOwnership: Owner can transfer ownership to another wallet
- renounceOwnership: Owner can resign and renounce ownership. After this function is called, no other owner functions can be called.
- excludeFromReward: Excludes accounts
- includeInReward: Includes accounts
- lock: Owner can lock the contract for a certain period.

Please do your due diligence before investing. Our audit report is never an investment advice.

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

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.

Email: info@rdauditors.com

Website: www.rdauditors.com