QuillAudits

Audit Report
September, 2021

DOGGIE

Contents

Scope of Audit

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

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.
SmartCheck.

Static Analysis
Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

Gas Consumption
In this step we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

High severity issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium level severity issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

Low level severity issues

Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	1	0	2	6
Acknowledged	0	0	0	0
Closed	0	0	0	0

Introduction

During the period of **August 30, 2021 to September 01, 2021** - QuillAudits Team performed a security audit for DOGGIE smart contracts.

The code for the audit was taken from following the official link:

Note	Date	Code
Version 1	August 30	https://bscscan.com/token/0xe926b137284ebd60fa232a12b8187a1b45ac7927?a=0xc4703cdd05835b58254001542d216ad709222d91#readContract

Issues Found – Code Review / Manual Testing

High severity issues

1. Bypass trading fee and burning fee

   Description

   The burnAmount and feeAmount are calculated by the following operations:

   burnAmount = amount.mul(tradeBurnRatio).div(10000);
   feeAmount = amount.mul(tradeFeeRatio).div(10000);

   Unfortunately, Solidity integer division truncates, therefore, if the amount.mul(tradeBurnRatio) is smaller than 10000. The burnAmount is always Zero, similar to the feeAmount.

   Moreover, the _tradeBurnRatio and _tradeFeeRatio are set by the following statements:

   require(_tradeBurnRatio >= 0 && _tradeBurnRatio <= 5000, "TRADE_BURN_RATIO_INVALID");
   require(_tradeFeeRatio >= 0 && _tradeFeeRatio <= 5000, "TRADE_FEE_RATIO_INVALID");

   The values between this range (0->5000) does not help to prevent the issue, if the amount.mul(tradeBurnRatio) < 10000, the operation will return 0.

   Remediation
   Consider implementing if statements where appropriate to validate all user-controlled input, including constructor, to avoid the potential for erroneous values to result in unexpected behaviors or wasted gas. Please ensure that the amount.mul(tradeBurnRatio) and amount.mul(tradeFeeRatio) are always greater than 10000.

   Status: Open

Medium severity issues

No issues were found.

Low level severity issues

2. Uninformative revert messages in require statements

Description

There is an instance in the codebase where the require statement has ambiguous or imprecise error messages.

L94: require(!_INITIALIZED_, "DODO_INITIALIZED");

Uninformative error messages greatly damage the overall user experience, thus lowering the system's quality. Consider not only fixing the specific instance mentioned above, but also reviewing the entire codebase to make sure every error message is informative and user-friendly.

Status: Open

3. Missing zero address validation

Description

We've detected missing zero address validation for the following parameters:

- _FeeAddress in init()
- user in mint()
- newTeam in changeTeamAccount()

Remediation

Consider implementing require statements where appropriate to validate all user-controlled input, including constructor, to avoid the potential for erroneous values to result in unexpected behaviors or wasted gas.

Status: Open

Informational

4. Typo

Description
There is a typo in MINTABEL.

Remediation
We recommend correcting and changing MINTABEL to MINTABLE.

Status: Open

5. Incorrect versions of Solidity

Solidity version used: 0.6.9.

Description
solc frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex pragma statements.

Remediation
Deploy with any of the following Solidity versions:
- 0.5.16 - 0.5.17
- 0.6.11 - 0.6.12
- 0.7.5 - 0.7.6

Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

Status: Open

6. State Variable Default Visibility

L139: mapping(address => uint256) balances;

Description
The Visibility of the aforementioned variable is not defined. Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

The default is internal for state variables, but it should be made explicit.

Status: Open

7. Public function that could be declared external

Description

Using the external attribute for functions never called from the contract. Therefore, the init() public function that is never called by the contract should be declared external to save gas.

Status: Open

8. Inconsistent coding style

Description

Deviations from the Solidity Style Guide were identified throughout the entire codebase. Taking into consideration how much value a consistent coding style adds to the project's readability, enforcing a standard coding style with the help of linter tools such as Solhint is recommended.

Status: Open

9. Missing docstrings

Description

It is extremely difficult to locate any contracts or functions, as they lack documentation. One consequence of this is that reviewers' understanding of the code's intention is impeded, which is significant because it is necessary to accurately determine both security and correctness.

They are additionally more readable and easier to maintain when wrapped in docstrings. The functions should be documented so that users can understand the purpose or intention of each function, as well as the situations in which it may fail, who is allowed to call it, what values it returns, and what events it emits.

Status: Open

Functional test

Function Names	Testing results
approve()	Passed
burn()	Passed
changeTeamAccount()	Passed
claimOnewership()	Passed
init()	Passed
initOwner()	Passed
mint()	Passed
transfer()	Passed
transferFrom()	Passed
transferOwnership()	Passed

Automated Testing

Slither

```
INFO:Detectors:
CustomERC20.init(address,uint256,string,string,uint8,uint256,uint256,address,bool) (customERC20.sol#150-174) contains a tautology or contradiction:
        - require(bool,string)(_tradeFeeRatio >= 0 && _tradeFeeRatio <= 5000,TRADE_FEE_RATIO_INVALID) (customERC20.sol#168)
CustomERC20.init(address,uint256,string,string,uint8,uint256,uint256,address,bool) (customERC20.sol#150-174) contains a tautology or contradiction:
        - require(bool,string)(_tradeBurnRatio >= 0 && _tradeBurnRatio <= 5000,TRADE_BURN_RATIO_INVALID) (customERC20.sol#167)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction
INFO:Detectors:
CustomERC20._transfer(address,address,uint256).burnAmount (customERC20.sol#218) is a local variable never initialized
CustomERC20._transfer(address,address,uint256).feeAmount (customERC20.sol#219) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
InitializableOwnable.initOwner(address).newOwner (customERC20.sol#105) lacks a zero-check on :
                - _OWNER_ = newOwner (customERC20.sol#107)
InitializableOwnable.transferOwnership(address).newOwner (customERC20.sol#110) lacks a zero-check on :
                - _NEW_OWNER_ = newOwner (customERC20.sol#112)
CustomERC20.init(address,uint256,string,string,uint8,uint256,uint256,address,bool)._team (customERC20.sol#158) lacks a zero-check on :
                - team = _team (customERC20.sol#171)
CustomERC20.changeTeamAccount(address).newTeam (customERC20.sol#256) lacks a zero-check on :
                - team = newTeam (customERC20.sol#259)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
SafeMath.divCeil(uint256,uint256) (customERC20.sol#41-49) is never used and should be removed
SafeMath.sqrt(uint256) (customERC20.sol#62-69) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.6.9 (customERC20.sol#14) is known to contain severe issues (https://solidity.readthedocs.io/en/latest/bugs.html)
solc-0.6.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable InitializableOwnable._OWNER_ (customERC20.sol#81) is not in mixedCase
Variable InitializableOwnable._NEW_OWNER_ (customERC20.sol#82) is not in mixedCase
Variable InitializableOwnable._INITIALIZED_ (customERC20.sol#83) is not in mixedCase
Parameter CustomERC20.init(address,uint256,string,string,uint8,uint256,uint256,address,bool)._creator (customERC20.sol#151) is not in mixedCase
Parameter CustomERC20.init(address,uint256,string,string,uint8,uint256,uint256,address,bool)._initSupply (customERC20.sol#152) is not in mixedCase
Parameter CustomERC20.init(address,uint256,string,string,uint8,uint256,uint256,address,bool)._name (customERC20.sol#153) is not in mixedCase
Parameter CustomERC20.init(address,uint256,string,string,uint8,uint256,uint256,address,bool)._symbol (customERC20.sol#154) is not in mixedCase
Parameter CustomERC20.init(address,uint256,string,string,uint8,uint256,uint256,address,bool)._decimals (customERC20.sol#155) is not in mixedCase
Parameter CustomERC20.init(address,uint256,string,string,uint8,uint256,uint256,address,bool)._tradeBurnRatio (customERC20.sol#156) is not in mixedCase
Parameter CustomERC20.init(address,uint256,string,string,uint8,uint256,uint256,address,bool)._tradeFeeRatio (customERC20.sol#157) is not in mixedCase
Parameter CustomERC20.init(address,uint256,string,string,uint8,uint256,uint256,address,bool)._team (customERC20.sol#158) is not in mixedCase
Parameter CustomERC20.init(address,uint256,string,string,uint8,uint256,uint256,address,bool)._isMintable (customERC20.sol#159) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
transferOwnership(address) should be declared external:
        - InitializableOwnable.transferOwnership(address) (customERC20.sol#110-113)
claimOwnership() should be declared external:
        - InitializableOwnable.claimOwnership() (customERC20.sol#115-120)
init(address,uint256,string,string,uint8,uint256,uint256,address,bool) should be declared external:
        - CustomERC20.init(address,uint256,string,string,uint8,uint256,uint256,address,bool) (customERC20.sol#150-174)
transfer(address,uint256) should be declared external:
        - CustomERC20.transfer(address,uint256) (customERC20.sol#176-179)
balanceOf(address) should be declared external:
        - CustomERC20.balanceOf(address) (customERC20.sol#181-183)
transferFrom(address,address,uint256) should be declared external:
        - CustomERC20.transferFrom(address,address,uint256) (customERC20.sol#185-194)
approve(address,uint256) should be declared external:
        - CustomERC20.approve(address,uint256) (customERC20.sol#196-200)
allowance(address,address) should be declared external:
        - CustomERC20.allowance(address,address) (customERC20.sol#202-204)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Mythril

```
==== Potential denial-of-service if block gas limit is reached ====
SWC ID: 128
Severity: Low
Contract: CustomERC20
Function name: transfer(address,uint256)
PC address: 6687
Estimated Gas Usage: 8714 - 29759
Potential denial-of-service if block gas limit is reached.
A storage modification is executed in a loop. Be aware that the transaction may fail to execute if the loop is unbounded and the necessary gas exceeds the block gas limit.
--------------------
In file: customERC20.sol:230

amount.sub(burnAmount).sub(feeAmount)

--------------------
Initial State:

Account: [CREATOR], balance: 0x1211, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x1, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x4440120000000, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], data: [CONTRACT CREATION], value: 0x0
Caller: [ATTACKER], function: transfer(address,uint256), txdata: 0xa9059cbb000000000000000000000000deadbeefdeadbeefdeadbeefdeadbeefdeadbeef0000000000000000000000000000000000000000000000000000000000000000, value: 0x0
```

Solhint Linter

```
customERC20.sol
   14:1  error    Compiler version 0.6.9 does not satisfy the ^0.5.8 semver requirement  compiler-version
   81:5  warning  Variable name must be in mixedCase                                     var-name-mixedcase
   82:5  warning  Variable name must be in mixedCase                                     var-name-mixedcase
   83:5  warning  Variable name must be in mixedCase                                     var-name-mixedcase
  139:5  warning  Explicitly mark visibility of state                                    state-visibility
  212:9  warning  Error message for require is too long                                  reason-string
  213:9  warning  Error message for require is too long                                  reason-string
  214:9  warning  Error message for require is too long                                  reason-string

✖ 8 problems (1 error, 7 warnings)
```

Results

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the DOGGIE platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the DOGGIE Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Closing Summary

In this report, we have considered the security of the DOGGIE platform. We performed our audit according to the procedure described above.

The audit showed several high, low, and informational severity issues. We highly recommend addressing them. Also, we recommend analyzing the latest version of the platform.

DOGGIE

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com