QuillAudits

Audit Report
May, 2021

CoinDogg

Contents

Audit Details and Target

1. Contract Link :

https://etherscan.io/address/0x620389053d327a103b573b414be1aab0fbed32b8#code

2. Audit Target :

- To find the security bugs and issues regarding security, potential risks and critical bugs.
- Check gas optimization and check gas consumption.
- Check function reusability and code optimisation.
- Test the limit for token transfer and check the accuracy in decimals.
- Check the functions and naming conventions.
- Check the code for proper checks before every function call.
- Event trigger checks for security and logs.
- Checks for constant and function visibility and dependencies.
- Validate the standard functions and checks.
- Check the business logic and correct implementation.
- Automated script testing for multiple use cases including transfers, including values and multi transfer check.
- Automated script testing to check the validations and limit tests before every function call.
- Check the use of data type and storage optimisation.
- Calculation checks for different use cases based on the transaction, calculations and reflected values.

Functions list and audit details

1. Read Functions in Contract:

- totalSupply()
- balanceOf()

- allowance()
- name()
- symbol()
- decimals()
- totalSupply()
- balanceOf()
- owner()

2. Write Function in Contract:

- transfer()
- allowance()
- approve()
- transferFrom()
- increaseAllowance()
- decreaseAllowance()
- renounceOwnership()
- transferOwnership()
- mint()

Overview of The Contract

CoinDogg is an ERC20 compatible token with burn, mint, token transfer, hold token, check balance functionality. This contract follows all ERC20 Standard protocols. Smart contract checks all the balances, addresses before performing any operation.
This smart contract is designed with ownable functions to manage the owner of the smart contract.
Tokens are compatible with all types of ETH supporting wallets. These tokens can be used on all standard trading, staking or exchange platforms.
Smart contracts have checks before sending, transfer, approve and burn the tokens for preventing any unexpected loss to the funds and minimizing the chance of mistake during any transaction.
Gas usage is optimal and minimal for all the functions. The gas price depends on the complexity, time and blocks. This is checked in the ETH standard test net environment of ethereum.

Tokenomics

As per the information provided, the tokens generated will be initially transferred to the contract owner and then further division will be done based on the business logic of the application. Tokens cannot be directly purchased from the smart contract, so there will be an additional or third-party platform that will help users to purchase the tokens. Tokens can be held, transferred and delegated freely. Contract owners can increase and decrease supply based on the business use case and supply management.

Scope of Audit

The scope of this audit was to analyse CoinDogg smart contracts codebase for quality, security, and correctness.

Checked Vulnerabilities

The smart contract is scanned and checked for multiple types of possible bugs and issues. This mainly focuses on issues regarding security, attacks, mathematical errors, logical and business logic issues. Here are some of the commonly known vulnerabilities that are considered:

- TimeStamp dependencies.
- Variable and overflow
- Calculations and checks
- SHA values checks
- Vulnerabilities check for use case
- Standard function checks
- Checks for functions and required checks
- Gas optimisations and utilisation
- Check for token values after transfer
- Proper value updates for decimals
- Array checks
- Safemath checks
- Variable visibility and checks
- Error handling and crash issues
- Code length and function failure check
- Check for negative cases
- D-DOS attacks
- Comments and relevance
- Address hardcoded
- Modifiers check
- Library function use check
- Throw and inline assembly functions
- Locking and unlocking (if any)
- Ownable functions and transfer ownership checks

- Array and integer overflow possibility checks
- Revert or Rollback transactions check

Techniques and Methods

- Manual testing for each and every test cases for all functions.
- Running the functions, getting the outputs and verifying manually for multiple test cases.
- Automated script to check the values and functions based on automated test cases written in JS frameworks.
- Checking standard function and check compatibility with multiple wallets and platforms
- Checks with negative and positive test cases.
- Checks for multiple transactions at the same time and checks d-dos attacks.
- Validating multiple addresses for transactions and validating the results in managed excel.
- Get the details of failed and success cases and compare them with the expected output.
- Verifying gas usage and consumption and comparing with other standard token platforms and optimizing the results.
- Validate the transactions before sending and test the possibilities of

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, Mythril, Slither, SmartCheck.

Issue Categories

High severity issues

Issues that must be fixed before deployment else they can create major issues.

Medium level severity issues

These issues will not create major issues in working but affect the performance of the smart contract.

Low level severity issues

These issues are more suggestions that should be implemented to refine the code in terms of gas, fees, speed and code accuracy

Informational

- Code is written in a very concise way which can be more informative and secure by more checks and modifier use.
- The use of variables and naming conventions can be minimized
- Gas use is proper and optimized as per the testing on the Ropsten test network and checking of the old transaction from this contract.
- **Suggestion**: There must be some locking function in case of any loss or security issue so that the admin can lock the contract in case of any security issue to the token.

Number of issues per severity

	High	Medium	Low	Total Issues
Open	0	0	2	2
Closed	0	0	0	0

Issues Found – Code Review / Manual Testing

High severity issues

No Issue found under this category

Medium severity issues

No Issue found under this category

Low level severity issues

1. The use of SafeMath is recommended for avoiding calculation error.
   Line 287, 388

```
386         );
387         _balances[sender] -= amount;
388         _balances[recipient] += amount;
389
```

```
406
407             _totalSupply += amount;
408             _balances[account] += amount;
```

   Solidity 0.8.0 will revert in case of overflow, but will not let you customize the revert reason. In some cases people may want to pass a reason message such as "ERC20: transfer amount exceeds balance". SafeMath provides these functions.

2. Best practice to not define visibility of constructor

```
554 contract CoinDogg is ERC20, Ownable {
555     constructor(address owner) public ERC20("CoinDogg", "DOG") {
556         _mint(owner, 10000000000 * 10**18);
557
Warning: Visibility for constructor is ignored. If you want the contract to be non-deployable, making it "abstract" is sufficient.
  --> test.sol:555:5:
   |
555|     constructor(address owner) public ERC20("CoinDogg", "DOG") {
   |     ^ (Relevant source part starts here and spans across multiple lines).
```

Closing Summary

Apart from the issue and suggestion, the contract is working fine for the business logic and the related functions. Code can be optimized by minor changes for gas optimization. Passing all checks and test cases for contract security and gas fees.
Compatibility tested and passed with most of the ethereum based wallets and platforms.

Disclaimer

QuillHash audit is not a security warranty, investment advice, or an endorsement of the CoinDogg 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 CoinDogg Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

CoinDogg

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com