



Valid Proof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

Audit

Security Assessment

January 2022

For



Disclaimer	3
About Us	4
Description	5
Project Engagement	5
Logo	5
Contract Link	5
Vulnerability & Risk Level	6
Methodology	7
Used Code from other Frameworks/Smart Contracts (direct imports)	8
Tested Contract Files	9
Source Lines	10
Risk Level	10
Capabilities	11
Scope of Work	12
Verify Claims	13
Write functions of contract	14
Deployer cannot mint any new tokens	15
Deployer cannot burn or lock user funds	16
Deployer cannot pause the contract	17
Overall checkup (Smart Contract Security)	18
Source Units in Scope	19
Audit Results	20
Audit Comments	21
SWC Attacks	22-25

Disclaimer

Valid Proof reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team. Valid Proof do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc’...)

Valid Proof Audits do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. Valid Proof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

Valid Proof Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. Valid Proof’s position is that each company and individual are responsible for their own due diligence and continuous security. Valid Proof in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	January 2022	<ul style="list-style-type: none">• Layout project• Automated- /Manual-Security• Testing• Summary

Network

Binance Smart Chain (BEP20)

Website

<http://dogecolony.io/>

WhitePaper

http://dogecolony.io/dogecolony_WhitePaper.pdf

Telegram

<https://t.me/dogecolony>

Twitter

https://twitter.com/dogecolony_io

Reddit

<https://www.reddit.com/r/dogecolony>

Description

Doge Colony is designed as a game based passive income generating cryptocurrency. The game is very simple, it is planned to be fun and enjoyable for everyone to play.

DogeCo is a play to earn game about the struggle of our character in 3 different worlds. It is possible to both have fun and win with the coins that the character collects in the game. The game can now be played as a Demo at Level 1-30. Soon the full version will be completed and published.

Project Engagement

During the 23th of January 2022, Doge Colony Team engaged Valid Proof to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. They provided Valid Proof with access to their code repository and whitepaper.

Logo



Contract Link

v1.0

Contract: <https://bscscan.com/>

address/0x7b1ebc0c4bc34964a0673cbef4e1de868e8a8b6

Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Information	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as they were discovered.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:

- i) Review of the specifications, sources, and instructions provided to Valid Proof to make sure we understand the size, scope, and functionality of the smart contract.
- ii) Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- iii) Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Valid Proof describe.

2. Testing and automated analysis that includes the following:

- i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
- ii) Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.

3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

Used Code from other Frameworks/Smart Contracts (direct imports)

Imported packages:

IERC20

IERC20Metadata

Context

Ownable

SafeMath

SafeMathInt

SafeMathUint

Tested Contract Files

All files and codes of the contract, whose address is below, have been tested.

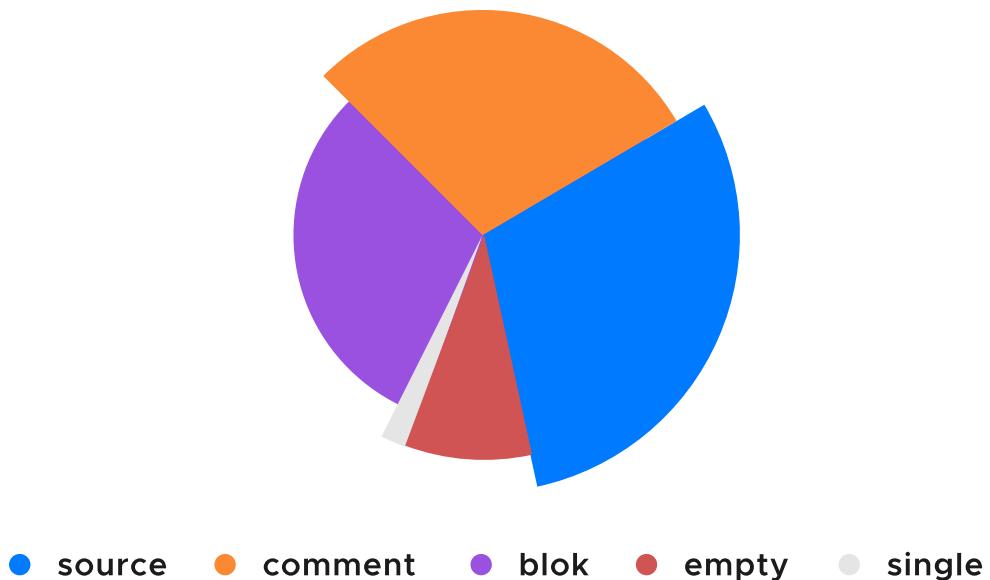
v1.0

File Name	Address
Contract Source Code	0x7b1ebc0c4bc34964a0673cbef4e1de868e8a8b6

Metrics

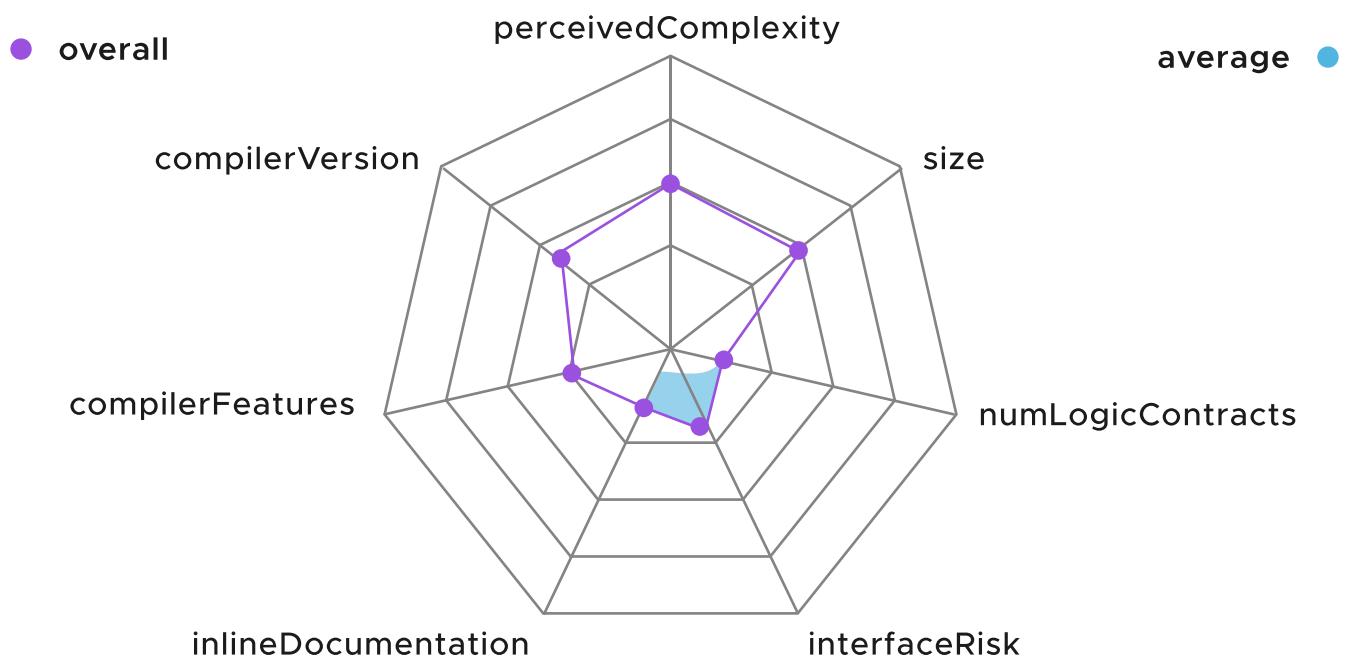
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	^0.8.7			**** (Oasm blocks)	

Version	Transfers ETH	Low- Level Calls	Delega teCall	Uses Hash Functions	ECRec over	New/ Create/ Create 2
1.0	yes					

Scope of Work

The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

1. Correct implementation of Token standard
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Overall checkup (Smart Contract Security)

Verify Claims

Correct implementation of Token standard

Tested	Verified
✓	✓

Function	Description	Exist	Tested	Verified
TotalSupply	provides information about the total token supply	✓	✓	✓
BalanceOf	provides account balance of the owner's account	✓	✓	✓
Transfer	executes transfers of a specified number of tokens to a specified address	✓	✓	✓
TransferFrom	executes transfers of a specified number of tokens from a specified address	✓	✓	✓
Approve	allow a spender to withdraw a set number of tokens from a specified account	✓	✓	✓
Allowance	returns a set number of tokens from a spender to the owner	✓	✓	✓

Write functions of contract

1. approve

13. lock

2. blacklistAddress

14. renounceOwnership

3. checkForCoolDown

15. setFees

4. decreaseAllowance

16. setMarketingWallet

5. deliver

17. setMaxBuyTxAmount

6. enableCooldown

18. setMaxSellTxAmount

7. excludeFromFee

19. setMaxWalletTokend

8. excludeFromReward

20. setMinimumTokensBeforeSwap

9. excludeMultipleAccountsFromFees

21. setSaleFees

10. includeInFee

22. setSellCooldownperiod

11. includeInReward

23. setSwapAndLiquifyEnabled

12. increaseAllowance

24. transfer

25. transferFrom

26. transferOwnership

27. unlock

Deployer cannot mint any new tokens

Name	Exist	Tested	Verified	File
Deployer cannot mint	✓	✓	✓	Main
Comment	Line: -			

Max / Total Supply: 1,000,000,000,000

Deployer cannot burn or lock user funds

Name	Exist	Tested	Verified
Deployer cannot mint	✓	✓	✓
Deployer cannot burn	✓	✓	✓

Deployer cannot pause the contract

Name	Exist	Tested	Verified
Deployer cannot pause	✓	✓	✓



Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Legend

Attribute	Symbol
Verified / Checked	✓
Partly Verified	🚩
Unverified / Not checked	✗
Not available	—

Source Units in Scope

v1.0

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
nSLOC	normalized source lines of code (only source-code lines; no comments, no blank lines)
Comment Lines	lines containing single or block comments
Complexity Score	a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

Audit Results

AUDIT PASSED



Security Issues

Critical Severity Issues

No critical severity issues found.

High Severity Issues

No medium severity issues found.

Medium Severity Issues

No medium severity issues found.

Low Severity Issues

No low severity issues found.

Audit Comments

23 January 2022 :

- It is not a contract that will cause problems for the user. This contract, which protects users and consists of certain limits, is reliable in our opinion.

SWC Attacks

ID	Title	Relationships	Type
SW C-13 6	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SW C-13 5	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SW C-13 4	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SW C-13 3	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SW C-13 2	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SW C-13 1	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SW C-13 0	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SW C-12 9	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SW C-12 8	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

<u>SW C-12 7</u>	Arbitrary Jump with Function Type Variable	<u>CWE-695: Use of Low-Level Functionality</u>	PASSED
<u>SW C-12 5</u>	Incorrect Inheritance Order	<u>CWE-696: Incorrect Behavior Order</u>	PASSED
<u>SW C-12 4</u>	Write to Arbitrary Storage Location	<u>CWE-123: Write-what-where Condition</u>	PASSED
<u>SW C-12 3</u>	Requirement Violation	<u>CWE-573: Improper Following of Specification by Caller</u>	PASSED
<u>SW C-12 2</u>	Lack of Proper Signature Verification	<u>CWE-345: Insufficient Verification of Data Authenticity</u>	PASSED
<u>SW C-12 1</u>	Missing Protection against Signature Replay Attacks	<u>CWE-347: Improper Verification of Cryptographic Signature</u>	PASSED
<u>SW C-12 0</u>	Weak Sources of Randomness from Chain Attributes	<u>CWE-330: Use of Insufficiently Random Values</u>	PASSED
<u>SW C-11 9</u>	Shadowing State Variables	<u>CWE-710: Improper Adherence to Coding Standards</u>	PASSED
<u>SW C-11 8</u>	Incorrect Constructor Name	<u>CWE-665: Improper Initialization</u>	PASSED
<u>SW C-11 7</u>	Signature Malleability	<u>CWE-347: Improper Verification of Cryptographic Signature</u>	PASSED

SW C-11 6	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 5	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SW C-11 4	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SW C-11 3	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SW C-11 2	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 1	Use of Deprecated Solidity Functions	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SW C-11 0	Assert Violation	CWE-824: Access of Uninitialized Pointer	PASSED
SW C-10 9	Uninitialized Storage Pointer	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-10 8	State Variable Default Visibility	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SW C-10 7	Reentrancy	CWE-284: Improper Access Control	PASSED

<u>SW C-10 5</u>	Unprotected Ether Withdrawal	<u>CWE-284: Improper Access Control</u>	PASSED
<u>SW C-10 4</u>	Unchecked Call Return Value	<u>CWE-252: Unchecked Return Value</u>	PASSED
<u>SW C-10 3</u>	Floating Pragma	<u>CWE-664: Improper Control of a Resource Through its Lifetime</u>	PASSED
<u>SW C-10 2</u>	Outdated Compiler Version	<u>CWE-937: Using Components with Known Vulnerabilities</u>	PASSED
<u>SW C-10 1</u>	Integer Overflow and Underflow	<u>CWE-682: Incorrect Calculation</u>	PASSED
<u>SW C-10 0</u>	Function Default Visibility	<u>CWE-710: Improper Adherence to Coding Standards</u>	PASSED



Valid Proof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC