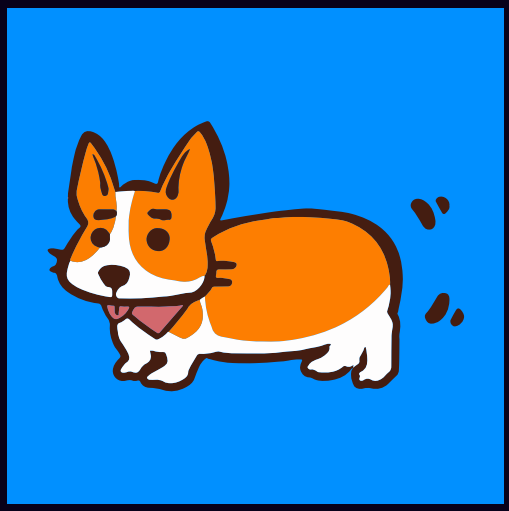

SMART CONTRACT REVIEW

zokyo

March 28th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passed a security audit.

ZOKYO AUDIT SCORING CORGI

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 2 Medium issues: 2 acknowledged = - 6 points deducted
- 1 Low issue: 1 acknowledged = - 2 points deducted
- 0 Informational issues: 0 points deducted

Thus, 100 - 6 - 2 = 92

TECHNICAL SUMMARY

This document outlines the overall security of the CORGI smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the CORGI smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the CORGI team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

Auditing Strategy and Techniques Applied 5

Executive Summary 7

Structure and Organization of the Document 8

Complete Analysis 9

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the CORGI repository:
Repo: https://github.com/FileCorgiPortal/CorgiContract

Last commit: 41bbcde91fff3d30d6c84468a3b0d3bc6d6b3cae

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- Contract.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of CORGI smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contract/s by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

The Zokyo team has performed a security audit of the provided codebase. The contracts submitted for auditing are well-crafted and organized. Detailed findings from the audit process are outlined in the "Complete Analysis" section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed.Acknowledged means that the issue was sent to the CORGI team and the CORGI team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Missing sanity checks and division by Zero Error	Medium	Acknowledged
2	Centralization issue	Medium	Acknowledged
3	Fee is charged on tokens even during Self-transfer	Low	Acknowledged

Missing sanity checks and division by Zero Error

In contract **CORGI,** there are missing sanity checks for **txFee, burnFee** and **FeeAddress**. The **txFee** and **burnFee** can be arbitrarily set to any value, including value above 100. The same sanity checks are also missing in the **updateFee()** function.

```
txFee = _txFee;
burnFee = _burnFee;
FeeAddress = _FeeAddress;
```

If any of **txFee** or **burnFee** is set to a value greater than **100**, then it can lead to **division by Zero Safemath** error.

Line: 421 `uint256 DenverDeflaionaryDecay = tempValue.div(uint256(100 / txFee));`
Line: 428 `uint256 Burnvalue = tempValue.div(uint256(100 / burnFee));`

Additionally, there is missing **zero address** check for **FeeAddress**.

Recommendation:

1. Add sanity checks to ensure that txFee & burnFee cannot be greater than 100 in both the constructor and the updateFee() function.
2. Add a non-zero address check for FeeAddress in the constructor.

Client comment:

The contract has been renounced. so the dev can't change anything in the smart contract.

Centralization issue

In contract **CORGI**, the **txFee & burnFee** can be set to a value that is **arbitrarily** very **high**. This can be done anytime by a malicious owner or compromised admin of the contract.

This can lead to changing of transfer & burn fees without user's notice or consent and can lead to users losing most of their tokens during transfer.

Recommendation:

It is advised to:

a. Add a **MAX** fee value for **txFee** and **burnFee** in **BOTH** the **constructor** and the **updateFee** function. Ideally the MaxFee should be less than **50%** of the tokens being transferred.
b. It is advised to use a multisig wallet for the owner of the contract with configuration of at least 2/3 or 3/5 owners to mitigate accidental loss of private keys. And ensure sufficient decentralization.

Client comment:

The contract has been renounced, so the dev can't change anything in the smart contract.

Fee is charged on tokens even during Self-transfer

There is a transaction and burnFee that **CORGI** token levies when transferring the tokens (when the sender is not **FeeAddress**).

But when a user self transfers tokens i.e. transfers the tokens to himself, the transaction fees is still levied.

This might not be ideal as the transfer fees should only be levied when user transfers tokens to other addresses. This **deviates** from the behaviour of an ideal **ERC20** token.

The following is the **ERC20 Property** failing of the **CORGI** tokens on self transfer of tokens:

```
1) Contract: TestCORGITransferable
     Self transfering tokens using transferFrom works as expected.:

    AssertionError: expected false to equal true
    + expected - actual

    -false
    +true

    at Context.<anonymous> (test/crytic/TestCORGITransferable.js:80:10)
    at processTicksAndRejections (node:internal/process/task_queues:95:5)

2) Contract: TestCORGITransferable
     Self transfering tokens using transfer works as expected.:

    AssertionError: expected false to equal true
    + expected - actual

    -false
    +true

    at Context.<anonymous> (test/crytic/TestCORGITransferable.js:98:10)
    at processTicksAndRejections (node:internal/process/task_queues:95:5)
```

Recommendation:

It is advised to not levy any fees when self transferring of CORGI tokens.

Client comment:
The contract has been renounced, so the dev can't change anything in the smart contract.

	Contract.sol
Reentrance	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the CORGI team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the CORGI team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo