HOGE Finance

Code Security Assessment

PREPARED BY:

THE AUDIT INSTITUTE ANALYST TEAM

PREPARED FOR:

THE HOGE.FINANCE TEAM

PREPARED ON:

MARCH 10TH 2021

THE AUDIT INSTITUTE

Report Version 1.1

Table of Contents

Disclaimer

The Audit Institute 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 or project that contracts The Audit Institute to perform a security review.

The Audit Institute Reports 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’s proprietors, business, business model or legal compliance.

The Audit Institute Reports 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.

The Audit Institute 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. The Audit Institute's position is that each company and individual are responsible for their own due diligence and continuous security. The Audit Institute'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 functionality of the technology we agree to analyze. View our full legal terms and conditions at https://audit.institute/

What is included in a report by *The Audit Institute*?

- A document describing the detailed analysis of a particular piece(s) of source code provided to The Audit Institute by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of The Audit Institute has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name & Website	**HOGE** - [Hoge.Finance]
Project Description	HOGE Finance combines meme and frictionless yield farming. There is a token burn on every transaction and HOGE holders get a share of the transaction as a reward.
Platform	Ethereum, Solidity
Compiler Version	v0.6.12
Mainnet Address	0xfAd45E47083e4607302aa43c65fB3106F1cd7607
Delivery Date	March 10th 2021
Method of Audit	Static Analysis, Fuzzing, Manual Review
Consultants Engaged	3

Summary of Findings

Critical	0
Medium	0
Informational	4
Total Issues	**4**

Executive Summary

The Audit Institute analyst team reviewed the code that was deployed on mainnet for the HOGE Finance (token) contract written in Solidity. There is an initial total supply of 1 Trillion HOGE. As this project has already been deployed, at the time of writing this report, the circulating supply is approximately 436 Billion HOGE. This project features a frictionless fee redistribution method. The contract contains a mechanism that rewards users for holding HOGE Token. These rewards are derived from transfer fees that are accumulated from all transactions (except those that are excluded by the owner).

Disclosed in the report below is a full analytical review of the Contract after undergoing various test scenarios and code review. The findings varied in criticality as some were related to solidity code standards and optimization, while others can result in a slight loss of precision.

Contracts in Scope

CONTRACT NAME	CONTRACT DESCRIPTION
HOGE.sol	The HOGE Token Contract

External Vulnerability Findings

Vulnerability Category	Notes	Results
Arbitrary Storage Write	N/A	PASS
Arbitrary Jump	N/A	PASS
Delegate Call to Untrusted Contract	N/A	PASS
Dependence on Predictable Variables	N/A	PASS
Deprecated Opcodes	N/A	PASS
Ether / Token Theft	N/A	PASS
Exceptions	N/A	PASS
External Calls	N/A	PASS
External Service Providers	N/A	PASS
Flash Loans	N/A	PASS
Inconsistent Emission of Events	N/A	PASS
Integer Over/Underflow	N/A	PASS
Multiple Sends	N/A	PASS
Oracles	N/A	PASS
Reentrancy Issues	N/A	PASS
Unchecked Retval	N/A	PASS
Suicide	N/A	PASS
State Change External Calls	N/A	PASS
Unchecked Retval	N/A	PASS

Findings

Finding Name	Criticality	Analyst Notes
Multiplication should be done before division	Informational	_getTValues(uint256) (Line #637-641) performs a multiplication on the result of a division: uint256 tFee = tAmount.div(100).mul(2) (Line #638) Dividing usually leads to integer truncation, which can result in calculations that are less precise. We recommend multiplying before division to reduce the risk as much as possible. *Recommendation: The line can be re-written as follows:* uint256 tFee = tAmount.mul(2).div(100)
Redundant Expression	Informational	The last `else` branch in this `if` statement will never be called as all the possible logical combinations of `_isExcluded[sender]` and `_isExcluded[recipient]` are covered by 4 `else if` branches. We recommend removing the highlighted portion of the code (below) as the body of this `else if` matches with the body of the `else`, so in the event that `!_isExcluded[sender] && !_isExcluded[recipient]` is `true`, this condition will be caught by the `else` block and the intended code will fire.

```
function _transfer(address sender, address recipient, uint256 amount) private {
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");
    require(amount > 0, "Transfer amount must be greater than zero");
    if (_isExcluded[sender] && !_isExcluded[recipient]) {
        _transferFromExcluded(sender, recipient, amount);
    } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
        _transferToExcluded(sender, recipient, amount);
    } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
        _transferStandard(sender, recipient, amount);
    } else if (_isExcluded[sender] && _isExcluded[recipient]) {
        _transferBothExcluded(sender, recipient, amount);
    } else {
        _transferStandard(sender, recipient, amount);
    }
}
```

Findings (Continued)

Functions should be `external`	Informational	name() (Line #457-459) symbol() (Line #461-463) decimals() (Line #465-467) totalSupply() (Line #469-471) balanceOf(address) (Line #473-476) transfer(address,uint256) (Line #478-481) allowance(address,address) (Line #483-485) approve(address,uint256) (Line #487-490) transferFrom(address,address,uint256) (Line #492-496) increaseAllowance(address,uint256) (Line #498-501) decreaseAllowance(address,uint256) (Line #503-506) isExcluded(address) (Line #508-510) totalFees() (Line #512-514) reflect(uint256) (Line #516-523) reflectionFromToken(uint256,bool) (Line #525-534) **Recommendation: set these Functions as external to slightly reduce gas cost.*

Variables should be `constant`	Informational	The following variables should be set to constant:
		`HOGE.decimals`
		`HOGE.name`
		`HOGE.symbol`
		**Recommendation: set these variables as constant to slightly reduce gas cost.*

Functions Overview

($) = payable function
 # = non-constant function
Int = Internal
Ext = External
Pub = Public

+ [Int] IERC20
- [Ext] totalSupply
- [Ext] balanceOf
- [Ext] transfer #
- [Ext] allowance
- [Ext] approve #
- [Ext] transferFrom #

+ [Lib] SafeMath
- [Int] add
- [Int] sub
- [Int] sub
- [Int] mul
- [Int] div
- [Int] div
- [Int] mod
- [Int] mod

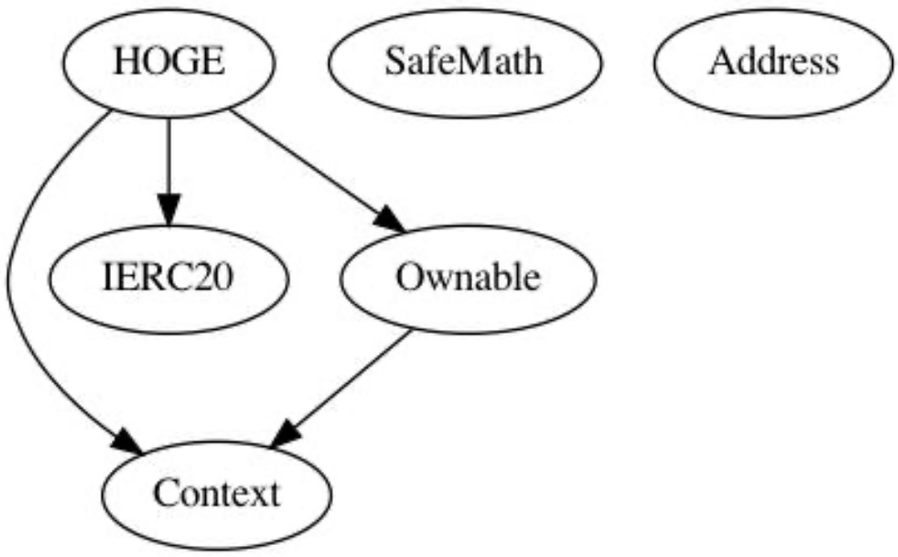
+ Context
- [Int] _msgSender
- [Int] _msgData

+ Ownable (Context)
- [Int] <Constructor> #
- [Pub] owner
- [Pub] renounceOwnership #
  - modifiers: onlyOwner
- [Pub] transferOwnership #
  - modifiers: onlyOwner

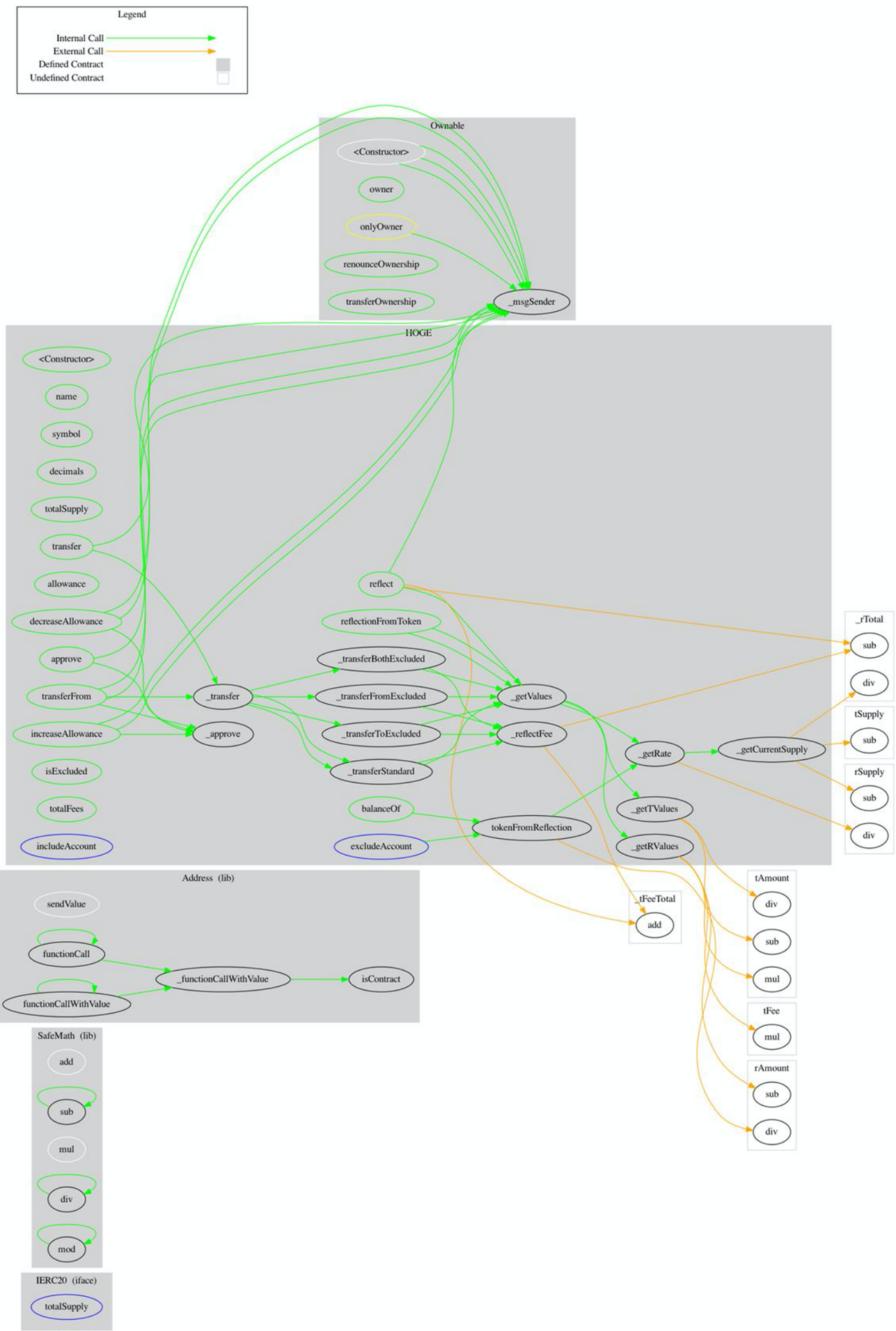
+ [Lib] Address
- [Int] isContract
- [Int] sendValue #
- [Int] functionCall #
- [Int] functionCall #
- [Int] functionCallWithValue #
- [Int] functionCallWithValue #
- [Prv] _functionCallWithValue #

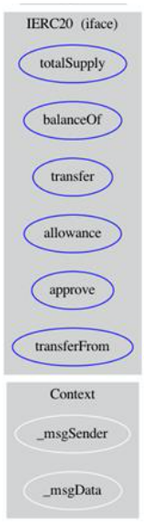
+ HOGE (Context, IERC20, Ownable)
- [Pub] <Constructor> #
- [Pub] name
- [Pub] symbol
- [Pub] decimals
- [Pub] totalSupply
- [Pub] balanceOf
- [Pub] transfer #
- [Pub] allowance
- [Pub] approve #
- [Pub] transferFrom #
- [Pub] increaseAllowance #
- [Pub] decreaseAllowance #
- [Pub] isExcluded
- [Pub] totalFees
- [Pub] reflect #
- [Pub] reflectionFromToken
- [Pub] tokenFromReflection
- [Ext] excludeAccount #
  - modifiers: onlyOwner
- [Ext] includeAccount #
  - modifiers: onlyOwner
- [Prv] _approve #
- [Prv] _transfer #
- [Prv] _transferStandard #
- [Prv] _transferToExcluded #
- [Prv] _transferFromExcluded #
- [Prv] _transferBothExcluded #
- [Prv] _reflectFee #
- [Prv] _getValues
- [Prv] _getTValues
- [Prv] _getRValues
- [Prv] _getRate
- [Prv] _getCurrentSupply

Inheritance Graph



Control Flow





END OF REPORT