Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Render Token
Website:	rendernetwork.com
Platform:	Ethereum
Language:	Solidity
Date:	April 14th, 2024

Table of contents

`

Introduction

As part of EtherAuthority's community smart contract audit initiatives, the smart contract of Render Token from rendernetwork.com was audited. 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 April 14th, 2024.

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

- RenderToken is an ERC20 mintable token that will only be minted through the crowdsale contract. Here's a breakdown of each contract:
    - **ERC20Basic:** An interface defining the basic ERC20 token functions: `totalSupply()`, `balanceOf()`, and `transfer()`.
    - **ERC20:** Extends ERC20Basic and adds functions for token allowance management: `allowance()`, `transferFrom()`, and `approve()`.
    - **SafeERC20:** A library providing safe methods for interacting with ERC20 tokens to prevent common errors like reentrancy and failed transfers.
    - **Migratable:** A contract providing functionality for managing migrations between different contract versions. It includes methods for initialization, migration, and checking migration status.
    - **Ownable:** Extends Migratable and adds ownership functionality, allowing only the owner to execute certain functions.
    - **SafeMath:** A library providing safe arithmetic operations to prevent overflows and underflows.
    - **Escrow:** Extends Migratable and Ownable. It implements an escrow mechanism for holding tokens and disbursing them according to specific conditions.
    - **MigratableERC20:** Extends Migratable and provides functionality for migrating ERC20 tokens from one contract to another during version upgrades.

    - **BasicToken:** Implements basic functionality for ERC20 tokens, including balances and transfers.
    - **StandardToken:** Extends BasicToken and adds allowance functionality for approving token transfers.
    - **RenderToken:** Extends Migratable, MigratableERC20, Ownable, and StandardToken. It represents the main token contract, adding specific functionality for a token named "Render Token," including holding tokens in escrow and migrating legacy tokens to the new contract.
- Each contract serves a specific purpose in managing tokens and their interactions within the system. The contracts are designed to be modular and reusable, adhering to best practices for security and efficiency in Ethereum smart contract development.
- Overall, this contract facilitates the management of RNDR tokens, including token transfers, migration from legacy tokens, and escrow functionality for holding tokens during job execution. It is designed to work in coordination with an escrow contract for handling token transactions related to rendering jobs.

Audit scope

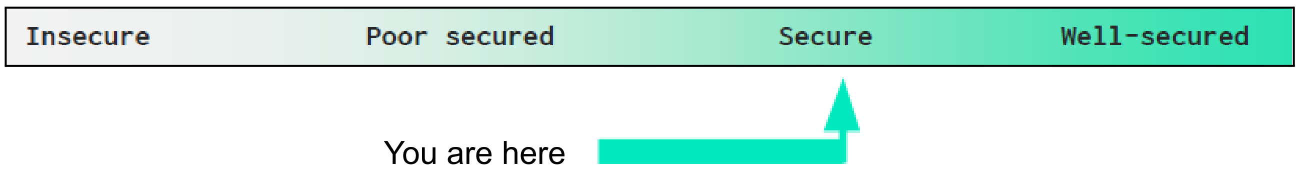
Name	**Code Review and Security Analysis Report for Render Token Smart Contract**
Platform	**Ethereum**
File	RenderToken.sol
Smart Contract Code	0x1a1fdf27c5e6784d1cebf256a8a5cc0877e73af0
Audit Date	April 14th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File : RenderToken.sol **Tokenomics:** • Name: Render Token • Symbol: RNDR • Decimals: 18 **Ownership control:** • Set the address of the escrow contract. • Change the address authorized to distribute tokens for completed jobs. • Change the address allowances will be sent to after job completion. • Disburse the Job by the authorized to disburse funds. • Add RNDR tokens to a job by the authorized. • Allows the current owner to transfer control of the contract to a new owner.	**YES, This is valid. We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contract is **"Secured"**. Also, this contract contains owner control, which does not make it fully decentralized.



We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a 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. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 1 low, and 5 very low level issues.

Investor 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	The solidity version is not specified	Passed
	The solidity version is too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	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

Business Risk Analysis

Category	Result
● Buy Tax	0%
● Sell Tax	0%
● Cannot Buy	No
● Cannot Sell	No
● Max Tax	0%
● Modify Tax	Not Detected
● Fee Check	No
● Is Honeypot	Not Detected
● Trading Cooldown	Not Detected
● Can Pause Trade?	No
● Pause Transfer?	Not Detected
● Max Tax?	No
● Is it Anti-whale?	Not Detected
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	Not Detected
● Blacklist Check	No
● Can Mint?	Not Detected
● Is it Proxy?	Yes
● Can Take Ownership?	Not Detected
● Hidden Owner?	Not Detected
● Self Destruction?	Not Detected
● Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract file. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Render 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 Render Token.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contract. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Render Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are well commented on. and the logic is straightforward. So it is easy 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 used in this smart contract infrastructure that are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

RenderToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initialize	write	Critical operation lacks event log, Missing-zero-check	Refer Audit Findings
3	holdInEscrow	write	Optimization	Refer Audit Findings
4	_mintMigratedTokens	internal	Passed	No Issue
5	setEscrowContractAddress	write	Centralized Ownership and Privileges Management	Refer Audit Findings
6	transferFrom	write	Passed	No Issue
7	approve	write	Passed	No Issue
8	allowance	read	Passed	No Issue
9	increaseApproval	write	Passed	No Issue
10	decreaseApproval	write	Passed	No Issue
11	totalSupply	read	Passed	No Issue
12	transfer	write	Passed	No Issue
13	balanceOf	read	Passed	No Issue
14	initialize	write	Warning	Refer Audit Findings
15	migrate	write	Passed	No Issue
16	migrateToken	write	Passed	No Issue
17	migrateTokenTo	write	Passed	No Issue
18	_mintMigratedTokens	internal	Passed	No Issue
19	canDisburse	modifier	Passed	No Issue
20	initialize	write	is Initializer	No Issue
21	changeDisbursalAddress	external	Missing-zero-check, Centralized Ownership and Privileges Management	Refer Audit Findings
22	changeRenderTokenAddress	external	Centralized Ownership and Privileges Management	Refer Audit Findings
23	disburseJob	external	Centralized Ownership and Privileges Management	Refer Audit Findings
24	fundJob	external	Passed	No Issue
25	jobBalance	external	Passed	No Issue

26	onlyOwner	modifier	Passed	No Issue
27	transferOwnership	write	Centralized Ownership and Privileges Management	Refer Audit Findings
28	isInitializer	modifier	Passed	No Issue
29	isMigration	modifier	Passed	No Issue
30	isMigrated	read	Passed	No Issue
31	initialize	write	Passed	No Issue
32	validateMigrationIsPending	write	Passed	No Issue
33	allowance	read	Passed	No Issue
34	transferFrom	write	Passed	No Issue
35	approve	write	Passed	No Issue
36	totalSupply	read	Passed	No Issue
37	balanceOf	read	Passed	No Issue
38	transfer	write	Passed	No Issue

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 a 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 a 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 Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

(1) Critical operation lacks event log: **RenderToken.sol**

```
function initialize(address _sender) public isInitializer("Ownable",
"1.9.0") {
    owner = _sender;
  }
```

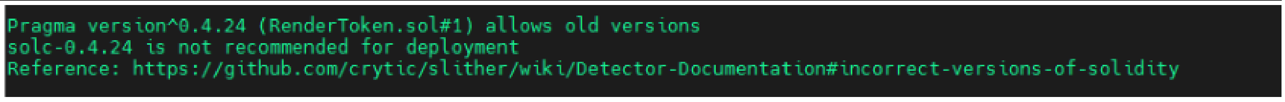
Detecting missing events for critical access control parameters initialize() has no event, so it is difficult to track off-chain owner changes.

Resolution: We suggest emitting an event for critical parameter changes.

Very Low / Informational / Best practices:

(1) Use the Latest Solidity Compiler Version for Enhanced Security: **RenderToken.sol**

```
pragma solidity ^0.4.24;
```



```
Pragma version^0.4.24 (RenderToken.sol#1) allows old versions
solc-0.4.24 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

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

Resolution: Deploy with any of the following Solidity versions:

0.8.18

The recommendations take into account:

- Risks related to recent releases
- Risks of complex code generation changes
- Risks of new language features
- Risks of known bugs

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

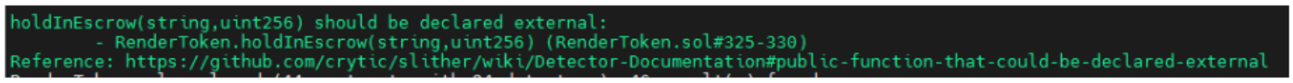
(2) Missing-zero-check: **RenderToken.sol**

```
function changeDisbursalAddress(address _newDisbursalAddress)
external onlyOwner {
    disbursalAddress = _newDisbursalAddress;

    emit DisbursalAddressUpdate(disbursalAddress);
  }
function initialize(address _sender) public isInitializer("Ownable",
"1.9.0") {
    owner = _sender;
  }
```

Detects missing zero address validation.

Resolution: Check that the address is not zero.

(3) Optimization: **RenderToken.sol**



```
holdInEscrow(string,uint256) should be declared external:
        - RenderToken.holdInEscrow(string,uint256) (RenderToken.sol#325-330)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

public functions that are never called by the contract should be declared external, and its immutable parameters should be located in call data to save gas.   (holdInEscrow()).

Resolution: Use the external attribute for functions never called from the contract, and change the location of immutable parameters to call data to save gas.

(4) Centralized Ownership and Privileges Management: **RenderToken.sol**

In the contract only the owner as owner authority on the following functions:

- transferOwnership
- changeDisbursalAddress
- changeRenderTokenAddress
- setEscrowContractAddress

The contract can disburse as an owner authority on the following function:

- disburseJob

Resolution: We suggest carefully managing the onlyOwner, canDisburse private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices.

(5) Warning: **RenderToken.sol**

```
function initialize() isInitializer("Migratable", "1.2.1") public {
 }
```

Visibility modifiers must be first in the list of modifiers.

Resolution: We suggest following proper coding guidelines.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble. The following are Admin functions:

RenderToken.sol

- setEscrowContractAddress: Set the address of the escrow contract by the owner.

Escrow.sol

- changeDisbursalAddress: Change the address authorized to distribute tokens for completed jobs by the owner.
- changeRenderTokenAddress: Change the address allowances will be sent to after job completion by the owner.
- disburseJob: Disburse the Job by the authorized to disburse funds.
- fundJob: Add RNDR tokens to a job by the authorized.

Ownable.sol

- transferOwnership: Allows the current owner to transfer control of the contract to a new owner by the current owner.

To make the smart contract 100% decentralized, we suggest renouncing ownership of the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of Etherscan web links. And we have used all possible tests based on given objects as files. We observed 1 low and 5 informational issues in the smart contract. And those issues are not critical. So, **it's good to go for the 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 the 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 the As-is overview section of the report.

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

The security state of the reviewed smart 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.

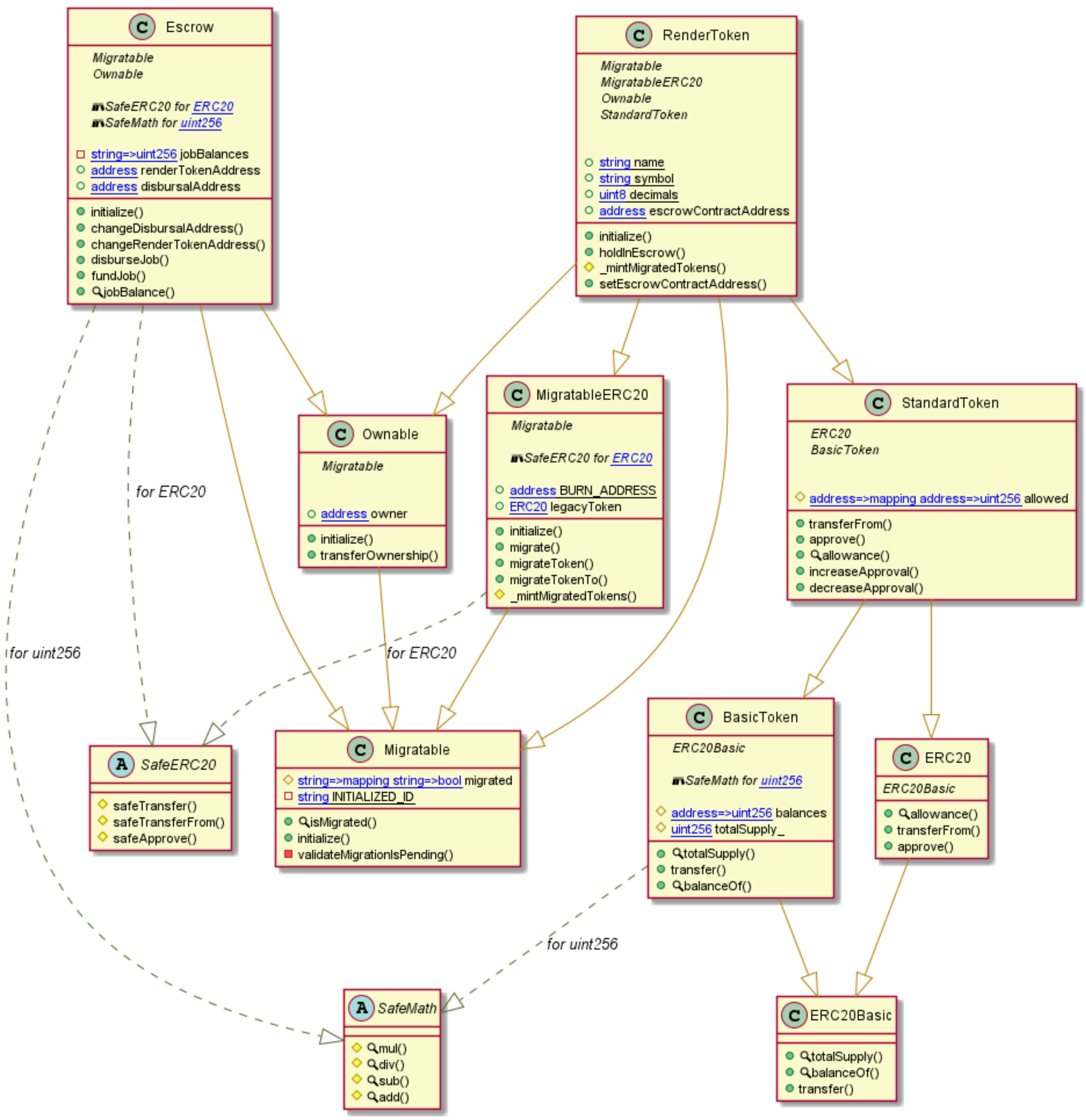
Technical Disclaimer

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 - Render Token

Render Token Diagram

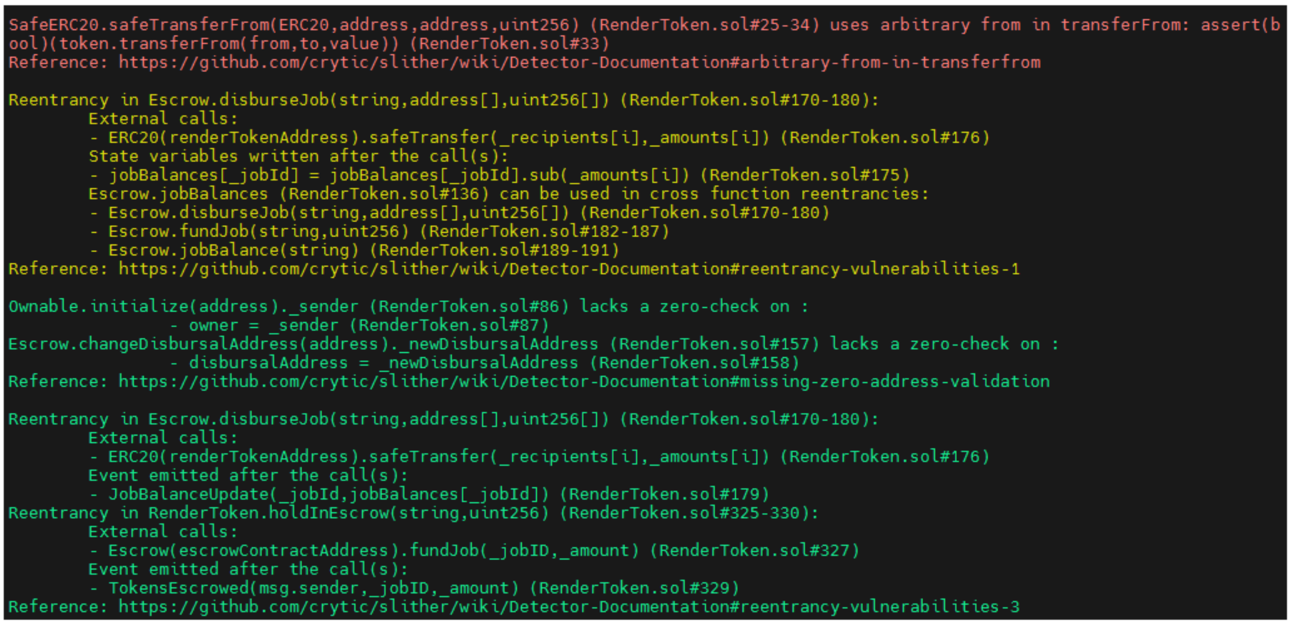


Slither Results Log

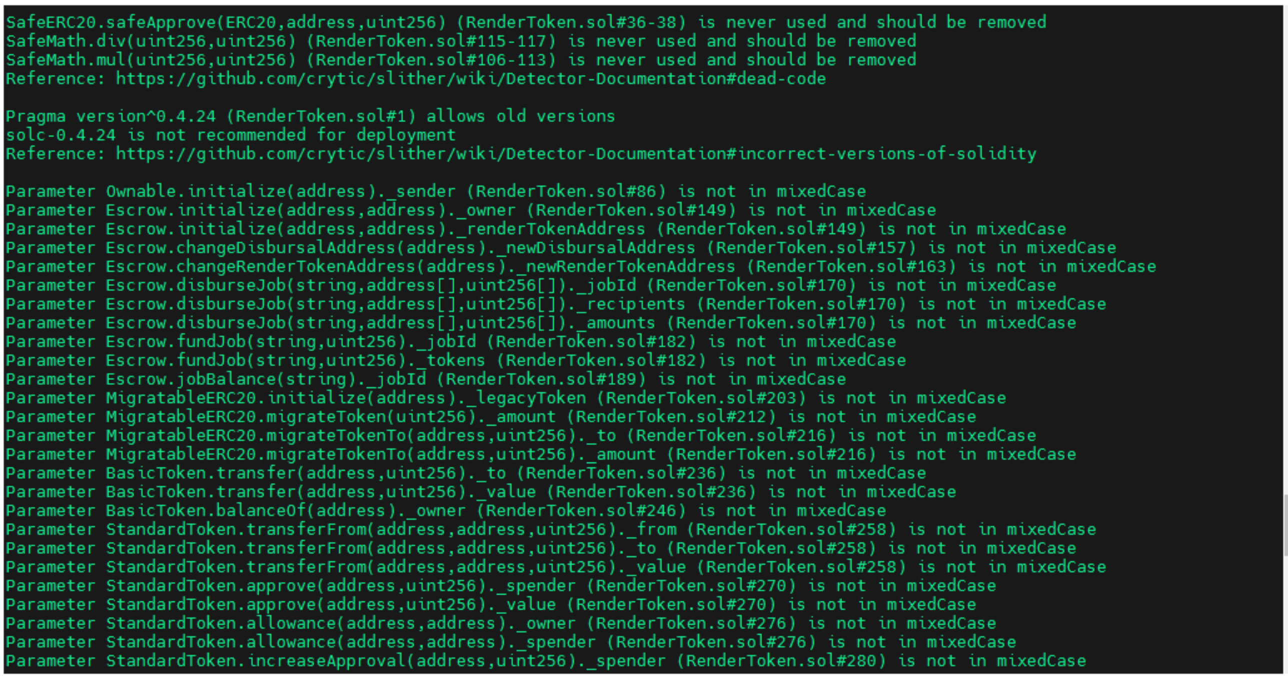
Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

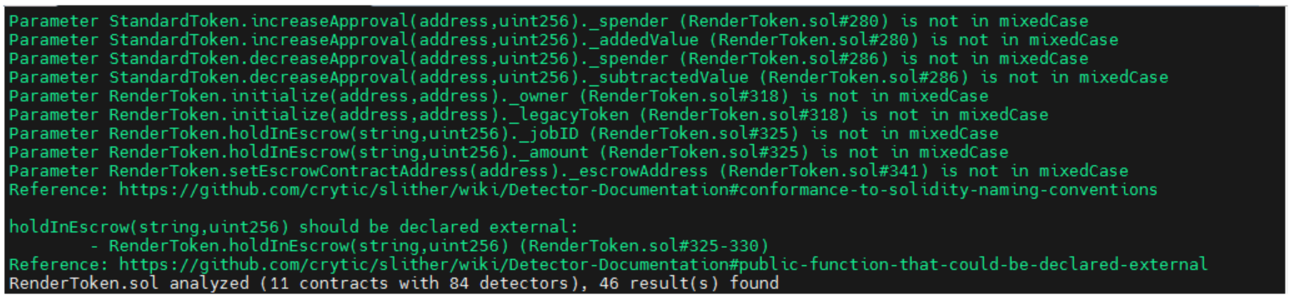
Slither Log >> RenderToken.sol



```
SafeERC20.safeTransferFrom(ERC20,address,address,uint256) (RenderToken.sol#25-34) uses arbitrary from in transferFrom: assert(b
ool)(token.transferFrom(from,to,value)) (RenderToken.sol#33)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom

Reentrancy in Escrow.disburseJob(string,address[],uint256[]) (RenderToken.sol#170-180):
        External calls:
        - ERC20(renderTokenAddress).safeTransfer(_recipients[i],_amounts[i]) (RenderToken.sol#176)
        State variables written after the call(s):
        - jobBalances[_jobId] = jobBalances[_jobId].sub(_amounts[i]) (RenderToken.sol#175)
        Escrow.jobBalances (RenderToken.sol#136) can be used in cross function reentrancies:
        - Escrow.disburseJob(string,address[],uint256[]) (RenderToken.sol#170-180)
        - Escrow.fundJob(string,uint256) (RenderToken.sol#182-187)
        - Escrow.jobBalance(string) (RenderToken.sol#189-191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

Ownable.initialize(address)._sender (RenderToken.sol#86) lacks a zero-check on :
                - owner = _sender (RenderToken.sol#87)
Escrow.changeDisbursalAddress(address)._newDisbursalAddress (RenderToken.sol#157) lacks a zero-check on :
                - disbursalAddress = _newDisbursalAddress (RenderToken.sol#158)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in Escrow.disburseJob(string,address[],uint256[]) (RenderToken.sol#170-180):
        External calls:
        - ERC20(renderTokenAddress).safeTransfer(_recipients[i],_amounts[i]) (RenderToken.sol#176)
        Event emitted after the call(s):
        - JobBalanceUpdate(_jobId,jobBalances[_jobId]) (RenderToken.sol#179)
Reentrancy in RenderToken.holdInEscrow(string,uint256) (RenderToken.sol#325-330):
        External calls:
        - Escrow(escrowContractAddress).fundJob(_jobID,_amount) (RenderToken.sol#327)
        Event emitted after the call(s):
        - TokensEscrowed(msg.sender,_jobID,_amount) (RenderToken.sol#329)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```



```
SafeERC20.safeApprove(ERC20,address,uint256) (RenderToken.sol#36-38) is never used and should be removed
SafeMath.div(uint256,uint256) (RenderToken.sol#115-117) is never used and should be removed
SafeMath.mul(uint256,uint256) (RenderToken.sol#106-113) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.4.24 (RenderToken.sol#1) allows old versions
solc-0.4.24 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter Ownable.initialize(address)._sender (RenderToken.sol#86) is not in mixedCase
Parameter Escrow.initialize(address,address)._owner (RenderToken.sol#149) is not in mixedCase
Parameter Escrow.initialize(address,address)._renderTokenAddress (RenderToken.sol#149) is not in mixedCase
Parameter Escrow.changeDisbursalAddress(address)._newDisbursalAddress (RenderToken.sol#157) is not in mixedCase
Parameter Escrow.changeRenderTokenAddress(address)._newRenderTokenAddress (RenderToken.sol#163) is not in mixedCase
Parameter Escrow.disburseJob(string,address[],uint256[])._jobId (RenderToken.sol#170) is not in mixedCase
Parameter Escrow.disburseJob(string,address[],uint256[])._recipients (RenderToken.sol#170) is not in mixedCase
Parameter Escrow.disburseJob(string,address[],uint256[])._amounts (RenderToken.sol#170) is not in mixedCase
Parameter Escrow.fundJob(string,uint256)._jobId (RenderToken.sol#182) is not in mixedCase
Parameter Escrow.fundJob(string,uint256)._tokens (RenderToken.sol#182) is not in mixedCase
Parameter Escrow.jobBalance(string)._jobId (RenderToken.sol#189) is not in mixedCase
Parameter MigratableERC20.initialize(address)._legacyToken (RenderToken.sol#203) is not in mixedCase
Parameter MigratableERC20.migrateToken(uint256)._amount (RenderToken.sol#212) is not in mixedCase
Parameter MigratableERC20.migrateTokenTo(address,uint256)._to (RenderToken.sol#216) is not in mixedCase
Parameter MigratableERC20.migrateTokenTo(address,uint256)._amount (RenderToken.sol#216) is not in mixedCase
Parameter BasicToken.transfer(address,uint256)._to (RenderToken.sol#236) is not in mixedCase
Parameter BasicToken.transfer(address,uint256)._value (RenderToken.sol#236) is not in mixedCase
Parameter BasicToken.balanceOf(address)._owner (RenderToken.sol#246) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._from (RenderToken.sol#258) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._to (RenderToken.sol#258) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._value (RenderToken.sol#258) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._spender (RenderToken.sol#270) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._value (RenderToken.sol#270) is not in mixedCase
Parameter StandardToken.allowance(address,address)._owner (RenderToken.sol#276) is not in mixedCase
Parameter StandardToken.allowance(address,address)._spender (RenderToken.sol#276) is not in mixedCase
Parameter StandardToken.increaseApproval(address,uint256)._spender (RenderToken.sol#280) is not in mixedCase
```

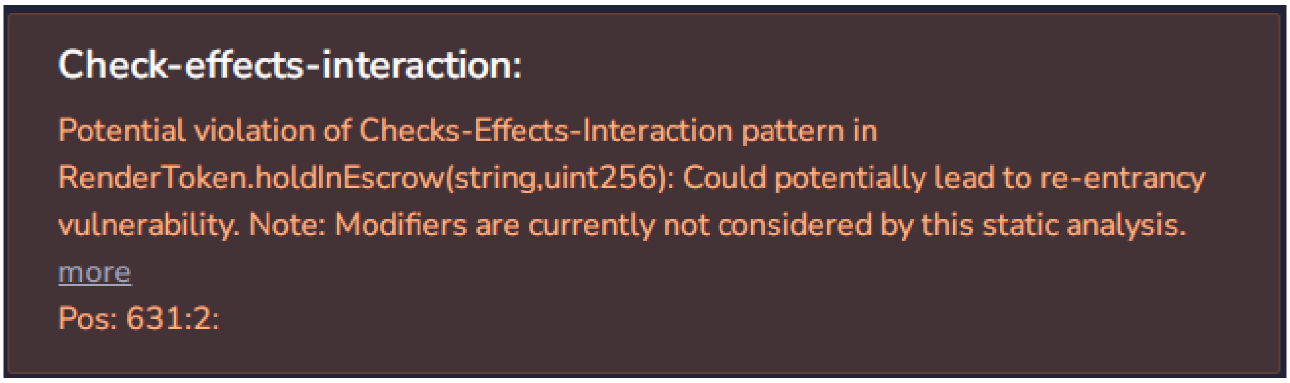


```
Parameter StandardToken.increaseApproval(address,uint256)._spender (RenderToken.sol#280) is not in mixedCase
Parameter StandardToken.increaseApproval(address,uint256)._addedValue (RenderToken.sol#280) is not in mixedCase
Parameter StandardToken.decreaseApproval(address,uint256)._spender (RenderToken.sol#286) is not in mixedCase
Parameter StandardToken.decreaseApproval(address,uint256)._subtractedValue (RenderToken.sol#286) is not in mixedCase
Parameter RenderToken.initialize(address,address)._owner (RenderToken.sol#318) is not in mixedCase
Parameter RenderToken.initialize(address,address)._legacyToken (RenderToken.sol#318) is not in mixedCase
Parameter RenderToken.holdInEscrow(string,uint256)._jobID (RenderToken.sol#325) is not in mixedCase
Parameter RenderToken.holdInEscrow(string,uint256)._amount (RenderToken.sol#325) is not in mixedCase
Parameter RenderToken.setEscrowContractAddress(address)._escrowAddress (RenderToken.sol#341) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

holdInEscrow(string,uint256) should be declared external:
        - RenderToken.holdInEscrow(string,uint256) (RenderToken.sol#325-330)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
RenderToken.sol analyzed (11 contracts with 84 detectors), 46 result(s) found
```

Solidity Static Analysis

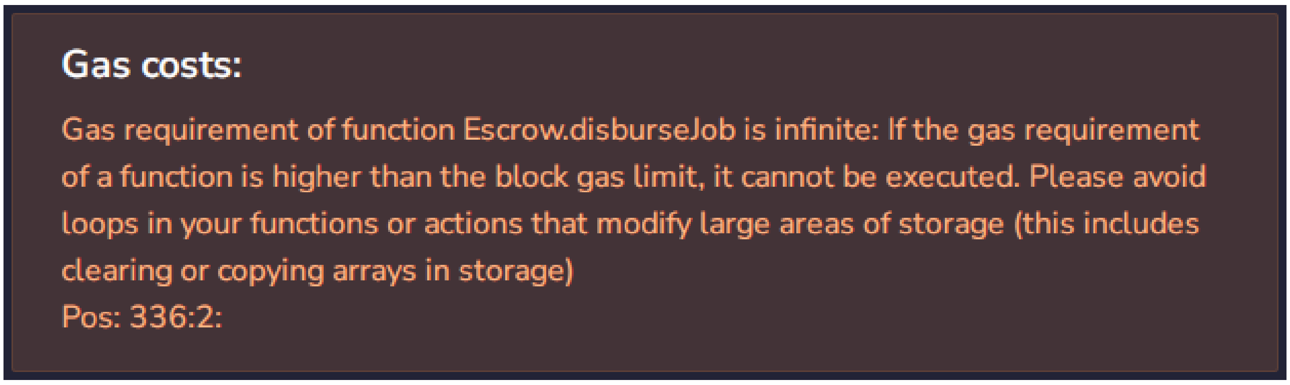
Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

RenderToken.sol



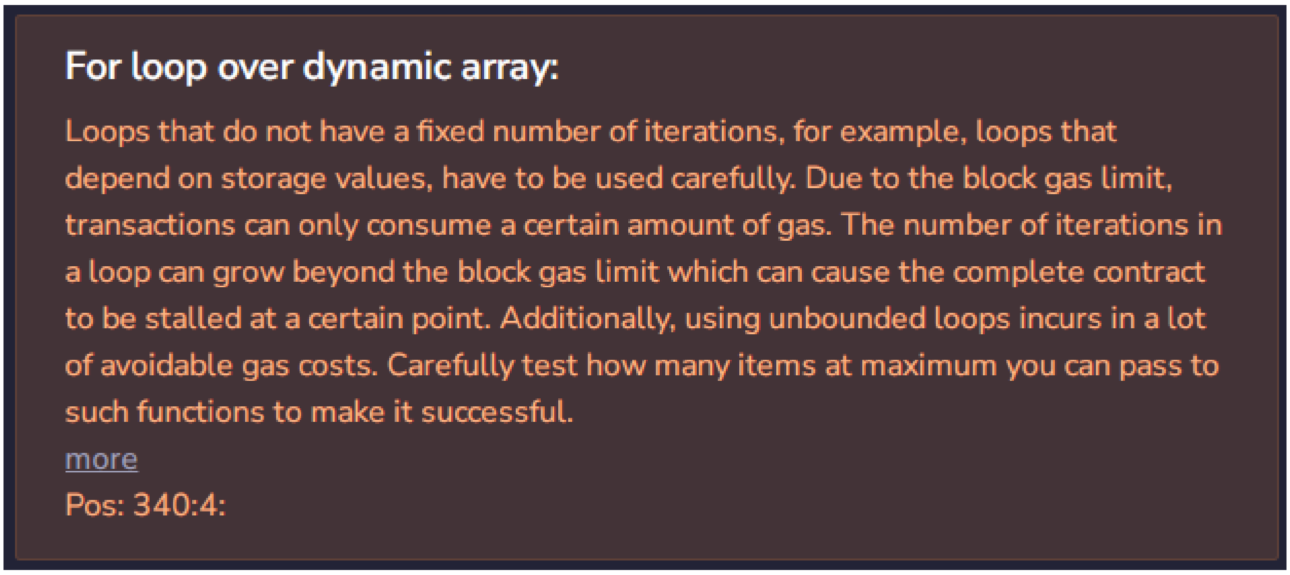
Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in RenderToken.holdInEscrow(string,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 631:2:



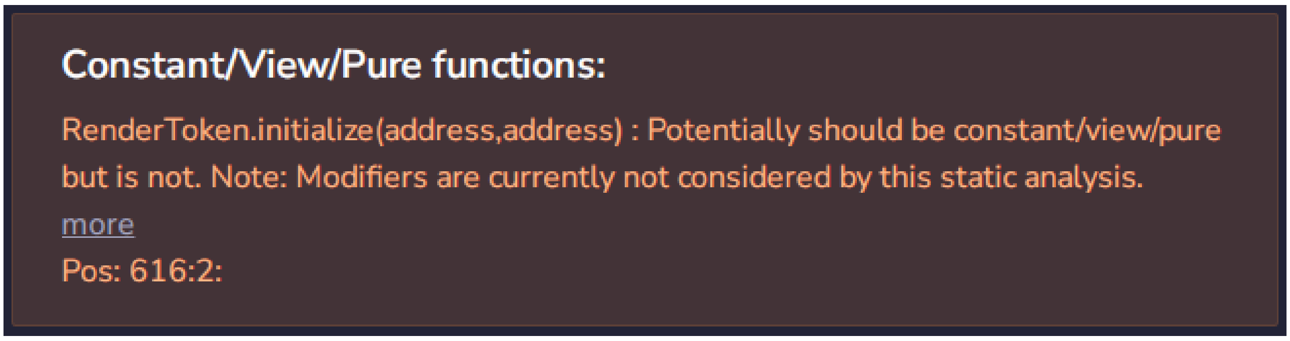
Gas costs:

Gas requirement of function Escrow.disburseJob is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 336:2:



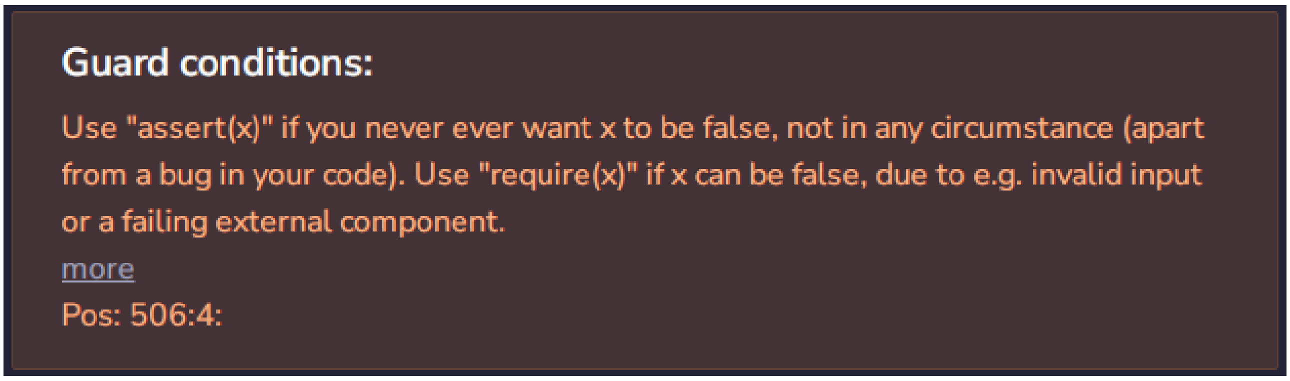
For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 340:4:



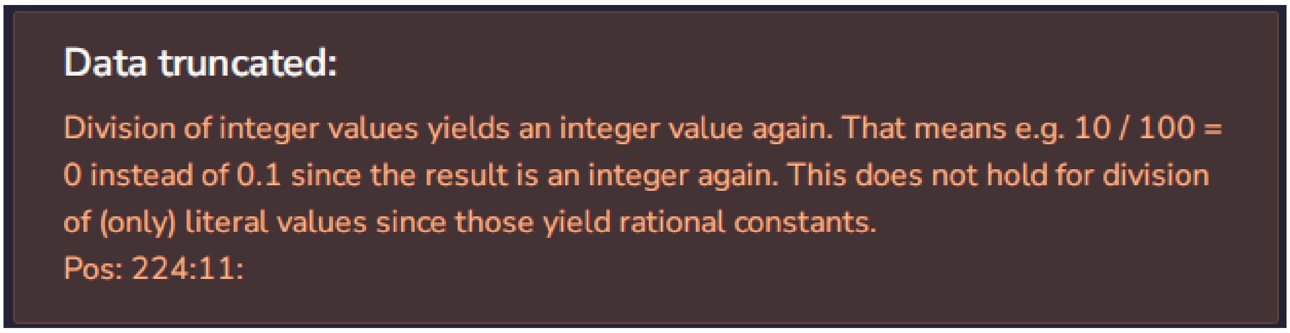
Constant/View/Pure functions:

RenderToken.initialize(address,address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 616:2:



Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 506:4:



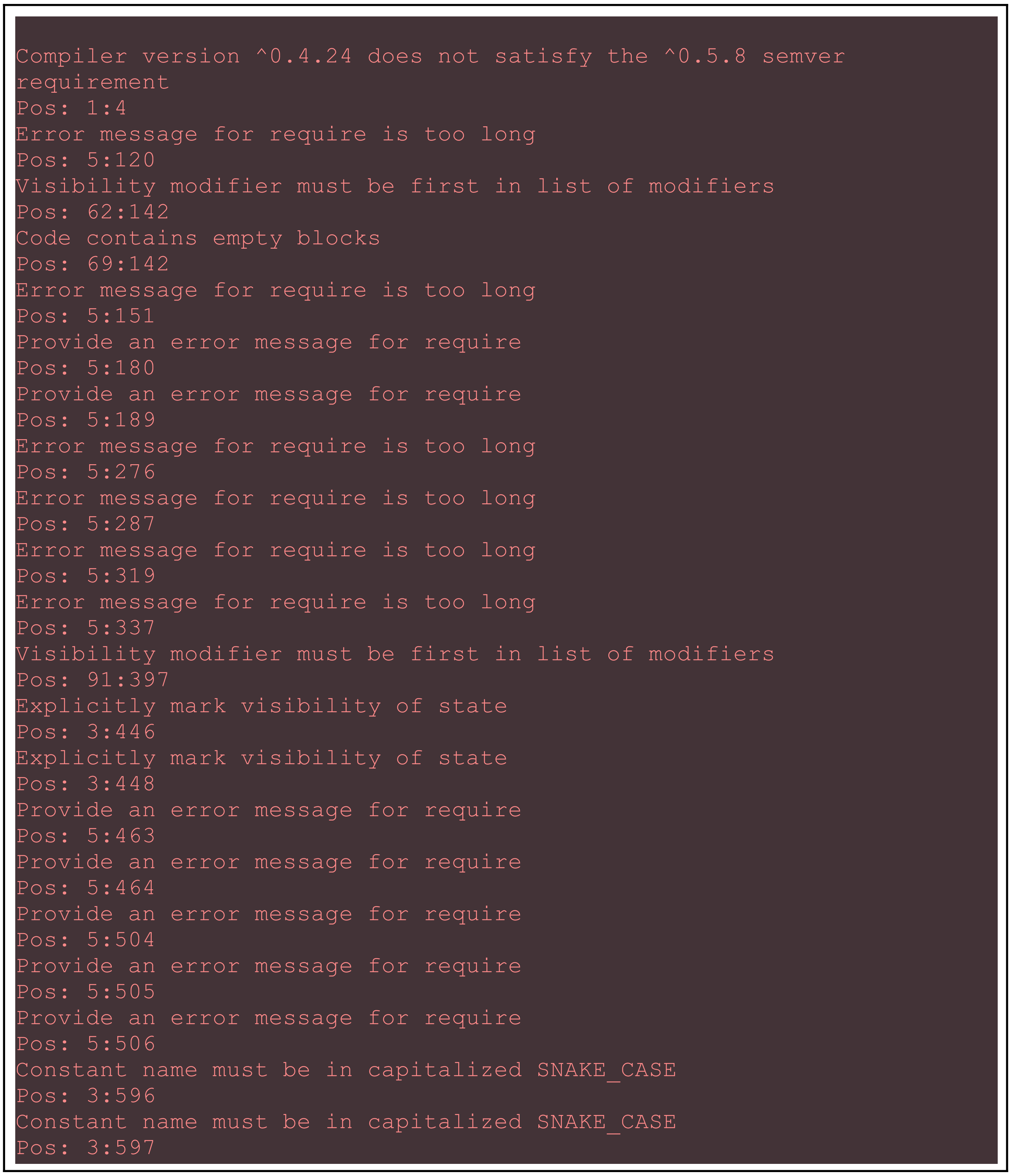
Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 224:11:

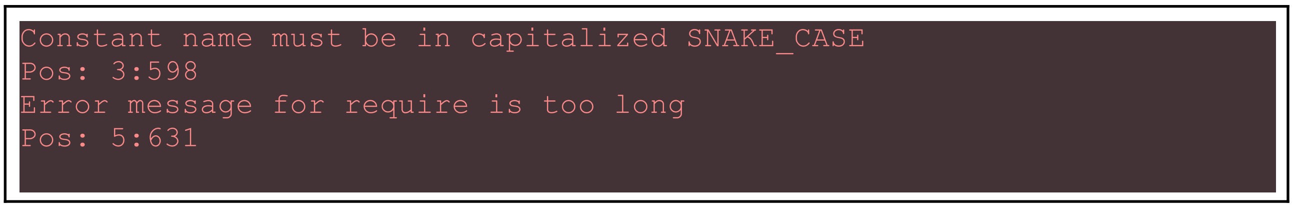
Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

RenderToken.sol



```
Compiler version ^0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:4
Error message for require is too long
Pos: 5:120
Visibility modifier must be first in list of modifiers
Pos: 62:142
Code contains empty blocks
Pos: 69:142
Error message for require is too long
Pos: 5:151
Provide an error message for require
Pos: 5:180
Provide an error message for require
Pos: 5:189
Error message for require is too long
Pos: 5:276
Error message for require is too long
Pos: 5:287
Error message for require is too long
Pos: 5:319
Error message for require is too long
Pos: 5:337
Visibility modifier must be first in list of modifiers
Pos: 91:397
Explicitly mark visibility of state
Pos: 3:446
Explicitly mark visibility of state
Pos: 3:448
Provide an error message for require
Pos: 5:463
Provide an error message for require
Pos: 5:464
Provide an error message for require
Pos: 5:504
Provide an error message for require
Pos: 5:505
Provide an error message for require
Pos: 5:506
Constant name must be in capitalized SNAKE_CASE
Pos: 3:596
Constant name must be in capitalized SNAKE_CASE
Pos: 3:597
```



```
Constant name must be in capitalized SNAKE_CASE
Pos: 3:598
Error message for require is too long
Pos: 5:631
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

EtherAuthority

Email: audit@EtherAuthority.io