



DogeBNB.org Smart Contract, Code Review and Security Analysis Report

Customer: DogeBNB.org
Prepared on: 24 Nov 2021
Platform: Binance Smart Chain
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer	2
Document	3
Introduction	5
Project Scope	6
Executive Summary	7
Code Quality	8
Documentation	9
Use of Dependencies	10
AS-IS Overview	11
Severity Definitions	17
Audit Findings	18
Conclusion	19
Note For Contract Users	20
Our Methodology	21
Disclaimers	23

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 DogeBNB.org
Platform	BSC / Solidity
File 1	BafNft721.sol
MD5 hash	67F5386D6FB0F3F017139859BAB8EFAC
SHA256 hash	391B71192387503579596EB97CF3044C66F81CEFE223DA33C45E5DF 212DBCDE4
File 2	BafNft1155.sol
MD5 hash	08E90D53D8C6310EC528B3E9FEEF159F
SHA256 hash	2D896D12EFF83099D2234F8C3258DD816506F44A7AA05A9E4209B D45C5960C81
File 3	TransferProxy.sol
MD5 hash	016B62097166DC002B5B82C1DB459F07
SHA256 hash	6DC4F9CAC23FC9B0E2BD616C36193D44EAAA5C85714C3FDD33B7 581EE565237B
File 4	Trade.sol

MD5 hash	516FC0B0BFFA62752DCE4D7AE03D1098
SHA256 hash	358DA1C14DA58C3349135F6C6978CE692BA8B27990B5FFA909E535 B059765137
Date	24/11/2021

Introduction

RD Auditors (Consultant) were contracted by DogeBNB.org (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 17 - 24 Nov 2021.

This contract consists of four files.

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**.



You are Here

 Insecure






 Poorly Secured

 Secure

 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, the 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	0

Code Quality

Please note that within this report

Strings,EnumerableMap,EnumerableSet,Address, SafeMath, IERC165, Context are taken from the popular OpenZeppelin library.

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 DogeBNB.org 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 on. Commenting can provide rich documentation for functions, return variables and more. Use of 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 in the 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

DogeBNB.org

File And Function Level Report

File: BafNft721.sol
Contract: ERC721
Inherit: Context, ERC165, IERC721, IERC721Metadata, IERC721Enumerable
Observation: Passed
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	balanceOf	read	Passed	All Passed	No Issue	Passed
2	ownerOf	read	Passed	All Passed	No Issue	Passed
3	name	read	Passed	All Passed	No Issue	Passed
4	symbol	read	Passed	All Passed	No Issue	Passed
5	tokenURI	read	Passed	All Passed	No Issue	Passed
6	baseURI	read	Passed	All Passed	No Issue	Passed
7	royaltyFee	read	Passed	All Passed	No Issue	Passed
8	getCreator	read	Passed	All Passed	No Issue	Passed
9	tokenOfOwner ByIndex	read	Passed	All Passed	No Issue	Passed
10	totalSupply	read	Passed	All Passed	No Issue	Passed
11	tokenByIndex	read	Passed	All Passed	No Issue	Passed
12	approve	write	Passed	All Passed	No Issue	Passed
13	getApproved	read	Passed	All Passed	No Issue	Passed

14	setApprovalForAll	write	Passed	All Passed	No Issue	Passed
15	isApprovedForAll	read	Passed	All Passed	No Issue	Passed
16	transferFrom	write	Passed	All Passed	No Issue	Passed
17	safeTransferFrom	write	Passed	All Passed	No Issue	Passed
18	safeTransferFrom	write	Passed	All Passed	No Issue	Passed
19	_safeTransfer	write	Passed	All Passed	No Issue	Passed
20	_exists	read	Passed	All Passed	No Issue	Passed
21	_isApprovedOrOwner	read	Passed	All Passed	No Issue	Passed
22	_safeMint	write	Passed	All Passed	No Issue	Passed
23	_safeMint	write	Passed	All Passed	No Issue	Passed
24	_mint	write	Passed	All Passed	No Issue	Passed
25	_burn	write	Passed	All Passed	No Issue	Passed
26	_transfer	write	Passed	All Passed	No Issue	Passed
27	_setTokenURI	write	Passed	All Passed	No Issue	Passed
28	_setBaseURI	write	Passed	All Passed	No Issue	Passed
29	_checkOnERC721Received	write	Passed	All Passed	No Issue	Passed
30	_approve	write	Passed	All Passed	No Issue	Passed
31	ownerTransfership	write	Passed	All Passed	No Issue	Passed
32	verifySign	write	Passed	All Passed	No Issue	Passed
33	createCollectible	write	Passed	All Passed	No Issue	Passed
34	setBaseURI	write	Passed	All Passed	No Issue	Passed
35	burn	write	Passed	All Passed	No Issue	Passed

File: BafNft1155.sol
Contract: ERC1155
Inherit: Context, ERC165, IERC1155, IERC1155Metadata,
IERC1155Enumerable
Observation: Passed
Test Report: 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	_setTokenURI	write	Passed	All Passed	No Issue	Passed
4	royaltyFee	read	Passed	All Passed	No Issue	Passed
5	getCreator	read	Passed	All Passed	No Issue	Passed
6	_setTokenURIPr efix	write	Passed	All Passed	No Issue	Passed
7	tokenURI	read	Passed	All Passed	No Issue	Passed
8	_exists	read	Passed	All Passed	No Issue	Passed
9	balanceOf	read	Passed	All Passed	No Issue	Passed
10	balanceOfBatc h	read	Passed	All Passed	No Issue	Passed
11	setApprovalFor All	write	Passed	All Passed	No Issue	Passed
12	isApprovedForA ll	read	Passed	All Passed	No Issue	Passed
13	safeTransferFro m	write	Passed	All Passed	No Issue	Passed
14	safeBatchTransf erFrom	write	Passed	All Passed	No Issue	Passed

15	_mint	write	Passed	All Passed	No Issue	Passed
16	_mintBatch	write	Passed	All Passed	No Issue	Passed
17	_burn	write	Passed	All Passed	No Issue	Passed
18	_burnBatch	write	Passed	All Passed	No Issue	Passed
19	_beforeTokenTransfer	write	Passed	All Passed	No Issue	Passed
20	_doSafeTransferAcceptanceCheck	write	Passed	All Passed	No Issue	Passed
21	_doSafeBatchTransferAcceptanceCheck	write	Passed	All Passed	No Issue	Passed
22	_asSingletonArray	write	Passed	All Passed	No Issue	Passed
23	ownerTransfership	write	Passed	All Passed	No Issue	Passed
24	verifySign	write	Passed	All Passed	No Issue	Passed
25	mint	write	Passed	All Passed	No Issue	Passed
26	setBaseURI	write	Passed	All Passed	No Issue	Passed
27	burn	write	Passed	All Passed	No Issue	Passed
28	burnBatch	write	Passed	All Passed	No Issue	Passed

File: Trade.sol
Contract: Trade
Observation: Passed
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	buyerServiceFee	read	Passed	All Passed	No Issue	Passed
2	sellerServiceFee	read	Passed	All Passed	No Issue	Passed
3	setBuyerServiceFee	write	Passed	All Passed	No Issue	Passed
4	setSellerServiceFee	write	Passed	All Passed	No Issue	Passed
5	ownerTransfership	write	Passed	All Passed	No Issue	Passed
6	getSigner	read	Passed	All Passed	No Issue	Passed
7	verifySellerSign	write	Passed	All Passed	No Issue	Passed
8	verifyBuyerSign	write	Passed	All Passed	No Issue	Passed
9	getFees	read	Passed	All Passed	No Issue	Passed
10	tradeAsset	write	Passed	All Passed	No Issue	Passed
11	buyAsset	write	Passed	All Passed	No Issue	Passed
12	executeBid	write	Passed	All Passed	No Issue	Passed

File: TransferProxy.sol
Contract: TransferProxy
Observation: Passed
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	changeOperator	write	Passed	All Passed	No Issue	Passed
2	ownerTransfership	write	Passed	All Passed	No Issue	Passed
3	erc721safeTransferFrom	write	Passed	All Passed	No Issue	Passed
4	erc1155safeTransferFrom	write	Passed	All Passed	No Issue	Passed
5	erc20safeTransferFrom	write	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:

No very low severity vulnerabilities were found.

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 Users

Owner has full control over the smart contract. Thus, technical auditing does not guarantee the project's ethical side.

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

RD Auditors Disclaimer

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

