EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Reign Token
Website:	https://reignprotocol.io/
Platform:	Binance Smart Chain
Language:	Solidity
Date:	July 29th, 2022

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 THE PUBLIC AFTER ISSUES ARE RESOLVED.

Introduction

EtherAuthority was contracted by the Reign Token team to perform the Security audit of the Reign 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 July 29th, 2022.

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

- Reign Contract is a smart contract, having functions like setFees, mint, checkFeeExempt, bonusTime, setFees, swapBack, stake, claim, unstake, setBUSD, etc.
- The Reign contract inherits OwnableUpgradeable, ReentrancyGuardUpgradeable, ERC20Upgradeable, IERC20, SafeMathUpgradeable standard smart contracts from the OpenZeppelin library. And inherits VRFCoordinatorV2Interface standard smart contracts from the chainlink library. And also inherits the console library from standard smart contracts from the hardhat library.
- These OpenZeppelin contracts, chainlink contracts and hardhat contracts are considered community audited and time tested, and hence are not part of the audit scope.

Audit scope

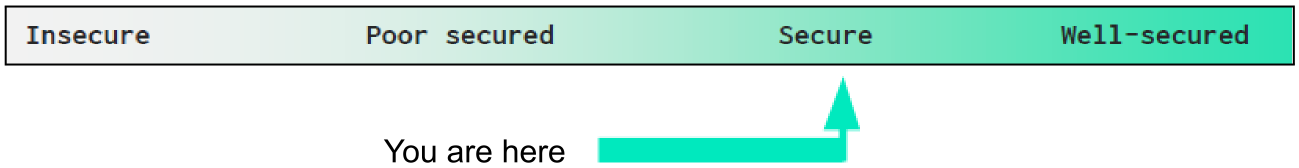
Name	**Code Review and Security Analysis Report for Reign Token Smart Contract**
Platform	**BSC / Solidity**
File	ReignERC20.sol
File MD5 Hash	64D06AA68EBF246AD8CE1C7C32CE5643
Updated File MD5 Hash	74C64556CE5FCE304524A2932F4C723C
Audit Date	July 29th, 2022
Revise Audit Date	August 2nd, 2022

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Name: Reign ● Symbol: REIGN ● Decimals: 18 ● Initial Fragments Supply: 0.4 Million	**YES, This is valid.**
Ownership Control: ● The owner can set a blacklist of addresses. ● The owner can set the next rebase value. ● The owner can set the fee exempt, buy fees, sell fees, transfer fees.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **"Secured".** This token contract does contain 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 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, 2 medium and 3 low and some very low level issues.

All the issues have been resolved/acknowledged in the revised 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	Passed
	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	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	Moderated
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. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the Reign 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 Reign Token.

The Reign 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.

Code parts are not well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Reign Token smart contract code in the form of a file. The hash of that code is mentioned above in the table.

As mentioned above, code parts are not commented on. So it is not 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.

Another source of information was its official website https://reignprotocol.io/ which provided rich information about the project architecture.

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.

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

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	__ReentrancyGuard_init	internal	access only Initializing	No Issue
3	__ReentrancyGuard_init_unchained	internal	access only Initializing	No Issue
4	nonReentrant	modifier		No Issue
5	_nonReentrantBefore	write	Passed	No Issue
6	_nonReentrantAfter	write	Passed	No Issue
7	__Ownable_init	internal	access only Initializing	No Issue
8	__Ownable_init_unchained	internal	access only Initializing	No Issue
9	onlyOwner	modifier	Passed	No Issue
10	owner	read	Passed	No Issue
11	_checkOwner	internal	Passed	No Issue
12	renounceOwnership	write	access only Owner	No Issue
13	transferOwnership	write	access only Owner	No Issue
14	_transferOwnership	internal	Passed	No Issue
15	__ERC20_init	internal	access only Initializing	No Issue
16	__ERC20_init_unchained	internal	access only Initializing	No Issue
17	name	read	Passed	No Issue
18	symbol	read	Passed	No Issue
19	decimals	read	Passed	No Issue
20	totalSupply	read	Passed	No Issue
21	balanceOf	read	Passed	No Issue
22	transfer	write	Passed	No Issue
23	allowance	read	Passed	No Issue
24	approve	write	Passed	No Issue
25	transferFrom	write	Passed	No Issue
26	increaseAllowance	write	Passed	No Issue
27	decreaseAllowance	write	Passed	No Issue
28	_transfer	internal	Passed	No Issue
29	_mint	internal	Passed	No Issue
30	_burn	internal	Passed	No Issue
31	_approve	internal	Passed	No Issue
32	_spendAllowance	internal	Passed	No Issue
33	_beforeTokenTransfer	internal	Passed	No Issue
34	_afterTokenTransfer	internal	Passed	No Issue
35	swapping	modifier	Passed	No Issue
36	validRecipient	modifier	Passed	No Issue
37	initialize	write	access by initializer	No Issue

38	setBlacklist	external	access only Owner	No Issue
39	noBlacklist	modifier	Passed	No Issue
40	allowance	read	Passed	No Issue
41	balanceOf	read	Passed	No Issue
42	transfer	write	Passed	No Issue
43	_basicTransfer	internal	Passed	No Issue
44	_transferFrom	internal	Passed	No Issue
45	transferFrom	write	Passed	No Issue
46	totalSupply	read	Passed	No Issue
47	getCirculatingSupply	read	Passed	No Issue
48	mint	external	Mint doesn’t work without stake	Refer audit findings
49	decreaseAllowance	write	Passed	No Issue
50	increaseAllowance	write	Passed	No Issue
51	approve	write	Passed	No Issue
52	setNextRebase	external	access only Owner	No Issue
53	shouldRebase	internal	Passed	No Issue
54	_rebase	write	Passed	No Issue
55	setAutoRebase	external	access only Owner	No Issue
56	getYield	read	Passed	No Issue
57	manualRebase	external	Passed	No Issue
58	setRebaseFrequency	external	access only Owner	No Issue
59	setBotForBonus	external	access only Owner	No Issue
60	bonusTime	external	Missing Error Message	Refer audit findings
61	getNextRebaseToken	external	Passed	No Issue
62	setChainLinkParam	external	access only Owner	No Issue
63	rawFulfillRandomWords	external	Range validation is missing	Refer audit findings
64	fulfillRandomWords	internal	Range validation is missing	Refer audit findings
65	resetYieldStaking	external	Missing Error Message	Refer audit findings
66	setFeeExempt	external	access only Owner	No Issue
67	setSwapBackSettings	external	Division before multiplication	Refer audit findings
68	checkFeeExempt	external	Passed	No Issue
69	setFees	external	access only Owner	No Issue
70	shouldTakeFee	internal	Passed	No Issue
71	takeFee	internal	Division before multiplication	Refer audit findings
72	setRouter	external	Function input parameters lack of check	Refer audit findings
73	setFeeReceivers	external	Function input parameters lack of check	Refer audit findings
74	checkSwapThreshold	external	Passed	No Issue

75	shouldSwapBack	internal	Passed	No Issue
76	_swapAndLiquify	write	Passed	No Issue
77	_addLiquidityStable	write	Passed	No Issue
78	_swapTokensForStable	write	Passed	No Issue
79	swapBack	internal	Passed	No Issue
80	manualSwapBack	external	access only Owner	No Issue
81	stake	external	Passed	No Issue
82	claim	external	Passed	No Issue
83	unstake	external	Passed	No Issue
84	setStakingTypeCount	external	access only Owner	No Issue
85	setStakeDuration	external	access only Owner	No Issue
86	getStakingAmount	external	Passed	No Issue
87	getStakingUnlocked	external	Passed	No Issue
88	getStakingAmountInitial	external	Passed	No Issue
89	getTotalStaked	external	Passed	No Issue
90	getStakedTokens	external	Passed	No Issue
91	getNewRebaseStakedToken	external	Passed	No Issue
92	getRebaseDailyStakedToken	external	Passed	No Issue
93	getMaxSellAmount	read	Passed	No Issue
94	setPresalePeriod	external	access only Owner	No Issue
95	buyPresale	external	Passed	No Issue
96	setBUSD	external	Function input parameters lack of check	Refer audit findings
97	setAutomatedMarketMakerPair	write	AutomatedMarketMakerPair set manually after setting router	Refer audit findings
98	setInitialDistributionFinished	external	access only Owner	No Issue
99	clearStuckBalance	external	access only Owner	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 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 Severity

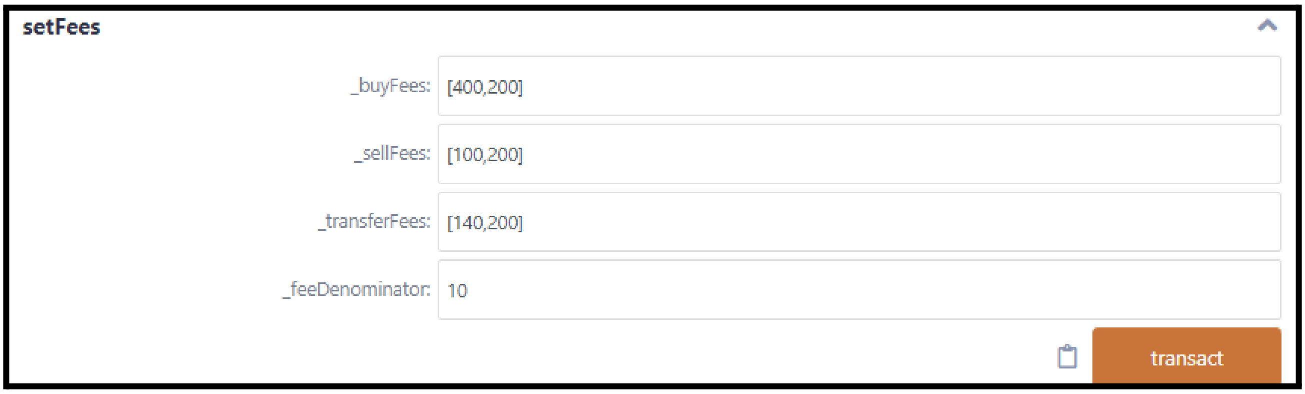
No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

(1) Fee Percentage limit is not set:



setFees

_buyFees: [400,200]

_sellFees: [100,200]

_transferFees: [140,200]

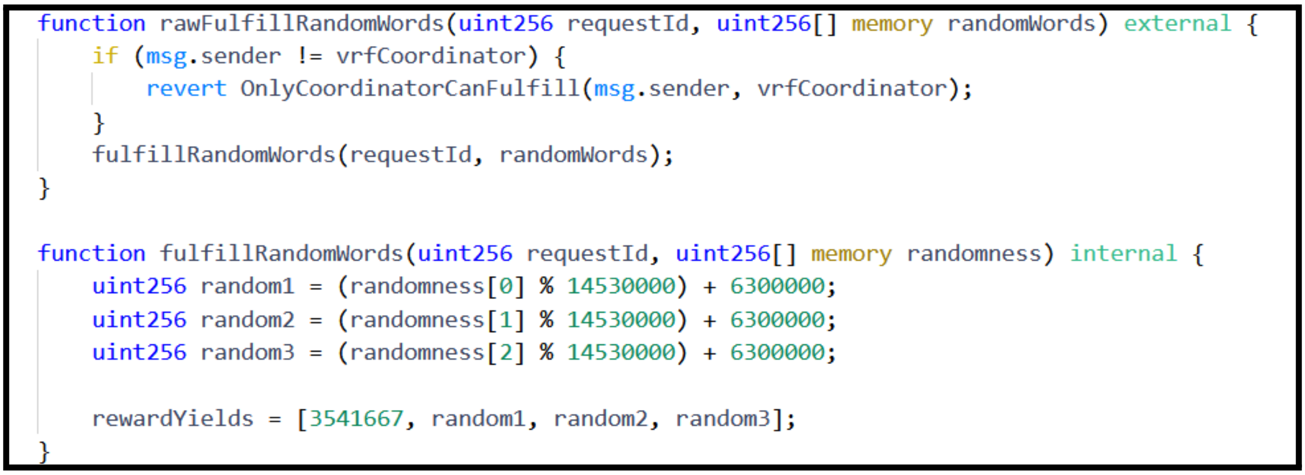
_feeDenominator: 10

The owner can set the individual fee percentage to any variable. This might deter investors as they could be wary that these fees might one day be set to 100% to force transfers to go to the contract owner.

Resolution: Consider adding an explicit limit to the fee percentage.

Status: Fixed

(2) Range validation is missing:



```
function rawFulfillRandomWords(uint256 requestId, uint256[] memory randomWords) external {
    if (msg.sender != vrfCoordinator) {
        revert OnlyCoordinatorCanFulfill(msg.sender, vrfCoordinator);
    }
    fulfillRandomWords(requestId, randomWords);
}

function fulfillRandomWords(uint256 requestId, uint256[] memory randomness) internal {
    uint256 random1 = (randomness[0] % 14530000) + 6300000;
    uint256 random2 = (randomness[1] % 14530000) + 6300000;
    uint256 random3 = (randomness[2] % 14530000) + 6300000;

    rewardYields = [3541667, random1, random2, random3];
}
```

A rawFulfillRandomWords function has a randomWords array input which requires 3 index input. It reverts if input is <3 array index.

Resolution: We suggest checking for the minimum length of the array.

Status: Acknowledged

Low

(1) Function input parameters lack of check:

Variable validation is not performed in below functions:

- setFeeReceivers = _liquidityReceiver, _treasuryReceiver, _teamReceiver, _burnReceiver
- setBUSD = _busdToken
- setRouter = _router

Resolution: We advise to put validation : int type variables should not be empty and greater than 0 and address type variables should not be address(0).

Status: Acknowledged

(2) AutomatedMarketMakerPair set manually after setting router:

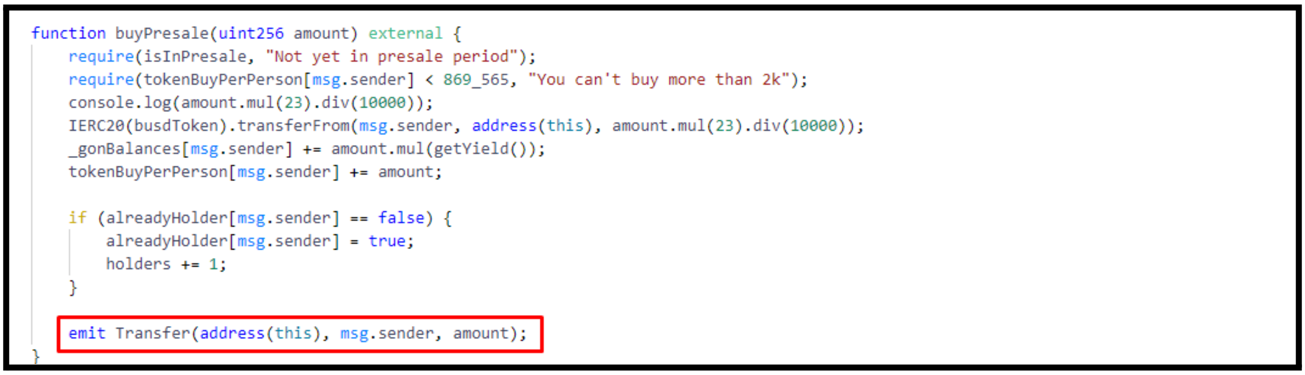
Currently, the token owner needs to take care to manually call setAutomatedMarketMakerPair whenever they call setRouter. If they forget to do this, then the automatedMarketMakerPairs mapping will fail to contain the updated liquidity pair address.

Resolution: Add a call to setAutomatedMarketMakerPair from setRouter.

Status: Acknowledged

(3) Invalid parameters:

In the buyPresale function, a transfer event has been logged for sending REIGN tokens from caller to contract address. But the actual transfer has been done from the contract address to the caller.



```
function buyPresale(uint256 amount) external {
    require(isInPresale, "Not yet in presale period");
    require(tokenBuyPerPerson[msg.sender] < 869_565, "You can't buy more than 2k");
    console.log(amount.mul(23).div(10000));
    IERC20(busdToken).transferFrom(msg.sender, address(this), amount.mul(23).div(10000));
    _gonBalances[msg.sender] += amount.mul(getYield());
    tokenBuyPerPerson[msg.sender] += amount;

    if (alreadyHolder[msg.sender] == false) {
        alreadyHolder[msg.sender] = true;
        holders += 1;
    }

    emit Transfer(address(this), msg.sender, amount);
}
```

Resolution: We suggest correcting the transfer event parameters.

Status: Acknowledged

Very Low / Informational / Best practices:

(1) Immutable variables:

rewardYieldDenominator is set only in the initialize function.

Resolution: We suggest declaring it as an Immutable variable. It will save some gas.

Status: Acknowledged

(2) Unused variables / Events:

Unused Variables:

- MAX_SUPPLY
- percentageForLessThanSevenDays
- percentageForMoreThanSevenDays

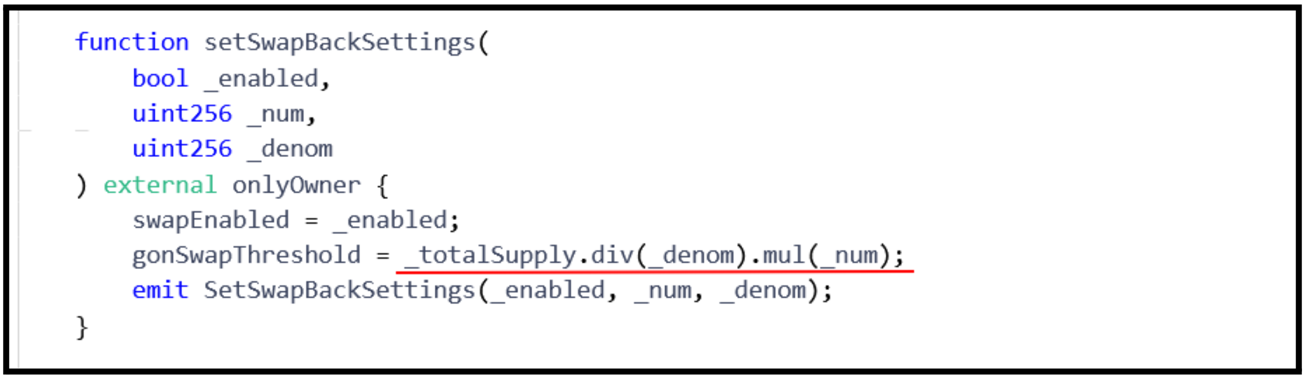
Unused Events:

- SetRewardYield
- SetIsLiquidityInBnb

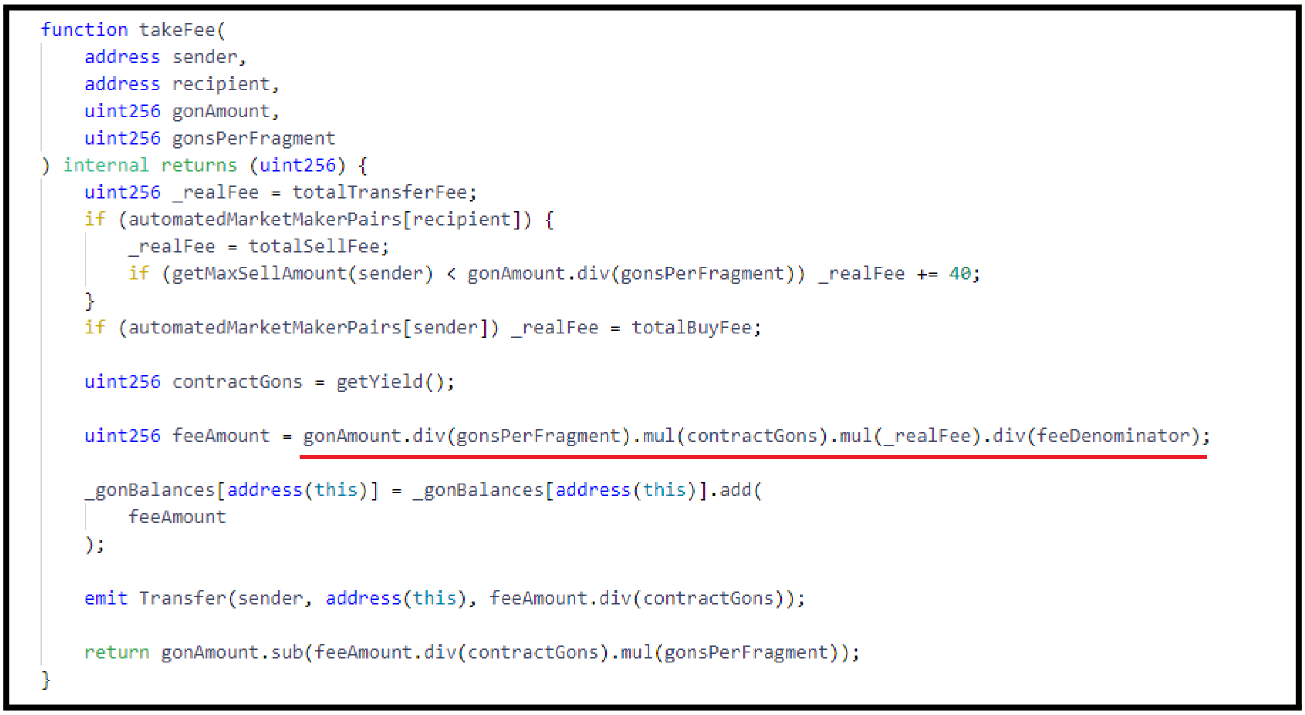
Resolution: We suggest removing unused variables and events.

Status: Acknowledged

(3) Division before multiplication:



```
function setSwapBackSettings(
    bool _enabled,
    uint256 _num,
    uint256 _denom
) external onlyOwner {
    swapEnabled = _enabled;
    gonSwapThreshold = _totalSupply.div(_denom).mul(_num);
    emit SetSwapBackSettings(_enabled, _num, _denom);
}
```



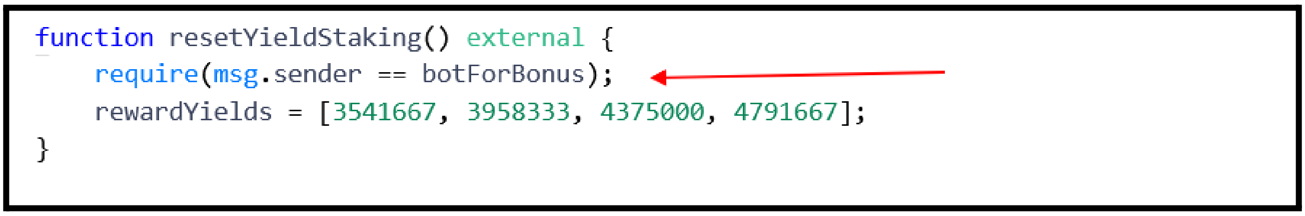
```
function takeFee(
    address sender,
    address recipient,
    uint256 gonAmount,
    uint256 gonsPerFragment
) internal returns (uint256) {
    uint256 _realFee = totalTransferFee;
    if (automatedMarketMakerPairs[recipient]) {
        _realFee = totalSellFee;
        if (getMaxSellAmount(sender) < gonAmount.div(gonsPerFragment)) _realFee += 40;
    }
    if (automatedMarketMakerPairs[sender]) _realFee = totalBuyFee;

    uint256 contractGons = getYield();

    uint256 feeAmount = gonAmount.div(gonsPerFragment).mul(contractGons).mul(_realFee).div(feeDenominator);

    _gonBalances[address(this)] = _gonBalances[address(this)].add(
        feeAmount
    );

    emit Transfer(sender, address(this), feeAmount.div(contractGons));

    return gonAmount.sub(feeAmount.div(contractGons).mul(gonsPerFragment));
}
```

Solidity being resource constraint language, dividing any amount and then multiplying will cause discrepancy in the outcome. Therefore always multiply the amount first and then divide it.

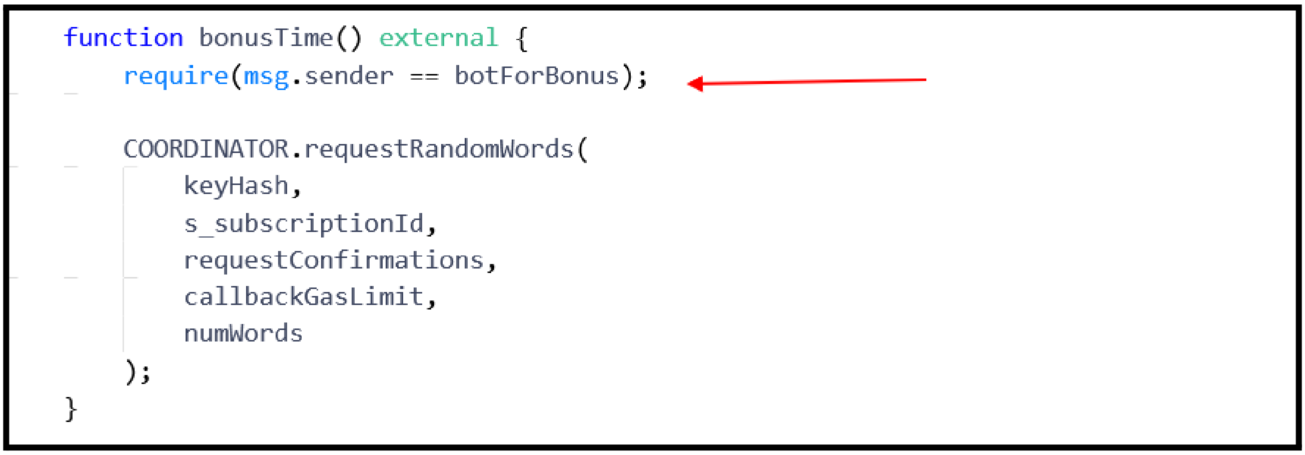
Resolution: Consider ordering multiplication before division.

Status: Acknowledged

(4) Missing Error Message:



```
function resetYieldStaking() external {
    require(msg.sender == botForBonus);
    rewardYields = [3541667, 3958333, 4375000, 4791667];
}
```



```
function bonusTime() external {
    require(msg.sender == botForBonus);

    COORDINATOR.requestRandomWords(
        keyHash,
        s_subscriptionId,
        requestConfirmations,
        callbackGasLimit,
        numWords
    );
}
```

Error Messages are missing in some functions. Requirements must have error messages.

Resolution: We suggest adding appropriate Error Messages if required.

Status: Acknowledged

(5) Mint doesn’t work without stake:

Mint reverts if the user has not staked any amount.

Resolution: If the user has any max limit to mint tokens then we suggest putting a validation or alert message to acknowledge the user.

Status: Acknowledged

Centralization

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

- setBlacklist: The owner can set Blacklist addresses.
- mint: The owner can mint a token.
- setNextRebase: The owner can set the next rebase.
- setAutoRebase: The owner can set the auto rebase.
- setRebaseFrequency: The owner can set rebase frequency value.
- setBotForBonus: The owner can set bot for bonuses.
- bonusTime: The owner can set bonus time.
- setChainLinkParam: The owner can set chain link parameter values.
- resetYieldStaking: The owner can reset yield staking values.
- setFeeExempt: The owner can set fee exempt addresses and values.
- setSwapBackSettings: The owner can set swap back settings value.
- setFees: The owner can set buy fee, sell fee and transfer fees.
- setRouter: The owner can set the router address.
- setFeeReceivers: The owner can set liquidity receiver fee, treasury receiver fee, team receiver fee, burn receiver fees.
- setStakeDuration: The owner can set stake duration values.
- setStakingTypeCount: The owner can set staking type count values.
- setPresalePeriod: The owner can set the presale period value.
- setBUSD: The owner can set BUSD value.
- setAutomatedMarketMakerPair: The owner can set an automated market maker pair address and value.
- setInitialDistributionFinished: The owner can set initial distribution finished values.
- clearStuckBalance: The owner can clear the stuck balance address.

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

Conclusion

We were given a contract code in the form of a file And we have used all possible tests based on given objects as files. We have observed 2 medium severity issues, 3 low severity issues and some informational issues. All the issues have been resolved/acknowledged in the revised code. 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 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).

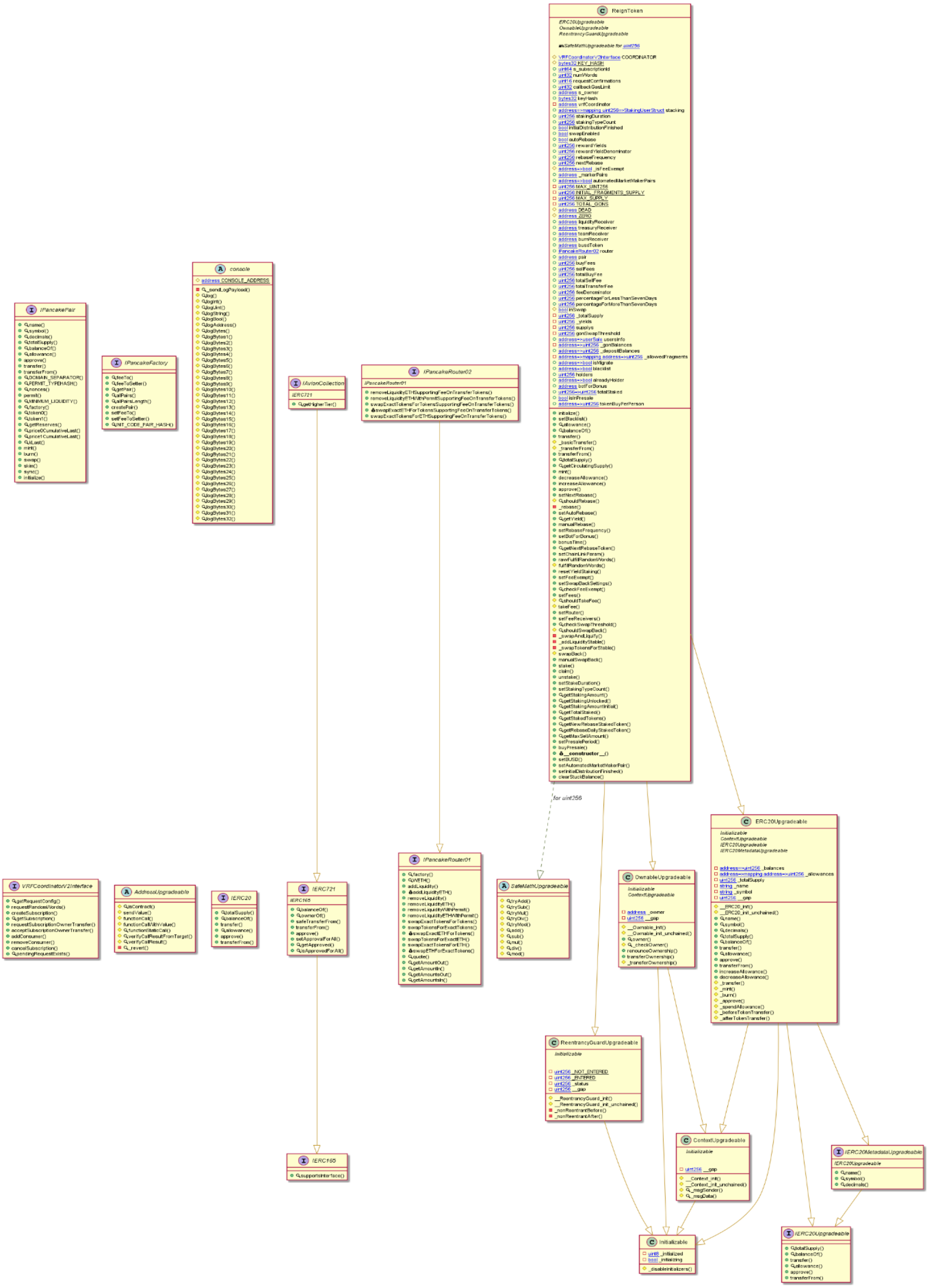
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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.

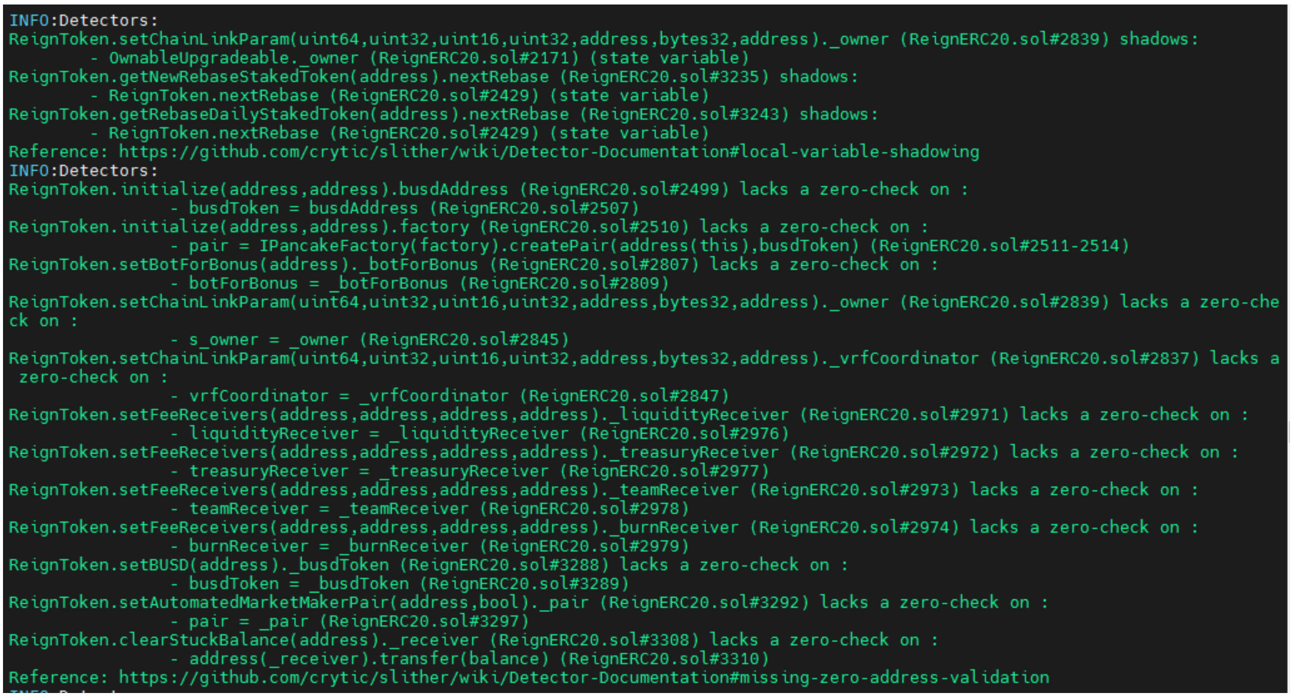
Appendix

Code Flow Diagram - Reign Token

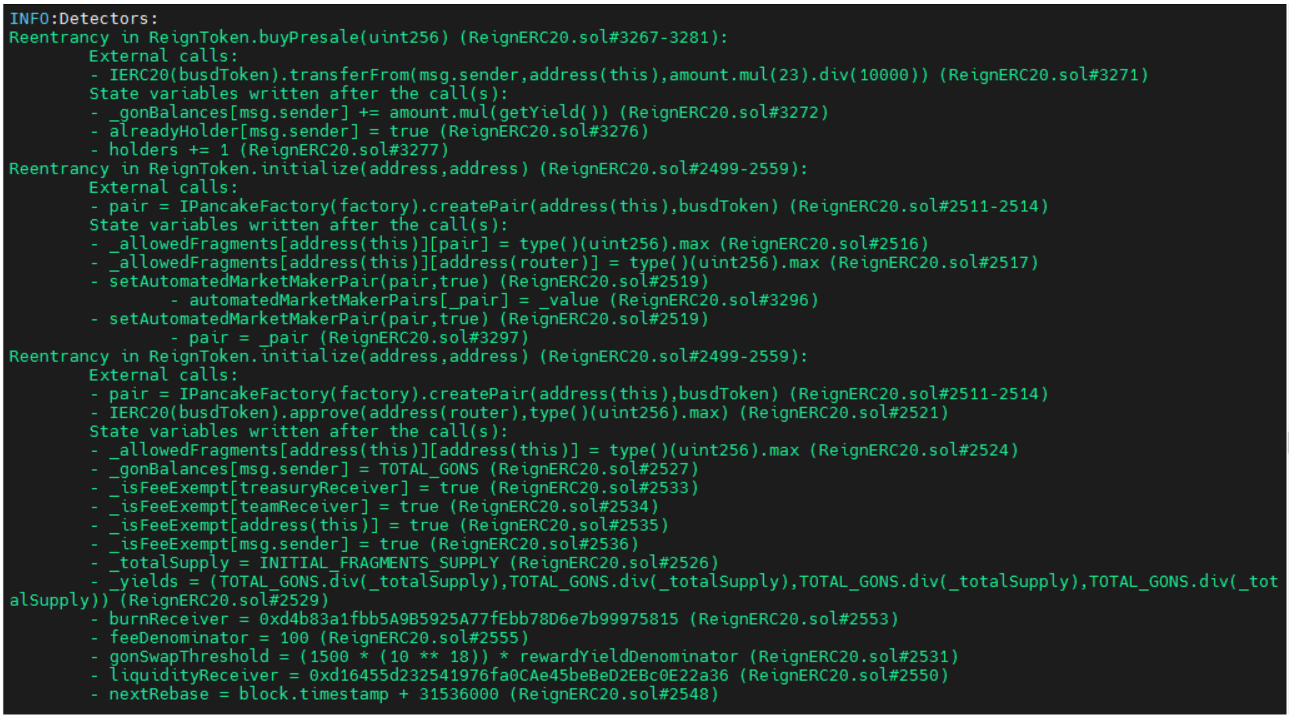


Slither Results Log

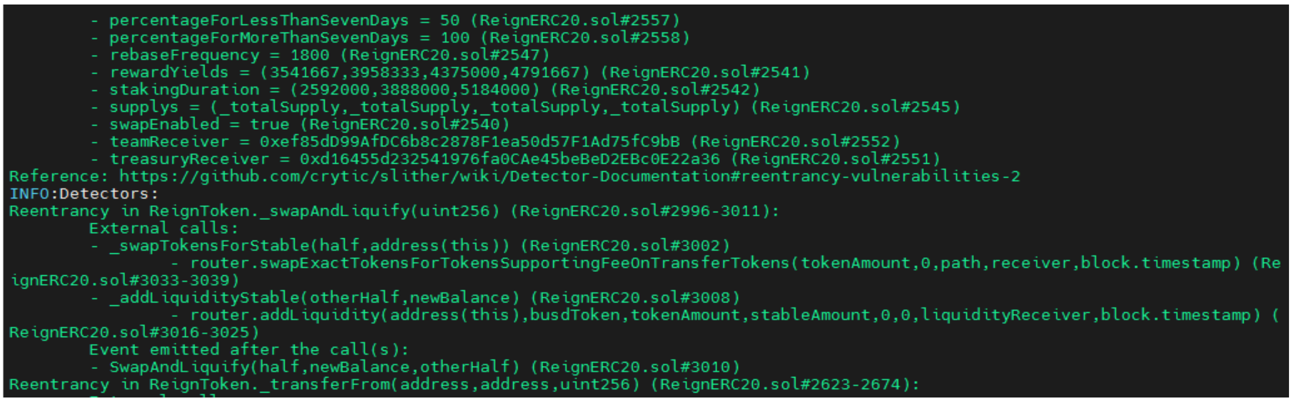
Slither Log >> ReignERC20.sol



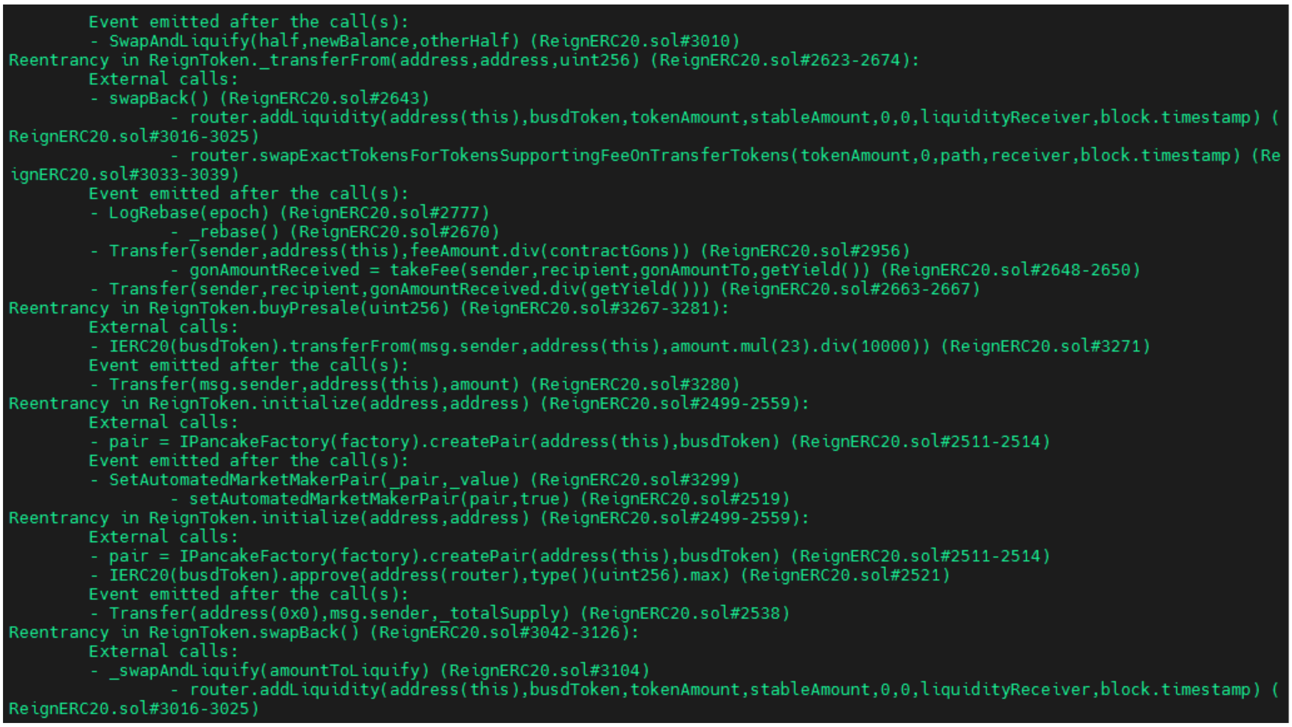
```
INFO:Detectors:
ReignToken.setChainLinkParam(uint64,uint32,uint16,uint32,address,bytes32,address)._owner (ReignERC20.sol#2839) shadows:
        - OwnableUpgradeable._owner (ReignERC20.sol#2171) (state variable)
ReignToken.getNewRebaseStakedToken(address).nextRebase (ReignERC20.sol#3235) shadows:
        - ReignToken.nextRebase (ReignERC20.sol#2429) (state variable)
ReignToken.getRebaseDailyStakedToken(address).nextRebase (ReignERC20.sol#3243) shadows:
        - ReignToken.nextRebase (ReignERC20.sol#2429) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
ReignToken.initialize(address,address).busdAddress (ReignERC20.sol#2499) lacks a zero-check on :
                - busdToken = busdAddress (ReignERC20.sol#2507)
ReignToken.initialize(address,address).factory (ReignERC20.sol#2510) lacks a zero-check on :
                - pair = IPancakeFactory(factory).createPair(address(this),busdToken) (ReignERC20.sol#2511-2514)
ReignToken.setBotForBonus(address)._botForBonus (ReignERC20.sol#2807) lacks a zero-check on :
                - botForBonus = _botForBonus (ReignERC20.sol#2809)
ReignToken.setChainLinkParam(uint64,uint32,uint16,uint32,address,bytes32,address)._owner (ReignERC20.sol#2839) lacks a zero-che
ck on :
                - s_owner = _owner (ReignERC20.sol#2845)
ReignToken.setChainLinkParam(uint64,uint32,uint16,uint32,address,bytes32,address)._vrfCoordinator (ReignERC20.sol#2837) lacks a
 zero-check on :
                - vrfCoordinator = _vrfCoordinator (ReignERC20.sol#2847)
ReignToken.setFeeReceivers(address,address,address,address)._liquidityReceiver (ReignERC20.sol#2971) lacks a zero-check on :
                - liquidityReceiver = _liquidityReceiver (ReignERC20.sol#2976)
ReignToken.setFeeReceivers(address,address,address,address)._treasuryReceiver (ReignERC20.sol#2972) lacks a zero-check on :
                - treasuryReceiver = _treasuryReceiver (ReignERC20.sol#2977)
ReignToken.setFeeReceivers(address,address,address,address)._teamReceiver (ReignERC20.sol#2973) lacks a zero-check on :
                - teamReceiver = _teamReceiver (ReignERC20.sol#2978)
ReignToken.setFeeReceivers(address,address,address,address)._burnReceiver (ReignERC20.sol#2974) lacks a zero-check on :
                - burnReceiver = _burnReceiver (ReignERC20.sol#2979)
ReignToken.setBUSD(address)._busdToken (ReignERC20.sol#3288) lacks a zero-check on :
                - busdToken = _busdToken (ReignERC20.sol#3289)
ReignToken.setAutomatedMarketMakerPair(address,bool)._pair (ReignERC20.sol#3292) lacks a zero-check on :
                - pair = _pair (ReignERC20.sol#3297)
ReignToken.clearStuckBalance(address)._receiver (ReignERC20.sol#3308) lacks a zero-check on :
                - address(_receiver).transfer(balance) (ReignERC20.sol#3310)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```



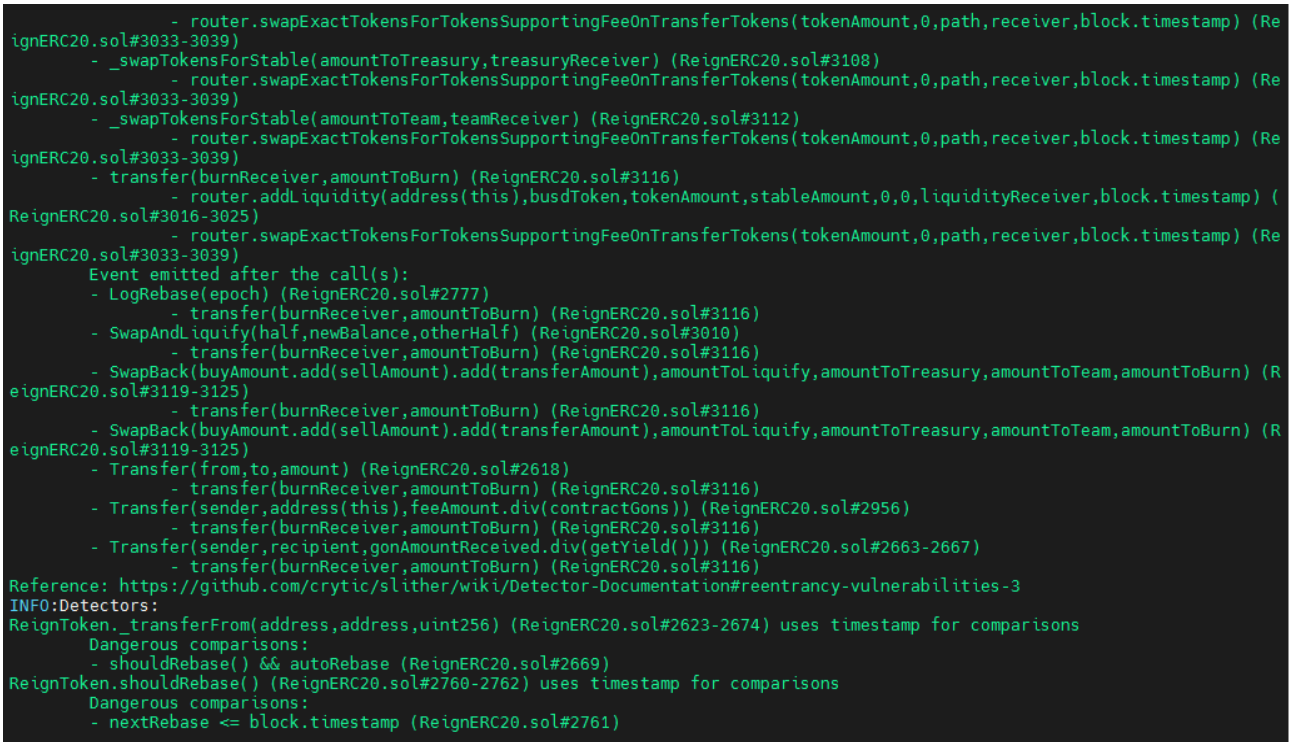
```
INFO:Detectors:
Reentrancy in ReignToken.buyPresale(uint256) (ReignERC20.sol#3267-3281):
        External calls:
        - IERC20(busdToken).transferFrom(msg.sender,address(this),amount.mul(23).div(10000)) (ReignERC20.sol#3271)
        State variables written after the call(s):
        - _gonBalances[msg.sender] += amount.mul(getYield()) (ReignERC20.sol#3272)
        - alreadyHolder[msg.sender] = true (ReignERC20.sol#3276)
        - holders += 1 (ReignERC20.sol#3277)
Reentrancy in ReignToken.initialize(address,address) (ReignERC20.sol#2499-2559):
        External calls:
        - pair = IPancakeFactory(factory).createPair(address(this),busdToken) (ReignERC20.sol#2511-2514)
        State variables written after the call(s):
        - _allowedFragments[address(this)][pair] = type()(uint256).max (ReignERC20.sol#2516)
        - _allowedFragments[address(this)][address(router)] = type()(uint256).max (ReignERC20.sol#2517)
        - setAutomatedMarketMakerPair(pair,true) (ReignERC20.sol#2519)
                - automatedMarketMakerPairs[_pair] = _value (ReignERC20.sol#3296)
        - setAutomatedMarketMakerPair(pair,true) (ReignERC20.sol#2519)
                - pair = _pair (ReignERC20.sol#3297)
Reentrancy in ReignToken.initialize(address,address) (ReignERC20.sol#2499-2559):
        External calls:
        - pair = IPancakeFactory(factory).createPair(address(this),busdToken) (ReignERC20.sol#2511-2514)
        - IERC20(busdToken).approve(address(router),type()(uint256).max) (ReignERC20.sol#2521)
        State variables written after the call(s):
        - _allowedFragments[address(this)][address(this)] = type()(uint256).max (ReignERC20.sol#2524)
        - _gonBalances[msg.sender] = TOTAL_GONS (ReignERC20.sol#2527)
        - _isFeeExempt[treasuryReceiver] = true (ReignERC20.sol#2533)
        - _isFeeExempt[teamReceiver] = true (ReignERC20.sol#2534)
        - _isFeeExempt[address(this)] = true (ReignERC20.sol#2535)
        - _isFeeExempt[msg.sender] = true (ReignERC20.sol#2536)
        - _totalSupply = INITIAL_FRAGMENTS_SUPPLY (ReignERC20.sol#2526)
        - _yields = (TOTAL_GONS.div(_totalSupply),TOTAL_GONS.div(_totalSupply),TOTAL_GONS.div(_totalSupply),TOTAL_GONS.div(_tot
alSupply)) (ReignERC20.sol#2529)
        - burnReceiver = 0xd4b83a1fbb5A9B5925A77fEbb78D6e7b99975815 (ReignERC20.sol#2553)
        - feeDenominator = 100 (ReignERC20.sol#2555)
        - gonSwapThreshold = (1500 * (10 ** 18)) * rewardYieldDenominator (ReignERC20.sol#2531)
        - liquidityReceiver = 0xd16455d232541976fa0CAe45beBeD2EBc0E22a36 (ReignERC20.sol#2550)
        - nextRebase = block.timestamp + 31536000 (ReignERC20.sol#2548)
```



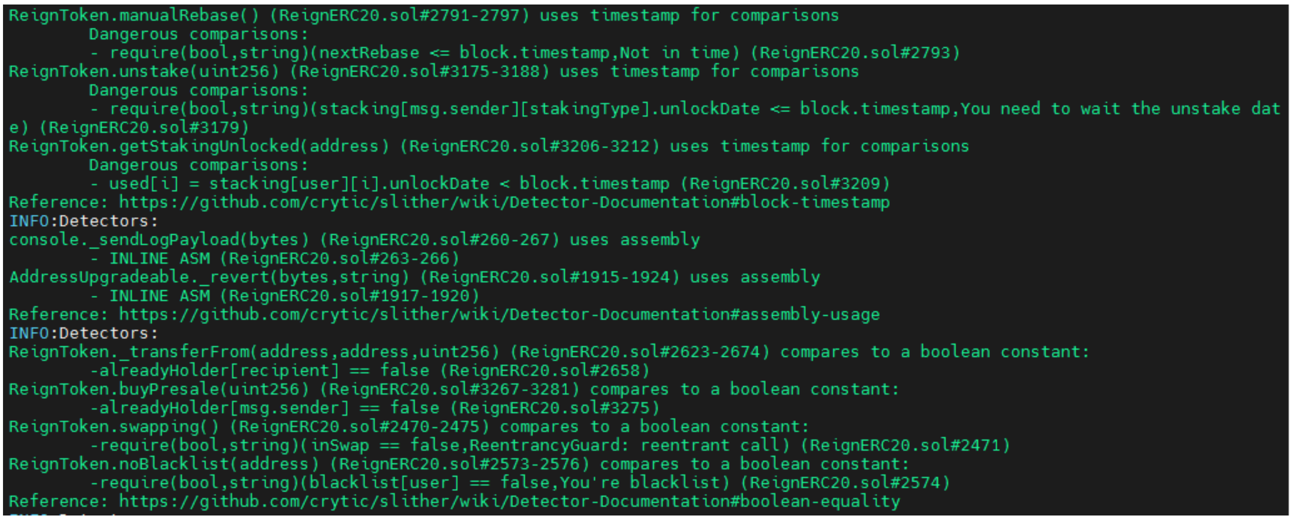
```
        - percentageForLessThanSevenDays = 50 (ReignERC20.sol#2557)
        - percentageForMoreThanSevenDays = 100 (ReignERC20.sol#2558)
        - rebaseFrequency = 1800 (ReignERC20.sol#2547)
        - rewardYields = (3541667,3958333,4375000,4791667) (ReignERC20.sol#2541)
        - stakingDuration = (2592000,3888000,5184000) (ReignERC20.sol#2542)
        - supplys = (_totalSupply,_totalSupply,_totalSupply,_totalSupply) (ReignERC20.sol#2545)
        - swapEnabled = true (ReignERC20.sol#2540)
        - teamReceiver = 0xef85dD99AfDC6b8c2878F1ea50d57F1Ad75fC9bB (ReignERC20.sol#2552)
        - treasuryReceiver = 0xd16455d232541976fa0CAe45beBeD2EBc0E22a36 (ReignERC20.sol#2551)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in ReignToken._swapAndLiquify(uint256) (ReignERC20.sol#2996-3011):
        External calls:
        - _swapTokensForStable(half,address(this)) (ReignERC20.sol#3002)
                - router.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,receiver,block.timestamp) (Re
ignERC20.sol#3033-3039)
        - _addLiquidityStable(otherHalf,newBalance) (ReignERC20.sol#3008)
                - router.addLiquidity(address(this),busdToken,tokenAmount,stableAmount,0,0,liquidityReceiver,block.timestamp) (
ReignERC20.sol#3016-3025)
        Event emitted after the call(s):
        - SwapAndLiquify(half,newBalance,otherHalf) (ReignERC20.sol#3010)
Reentrancy in ReignToken._transferFrom(address,address,uint256) (ReignERC20.sol#2623-2674):
```



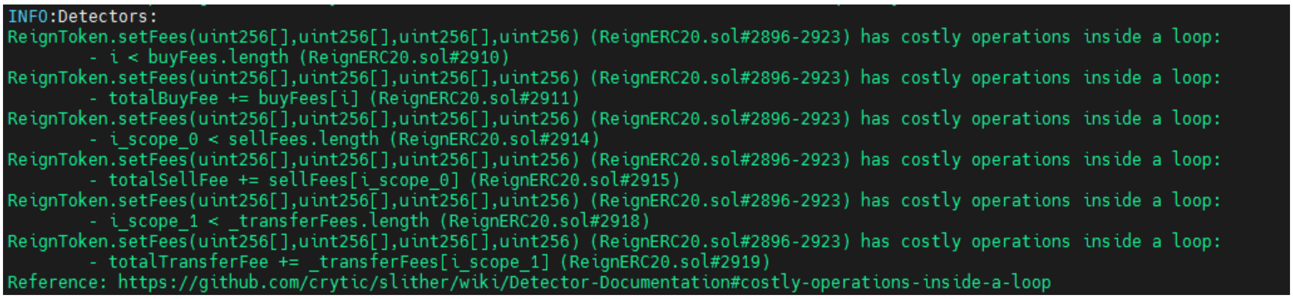
```
        Event emitted after the call(s):
        - SwapAndLiquify(half,newBalance,otherHalf) (ReignERC20.sol#3010)
Reentrancy in ReignToken._transferFrom(address,address,uint256) (ReignERC20.sol#2623-2674):
        External calls:
        - swapBack() (ReignERC20.sol#2643)
                - router.addLiquidity(address(this),busdToken,tokenAmount,stableAmount,0,0,liquidityReceiver,block.timestamp) (
ReignERC20.sol#3016-3025)
                - router.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,receiver,block.timestamp) (Re
ignERC20.sol#3033-3039)
        Event emitted after the call(s):
        - LogRebase(epoch) (ReignERC20.sol#2777)
                - _rebase() (ReignERC20.sol#2670)
        - Transfer(sender,address(this),feeAmount.div(contractGons)) (ReignERC20.sol#2956)
                - gonAmountReceived = takeFee(sender,recipient,gonAmountTo,getYield()) (ReignERC20.sol#2648-2650)
        - Transfer(sender,recipient,gonAmountReceived.div(getYield())) (ReignERC20.sol#2663-2667)
Reentrancy in ReignToken.buyPresale(uint256) (ReignERC20.sol#3267-3281):
        External calls:
        - IERC20(busdToken).transferFrom(msg.sender,address(this),amount.mul(23).div(10000)) (ReignERC20.sol#3271)
        Event emitted after the call(s):
        - Transfer(msg.sender,address(this),amount) (ReignERC20.sol#3280)
Reentrancy in ReignToken.initialize(address,address) (ReignERC20.sol#2499-2559):
        External calls:
        - pair = IPancakeFactory(factory).createPair(address(this),busdToken) (ReignERC20.sol#2511-2514)
        Event emitted after the call(s):
        - SetAutomatedMarketMakerPair(_pair,_value) (ReignERC20.sol#3299)
                - setAutomatedMarketMakerPair(pair,true) (ReignERC20.sol#2519)
Reentrancy in ReignToken.initialize(address,address) (ReignERC20.sol#2499-2559):
        External calls:
        - pair = IPancakeFactory(factory).createPair(address(this),busdToken) (ReignERC20.sol#2511-2514)
        - IERC20(busdToken).approve(address(router),type()(uint256).max) (ReignERC20.sol#2521)
        Event emitted after the call(s):
        - Transfer(address(0x0),msg.sender,_totalSupply) (ReignERC20.sol#2538)
Reentrancy in ReignToken.swapBack() (ReignERC20.sol#3042-3126):
        External calls:
        - _swapAndLiquify(amountToLiquify) (ReignERC20.sol#3104)
                - router.addLiquidity(address(this),busdToken,tokenAmount,stableAmount,0,0,liquidityReceiver,block.timestamp) (
ReignERC20.sol#3016-3025)
```



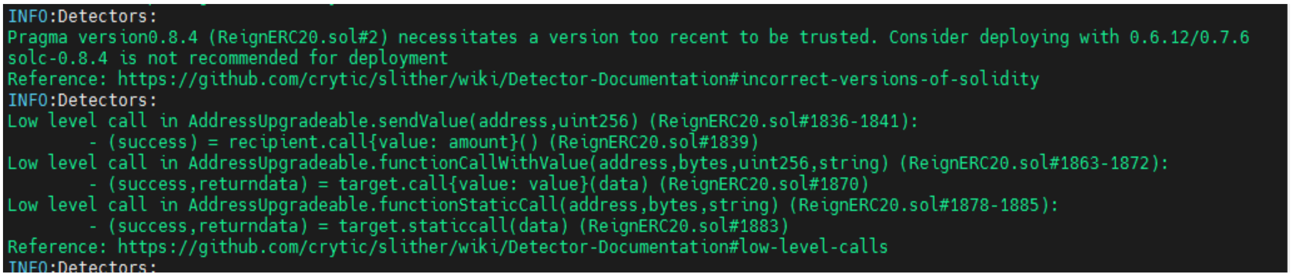
```
                - router.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,receiver,block.timestamp) (Re
ignERC20.sol#3033-3039)
        - _swapTokensForStable(amountToTreasury,treasuryReceiver) (ReignERC20.sol#3108)
                - router.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,receiver,block.timestamp) (Re
ignERC20.sol#3033-3039)
        - _swapTokensForStable(amountToTeam,teamReceiver) (ReignERC20.sol#3112)
                - router.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,receiver,block.timestamp) (Re
ignERC20.sol#3033-3039)
        - transfer(burnReceiver,amountToBurn) (ReignERC20.sol#3116)
                - router.addLiquidity(address(this),busdToken,tokenAmount,stableAmount,0,0,liquidityReceiver,block.timestamp) (
ReignERC20.sol#3016-3025)
                - router.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,receiver,block.timestamp) (Re
ignERC20.sol#3033-3039)
        Event emitted after the call(s):
        - LogRebase(epoch) (ReignERC20.sol#2777)
                - transfer(burnReceiver,amountToBurn) (ReignERC20.sol#3116)
        - SwapAndLiquify(half,newBalance,otherHalf) (ReignERC20.sol#3010)
                - transfer(burnReceiver,amountToBurn) (ReignERC20.sol#3116)
        - SwapBack(buyAmount.add(sellAmount).add(transferAmount),amountToLiquify,amountToTreasury,amountToTeam,amountToBurn) (R
eignERC20.sol#3119-3125)
                - transfer(burnReceiver,amountToBurn) (ReignERC20.sol#3116)
        - SwapBack(buyAmount.add(sellAmount).add(transferAmount),amountToLiquify,amountToTreasury,amountToTeam,amountToBurn) (R
eignERC20.sol#3119-3125)
        - Transfer(from,to,amount) (ReignERC20.sol#2618)
                - transfer(burnReceiver,amountToBurn) (ReignERC20.sol#3116)
        - Transfer(sender,address(this),feeAmount.div(contractGons)) (ReignERC20.sol#2956)
                - transfer(burnReceiver,amountToBurn) (ReignERC20.sol#3116)
        - Transfer(sender,recipient,gonAmountReceived.div(getYield())) (ReignERC20.sol#2663-2667)
                - transfer(burnReceiver,amountToBurn) (ReignERC20.sol#3116)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
ReignToken._transferFrom(address,address,uint256) (ReignERC20.sol#2623-2674) uses timestamp for comparisons
        Dangerous comparisons:
        - shouldRebase() && autoRebase (ReignERC20.sol#2669)
ReignToken.shouldRebase() (ReignERC20.sol#2760-2762) uses timestamp for comparisons
        Dangerous comparisons:
        - nextRebase <= block.timestamp (ReignERC20.sol#2761)
```



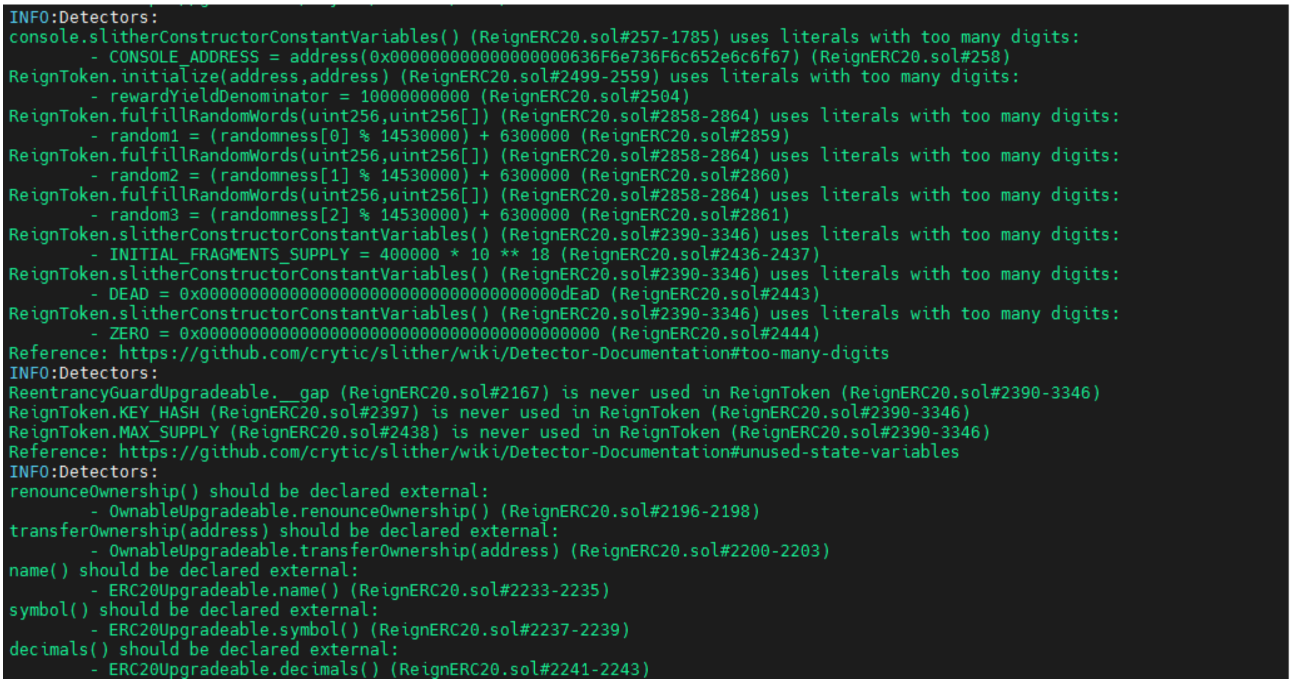
```
ReignToken.manualRebase() (ReignERC20.sol#2791-2797) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(nextRebase <= block.timestamp,Not in time) (ReignERC20.sol#2793)
ReignToken.unstake(uint256) (ReignERC20.sol#3175-3188) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(stacking[msg.sender][stakingType].unlockDate <= block.timestamp,You need to wait the unstake dat
e) (ReignERC20.sol#3179)
ReignToken.getStakingUnlocked(address) (ReignERC20.sol#3206-3212) uses timestamp for comparisons
        Dangerous comparisons:
        - used[i] = stacking[user][i].unlockDate < block.timestamp (ReignERC20.sol#3209)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
console._sendLogPayload(bytes) (ReignERC20.sol#260-267) uses assembly
        - INLINE ASM (ReignERC20.sol#263-266)
AddressUpgradeable._revert(bytes,string) (ReignERC20.sol#1915-1924) uses assembly
        - INLINE ASM (ReignERC20.sol#1917-1920)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
ReignToken._transferFrom(address,address,uint256) (ReignERC20.sol#2623-2674) compares to a boolean constant:
        -alreadyHolder[recipient] == false (ReignERC20.sol#2658)
ReignToken.buyPresale(uint256) (ReignERC20.sol#3267-3281) compares to a boolean constant:
        -alreadyHolder[msg.sender] == false (ReignERC20.sol#3275)
ReignToken.swapping() (ReignERC20.sol#2470-2475) compares to a boolean constant:
        -require(bool,string)(inSwap == false,ReentrancyGuard: reentrant call) (ReignERC20.sol#2471)
ReignToken.noBlacklist(address) (ReignERC20.sol#2573-2576) compares to a boolean constant:
        -require(bool,string)(blacklist[user] == false,You're blacklist) (ReignERC20.sol#2574)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```



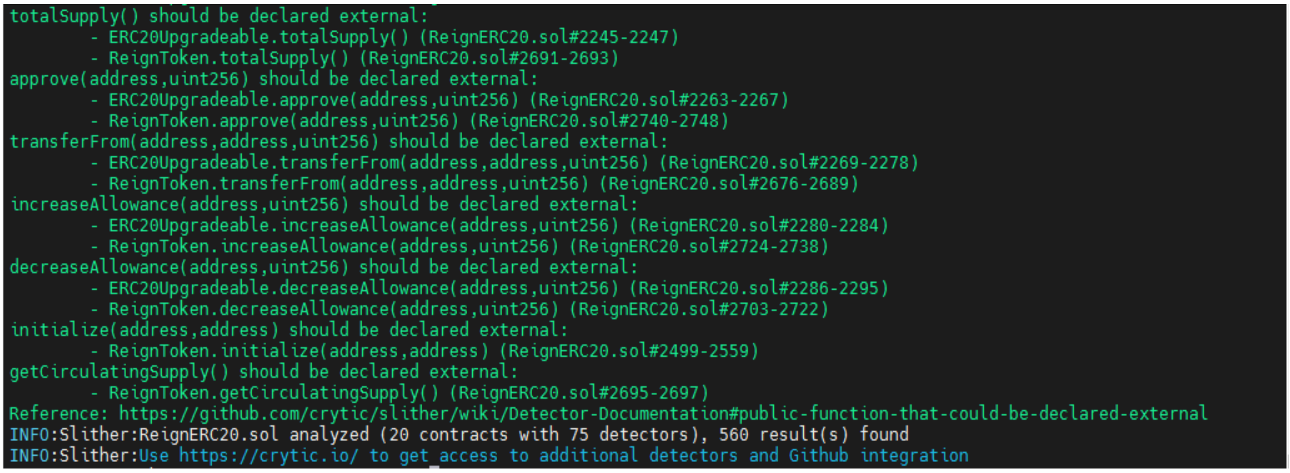
```
INFO:Detectors:
ReignToken.setFees(uint256[],uint256[],uint256[],uint256) (ReignERC20.sol#2896-2923) has costly operations inside a loop:
        - i < buyFees.length (ReignERC20.sol#2910)
ReignToken.setFees(uint256[],uint256[],uint256[],uint256) (ReignERC20.sol#2896-2923) has costly operations inside a loop:
        - totalBuyFee += buyFees[i] (ReignERC20.sol#2911)
ReignToken.setFees(uint256[],uint256[],uint256[],uint256) (ReignERC20.sol#2896-2923) has costly operations inside a loop:
        - i_scope_0 < sellFees.length (ReignERC20.sol#2914)
ReignToken.setFees(uint256[],uint256[],uint256[],uint256) (ReignERC20.sol#2896-2923) has costly operations inside a loop:
        - totalSellFee += sellFees[i_scope_0] (ReignERC20.sol#2915)
ReignToken.setFees(uint256[],uint256[],uint256[],uint256) (ReignERC20.sol#2896-2923) has costly operations inside a loop:
        - i_scope_1 < _transferFees.length (ReignERC20.sol#2918)
ReignToken.setFees(uint256[],uint256[],uint256[],uint256) (ReignERC20.sol#2896-2923) has costly operations inside a loop:
        - totalTransferFee += _transferFees[i_scope_1] (ReignERC20.sol#2919)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```



```
INFO:Detectors:
Pragma version0.8.4 (ReignERC20.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in AddressUpgradeable.sendValue(address,uint256) (ReignERC20.sol#1836-1841):
        - (success) = recipient.call{value: amount}() (ReignERC20.sol#1839)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (ReignERC20.sol#1863-1872):
        - (success,returndata) = target.call{value: value}(data) (ReignERC20.sol#1870)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (ReignERC20.sol#1878-1885):
        - (success,returndata) = target.staticcall(data) (ReignERC20.sol#1883)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
```



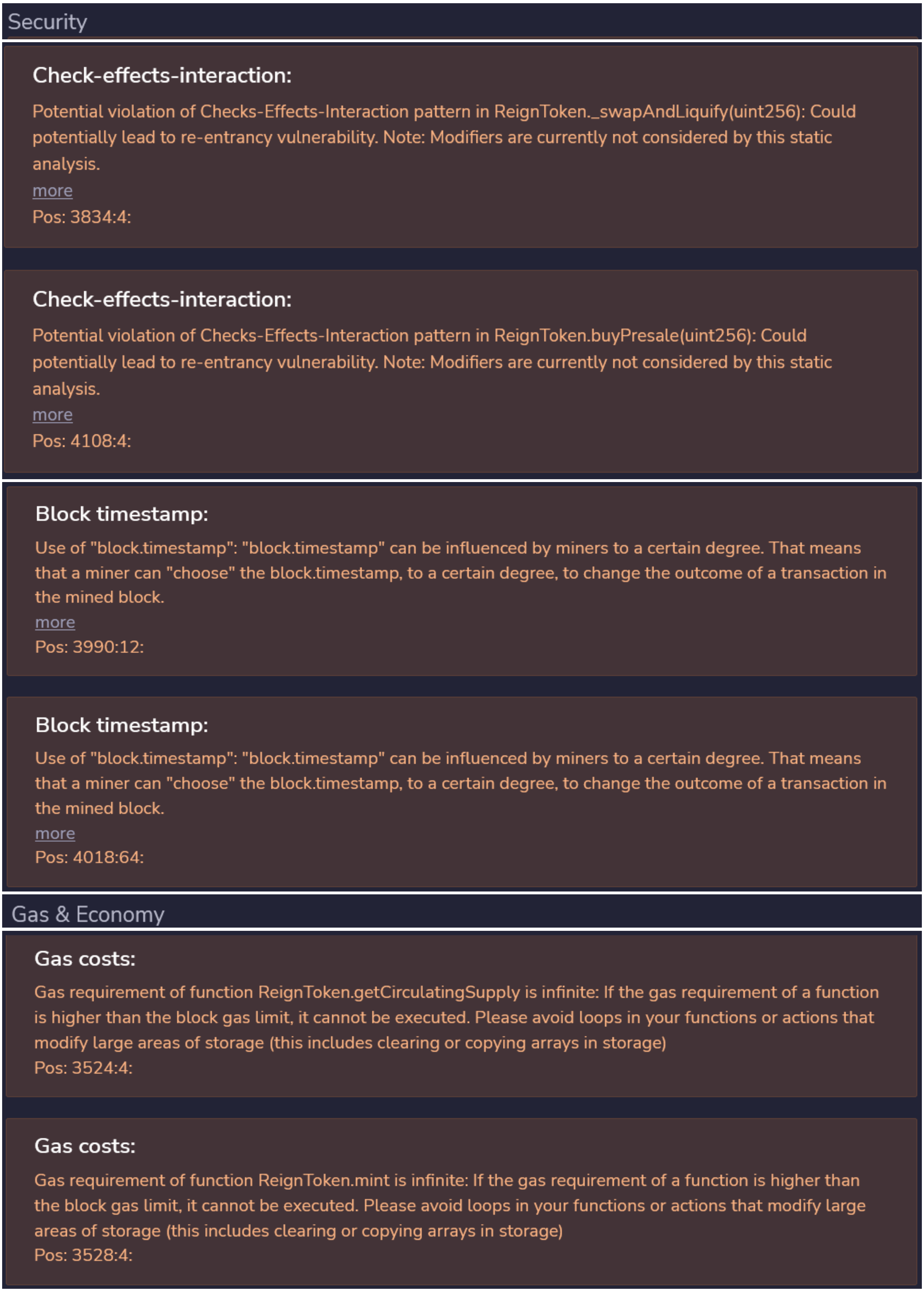
```
INFO:Detectors:
console.slitherConstructorConstantVariables() (ReignERC20.sol#257-1785) uses literals with too many digits:
        - CONSOLE_ADDRESS = address(0x000000000000000000636F6e736F6c652e6c6f67) (ReignERC20.sol#258)
ReignToken.initialize(address,address) (ReignERC20.sol#2499-2559) uses literals with too many digits:
        - rewardYieldDenominator = 10000000000 (ReignERC20.sol#2504)
ReignToken.fulfillRandomWords(uint256,uint256[]) (ReignERC20.sol#2858-2864) uses literals with too many digits:
        - random1 = (randomness[0] % 14530000) + 6300000 (ReignERC20.sol#2859)
ReignToken.fulfillRandomWords(uint256,uint256[]) (ReignERC20.sol#2858-2864) uses literals with too many digits:
        - random2 = (randomness[1] % 14530000) + 6300000 (ReignERC20.sol#2860)
ReignToken.fulfillRandomWords(uint256,uint256[]) (ReignERC20.sol#2858-2864) uses literals with too many digits:
        - random3 = (randomness[2] % 14530000) + 6300000 (ReignERC20.sol#2861)
ReignToken.slitherConstructorConstantVariables() (ReignERC20.sol#2390-3346) uses literals with too many digits:
        - INITIAL_FRAGMENTS_SUPPLY = 400000 * 10 ** 18 (ReignERC20.sol#2436-2437)
ReignToken.slitherConstructorConstantVariables() (ReignERC20.sol#2390-3346) uses literals with too many digits:
        - DEAD = 0x000000000000000000000000000000000000dEaD (ReignERC20.sol#2443)
ReignToken.slitherConstructorConstantVariables() (ReignERC20.sol#2390-3346) uses literals with too many digits:
        - ZERO = 0x0000000000000000000000000000000000000000 (ReignERC20.sol#2444)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
ReentrancyGuardUpgradeable.__gap (ReignERC20.sol#2167) is never used in ReignToken (ReignERC20.sol#2390-3346)
ReignToken.KEY_HASH (ReignERC20.sol#2397) is never used in ReignToken (ReignERC20.sol#2390-3346)
ReignToken.MAX_SUPPLY (ReignERC20.sol#2438) is never used in ReignToken (ReignERC20.sol#2390-3346)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
renounceOwnership() should be declared external:
        - OwnableUpgradeable.renounceOwnership() (ReignERC20.sol#2196-2198)
transferOwnership(address) should be declared external:
        - OwnableUpgradeable.transferOwnership(address) (ReignERC20.sol#2200-2203)
name() should be declared external:
        - ERC20Upgradeable.name() (ReignERC20.sol#2233-2235)
symbol() should be declared external:
        - ERC20Upgradeable.symbol() (ReignERC20.sol#2237-2239)
decimals() should be declared external:
        - ERC20Upgradeable.decimals() (ReignERC20.sol#2241-2243)
```



```
totalSupply() should be declared external:
        - ERC20Upgradeable.totalSupply() (ReignERC20.sol#2245-2247)
        - ReignToken.totalSupply() (ReignERC20.sol#2691-2693)
approve(address,uint256) should be declared external:
        - ERC20Upgradeable.approve(address,uint256) (ReignERC20.sol#2263-2267)
        - ReignToken.approve(address,uint256) (ReignERC20.sol#2740-2748)
transferFrom(address,address,uint256) should be declared external:
        - ERC20Upgradeable.transferFrom(address,address,uint256) (ReignERC20.sol#2269-2278)
        - ReignToken.transferFrom(address,address,uint256) (ReignERC20.sol#2676-2689)
increaseAllowance(address,uint256) should be declared external:
        - ERC20Upgradeable.increaseAllowance(address,uint256) (ReignERC20.sol#2280-2284)
        - ReignToken.increaseAllowance(address,uint256) (ReignERC20.sol#2724-2738)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20Upgradeable.decreaseAllowance(address,uint256) (ReignERC20.sol#2286-2295)
        - ReignToken.decreaseAllowance(address,uint256) (ReignERC20.sol#2703-2722)
initialize(address,address) should be declared external:
        - ReignToken.initialize(address,address) (ReignERC20.sol#2499-2559)
getCirculatingSupply() should be declared external:
        - ReignToken.getCirculatingSupply() (ReignERC20.sol#2695-2697)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:ReignERC20.sol analyzed (20 contracts with 75 detectors), 560 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

ReignERC20.sol



Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in ReignToken._swapAndLiquify(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 3834:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in ReignToken.buyPresale(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 4108:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 3990:12:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 4018:64:

Gas & Economy

Gas costs:

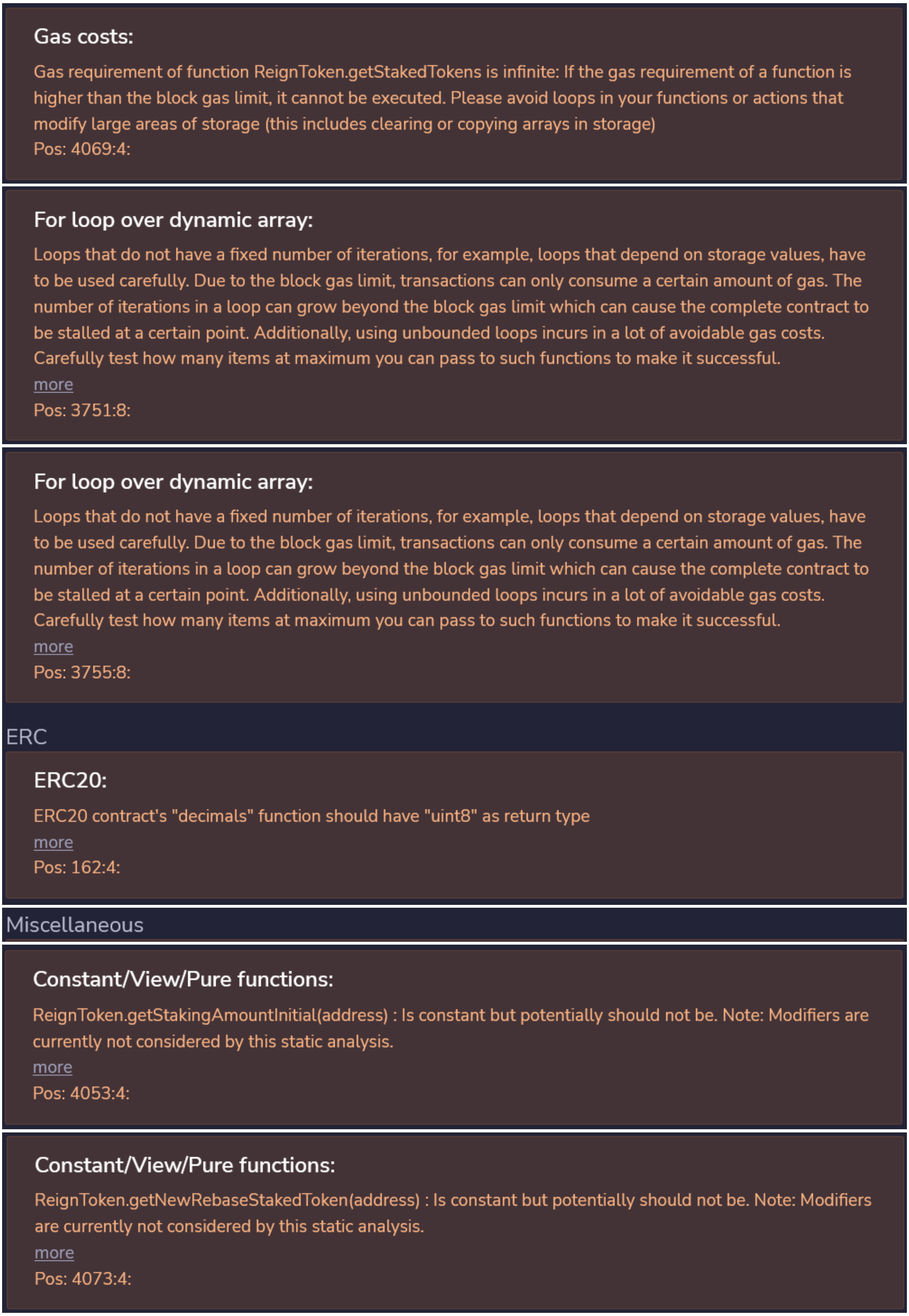
Gas requirement of function ReignToken.getCirculatingSupply is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 3524:4:

Gas costs:

Gas requirement of function ReignToken.mint 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: 3528:4:



Gas costs:

Gas requirement of function ReignToken.getStakedTokens 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: 4069:4:

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: 3751:8:

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: 3755:8:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 162:4:

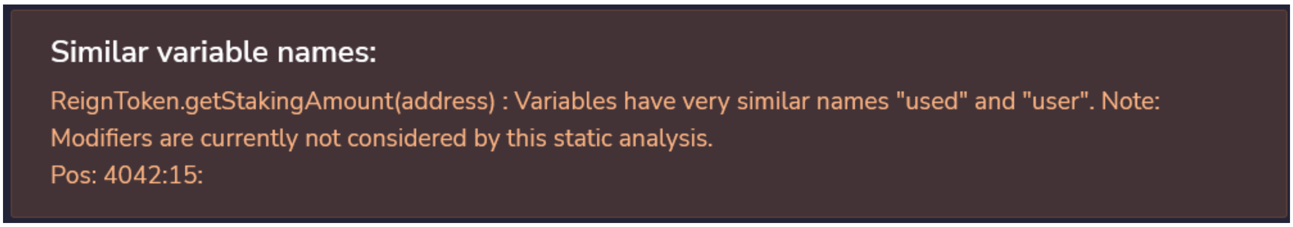
Miscellaneous

Constant/View/Pure functions:

ReignToken.getStakingAmountInitial(address) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 4053:4:

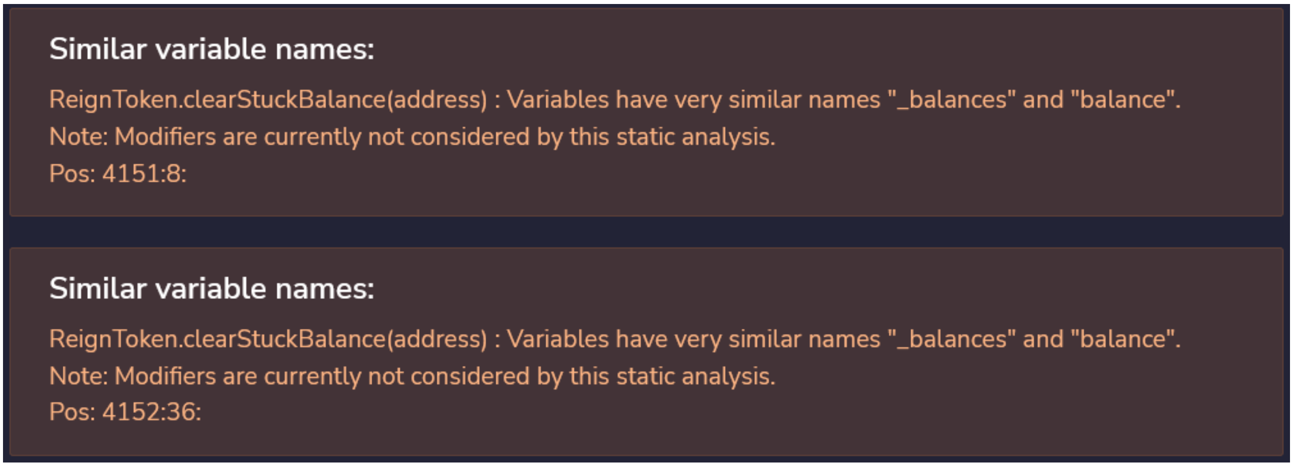
Constant/View/Pure functions:

ReignToken.getNewRebaseStakedToken(address) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 4073:4:



Similar variable names:

ReignToken.getStakingAmount(address) : Variables have very similar names "used" and "user". Note: Modifiers are currently not considered by this static analysis.
Pos: 4042:15:

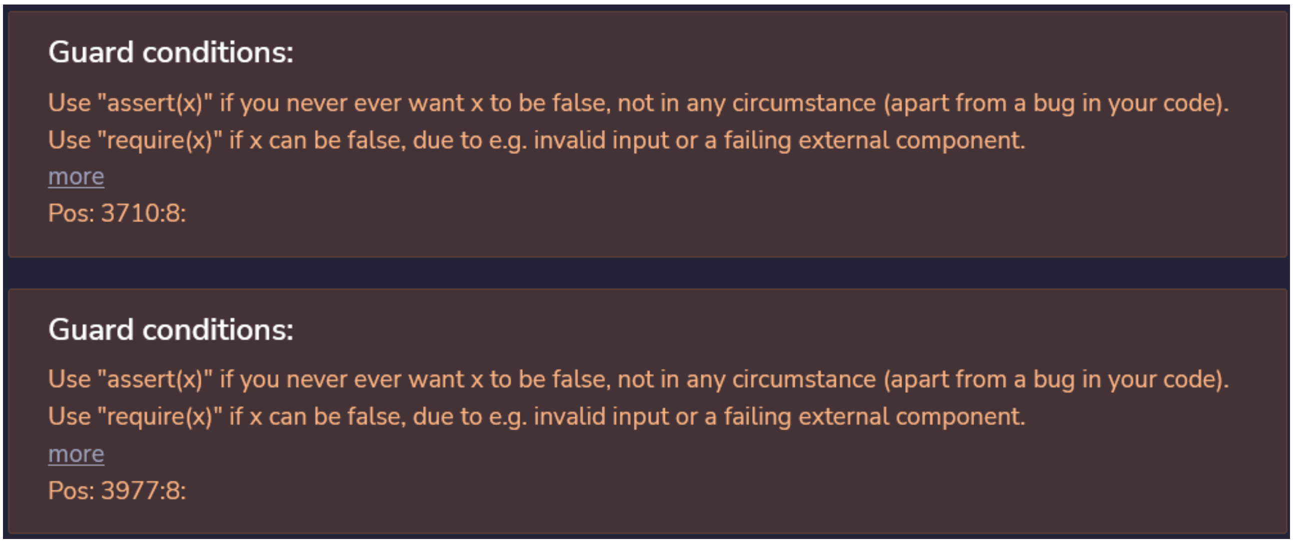


Similar variable names:

ReignToken.clearStuckBalance(address) : Variables have very similar names "_balances" and "balance". Note: Modifiers are currently not considered by this static analysis.
Pos: 4151:8:

Similar variable names:

ReignToken.clearStuckBalance(address) : Variables have very similar names "_balances" and "balance". Note: Modifiers are currently not considered by this static analysis.
Pos: 4152:36:

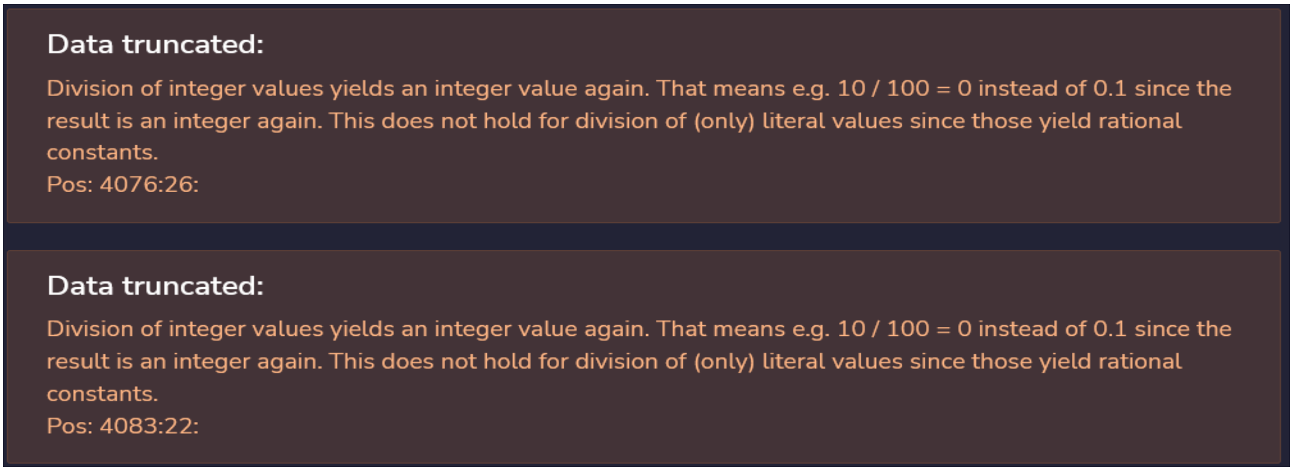


Guard conditions:

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: 3710:8:

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: 3977:8:

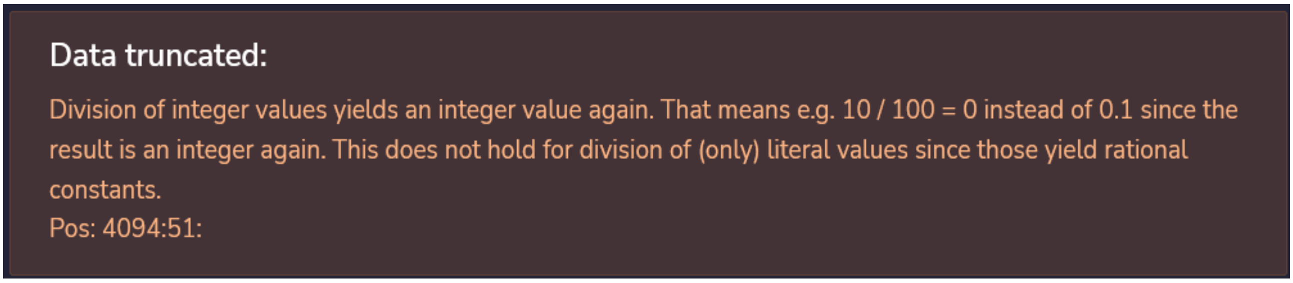


Data truncated:

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: 4076:26:

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: 4083:22:

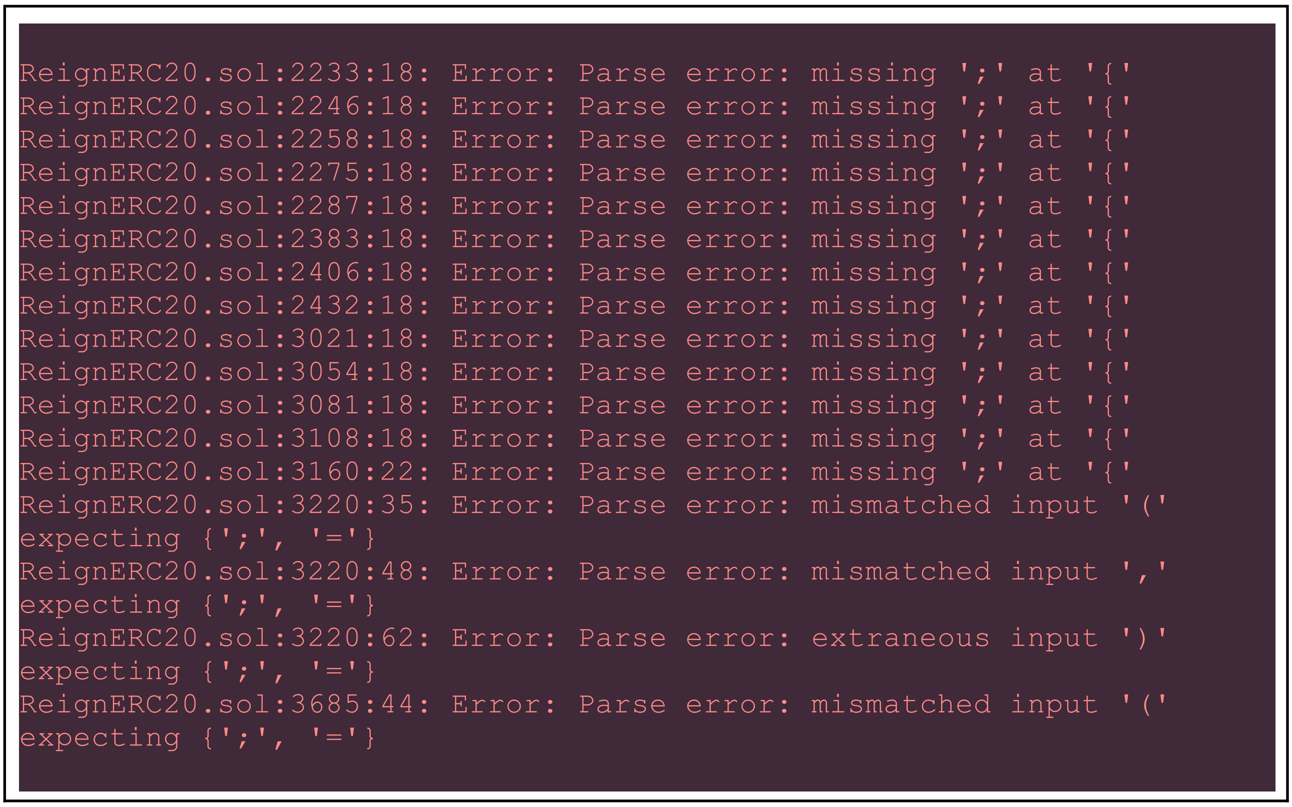


Data truncated:

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: 4094:51:

Solhint Linter

ReignERC20.sol



```
ReignERC20.sol:2233:18: Error: Parse error: missing ';' at '{'
ReignERC20.sol:2246:18: Error: Parse error: missing ';' at '{'
ReignERC20.sol:2258:18: Error: Parse error: missing ';' at '{'
ReignERC20.sol:2275:18: Error: Parse error: missing ';' at '{'
ReignERC20.sol:2287:18: Error: Parse error: missing ';' at '{'
ReignERC20.sol:2383:18: Error: Parse error: missing ';' at '{'
ReignERC20.sol:2406:18: Error: Parse error: missing ';' at '{'
ReignERC20.sol:2432:18: Error: Parse error: missing ';' at '{'
ReignERC20.sol:3021:18: Error: Parse error: missing ';' at '{'
ReignERC20.sol:3054:18: Error: Parse error: missing ';' at '{'
ReignERC20.sol:3081:18: Error: Parse error: missing ';' at '{'
ReignERC20.sol:3108:18: Error: Parse error: missing ';' at '{'
ReignERC20.sol:3160:22: Error: Parse error: missing ';' at '{'
ReignERC20.sol:3220:35: Error: Parse error: mismatched input '('
expecting {';', '='}
ReignERC20.sol:3220:48: Error: Parse error: mismatched input ','
expecting {';', '='}
ReignERC20.sol:3220:62: Error: Parse error: extraneous input ')'
expecting {';', '='}
ReignERC20.sol:3685:44: Error: Parse error: mismatched input '('
expecting {';', '='}
```

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