EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	Smart Doge
Website:	https://smartdoge.in
Platform:	Binance Smart Chain
Language:	Solidity
Date:	August 9th, 2021

Table of contents

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO PUBLIC AFTER ISSUES ARE RESOLVED.

Introduction

EtherAuthority was contracted by the Smart Doge Token team to perform the Security audit of the Smart Doge Token smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on August 9th, 2021.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

Smart Doge (SDOGE) Token is a BEP20 token smart contract based on Binance Smart Chain (BSC), having features like deflection, burn, etc. SDOGE token can be used as a backbone of the tokenomics.

Audit scope

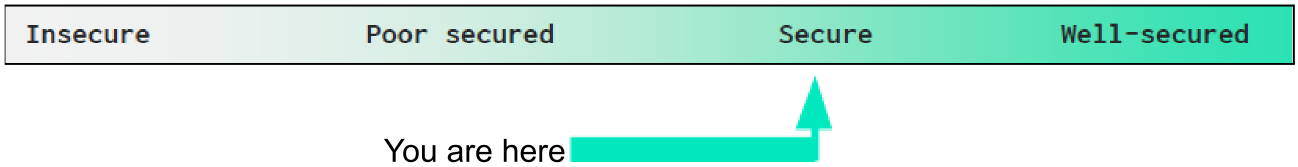
Name	**Code Review and Security Analysis Report for Smart Doge Token Smart Contract**
Platform	**BSC / Solidity**
File	Sdoge.sol
Smart Contract Online Code	https://bscscan.com/address/0x59ce867c6a6cdde4a857cf3271bd00043423f798#code
File MD5 Hash	2C26FE90F1A90FD082618F34FAF043DF
Audit Date	August 9th, 2021
Revised Contract Code	https://bscscan.com/address/0x7bdee7152d3ab496cc879cfac1546f0bbd8ad5f2#code
Revision Date	August 9th, 2021

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Name: SDOGE	**This can be: SMART DOGE**
Symbol: SMART DOGE	**This can be: SDOGE**
Decimals: 18	**YES, This is valid.**
Deflection: 10% on every token transfer	**YES, This is Valid. But this percentage can be changed by the owner anytime.**
● The Owner can access functions like changeBurnRate, transferownership	**YES, This is valid. The smart contract owner controls these functions, so the owner must handle the private key of the owner's wallet very securely. Because if the private key is compromised, then it will create problems.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **“Secured”**. These contracts also have owner functions (described in the centralization section below), which does not make everything 100% decentralized. Thus, the owner must execute those smart contract functions as per the business plan.



We used various tools like MythX, Slither and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 5 low and some very low level issues.

They fixed/acknowledged in the revised contract code.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Moderated
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Moderated
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Other code specification issues	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract. This smart contract also contains Libraries, Smart contracts inherits and Interfaces. These are compact and well written contracts.

The libraries in Smart Doge Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Smart Doge Token token.

The Smart Doge Token team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Some code parts are **not well** commented on smart contracts.

Documentation

We were given a Smart Doge smart contracts code in the form of a github code. The hashes of that code are mentioned above in the table.

As mentioned above, some code parts are **not well** commented. So it is difficult to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects. And their core code blocks are written well.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

Sdoge.sol

(1) Interface

(a) IERC20

(2) Usages

(a) using SafeMath for uint256;

(3) Functions

Sl.	Functions	Type	Observation	Conclusion
1	changeBurnRate	write	Function input parameters lack of check	No Issue
2	name	read	Passed	No Issue
3	symbol	read	Passed	No Issue
4	decimals	read	Passed	No Issue
5	totalSupply	read	Passed	No Issue
6	balanceOf	read	Passed	No Issue
7	transfer	write	SafeMath library, No error message	Refer Audit Findings
8	transferFrom	write	SafeMath library, No error message	Refer Audit Findings
9	approve	write	Passed	No Issue
10	allowance	read	Passed	No Issue
11	burn	write	SafeMath library	Refer Audit Findings
12	burnFrom	write	No error message	Refer Audit Findings
13	transferownership	write	Renounce Ownership	Refer Audit Findings

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical

No Critical severity vulnerabilities were found.

High

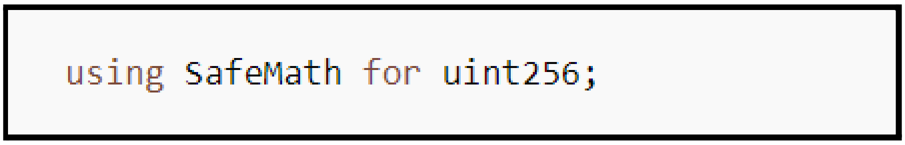
No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) SafeMath library:



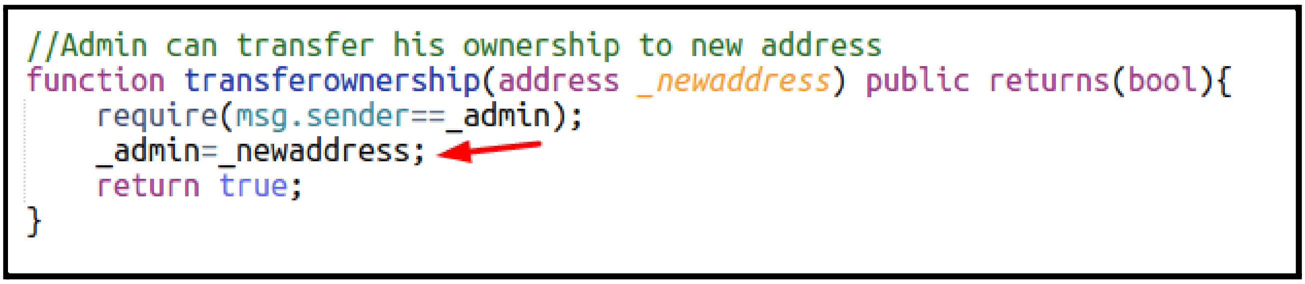
```
using SafeMath for uint256;
```

SafeMath library is used in the code. But Solidity version 0.8.0+ has in-built support for overflow/underflow prevention. And thus this library is not needed in that case.

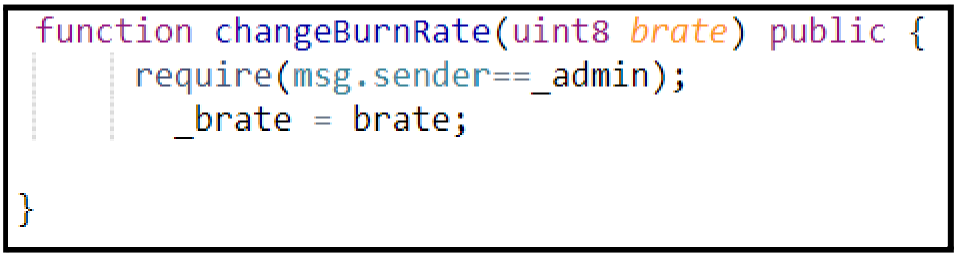
Resolution: We recommend removing the safemath library as it saves some gas.

Status: acknowledged.

(2) Function input parameters lack of check:



```
//Admin can transfer his ownership to new address
function transferownership(address _newaddress) public returns(bool){
    require(msg.sender==_admin);
    _admin=_newaddress;
    return true;
}
```



```
 function changeBurnRate(uint8 brate) public {
        require(msg.sender==_admin);
          _brate = brate;

}
```

Variable validation is not performed in below functions:transferownership,changeBurnRate.

Resolution: Put validation: for transferownership, variable must not address(0). And for changeBurnRate, set the variable's range...

Status: acknowledged.

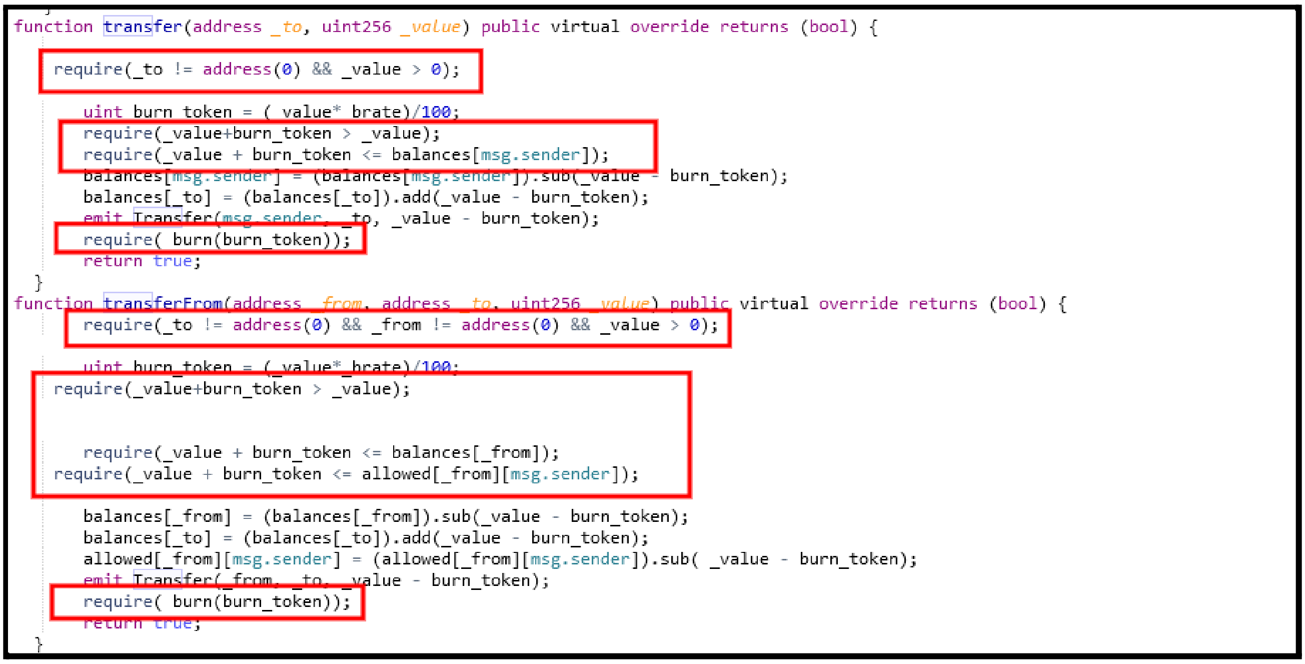
(3) Specify visibility external instead of public:

Resolution: All functions which are not called internally, must be declared as external. It is more efficient as sometimes it saves some gas.

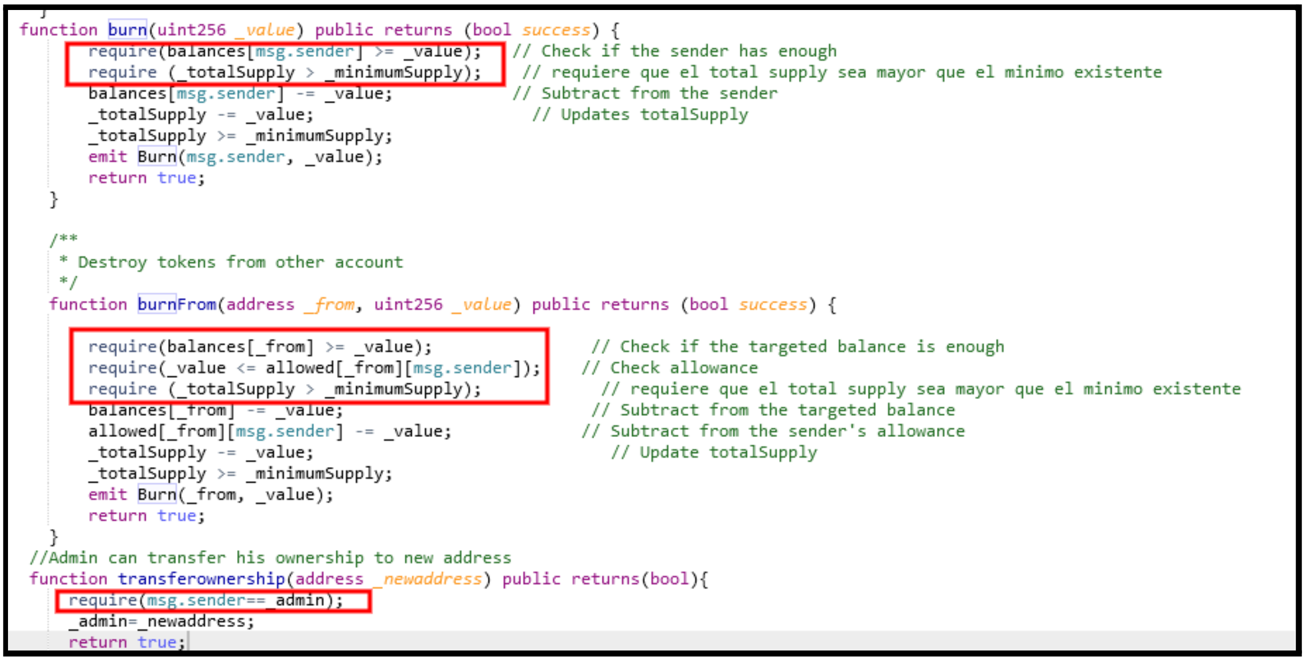
https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

Status: acknowledged.

(4) No error message:



```
function transfer(address _to, uint256 _value) public virtual override returns (bool) {

    require(_to != address(0) && _value > 0);

       uint burn_token = (_value* brate)/100;
       require(_value+burn_token > _value);
       require(_value + burn_token <= balances[msg.sender]);
       balances[msg.sender] = (balances[msg.sender]).sub(_value - burn_token);
       balances[_to] = (balances[_to]).add(_value - burn_token);
       emit Transfer(msg.sender, _to, _value - burn_token);
       require( burn(burn_token));
       return true;
  }
function transferFrom(address _from, address _to, uint256 _value) public virtual override returns (bool) {
       require(_to != address(0) && _from != address(0) && _value > 0);

       uint burn_token = (_value*_brate)/100;
    require(_value+burn_token > _value);


       require(_value + burn_token <= balances[_from]);
    require(_value + burn_token <= allowed[_from][msg.sender]);

       balances[_from] = (balances[_from]).sub(_value - burn_token);
       balances[_to] = (balances[_to]).add(_value - burn_token);
       allowed[_from][msg.sender] = (allowed[_from][msg.sender]).sub( _value - burn_token);
       emit Transfer(_from, _to, _value - burn_token);
       require( burn(burn_token));
       return true;
  }
```



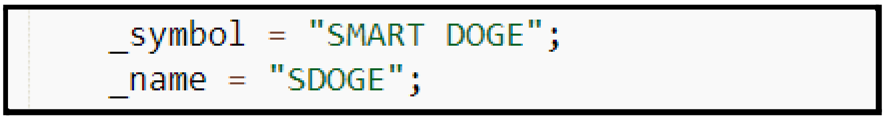
```
function burn(uint256 _value) public returns (bool success) {
        require(balances[msg.sender] >= _value);   // Check if the sender has enough
        require (_totalSupply > _minimumSupply);    // requiere que el total supply sea mayor que el minimo existente
        balances[msg.sender] -= _value;            // Subtract from the sender
        _totalSupply -= _value;                      // Updates totalSupply
        _totalSupply >= _minimumSupply;
        emit Burn(msg.sender, _value);
        return true;
    }

    /**
     * Destroy tokens from other account
     */
    function burnFrom(address _from, uint256 _value) public returns (bool success) {

        require(balances[_from] >= _value);                // Check if the targeted balance is enough
        require(_value <= allowed[_from][msg.sender]);    // Check allowance
        require (_totalSupply > _minimumSupply);            // requiere que el total supply sea mayor que el minimo existente
        balances[_from] -= _value;                         // Subtract from the targeted balance
        allowed[_from][msg.sender] -= _value;             // Subtract from the sender's allowance
        _totalSupply -= _value;                              // Update totalSupply
        _totalSupply >= _minimumSupply;
        emit Burn(_from, _value);
        return true;
    }
  //Admin can transfer his ownership to new address
  function transferownership(address _newaddress) public returns(bool){
      require(msg.sender==_admin);
      _admin=_newaddress;
      return true;
```

There are some places where require is used without giving proper error message

Resolution: There should be a relevant error message.

Status: acknowledged.

(5) Token name and symbol mismatch:



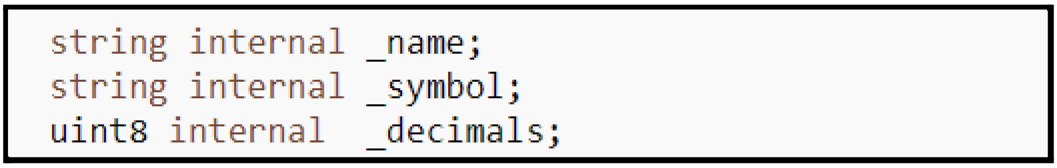
```
    _symbol = "SMART DOGE";
    _name = "SDOGE";
```

Name and symbol should be swapped. So, _symbol must be “SDOGE” and _name must be “SMART DOGE”.

Status: Fixed

Very Low / Discussion / Best practices:

(1) Make variable constant:



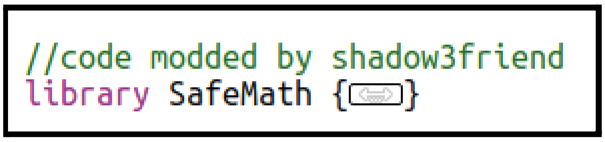
```
 string internal _name;
 string internal _symbol;
 uint8 internal  _decimals;
```

This variable's value will be unchanged. So, please make it constant. It will save some gas.

Resolution: Declare that variable as constant. Just put a constant keyword.

Status: acknowledged.

(2) Irrelevant comment:



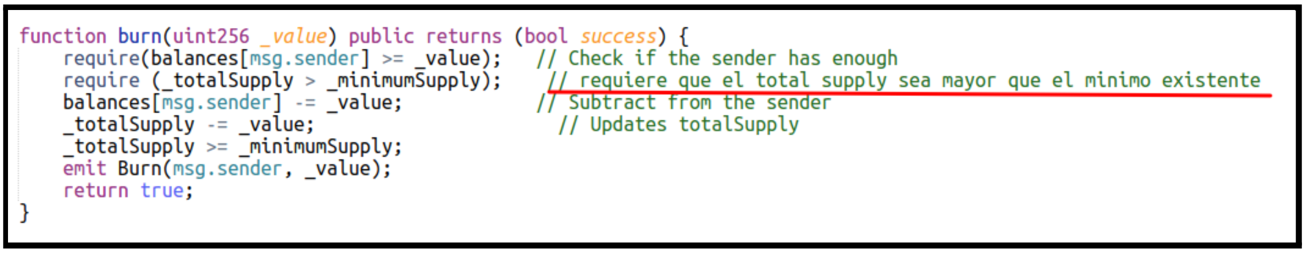
```
//code modded by shadow3friend
library SafeMath {⇔}
```

An irrelevant comment is there.

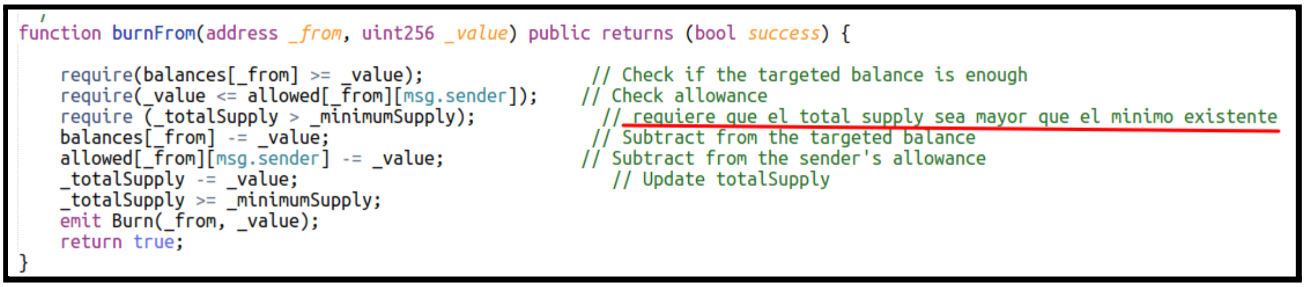
Resolution: Although this does not raise any vulnerability, but we recommend to remove any irrelevant code to make the code clean.

Status: acknowledged.

(3) Comment message in different language:



```
function burn(uint256 _value) public returns (bool success) {
    require(balances[msg.sender] >= _value);   // Check if the sender has enough
    require (_totalSupply > _minimumSupply);    // requiere que el total supply sea mayor que el minimo existente
    balances[msg.sender] -= _value;            // Subtract from the sender
    _totalSupply -= _value;                      // Updates totalSupply
    _totalSupply >= _minimumSupply;
    emit Burn(msg.sender, _value);
    return true;
}
```



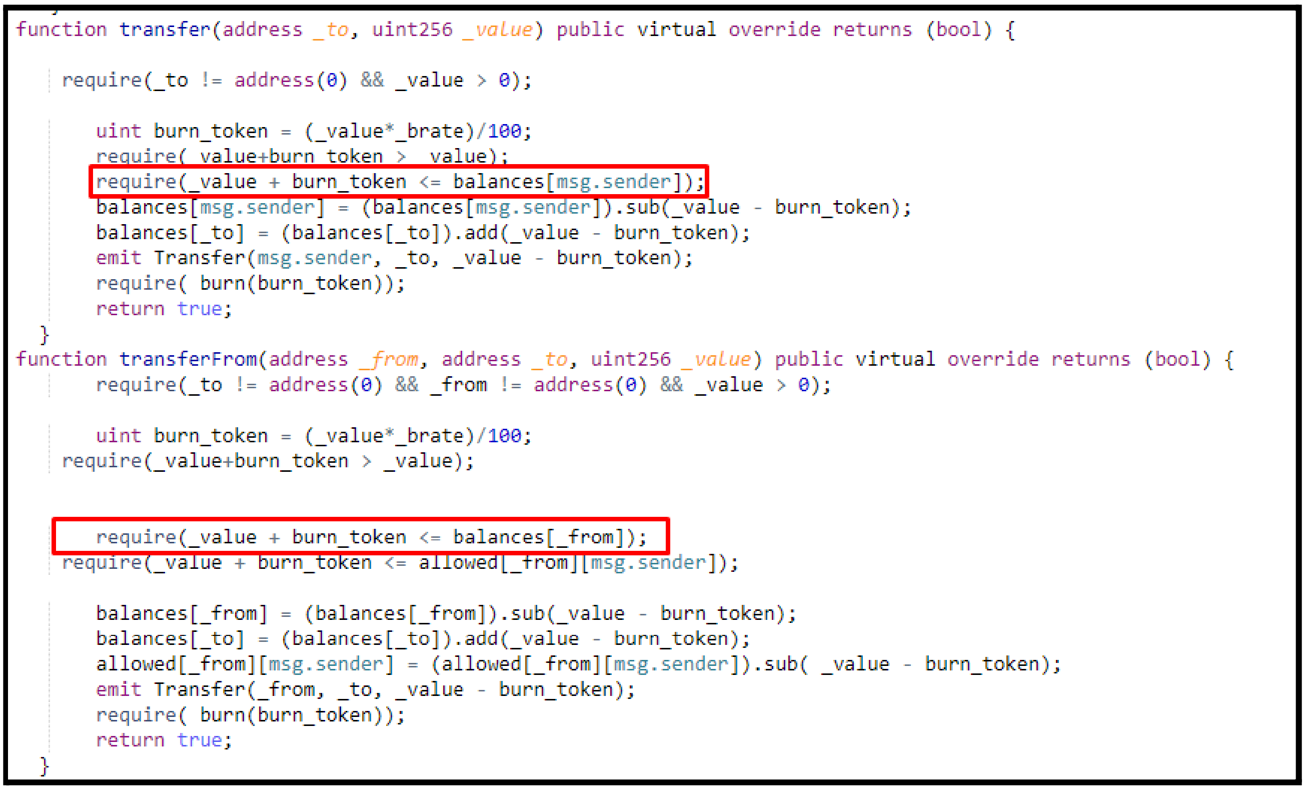
```
function burnFrom(address _from, uint256 _value) public returns (bool success) {

    require(balances[_from] >= _value);                // Check if the targeted balance is enough
    require(_value <= allowed[_from][msg.sender]);    // Check allowance
    require (_totalSupply > _minimumSupply);            //_requiere que el total supply sea mayor que el minimo existente
    balances[_from] -= _value;                         // Subtract from the targeted balance
    allowed[_from][msg.sender] -= _value;             // Subtract from the sender's allowance
    _totalSupply -= _value;                              // Update totalSupply
    _totalSupply >= _minimumSupply;
    emit Burn(_from, _value);
    return true;
}
```

Comment in Spanish Language.

Resolution: Consistency, Use the same language for all comments.

Status: acknowledged.

(4) Amount check while transfer:



```
function transfer(address _to, uint256 _value) public virtual override returns (bool) {

    require(_to != address(0) && _value > 0);

      uint burn_token = (_value*_brate)/100;
      require(_value+burn_token > _value);
      require(_value + burn_token <= balances[msg.sender]);
      balances[msg.sender] = (balances[msg.sender]).sub(_value - burn_token);
      balances[_to] = (balances[_to]).add(_value - burn_token);
      emit Transfer(msg.sender, _to, _value - burn_token);
      require( burn(burn_token));
      return true;
  }
function transferFrom(address _from, address _to, uint256 _value) public virtual override returns (bool) {
      require(_to != address(0) && _from != address(0) && _value > 0);

      uint burn_token = (_value*_brate)/100;
    require(_value+burn_token > _value);


      require(_value + burn_token <= balances[_from]);
    require(_value + burn_token <= allowed[_from][msg.sender]);

      balances[_from] = (balances[_from]).sub(_value - burn_token);
      balances[_to] = (balances[_to]).add(_value - burn_token);
      allowed[_from][msg.sender] = (allowed[_from][msg.sender]).sub( _value - burn_token);
      emit Transfer(_from, _to, _value - burn_token);
      require( burn(burn_token));
      return true;
  }
```

By this, the user cannot transfer his whole amount. Ideally, this burn amount should be deducted from the transfer amount. So, users can transfer the whole amount from their wallet.

Resolution: If this is a part of the plan, then okay.

Status: acknowledged.

Centralization

These smart contracts have some functions which can be executed by Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- changeBurnRate: The Owner can set token burn rate.
- transferownership: The Owner transfers ownership.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We observed some issues in the smart contracts and those issues are fixed in revised code. So, **it's good to go to production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

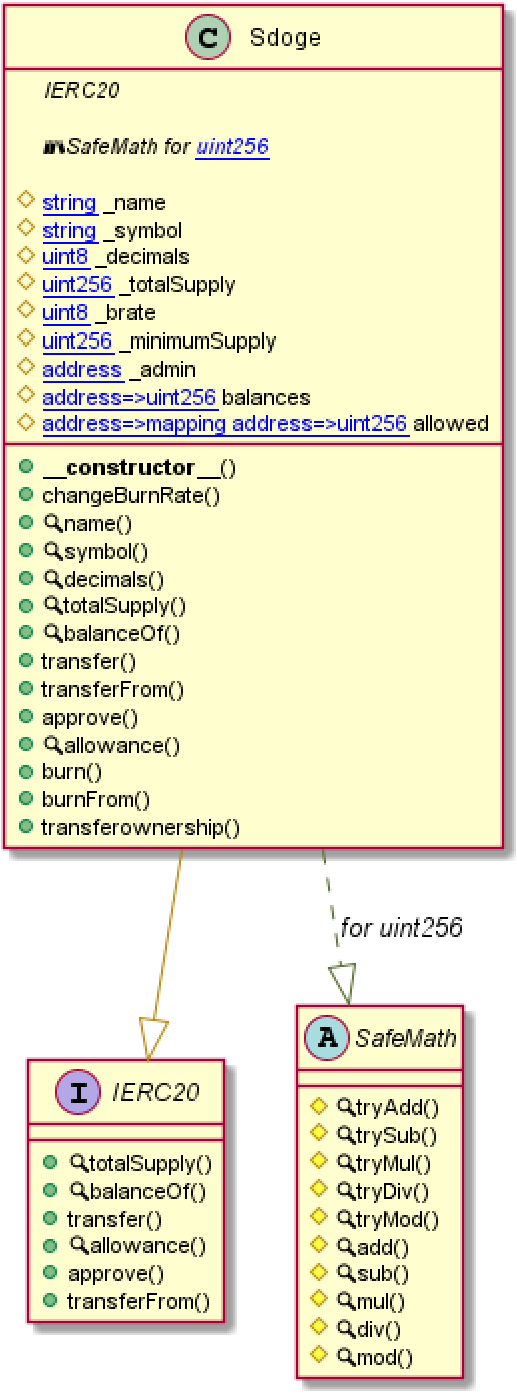
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

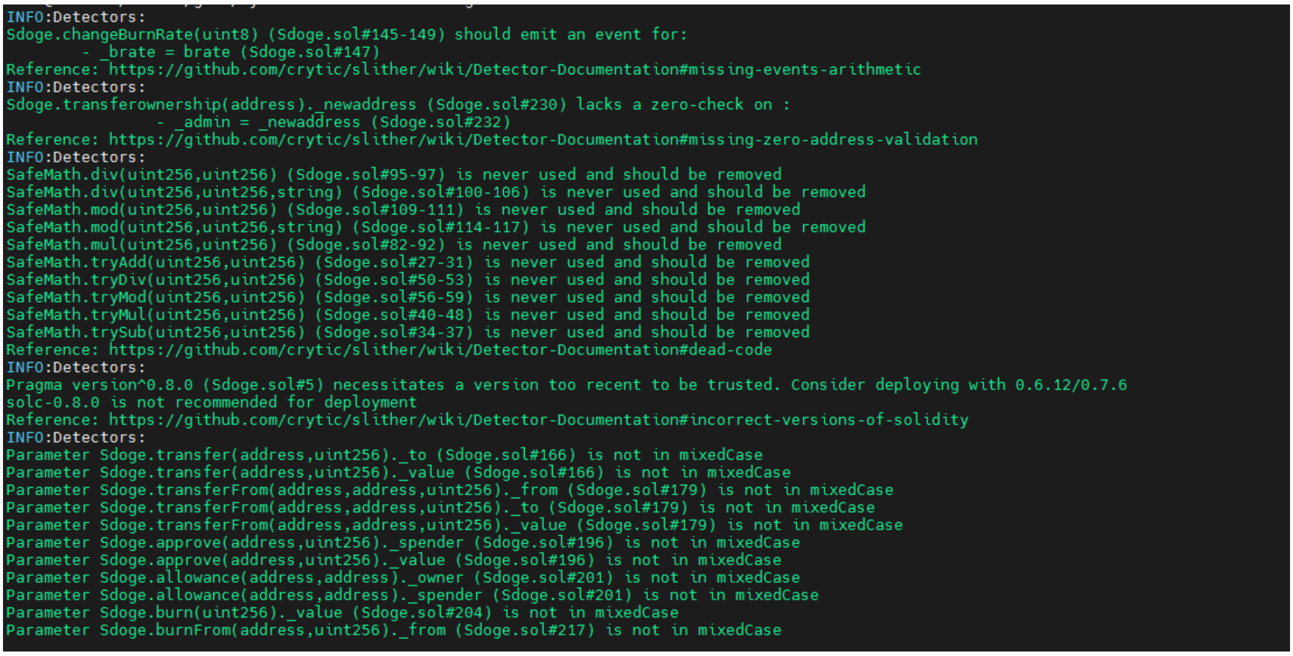
Appendix

Code Flow Diagram - Smart Doge

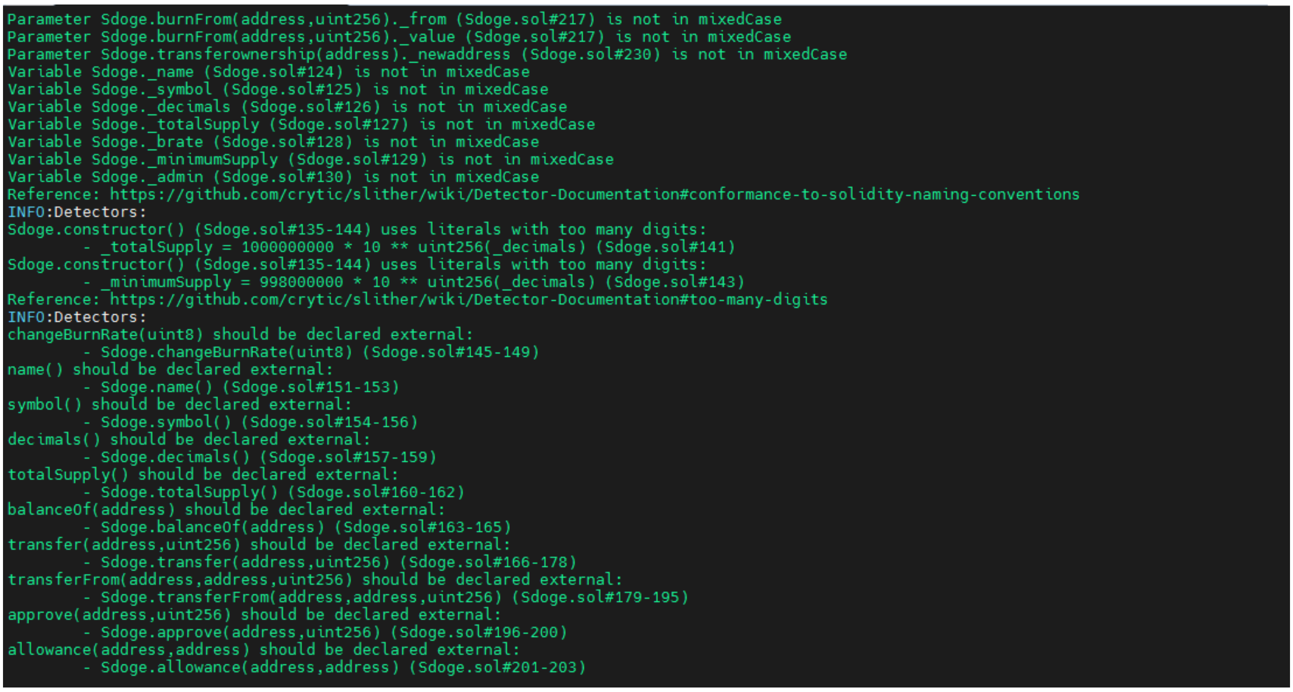


Slither Results Log

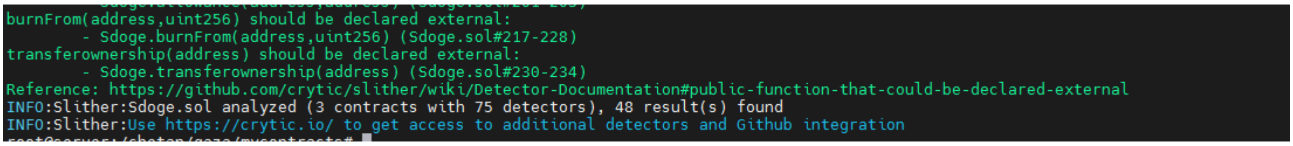
Slither log >> Sdoge.sol



```
INFO:Detectors:
Sdoge.changeBurnRate(uint8) (Sdoge.sol#145-149) should emit an event for:
        - _brate = brate (Sdoge.sol#147)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
Sdoge.transferownership(address)._newaddress (Sdoge.sol#230) lacks a zero-check on :
                - _admin = _newaddress (Sdoge.sol#232)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
SafeMath.div(uint256,uint256) (Sdoge.sol#95-97) is never used and should be removed
SafeMath.div(uint256,uint256,string) (Sdoge.sol#100-106) is never used and should be removed
SafeMath.mod(uint256,uint256) (Sdoge.sol#109-111) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (Sdoge.sol#114-117) is never used and should be removed
SafeMath.mul(uint256,uint256) (Sdoge.sol#82-92) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (Sdoge.sol#27-31) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (Sdoge.sol#50-53) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (Sdoge.sol#56-59) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (Sdoge.sol#40-48) is never used and should be removed
SafeMath.trySub(uint256,uint256) (Sdoge.sol#34-37) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (Sdoge.sol#5) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter Sdoge.transfer(address,uint256)._to (Sdoge.sol#166) is not in mixedCase
Parameter Sdoge.transfer(address,uint256)._value (Sdoge.sol#166) is not in mixedCase
Parameter Sdoge.transferFrom(address,address,uint256)._from (Sdoge.sol#179) is not in mixedCase
Parameter Sdoge.transferFrom(address,address,uint256)._to (Sdoge.sol#179) is not in mixedCase
Parameter Sdoge.transferFrom(address,address,uint256)._value (Sdoge.sol#179) is not in mixedCase
Parameter Sdoge.approve(address,uint256)._spender (Sdoge.sol#196) is not in mixedCase
Parameter Sdoge.approve(address,uint256)._value (Sdoge.sol#196) is not in mixedCase
Parameter Sdoge.allowance(address,address)._owner (Sdoge.sol#201) is not in mixedCase
Parameter Sdoge.allowance(address,address)._spender (Sdoge.sol#201) is not in mixedCase
Parameter Sdoge.burn(uint256)._value (Sdoge.sol#204) is not in mixedCase
Parameter Sdoge.burnFrom(address,uint256)._from (Sdoge.sol#217) is not in mixedCase
```



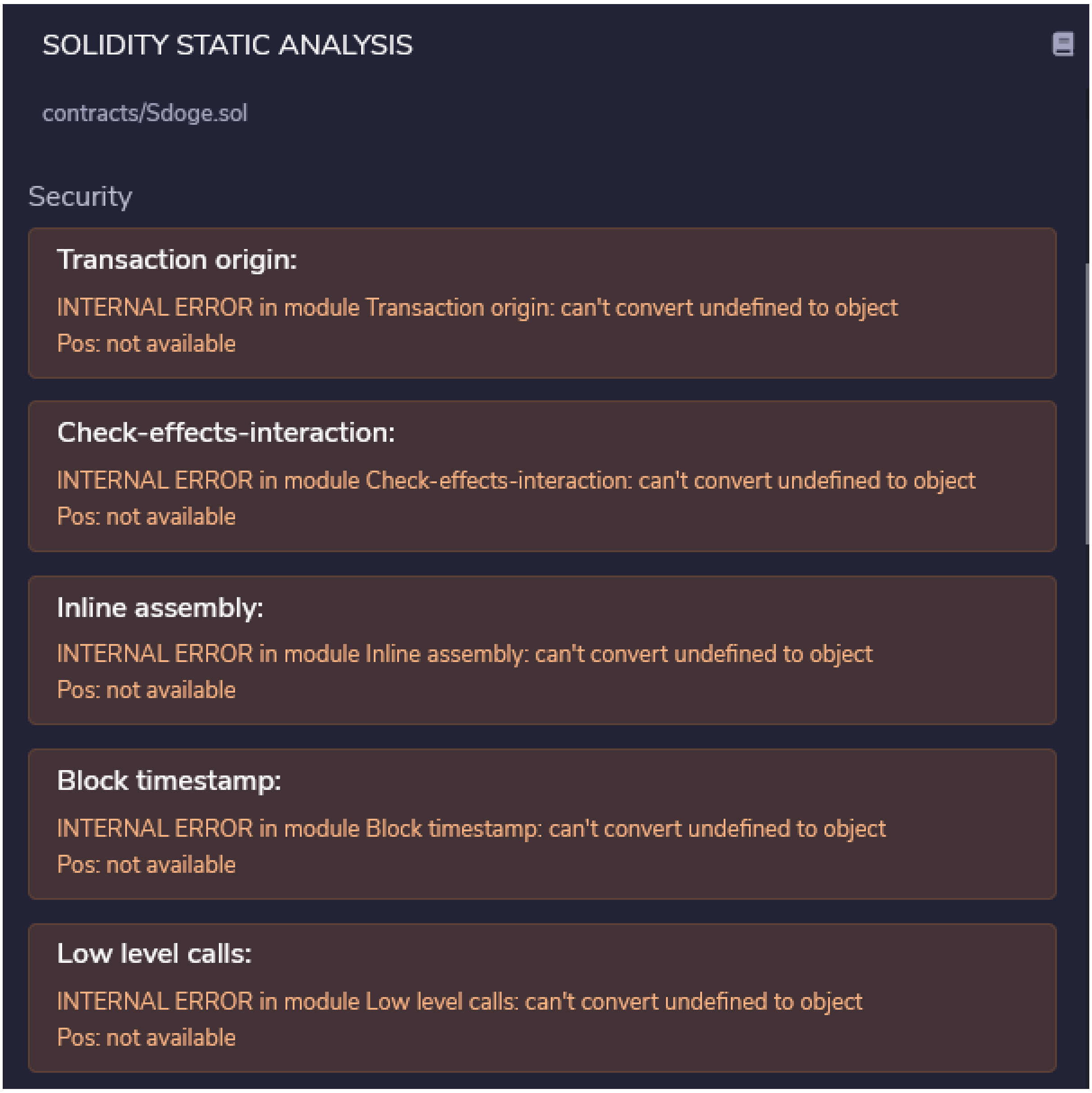
```
Parameter Sdoge.burnFrom(address,uint256)._from (Sdoge.sol#217) is not in mixedCase
Parameter Sdoge.burnFrom(address,uint256)._value (Sdoge.sol#217) is not in mixedCase
Parameter Sdoge.transferownership(address)._newaddress (Sdoge.sol#230) is not in mixedCase
Variable Sdoge._name (Sdoge.sol#124) is not in mixedCase
Variable Sdoge._symbol (Sdoge.sol#125) is not in mixedCase
Variable Sdoge._decimals (Sdoge.sol#126) is not in mixedCase
Variable Sdoge._totalSupply (Sdoge.sol#127) is not in mixedCase
Variable Sdoge._brate (Sdoge.sol#128) is not in mixedCase
Variable Sdoge._minimumSupply (Sdoge.sol#129) is not in mixedCase
Variable Sdoge._admin (Sdoge.sol#130) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Sdoge.constructor() (Sdoge.sol#135-144) uses literals with too many digits:
        - _totalSupply = 1000000000 * 10 ** uint256(_decimals) (Sdoge.sol#141)
Sdoge.constructor() (Sdoge.sol#135-144) uses literals with too many digits:
        - _minimumSupply = 998000000 * 10 ** uint256(_decimals) (Sdoge.sol#143)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
changeBurnRate(uint8) should be declared external:
        - Sdoge.changeBurnRate(uint8) (Sdoge.sol#145-149)
name() should be declared external:
        - Sdoge.name() (Sdoge.sol#151-153)
symbol() should be declared external:
        - Sdoge.symbol() (Sdoge.sol#154-156)
decimals() should be declared external:
        - Sdoge.decimals() (Sdoge.sol#157-159)
totalSupply() should be declared external:
        - Sdoge.totalSupply() (Sdoge.sol#160-162)
balanceOf(address) should be declared external:
        - Sdoge.balanceOf(address) (Sdoge.sol#163-165)
transfer(address,uint256) should be declared external:
        - Sdoge.transfer(address,uint256) (Sdoge.sol#166-178)
transferFrom(address,address,uint256) should be declared external:
        - Sdoge.transferFrom(address,address,uint256) (Sdoge.sol#179-195)
approve(address,uint256) should be declared external:
        - Sdoge.approve(address,uint256) (Sdoge.sol#196-200)
allowance(address,address) should be declared external:
        - Sdoge.allowance(address,address) (Sdoge.sol#201-203)
```



```
burnFrom(address,uint256) should be declared external:
        - Sdoge.burnFrom(address,uint256) (Sdoge.sol#217-228)
transferownership(address) should be declared external:
        - Sdoge.transferownership(address) (Sdoge.sol#230-234)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Sdoge.sol analyzed (3 contracts with 75 detectors), 48 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity static analysis

Sdoge.sol



SOLIDITY STATIC ANALYSIS

contracts/Sdoge.sol

Security

Transaction origin:

INTERNAL ERROR in module Transaction origin: can't convert undefined to object
Pos: not available

Check-effects-interaction:

INTERNAL ERROR in module Check-effects-interaction: can't convert undefined to object
Pos: not available

Inline assembly:

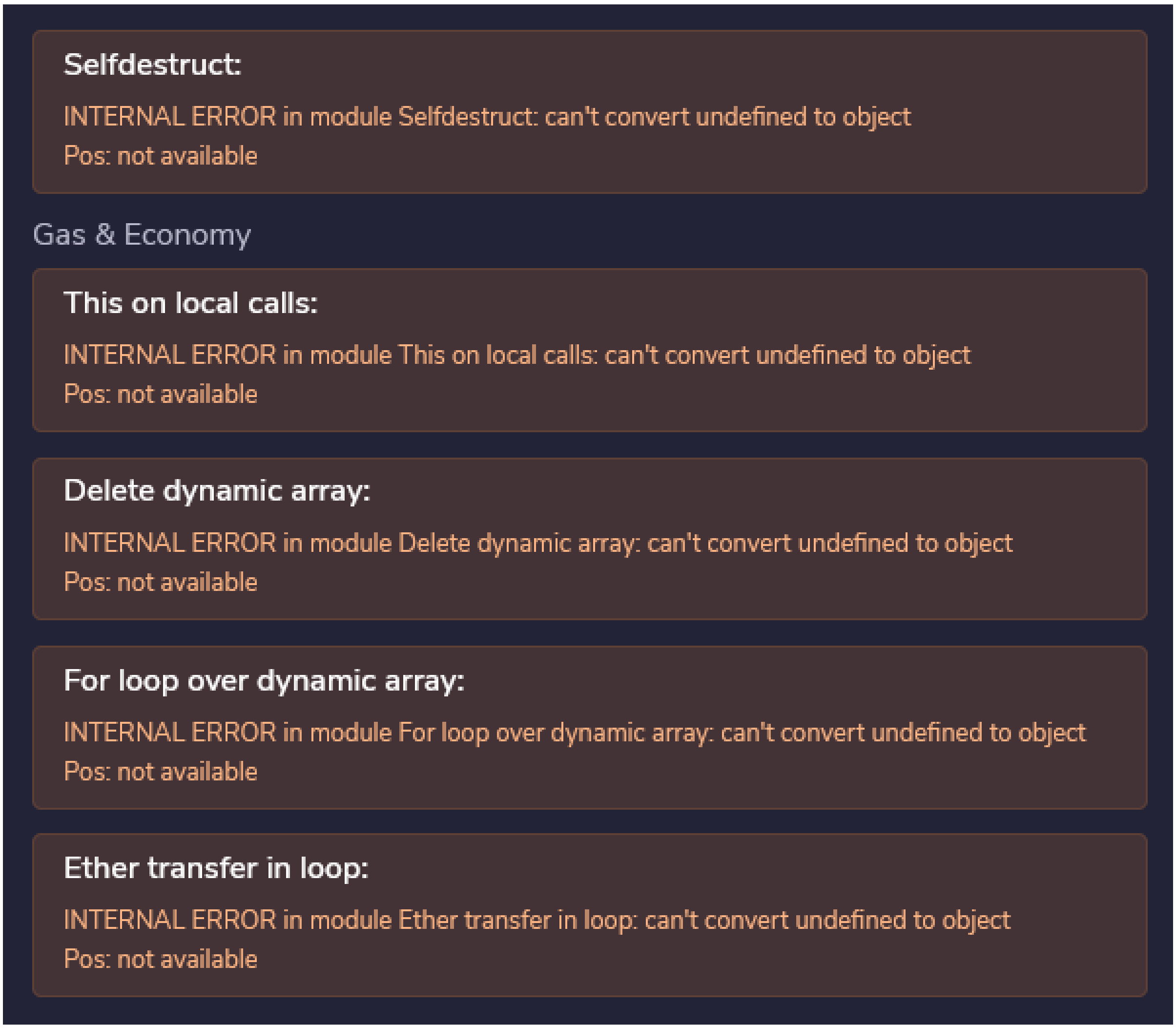
INTERNAL ERROR in module Inline assembly: can't convert undefined to object
Pos: not available

Block timestamp:

INTERNAL ERROR in module Block timestamp: can't convert undefined to object
Pos: not available

Low level calls:

INTERNAL ERROR in module Low level calls: can't convert undefined to object
Pos: not available



Selfdestruct:

INTERNAL ERROR in module Selfdestruct: can't convert undefined to object
Pos: not available

Gas & Economy

This on local calls:

INTERNAL ERROR in module This on local calls: can't convert undefined to object
Pos: not available

Delete dynamic array:

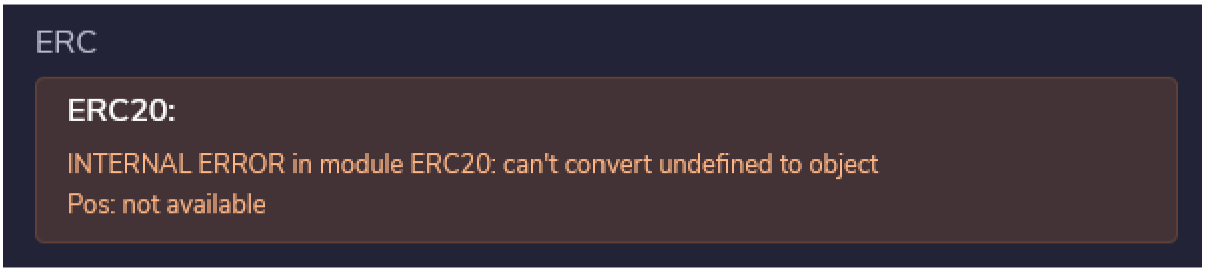
INTERNAL ERROR in module Delete dynamic array: can't convert undefined to object
Pos: not available

For loop over dynamic array:

INTERNAL ERROR in module For loop over dynamic array: can't convert undefined to object
Pos: not available

Ether transfer in loop:

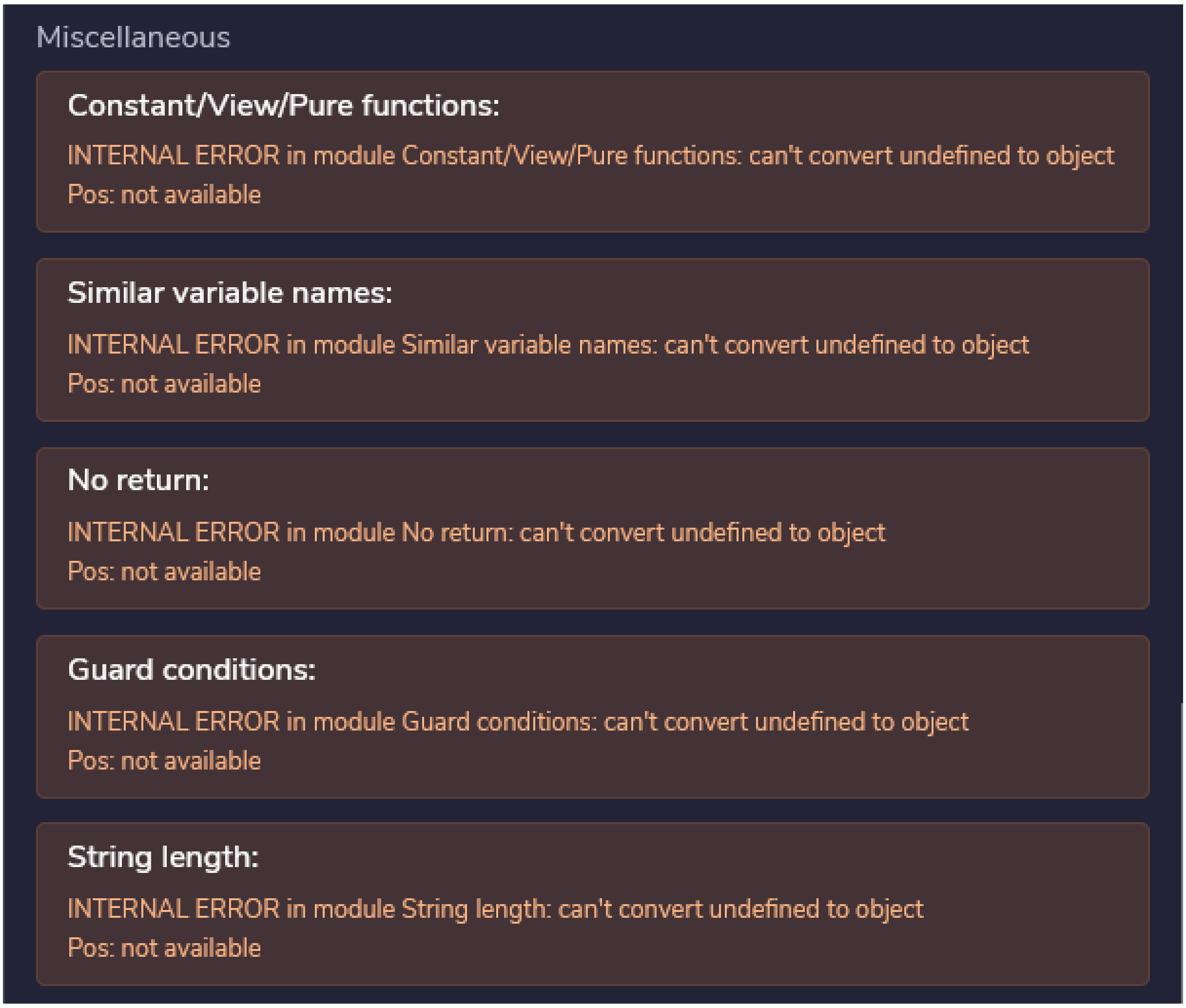
INTERNAL ERROR in module Ether transfer in loop: can't convert undefined to object
Pos: not available



ERC

ERC20:

INTERNAL ERROR in module ERC20: can't convert undefined to object
Pos: not available



Miscellaneous

Constant/View/Pure functions:

INTERNAL ERROR in module Constant/View/Pure functions: can't convert undefined to object

Pos: not available

Similar variable names:

INTERNAL ERROR in module Similar variable names: can't convert undefined to object

Pos: not available

No return:

INTERNAL ERROR in module No return: can't convert undefined to object

Pos: not available

Guard conditions:

INTERNAL ERROR in module Guard conditions: can't convert undefined to object

Pos: not available

String length:

INTERNAL ERROR in module String length: can't convert undefined to object

Pos: not available

Solhint Linter

Sdoge.sol

```
 contracts/Sdoge.sol:5:1: Error: Compiler version ^0.8.0 does not satisfy the r semver
requirement

 contracts/Sdoge.sol:135:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)

 contracts/Sdoge.sol:170:9: Error: Variable name must be in mixedCase

 contracts/Sdoge.sol:182:9: Error: Variable name must be in mixedCase
```

EtherAuthority

Email: audit@EtherAuthority.io