STAYSAFU
AUDIT

May 1st, 2023

DOGE CEO

TABLE OF CONTENTS

AUDIT SUMMARY

This report was written for Doge CEO in order to find flaws and vulnerabilities in the Doge CEO project's source code, as well as any contract dependencies that were not part of an officially recognized library.

A comprehensive examination has been performed, utilizing Static Analysis, Manual Review, and Doge CEO Deployment techniques. The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors
- Assessing the codebase to ensure compliance with current best practices and industry standards
- Ensuring contract logic meets the specifications and intentions of the client
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Through line-by-line manual review of the entire codebase by industry expert

AUDIT OVERVIEW

PROJECT SUMMARY

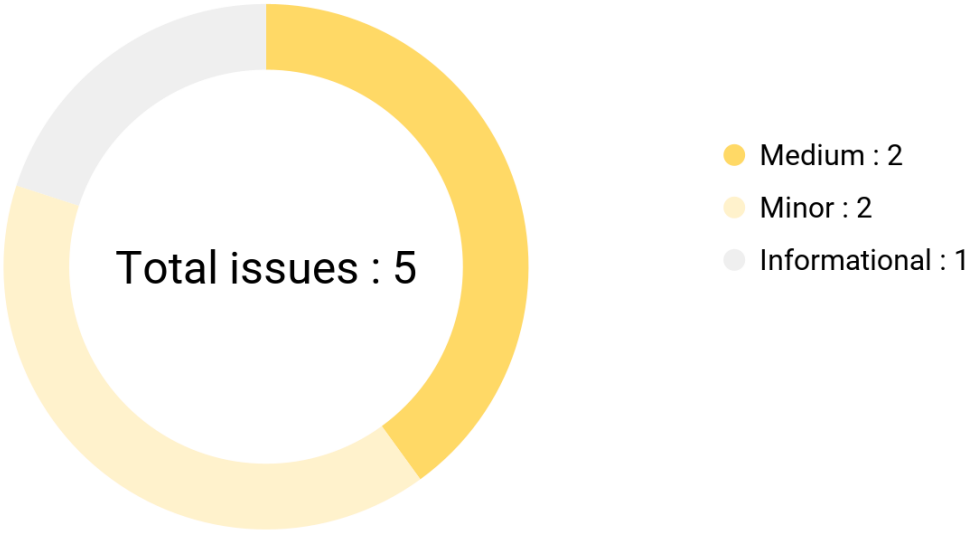
Project name	Doge CEO
Description	Doge CEO a Meme token in the BSC ecosystem, $DOGECEO is community-driven and cannot be controlled by anyone.
Platform	BSC
Language	Solidity
Codebase	https://bscscan.com/token/0x9cbB03eFfD6FB7d79c9baB1b0cEAF4232e957521#code

FINDINGS SUMMARY

Vulnerability	Total
Critical	0
Major	0
Medium	2
Minor	2
Informational	1

EXECUTIVE SUMMARY

Doge CEO is a community project with a goal to advance into every aspect and day to day living of people all over the world. Doge CEO aims to solve this issue by enabling these groups to have access to a secure, SAFE and completely decentralized finance, aimed at giving the power for wealth back to where it belongs – In The Hands of The People. Through harnessing the complete potential of community, decentralization and the Power of Blockchain



AUDIT FINDINGS

Code	Title	Severity
CENT-1	Centralization of major privileges	Medium
LACK-1	Lack of adequate controls over transaction fees	Medium
OPTI-1	Duplicate code	Minor
GAS-1	Optimization of gas costs	Minor
ERRM-1	Error management	Informational

CENT-1 | Centralization of major privileges

Description

The onlyOwner modifier of the smart contract(s) gives major privileges over it (change fees, transferOwnership, removeLimits…)*. This can be a problem, in the case of a hack, an attacker who has taken possession of this privileged account could damage the project and the investors.

**This list is not exhaustive but presents the most sensitive points*

Recommendation

We recommend at least to use a multi-sig wallet as the owner address, and at best to establish a community governance protocol to avoid such centralization. For more information, see https://solidity-by-example.org/app/multi-sig-wallet/

LACK-1 | Lack of adequate controls over transaction fees

Description

The _takeBuyback and _takeMarketing functions in the smart contract appear to control the allocation of transaction fees. Without proper controls, this could allow market manipulation or abuse by the contract owners.

Recommendation

Implement some form of decentralized governance or impose strict limits on these taxes. For example, for the marketing tax, you could set a maximum limit that cannot be exceeded:

uint256 public maxMarketingFee = 5; // Maximum 5% fee for marketing

function _takeMarketing(uint256 tMarketing) private { require(tMarketing <= maxMarketingFee, "Marketing fee exceeds the maximum limit"); // Rest of the function logic }

This ensures that the marketing fee cannot exceed 5%. A similar logic can be applied to _takeBuyback.

OPTI-1 | Duplicate code

Description

Duplicate code is a common problem in many smart contracts. It occurs when the same or very similar blocks of code are present in multiple places in the contract. This makes the contract more difficult to understand, maintain and update. In addition, if an error is present in the duplicated code, it also gets duplicated, which can make the contract vulnerable.

Recommendation

To solve the problem of duplicate code, a good practice is to create internal functions for blocks of code that are used repeatedly. For example, if several parts of the contract perform the same check or calculation, that logic can be extracted into an internal function.

The use of libraries and inheritance can also help avoid duplication of code. If certain functionality is used in multiple contracts, it can be put in a library or base contract, and the other contracts can reference or inherit it.

Finally, the use of static code analysis tools can help identify parts of the code that are duplicated. This allows duplication to be identified and phased out during contract maintenance.

GAS-1 | Optimization of gas costs

Description

The smart contract uses multiple for and while loops to perform various operations. This can lead to excessive gas consumption, as each operation in a loop consumes gas, and this consumption can increase rapidly if the number of iterations is large. In addition, the transfer function is used repeatedly to transfer tokens to different addresses, which also increases gas costs.

Recommendation

To optimize gas costs, it would be possible to reduce the number of operations in the loops or eliminate them altogether if possible. For example, if loops are used to iterate over a set of data, one could consider using a mapping instead of an array to store that data, allowing direct access to a specific value without having to traverse the entire array.

As for the repeated use of the `transfer` function, one solution would be to implement a `batchTransfer` function that would allow tokens to be transferred to multiple addresses in a single transaction. This would significantly reduce the gas costs compared to running multiple individual transactions.

ERRM-1 | Error management

Description

Error handling in the current contract is achieved through the use of the require statement. While this is common practice in smart contracts, the drawback here is that the error messages are not always descriptive. For example, require(owner != address(0), "ERC20: approve from the zero address"); does not provide clear information about the problem, namely why the transaction failed. Also, there is no error handling for situations where a transaction could result in overflow or underflow.

Recommendation

To improve error handling, it would be better to use more descriptive error messages. These would allow users to clearly understand why a transaction failed. For example, "ERC20: approve from the zero address" could be replaced by "ERC20: owner's address cannot be zero when approving".

In addition, the OpenZeppelin SafeMath library could be used to prevent overflow and underflow situations. SafeMath replaces standard arithmetic operations with functions that raise an error in case of overflow or underflow, which improves the security of the contract.

Global security warnings

These are safety issues for the whole project. They are not necessarily critical problems but they are inherent in the structure of the project itself. Potential attack vectors for these security problems should be monitored.

CENT-1 | Global SPOF (Single Point Of Failure)

The project's smart contract has a problem of centralized privileges. The owner system in particular can be subject to attack. To address this security issue we recommend using a multi-sig wallet, establishing secure project administration protocols and strengthening the security of project administrators.

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement.

The Company only to the extent permitted under the terms shall use this report provided in connection with the Services set forth in the Agreement and conditions set forth in the Agreement.

This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without StaySAFU's prior written consent. This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts StaySAFU to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project.

This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

StaySAFU's position is that each company and individual are responsible for their own due diligence and continuous security. StaySAFU's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims, any guarantee of security or fun.