EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	ShibaNova Token
Platform:	Binance Smart Chain
Language:	Solidity
Date:	September 24th, 2021

Table of contents

Introduction

EtherAuthority was contracted by the ShibaNova Token team to perform the Security audit of ShibaNova 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 September 24th, 2021.

The purpose of this audit was to address the following:

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

Project Background

Presale contract is set up by the ShibaNova team, thereby being the 'owner' of the contract. ShibaNova collects 1% of the BNB raised and project tokens deposited as a fee. The 'payee' is the wallet belonging to the project owner requesting the presale. They are responsible for depositing the project token. The presale is set up with a start block and end block to determine the duration of the presale. The first stage of the presale is for whitelisted Nova/sNova holders to be able to buy first, usually 30 minutes. Once the presale has finished (either through selling out or time ending), ShibaNova will send the 'addLiquidity' function,which creates and locks the LP, sends the fee to ShibaNova, allows purchasers to claim their tokens and allows project wallet to collect bnb.

Audit scope

Name	**Code Review and Security Analysis Report for ShibaNova-Presale Token Smart Contract**
Platform	**BSC / Solidity**
File	presalev2.sol
File MD5 Hash	6489B7CBD37DCA6F9CFF2C2DA3288384
Updated MD5 Hash	19DA1BAB231FB06E8C82FFBFC12BF69D
Online Code	https://github.com/ShibaNova/Contracts/blob/main/presalev2.sol

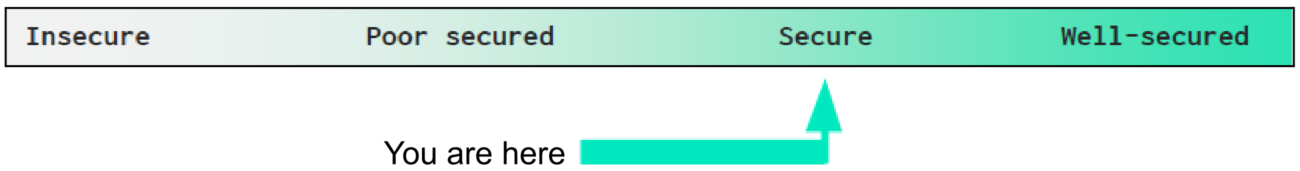
Audit Date	September 24th, 2021
Revised Audit Date	September 30th, 2021

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Fee: 1% ● Liquidity Percent: 10% ● Swap Rate: 10 ● Maximum Buy: 2 bnb ● Minimum Buy: 0.1 bnb ● Minimum SNova: 100 sNova ● Minimum Nova: 500 Nova	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. This token 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 critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 2 low and some very low level issues.

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	Moderate
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderate
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

Code Quality

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

The libraries in ShibaNova 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 ShibaNova Token.

The ShibaNova 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 smart contracts.

Documentation

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

As mentioned above, code parts are **not well** commented. 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.

Use of Dependencies

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

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

AS-IS overview

(1) Interface

(a) IERC20

(b) Router

(c) Factory

(2) Inherited contracts

(a) Context

(b) Ownable

(c) Whitelisted

(d) ReentrancyGuard

(3) Usages

(a) using Address for address payable;

(4) Events

(a) event Swap(address indexed user, uint256 inAmount, uint256 owedAmount);

(b) event Claim(address indexed user, uint256 amount);

(c) event PayeeTransferred(address indexed previousPayee, address indexed newPayee);

(d) event NewSwapRate (uint256 indexed previousRate, uint256 newSwapRate);

(e) event NewMaxBuy (uint256 indexed previousMaxBuy, uint256 newMaxBuy);

(f) event EmergencyWithdrawn(address indexed user, uint256 amount);

(g) event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

(5) Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	read	Function input parameters lack of check	Refer audit findings section below
2	swap	write	Passed	No Issue
3	claim	external	Check amount before transfer	Refer audit findings section below

4	addLiquidity	external	access only Owner	No Issue
5	widthdrawLiquidity	external	Payee can drain tokens and fund	Refer audit findings section below
6	updateEnd	write	access only Owner	No Issue
7	setClaim	internal	access only Owner	No Issue
8	setSwapRate	write	Passed	No Issue
9	setMaxBuy	write	Passed	No Issue
10	transferPayee	write	access only Owner	No Issue
11	withdrawBNB	external	Payee can drain tokens and fund	Refer audit findings section below
12	failPresale	write	access only Owner	No Issue
13	emergencyWithdraw	external	access by nonReentrant	No Issue
14	nonReentrant	modifier	Passed	No Issue
15	onlyWhitelisted	modifier	Passed	No Issue
16	whitelistAddresses	write	access only Owner	No Issue
17	deleteFromWhitelist	write	access only Owner	No Issue
18	isWhitelisted	read	Passed	No Issue
19	owner	read	Passed	No Issue
20	onlyOwner	modifier	Passed	No Issue
21	renounceOwnership	write	access only Owner	No Issue
22	transferOwnership	write	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

No Critical severity vulnerabilities were found.

High

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Function input parameters lack of check:

```
function setSwapRate(uint256 newRate) public onlyOwner {
    require(block.timestamp < startBlock); //cannot modify after presale starts
    emit NewSwapRate(swapRate, newRate);
    swapRate = newRate;
}

function setMaxBuy(uint256 newMax) public onlyOwner {
    emit NewMaxBuy(maxBuy, newMax);
    maxBuy = newMax;
}
```

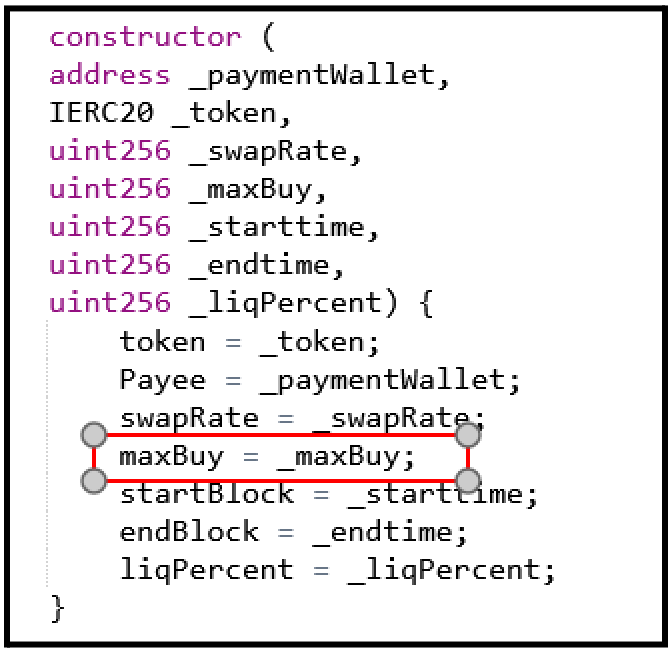
While setting values need to check for values that are not 0 or negative.

Resolution:

- setSwapRate() - newRate is not 0 or negative.
- setMaxBuy() - newMax is not 0 or negative and not less than minBuy.

Status: Fixed.

(2) Function input parameters lack of check:



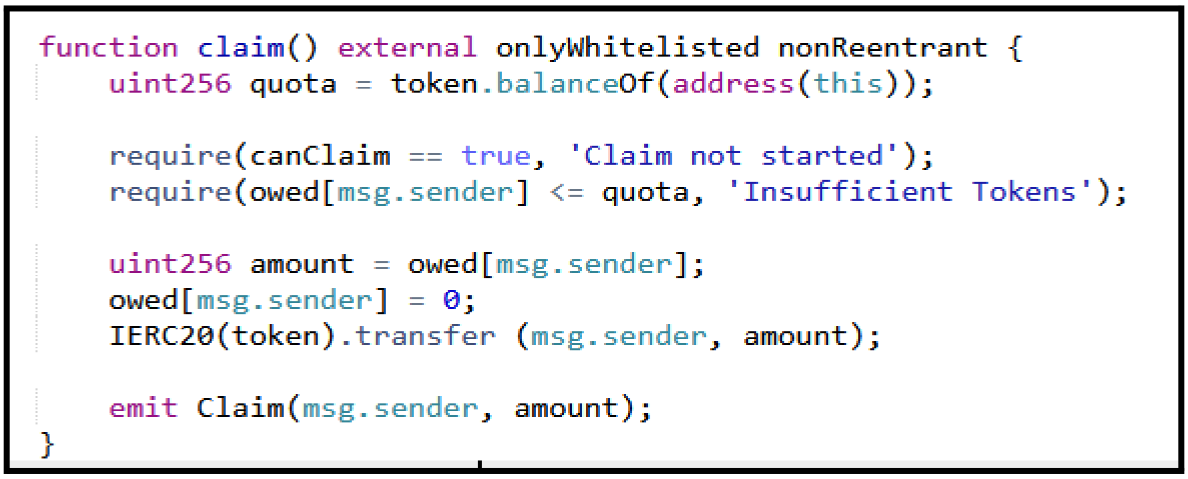
```
constructor (
address _paymentWallet,
IERC20 _token,
uint256 _swapRate,
uint256 _maxBuy,
uint256 _starttime,
uint256 _endtime,
uint256 _liqPercent) {
    token = _token;
    Payee = _paymentWallet;
    swapRate = _swapRate;
    maxBuy = _maxBuy;
    startBlock = _starttime;
    endBlock = _endtime;
    liqPercent = _liqPercent;
}
```

In constructor, maxBuy value should be checked not to less than minBuy.

Resolution: setSwapRate() newRate is not 0 or negative.

Status: Acknowledged.

(3) Check amount before transfer:

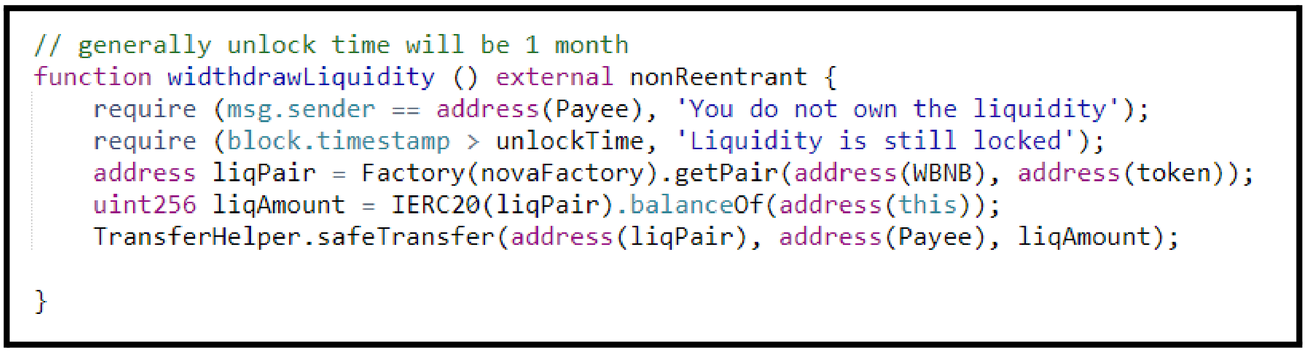


```
function claim() external onlyWhitelisted nonReentrant {
    uint256 quota = token.balanceOf(address(this));

    require(canClaim == true, 'Claim not started');
    require(owed[msg.sender] <= quota, 'Insufficient Tokens');

    uint256 amount = owed[msg.sender];
    owed[msg.sender] = 0;
    IERC20(token).transfer (msg.sender, amount);

    emit Claim(msg.sender, amount);
}
```



```
// generally unlock time will be 1 month
function widthdrawLiquidity () external nonReentrant {
    require (msg.sender == address(Payee), 'You do not own the liquidity');
    require (block.timestamp > unlockTime, 'Liquidity is still locked');
    address liqPair = Factory(novaFactory).getPair(address(WBNB), address(token));
    uint256 liqAmount = IERC20(liqPair).balanceOf(address(this));
    TransferHelper.safeTransfer(address(liqPair), address(Payee), liqAmount);

}
```

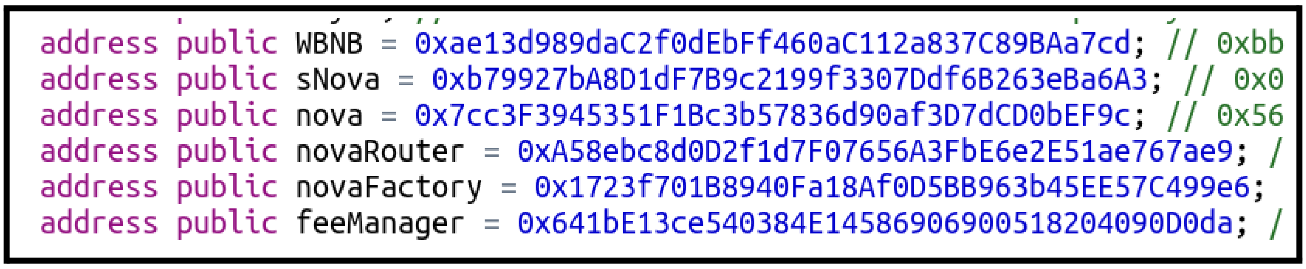
In the claim function, if the user has already claimed , then 0 ERC20 token has been transferred. In the withdrawLiquidity function ,liqAmount needs to check before using for transfer.

Resolution: In such cases, transactions should be reverted to save the gas fee of the user.

Status: Acknowledged.

Very Low / Informational / Best practices:

(1) Make variables constant:



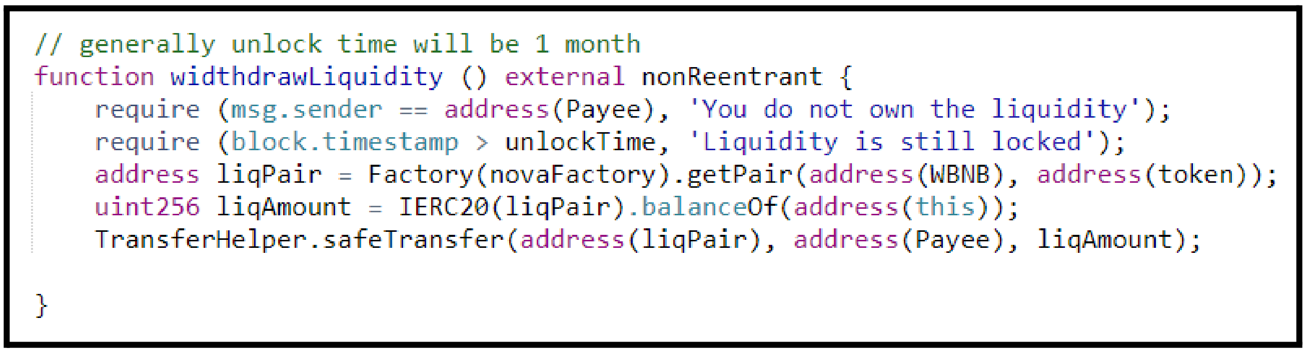
```
address public WBNB = 0xae13d989daC2f0dEbFf460aC112a837C89BAa7cd; // 0xbb
address public sNova = 0xb79927bA8D1dF7B9c2199f3307Ddf6B263eBa6A3; // 0x0
address public nova = 0x7cc3F3945351F1Bc3b57836d90af3D7dCD0bEF9c; // 0x56
address public novaRouter = 0xA58ebc8d0D2f1d7F07656A3FbE6e2E51ae767ae9; /
address public novaFactory = 0x1723f701B8940Fa18Af0D5BB963b45EE57C499e6;
address public feeManager = 0x641bE13ce540384E14586906900518204090D0da; /
```

WBNB, sNova, nova, novaRouter, novaFactory, feeManager and novaStage, fee, lockPeriod, minSNova, minNova, minBuy. These variables will be unchanged. So, please make them constant. It will save some gas.

Resolution: Declare those variables as constant. Just put a constant keyword.

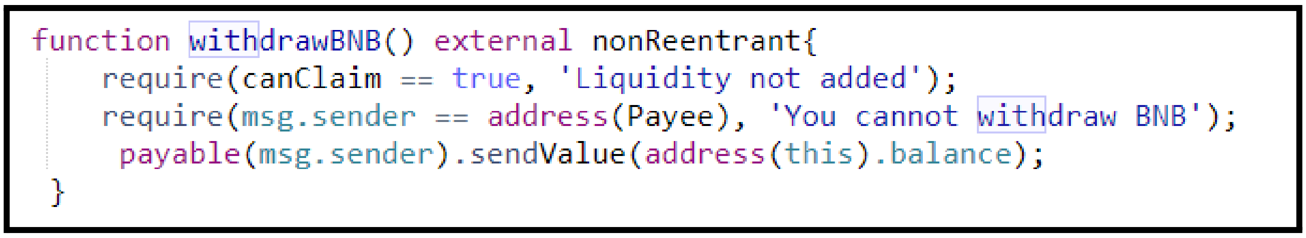
Status: Fixed.

(2) Payee can drain tokens and fund:



```
// generally unlock time will be 1 month
function widthdrawLiquidity () external nonReentrant {
    require (msg.sender == address(Payee), 'You do not own the liquidity');
    require (block.timestamp > unlockTime, 'Liquidity is still locked');
    address liqPair = Factory(novaFactory).getPair(address(WBNB), address(token));
    uint256 liqAmount = IERC20(liqPair).balanceOf(address(this));
    TransferHelper.safeTransfer(address(liqPair), address(Payee), liqAmount);

}
```



```
function withdrawBNB() external nonReentrant{
    require(canClaim == true, 'Liquidity not added');
    require(msg.sender == address(Payee), 'You cannot withdraw BNB');
     payable(msg.sender).sendValue(address(this).balance);
 }
```

While deploying, the owner can set his own address/ other as Payee and can drain all the funds and tokens. If the private key of the Payee's wallet is compromised, then it will create a problem.

Resolution: The Payee can accept this risk and handle the private key very securely.

Status: **Acknowledged.**

Centralization

These smart contracts have 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:

- swap: The Whitelisted owner can swap the Nova stage.
- claim: The Whitelisted owner can claim a token.
- addLiquidity: The Owner can add liquidity.
- updateEnd: The Owner can only end presale early, not extend.
- setSwapRate: The Owner can set the swap rate.
- setMaxBuy: The Owner can set a max buy rate.
- transferPayee: The Owner can transfer a new payee.
- failPresale: The Owner can failsafe for everyone to withdraw their BNB if an issue arises, project tokens cannot be withdrawn.
- emergencyWithdraw: The nonReentrant owner can withdraw emergency amounts.
- withdrawBNB: The nonReentrant owner can withdraw the BNB token.
- widthdrawLiquidity: The nonReentrant owner can withdraw liquidity amount.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We observed some issues in the smart contracts and those issues are not critical ones. So, **it's good to go to production**.

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

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

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

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

Our Methodology

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

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

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

Documenting Results:

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

Suggested Solutions:

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

Disclaimers

EtherAuthority.io Disclaimer

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

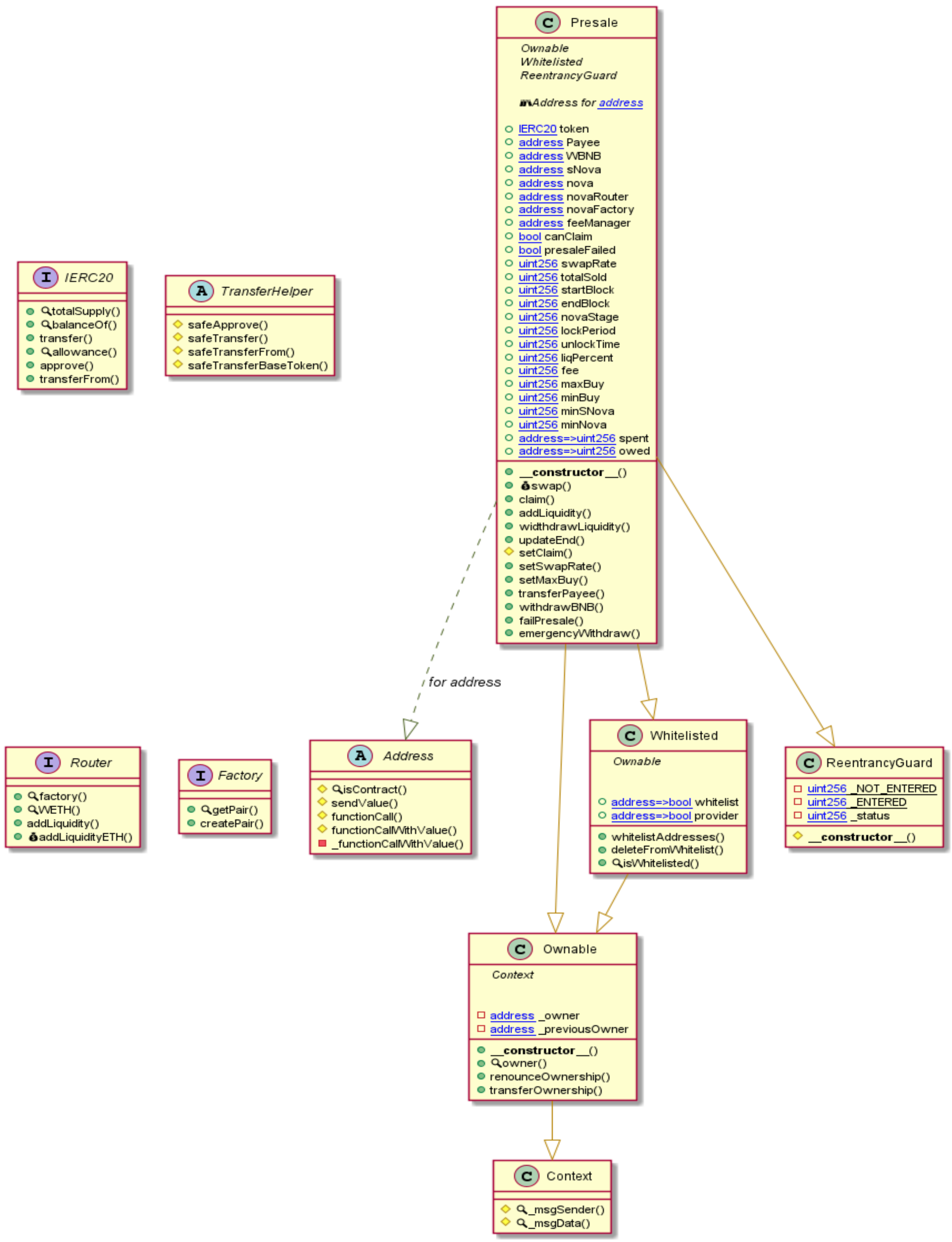
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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

Technical Disclaimer

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

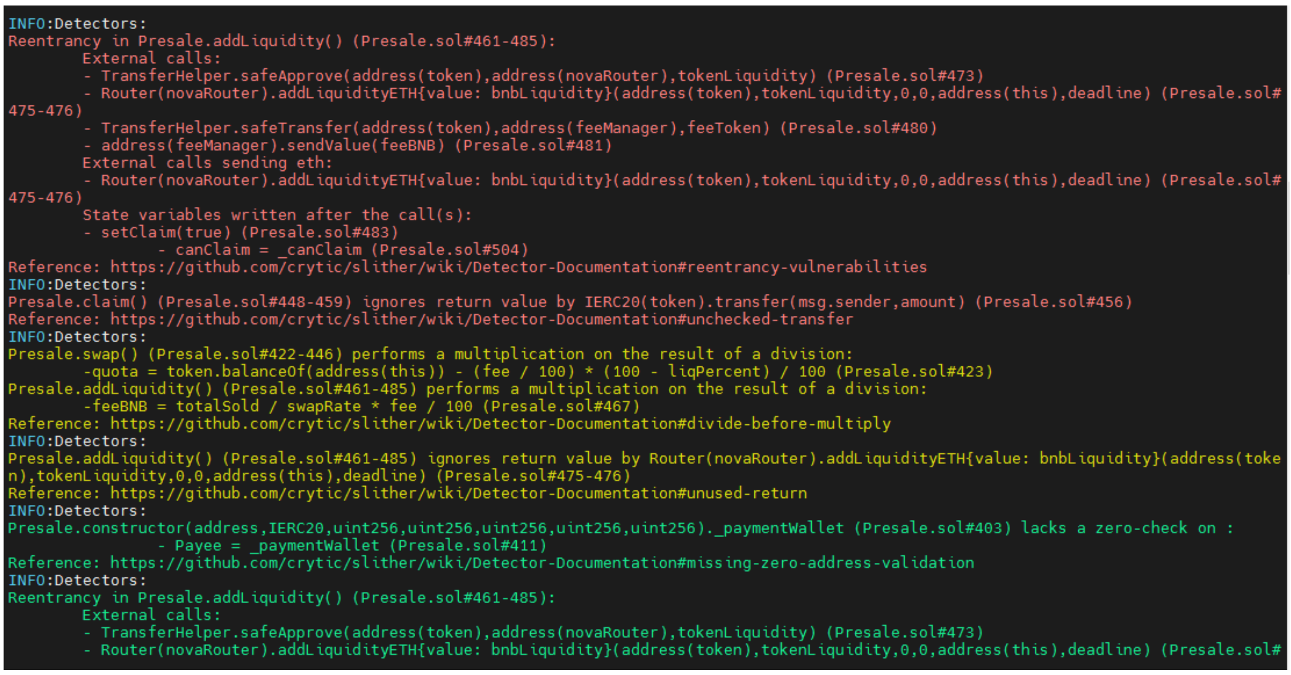
Appendix

Code Flow Diagram - ShibaNova Token

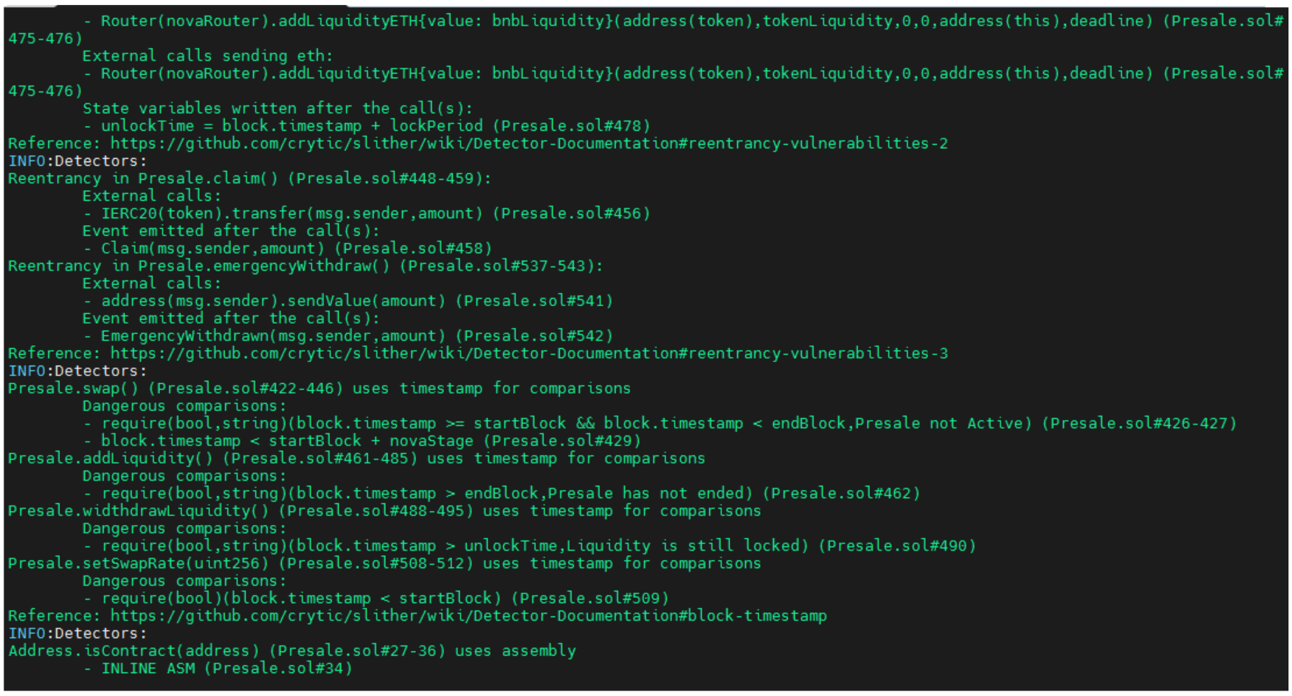


Slither Results Log

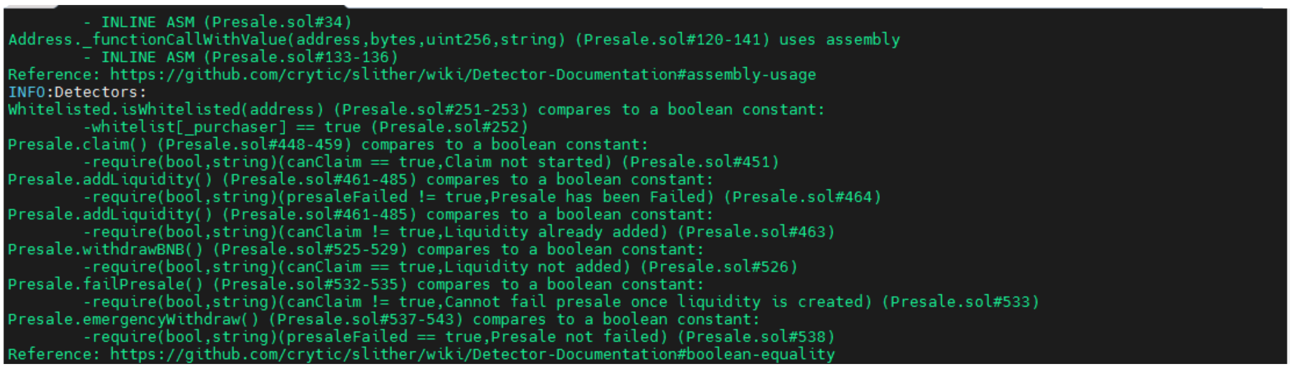
Slither log >> presalev2.sol



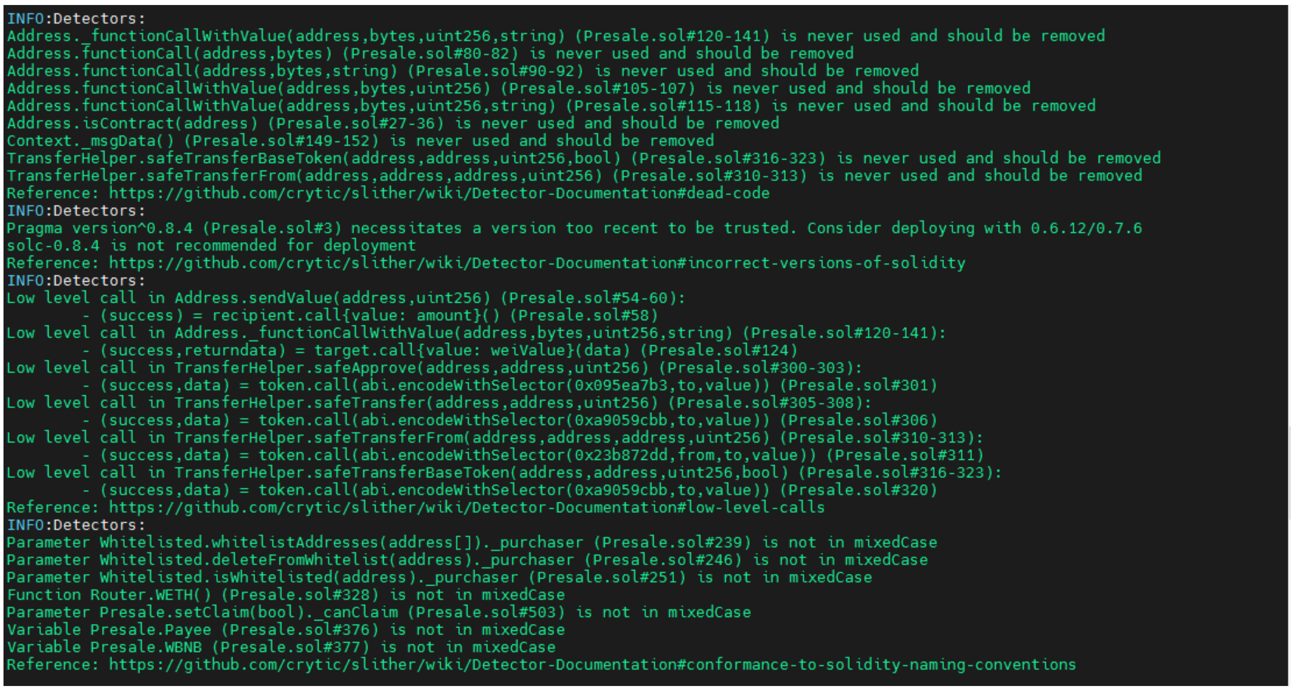
```
INFO:Detectors:
Reentrancy in Presale.addLiquidity() (Presale.sol#461-485):
        External calls:
        - TransferHelper.safeApprove(address(token),address(novaRouter),tokenLiquidity) (Presale.sol#473)
        - Router(novaRouter).addLiquidityETH{value: bnbLiquidity}(address(token),tokenLiquidity,0,0,address(this),deadline) (Presale.sol#
475-476)
        - TransferHelper.safeTransfer(address(token),address(feeManager),feeToken) (Presale.sol#480)
        - address(feeManager).sendValue(feeBNB) (Presale.sol#481)
        External calls sending eth:
        - Router(novaRouter).addLiquidityETH{value: bnbLiquidity}(address(token),tokenLiquidity,0,0,address(this),deadline) (Presale.sol#
475-476)
        State variables written after the call(s):
        - setClaim(true) (Presale.sol#483)
                - canClaim = _canClaim (Presale.sol#504)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
Presale.claim() (Presale.sol#448-459) ignores return value by IERC20(token).transfer(msg.sender,amount) (Presale.sol#456)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
Presale.swap() (Presale.sol#422-446) performs a multiplication on the result of a division:
        -quota = token.balanceOf(address(this)) - (fee / 100) * (100 - liqPercent) / 100 (Presale.sol#423)
Presale.addLiquidity() (Presale.sol#461-485) performs a multiplication on the result of a division:
        -feeBNB = totalSold / swapRate * fee / 100 (Presale.sol#467)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Presale.addLiquidity() (Presale.sol#461-485) ignores return value by Router(novaRouter).addLiquidityETH{value: bnbLiquidity}(address(toke
n),tokenLiquidity,0,0,address(this),deadline) (Presale.sol#475-476)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Presale.constructor(address,IERC20,uint256,uint256,uint256,uint256,uint256)._paymentWallet (Presale.sol#403) lacks a zero-check on :
                - Payee = _paymentWallet (Presale.sol#411)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in Presale.addLiquidity() (Presale.sol#461-485):
        External calls:
        - TransferHelper.safeApprove(address(token),address(novaRouter),tokenLiquidity) (Presale.sol#473)
        - Router(novaRouter).addLiquidityETH{value: bnbLiquidity}(address(token),tokenLiquidity,0,0,address(this),deadline) (Presale.sol#
```



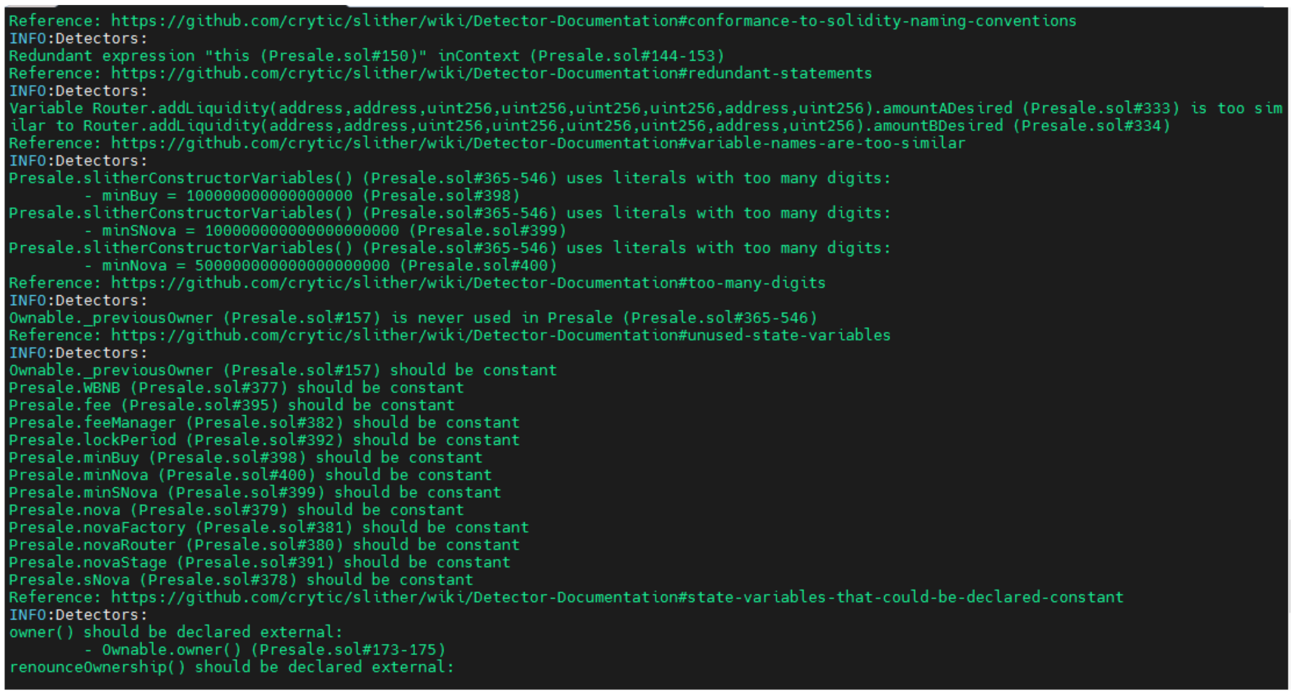
```
        - Router(novaRouter).addLiquidityETH{value: bnbLiquidity}(address(token),tokenLiquidity,0,0,address(this),deadline) (Presale.sol#
475-476)
        External calls sending eth:
        - Router(novaRouter).addLiquidityETH{value: bnbLiquidity}(address(token),tokenLiquidity,0,0,address(this),deadline) (Presale.sol#
475-476)
        State variables written after the call(s):
        - unlockTime = block.timestamp + lockPeriod (Presale.sol#478)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Presale.claim() (Presale.sol#448-459):
        External calls:
        - IERC20(token).transfer(msg.sender,amount) (Presale.sol#456)
        Event emitted after the call(s):
        - Claim(msg.sender,amount) (Presale.sol#458)
Reentrancy in Presale.emergencyWithdraw() (Presale.sol#537-543):
        External calls:
        - address(msg.sender).sendValue(amount) (Presale.sol#541)
        Event emitted after the call(s):
        - EmergencyWithdrawn(msg.sender,amount) (Presale.sol#542)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Presale.swap() (Presale.sol#422-446) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= startBlock && block.timestamp < endBlock,Presale not Active) (Presale.sol#426-427)
        - block.timestamp < startBlock + novaStage (Presale.sol#429)
Presale.addLiquidity() (Presale.sol#461-485) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > endBlock,Presale has not ended) (Presale.sol#462)
Presale.widthdrawLiquidity() (Presale.sol#488-495) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > unlockTime,Liquidity is still locked) (Presale.sol#490)
Presale.setSwapRate(uint256) (Presale.sol#508-512) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(block.timestamp < startBlock) (Presale.sol#509)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (Presale.sol#27-36) uses assembly
        - INLINE ASM (Presale.sol#34)
```



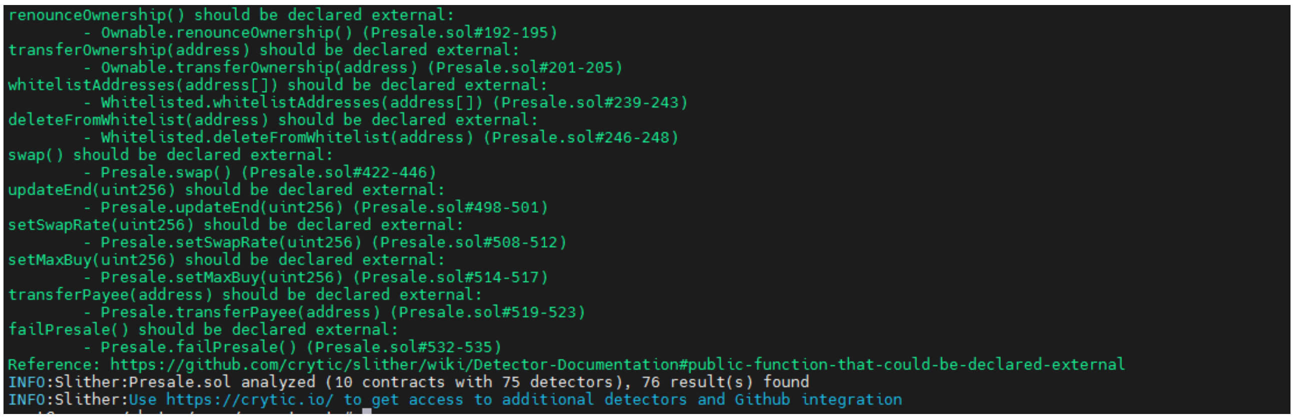
```
        - INLINE ASM (Presale.sol#34)
Address._functionCallWithValue(address,bytes,uint256,string) (Presale.sol#120-141) uses assembly
        - INLINE ASM (Presale.sol#133-136)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Whitelisted.isWhitelisted(address) (Presale.sol#251-253) compares to a boolean constant:
        -whitelist[_purchaser] == true (Presale.sol#252)
Presale.claim() (Presale.sol#448-459) compares to a boolean constant:
        -require(bool,string)(canClaim == true,Claim not started) (Presale.sol#451)
Presale.addLiquidity() (Presale.sol#461-485) compares to a boolean constant:
        -require(bool,string)(presaleFailed != true,Presale has been Failed) (Presale.sol#464)
Presale.addLiquidity() (Presale.sol#461-485) compares to a boolean constant:
        -require(bool,string)(canClaim != true,Liquidity already added) (Presale.sol#463)
Presale.withdrawBNB() (Presale.sol#525-529) compares to a boolean constant:
        -require(bool,string)(canClaim == true,Liquidity not added) (Presale.sol#526)
Presale.failPresale() (Presale.sol#532-535) compares to a boolean constant:
        -require(bool,string)(canClaim != true,Cannot fail presale once liquidity is created) (Presale.sol#533)
Presale.emergencyWithdraw() (Presale.sol#537-543) compares to a boolean constant:
        -require(bool,string)(presaleFailed == true,Presale not failed) (Presale.sol#538)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```



```
INFO:Detectors:
Address._functionCallWithValue(address,bytes,uint256,string) (Presale.sol#120-141) is never used and should be removed
Address.functionCall(address,bytes) (Presale.sol#80-82) is never used and should be removed
Address.functionCall(address,bytes,string) (Presale.sol#90-92) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Presale.sol#105-107) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (Presale.sol#115-118) is never used and should be removed
Address.isContract(address) (Presale.sol#27-36) is never used and should be removed
Context._msgData() (Presale.sol#149-152) is never used and should be removed
TransferHelper.safeTransferBaseToken(address,address,uint256,bool) (Presale.sol#316-323) is never used and should be removed
TransferHelper.safeTransferFrom(address,address,address,uint256) (Presale.sol#310-313) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.4 (Presale.sol#3) 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 Address.sendValue(address,uint256) (Presale.sol#54-60):
        - (success) = recipient.call{value: amount}() (Presale.sol#58)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (Presale.sol#120-141):
        - (success,returndata) = target.call{value: weiValue}(data) (Presale.sol#124)
Low level call in TransferHelper.safeApprove(address,address,uint256) (Presale.sol#300-303):
        - (success,data) = token.call(abi.encodeWithSelector(0x095ea7b3,to,value)) (Presale.sol#301)
Low level call in TransferHelper.safeTransfer(address,address,uint256) (Presale.sol#305-308):
        - (success,data) = token.call(abi.encodeWithSelector(0xa9059cbb,to,value)) (Presale.sol#306)
Low level call in TransferHelper.safeTransferFrom(address,address,address,uint256) (Presale.sol#310-313):
        - (success,data) = token.call(abi.encodeWithSelector(0x23b872dd,from,to,value)) (Presale.sol#311)
Low level call in TransferHelper.safeTransferBaseToken(address,address,uint256,bool) (Presale.sol#316-323):
        - (success,data) = token.call(abi.encodeWithSelector(0xa9059cbb,to,value)) (Presale.sol#320)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter Whitelisted.whitelistAddresses(address[])._purchaser (Presale.sol#239) is not in mixedCase
Parameter Whitelisted.deleteFromWhitelist(address)._purchaser (Presale.sol#246) is not in mixedCase
Parameter Whitelisted.isWhitelisted(address)._purchaser (Presale.sol#251) is not in mixedCase
Function Router.WETH() (Presale.sol#328) is not in mixedCase
Parameter Presale.setClaim(bool)._canClaim (Presale.sol#503) is not in mixedCase
Variable Presale.Payee (Presale.sol#376) is not in mixedCase
Variable Presale.WBNB (Presale.sol#377) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```



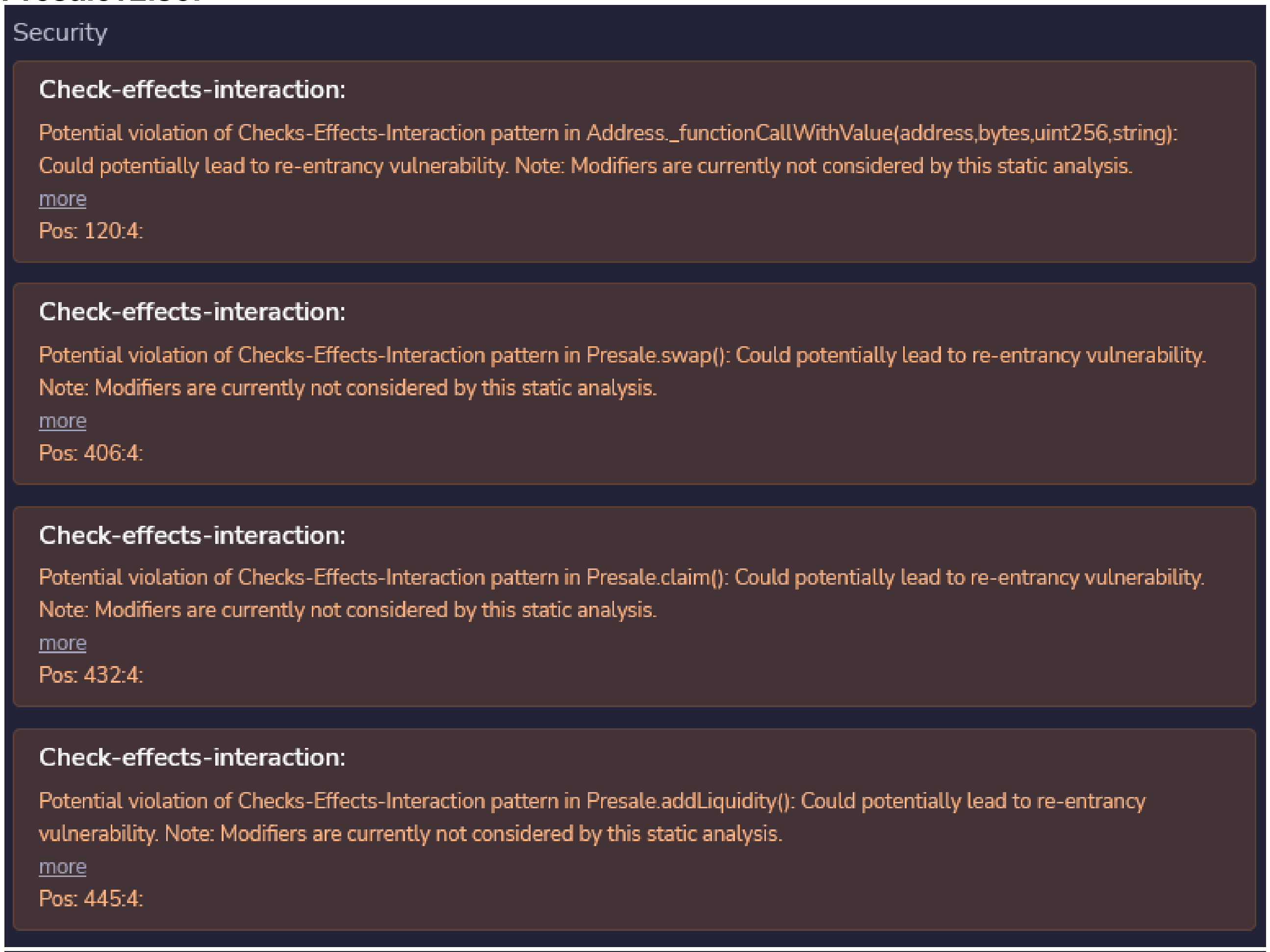
```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (Presale.sol#150)" inContext (Presale.sol#144-153)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Variable Router.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (Presale.sol#333) is too sim
ilar to Router.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (Presale.sol#334)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
Presale.slitherConstructorVariables() (Presale.sol#365-546) uses literals with too many digits:
        - minBuy = 100000000000000000 (Presale.sol#398)
Presale.slitherConstructorVariables() (Presale.sol#365-546) uses literals with too many digits:
        - minSNova = 100000000000000000000 (Presale.sol#399)
Presale.slitherConstructorVariables() (Presale.sol#365-546) uses literals with too many digits:
        - minNova = 500000000000000000000 (Presale.sol#400)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
Ownable._previousOwner (Presale.sol#157) is never used in Presale (Presale.sol#365-546)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
Ownable._previousOwner (Presale.sol#157) should be constant
Presale.WBNB (Presale.sol#377) should be constant
Presale.fee (Presale.sol#395) should be constant
Presale.feeManager (Presale.sol#382) should be constant
Presale.lockPeriod (Presale.sol#392) should be constant
Presale.minBuy (Presale.sol#398) should be constant
Presale.minNova (Presale.sol#400) should be constant
Presale.minSNova (Presale.sol#399) should be constant
Presale.nova (Presale.sol#379) should be constant
Presale.novaFactory (Presale.sol#381) should be constant
Presale.novaRouter (Presale.sol#380) should be constant
Presale.novaStage (Presale.sol#391) should be constant
Presale.sNova (Presale.sol#378) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
owner() should be declared external:
        - Ownable.owner() (Presale.sol#173-175)
renounceOwnership() should be declared external:
```



```
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Presale.sol#192-195)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Presale.sol#201-205)
whitelistAddresses(address[]) should be declared external:
        - Whitelisted.whitelistAddresses(address[]) (Presale.sol#239-243)
deleteFromWhitelist(address) should be declared external:
        - Whitelisted.deleteFromWhitelist(address) (Presale.sol#246-248)
swap() should be declared external:
        - Presale.swap() (Presale.sol#422-446)
updateEnd(uint256) should be declared external:
        - Presale.updateEnd(uint256) (Presale.sol#498-501)
setSwapRate(uint256) should be declared external:
        - Presale.setSwapRate(uint256) (Presale.sol#508-512)
setMaxBuy(uint256) should be declared external:
        - Presale.setMaxBuy(uint256) (Presale.sol#514-517)
transferPayee(address) should be declared external:
        - Presale.transferPayee(address) (Presale.sol#519-523)
failPresale() should be declared external:
        - Presale.failPresale() (Presale.sol#532-535)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Presale.sol analyzed (10 contracts with 75 detectors), 76 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

Presalev2.sol



Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Address._functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 120:4:

Check-effects-interaction:

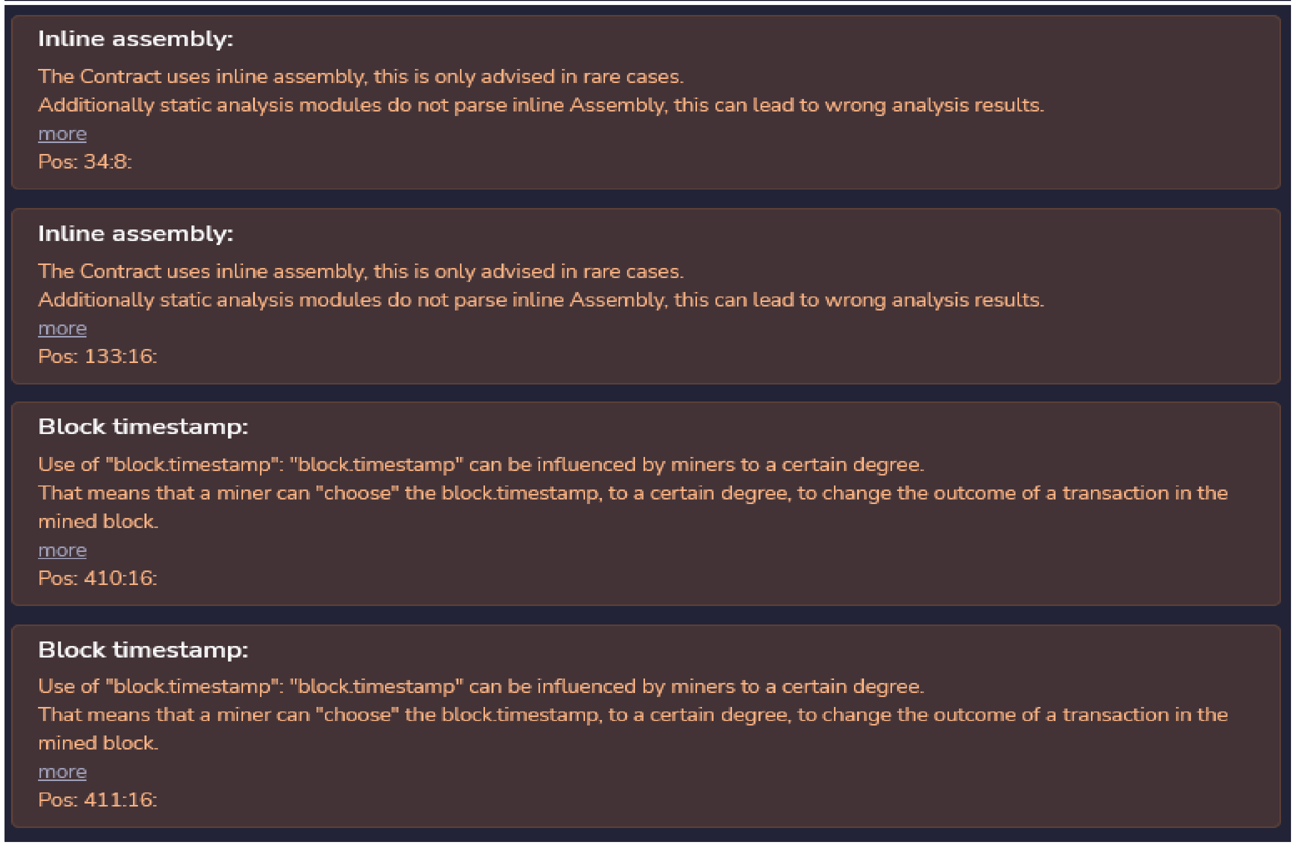
Potential violation of Checks-Effects-Interaction pattern in Presale.swap(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 406:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Presale.claim(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 432:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Presale.addLiquidity(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 445:4:



Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases.
Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.
more
Pos: 34:8:

Inline assembly:

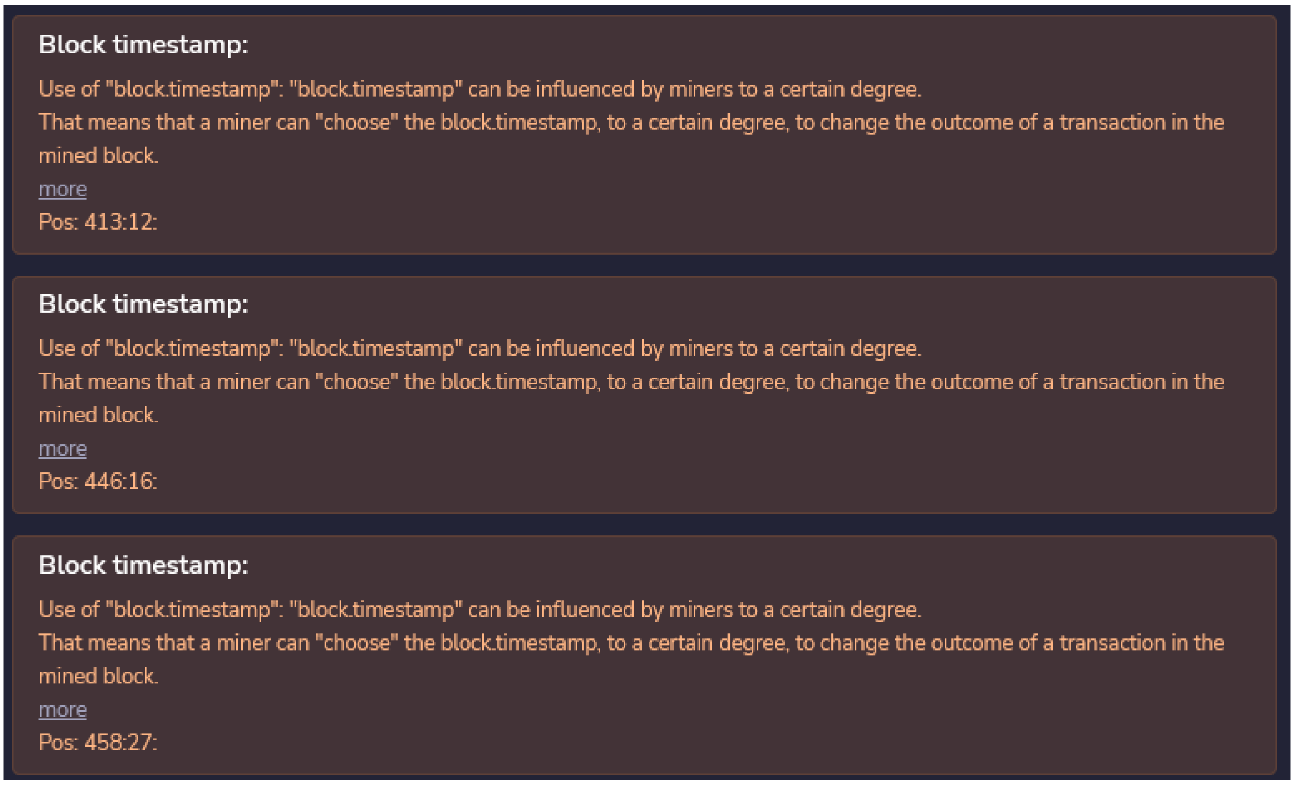
The Contract uses inline assembly, this is only advised in rare cases.
Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.
more
Pos: 133:16:

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: 410:16:

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: 411:16:



Block timestamp:

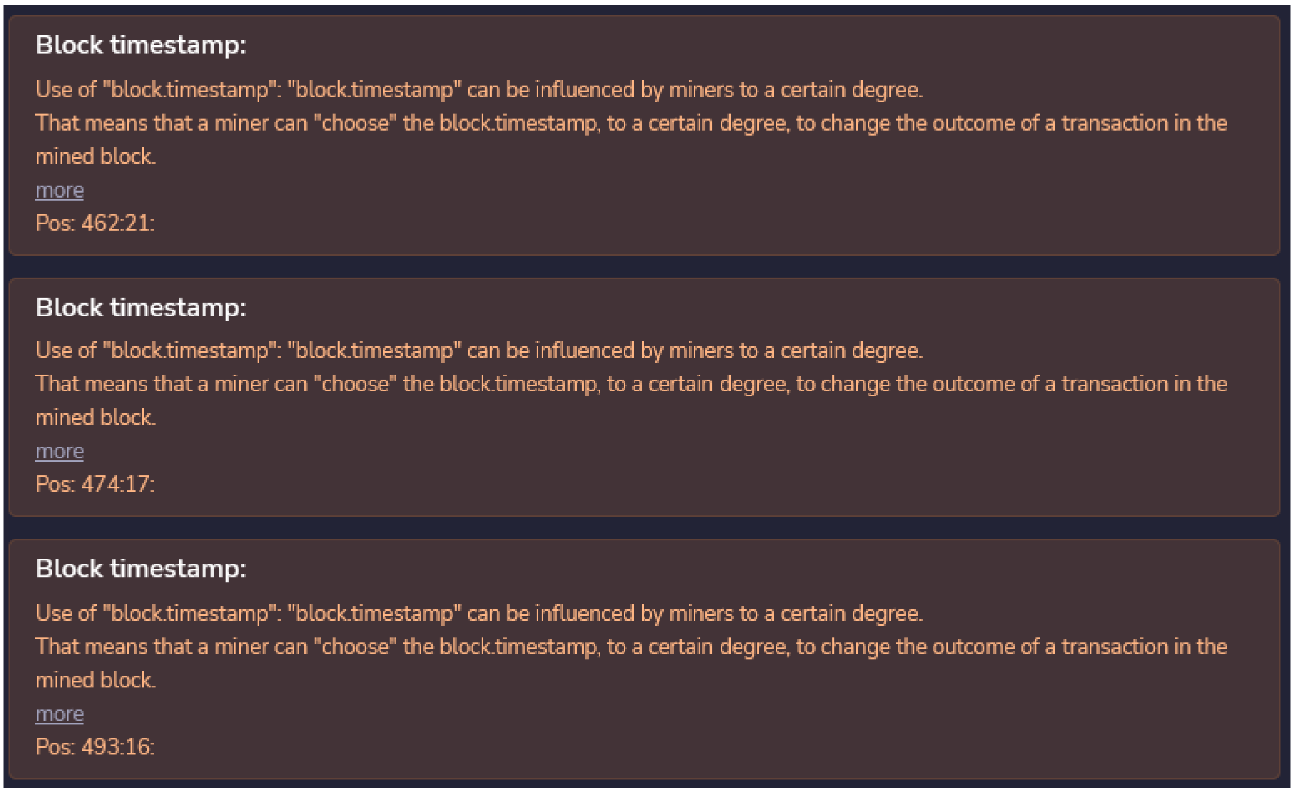
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 413:12:

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: 446:16:

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: 458:27:



Block timestamp:

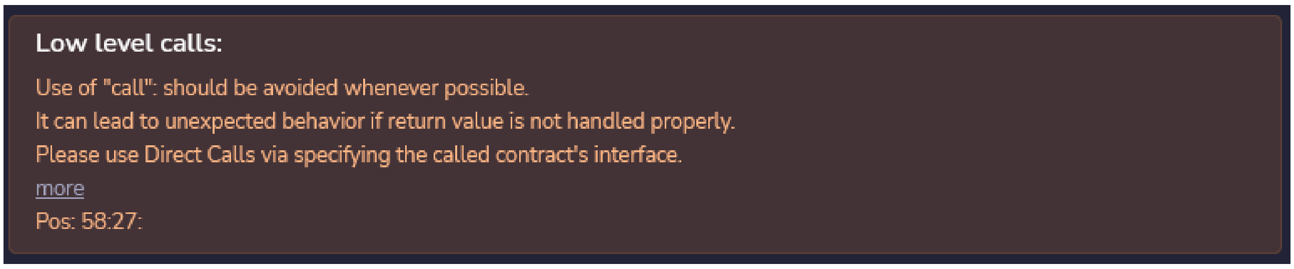
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 462:21:

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: 474:17:

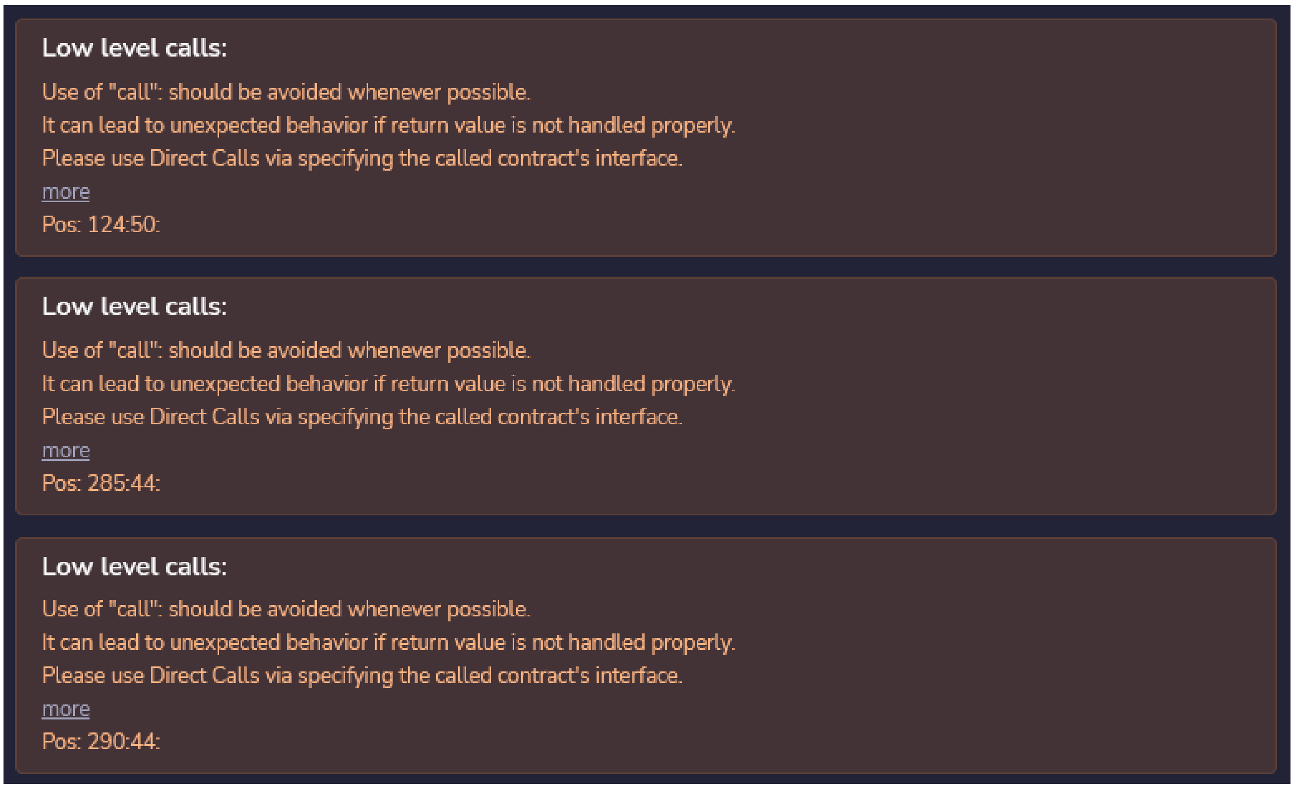
Block timestamp:

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: 493:16:



Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 58:27:



Low level calls:

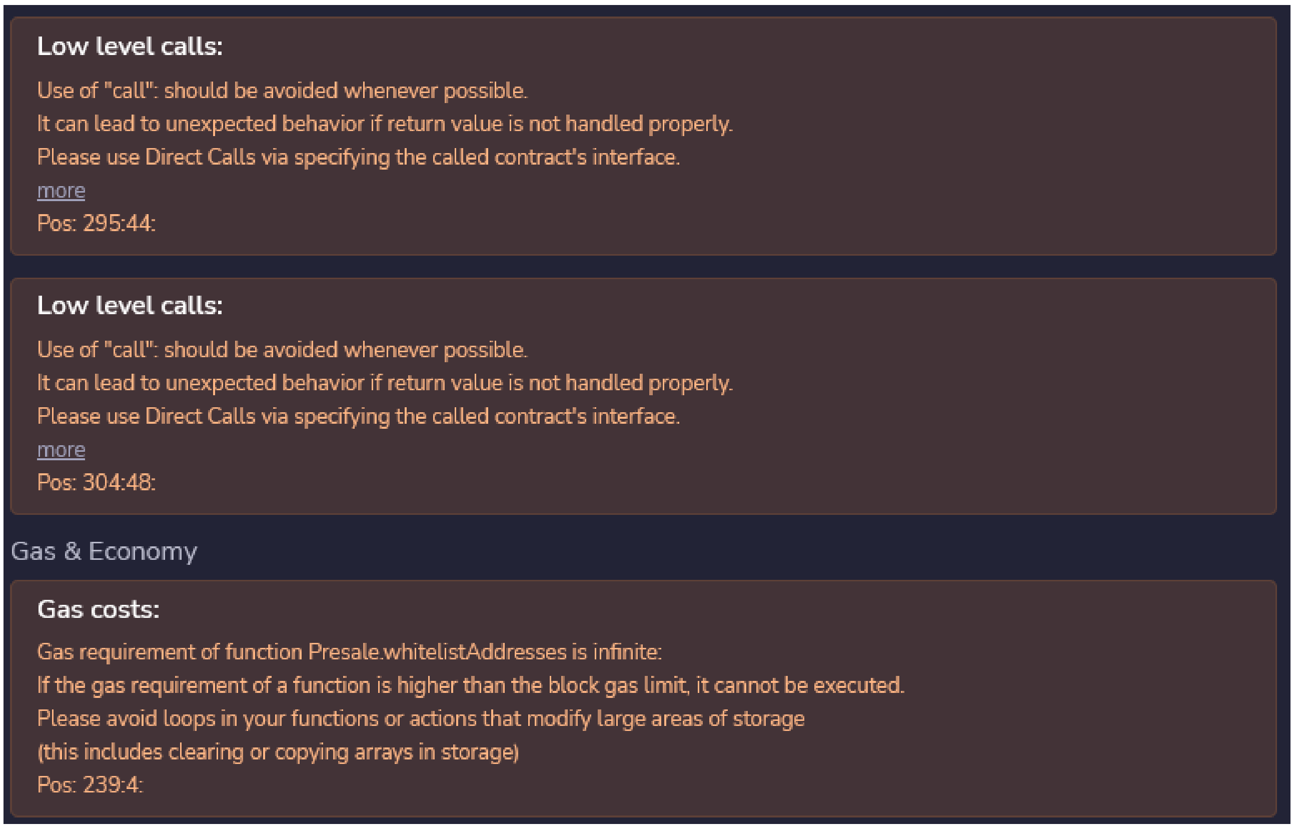
Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 124:50:

Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 285:44:

Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 290:44:



Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 295:44:

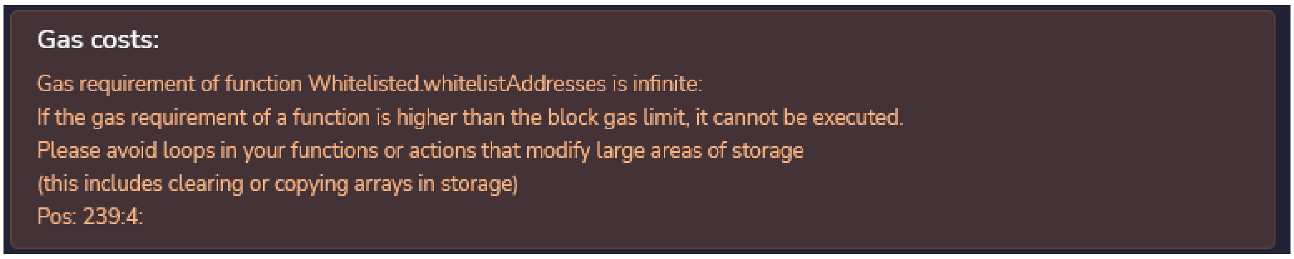
Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 304:48:

Gas & Economy

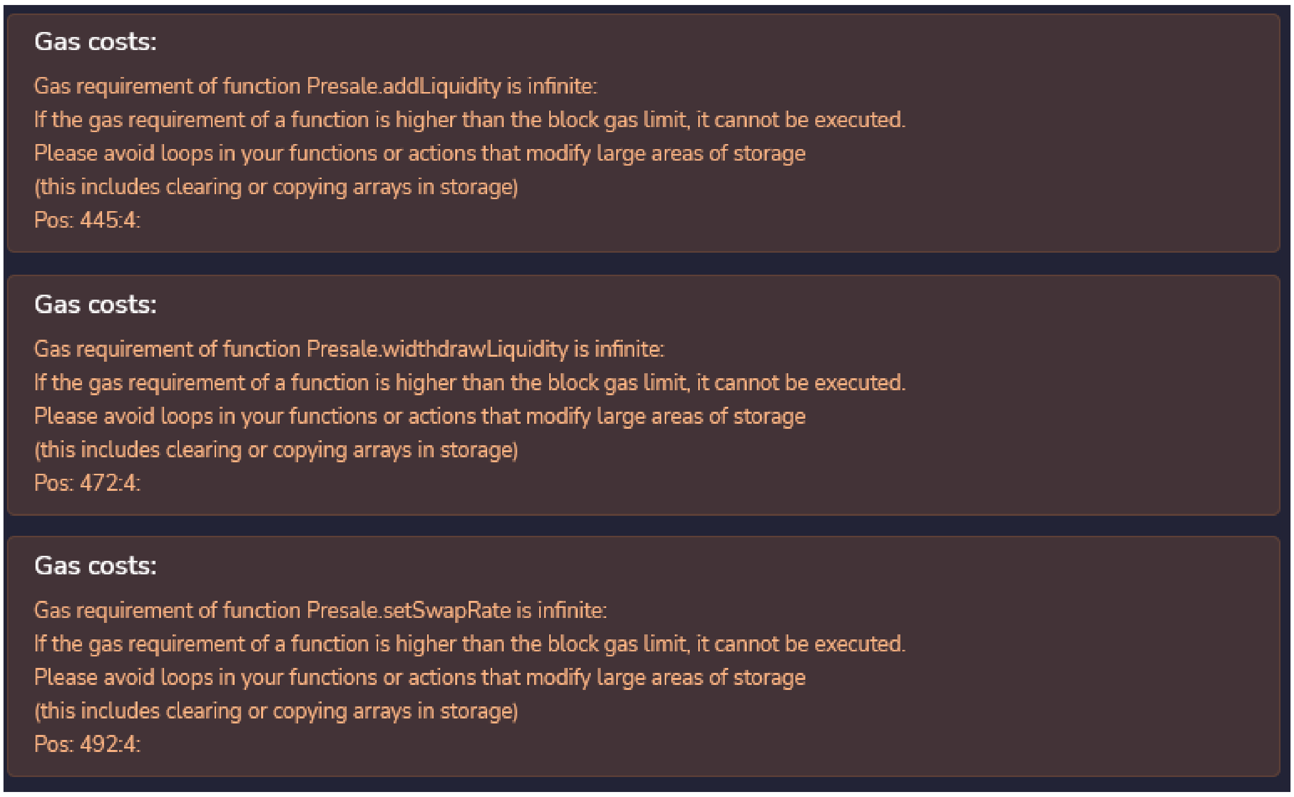
Gas costs:

Gas requirement of function Presale.whitelistAddresses 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: 239:4:



Gas costs:

Gas requirement of function Whitelisted.whitelistAddresses 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: 239:4:



Gas costs:

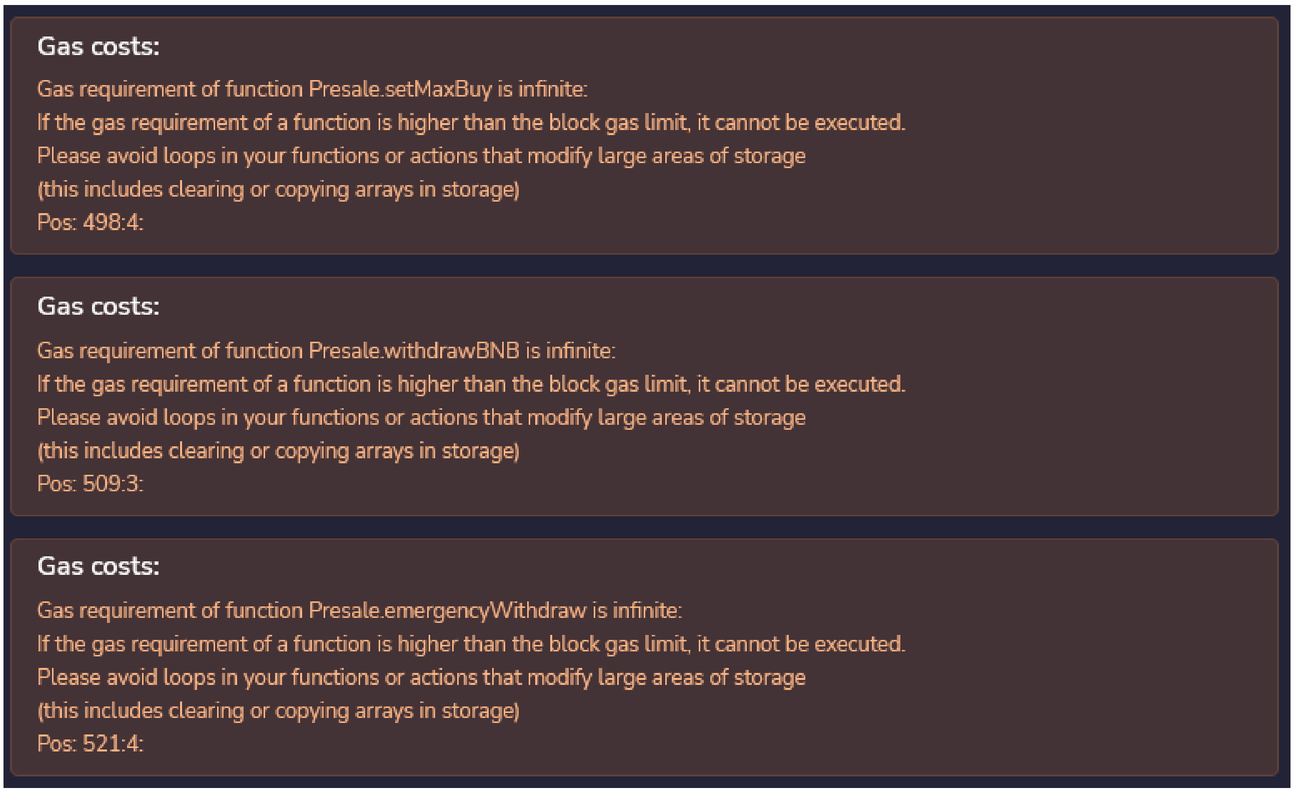
Gas requirement of function Presale.addLiquidity is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 445:4:

Gas costs:

Gas requirement of function Presale.widthdrawLiquidity 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: 472:4:

Gas costs:

Gas requirement of function Presale.setSwapRate 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: 492:4:



Gas costs:

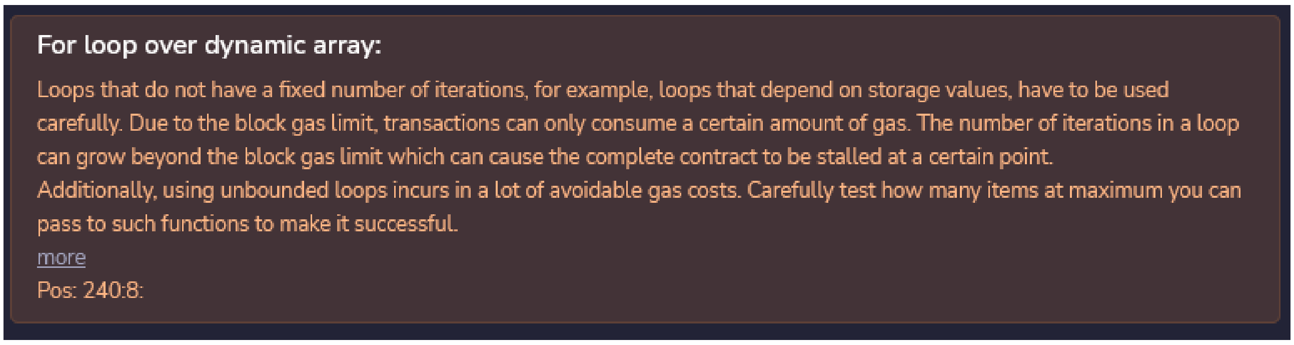
Gas requirement of function Presale.setMaxBuy is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 498:4:

Gas costs:

Gas requirement of function Presale.withdrawBNB 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: 509:3:

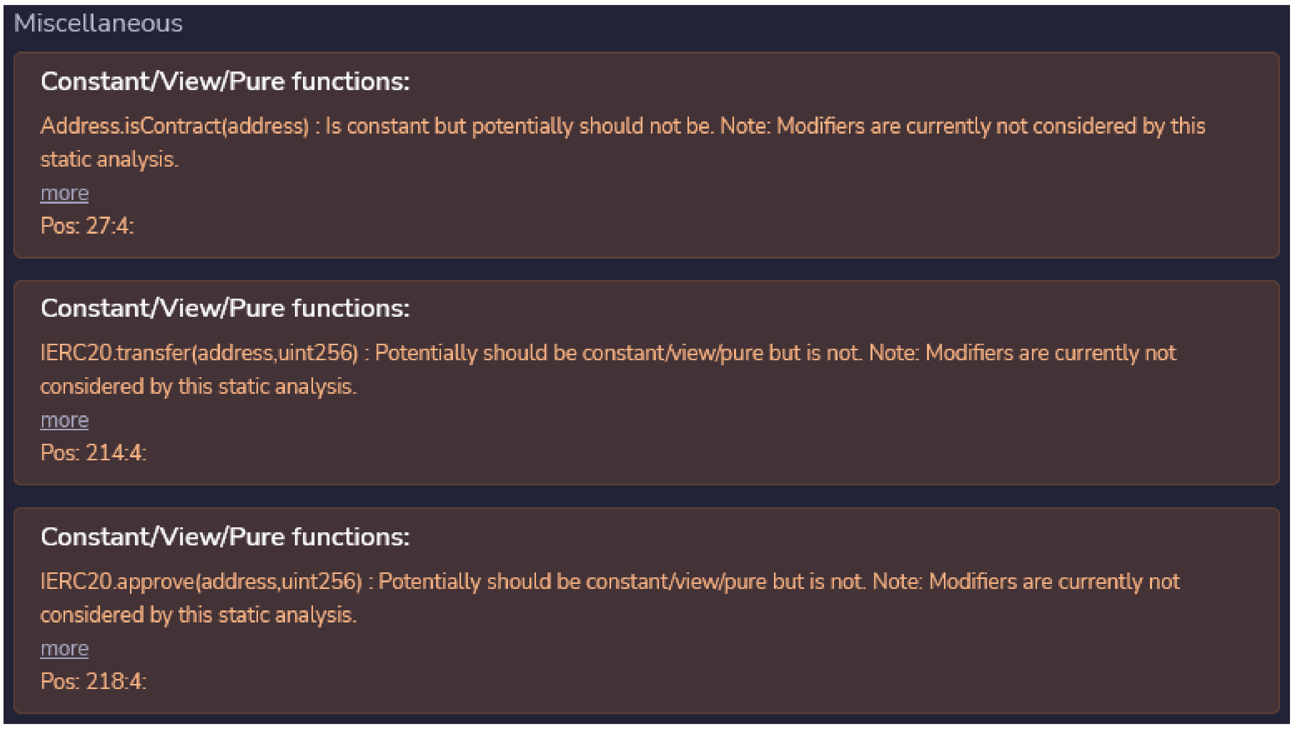
Gas costs:

Gas requirement of function Presale.emergencyWithdraw 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: 521:4:



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: 240:8:



Miscellaneous

Constant/View/Pure functions:

Address.isContract(address) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 27:4:

Constant/View/Pure functions:

IERC20.transfer(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

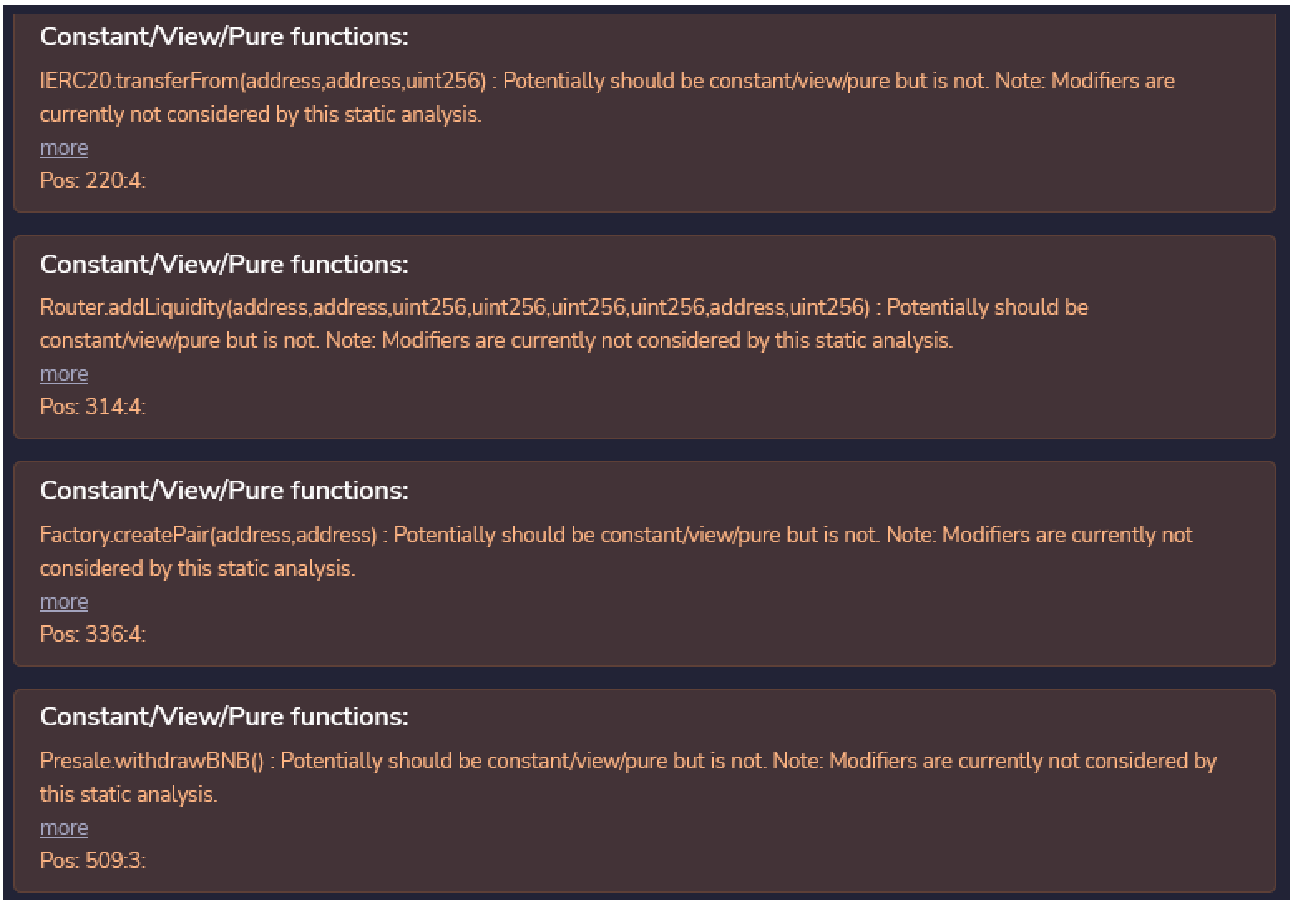
Pos: 214:4:

Constant/View/Pure functions:

IERC20.approve(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 218:4:



Constant/View/Pure functions:

IERC20.transferFrom(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 220:4:

Constant/View/Pure functions:

Router.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 314:4:

Constant/View/Pure functions:

Factory.createPair(address,address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

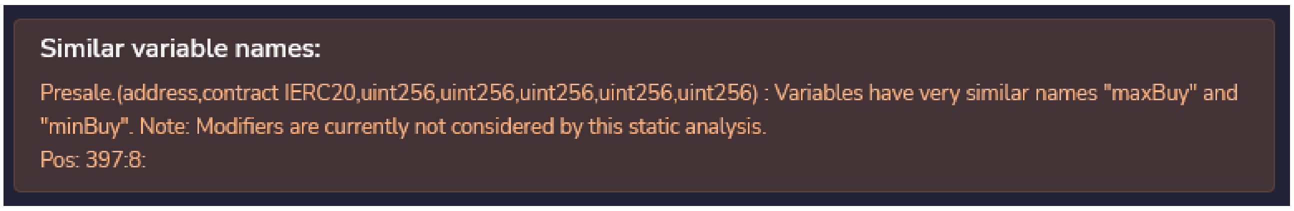
Pos: 336:4:

Constant/View/Pure functions:

Presale.withdrawBNB() : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

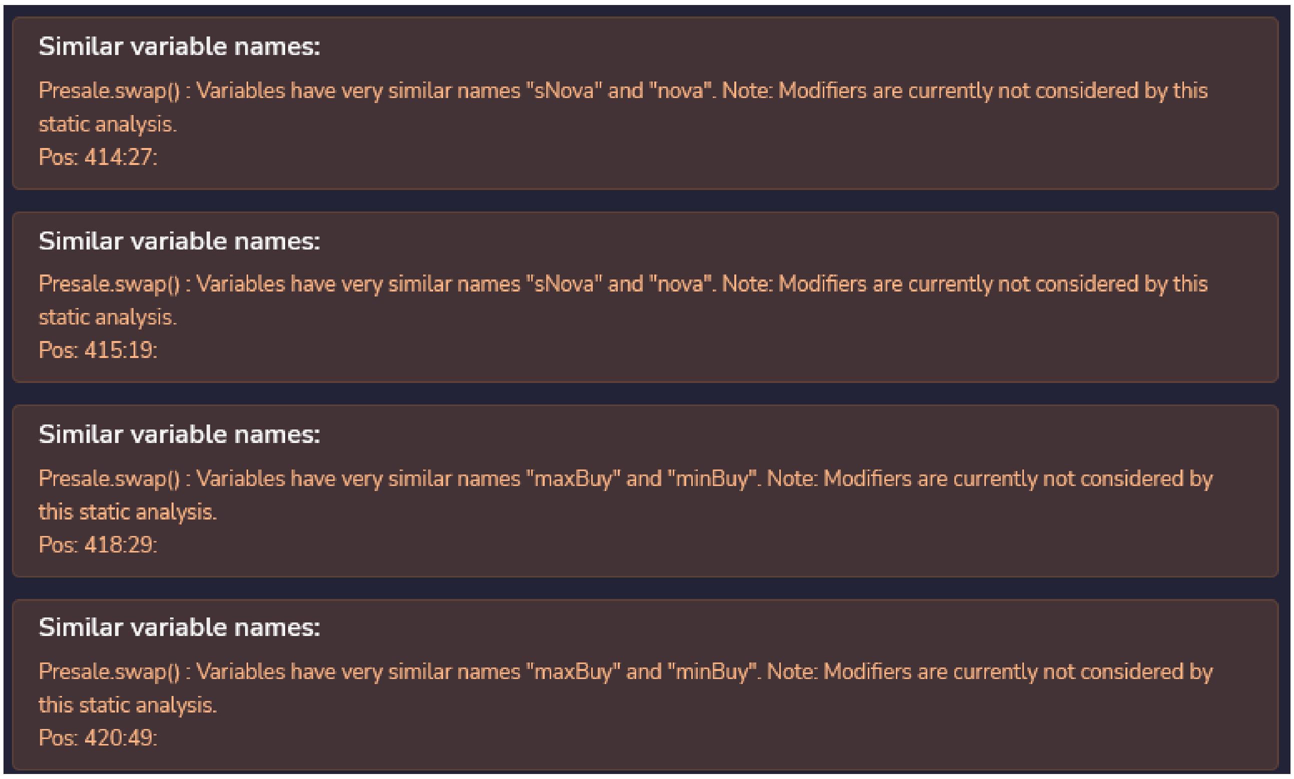
Pos: 509:3:



Similar variable names:

Presale.(address,contract IERC20,uint256,uint256,uint256,uint256,uint256) : Variables have very similar names "maxBuy" and "minBuy". Note: Modifiers are currently not considered by this static analysis.

Pos: 397:8:



Similar variable names:

Presale.swap() : Variables have very similar names "sNova" and "nova". Note: Modifiers are currently not considered by this static analysis.
Pos: 414:27:

Similar variable names:

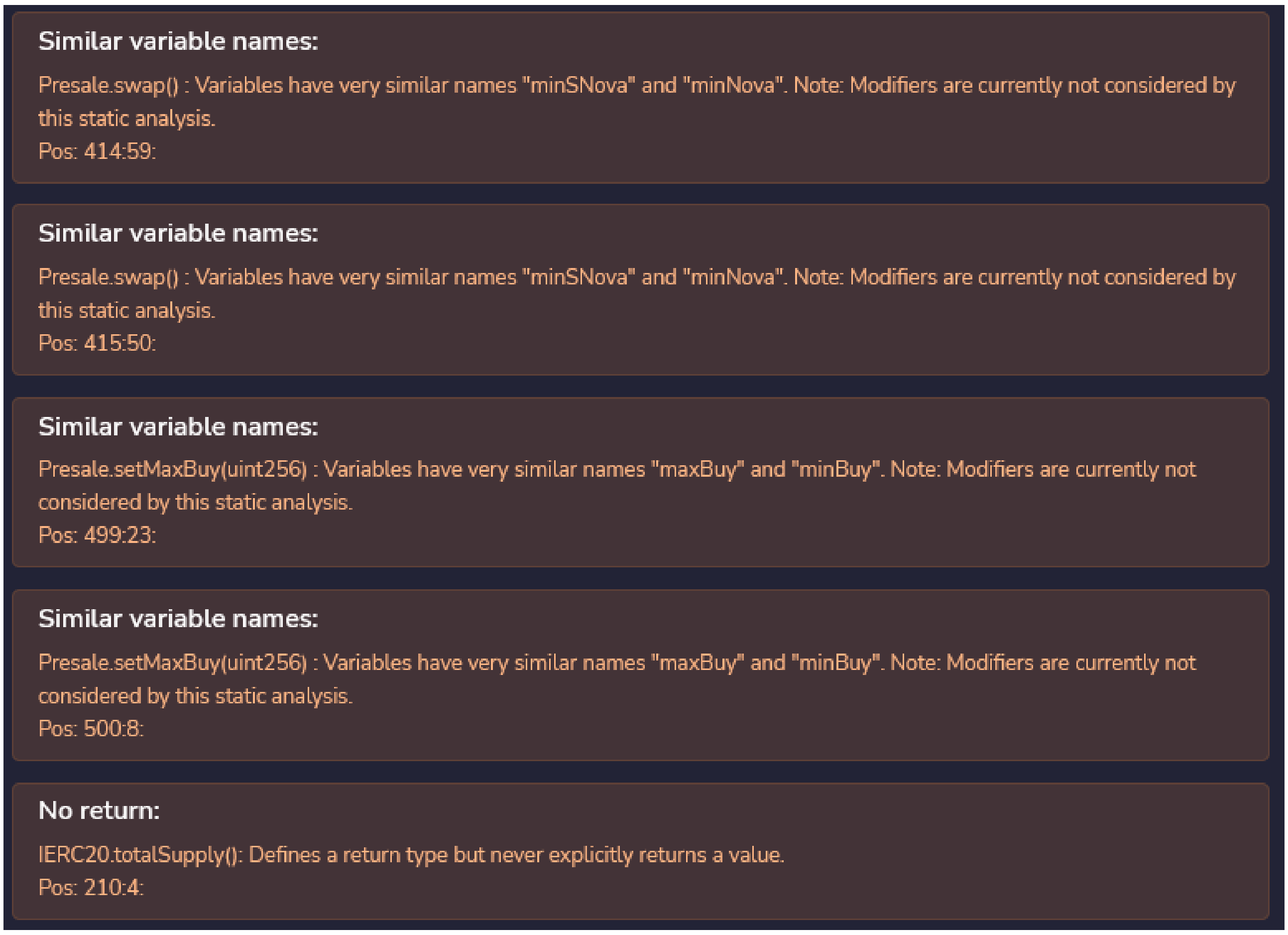
Presale.swap() : Variables have very similar names "sNova" and "nova". Note: Modifiers are currently not considered by this static analysis.
Pos: 415:19:

Similar variable names:

Presale.swap() : Variables have very similar names "maxBuy" and "minBuy". Note: Modifiers are currently not considered by this static analysis.
Pos: 418:29:

Similar variable names:

Presale.swap() : Variables have very similar names "maxBuy" and "minBuy". Note: Modifiers are currently not considered by this static analysis.
Pos: 420:49:



Similar variable names:

Presale.swap() : Variables have very similar names "minSNova" and "minNova". Note: Modifiers are currently not considered by this static analysis.
Pos: 414:59:

Similar variable names:

Presale.swap() : Variables have very similar names "minSNova" and "minNova". Note: Modifiers are currently not considered by this static analysis.
Pos: 415:50:

Similar variable names:

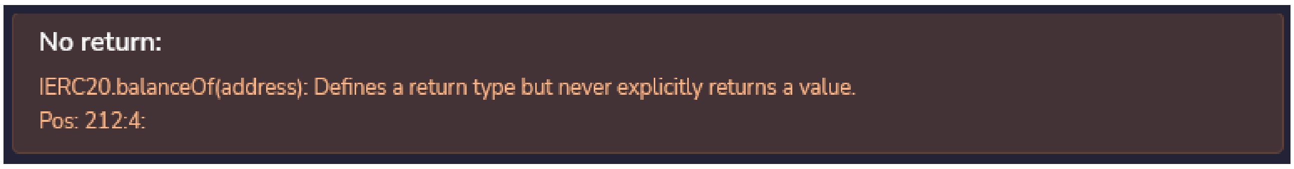
Presale.setMaxBuy(uint256) : Variables have very similar names "maxBuy" and "minBuy". Note: Modifiers are currently not considered by this static analysis.
Pos: 499:23:

Similar variable names:

Presale.setMaxBuy(uint256) : Variables have very similar names "maxBuy" and "minBuy". Note: Modifiers are currently not considered by this static analysis.
Pos: 500:8:

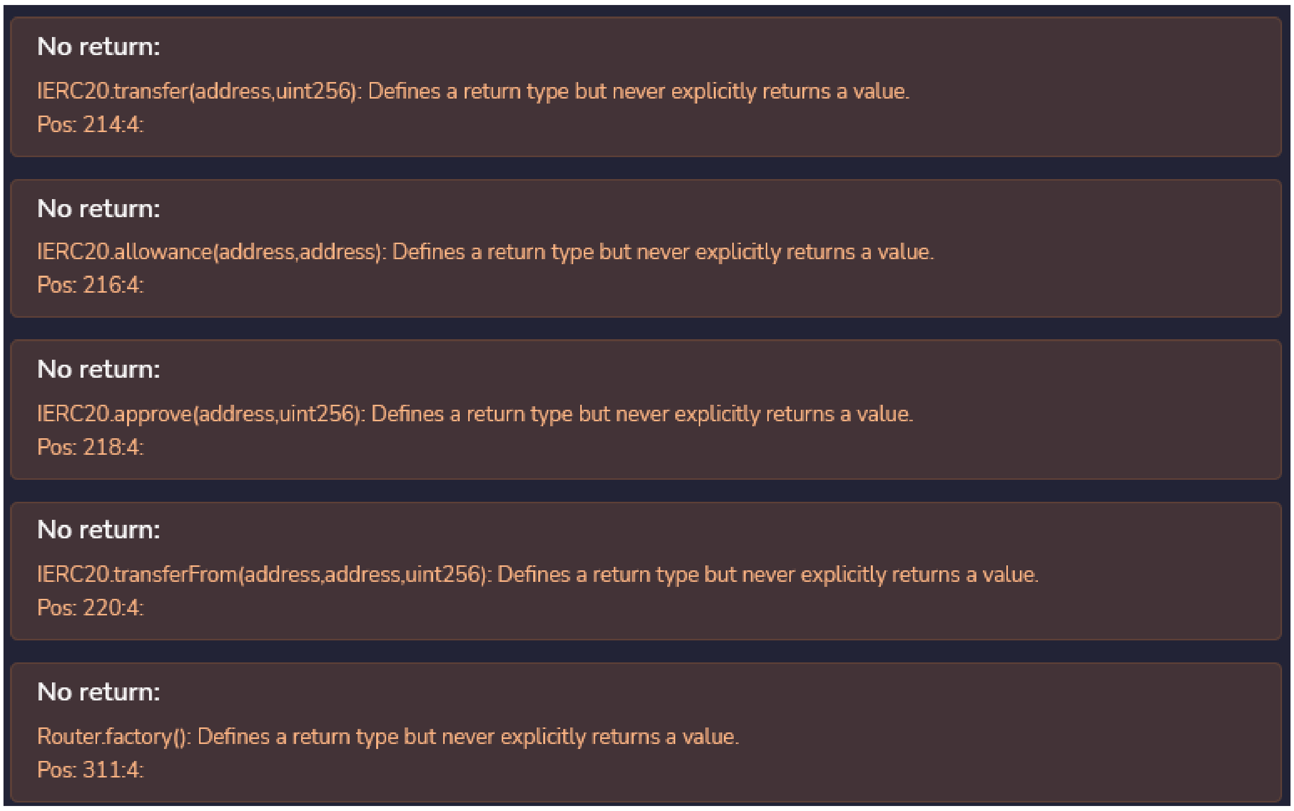
No return:

IERC20.totalSupply(): Defines a return type but never explicitly returns a value.
Pos: 210:4:



No return:

IERC20.balanceOf(address): Defines a return type but never explicitly returns a value.
Pos: 212:4:



No return:

IERC20.transfer(address,uint256): Defines a return type but never explicitly returns a value.
Pos: 214:4:

No return:

IERC20.allowance(address,address): Defines a return type but never explicitly returns a value.
Pos: 216:4:

No return:

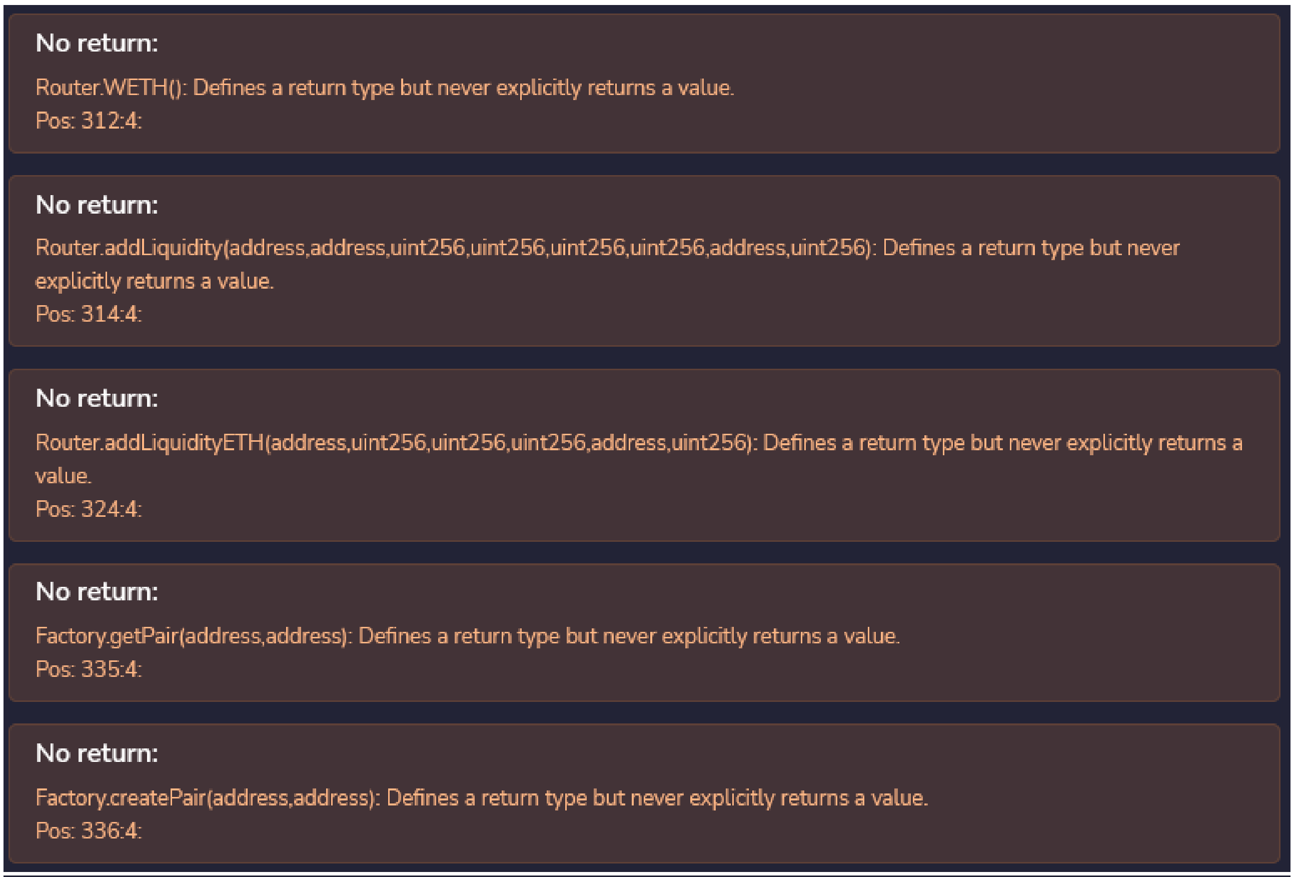
IERC20.approve(address,uint256): Defines a return type but never explicitly returns a value.
Pos: 218:4:

No return:

IERC20.transferFrom(address,address,uint256): Defines a return type but never explicitly returns a value.
Pos: 220:4:

No return:

Router.factory(): Defines a return type but never explicitly returns a value.
Pos: 311:4:



No return:

Router.WETH(): Defines a return type but never explicitly returns a value.
Pos: 312:4:

No return:

Router.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256): Defines a return type but never explicitly returns a value.
Pos: 314:4:

No return:

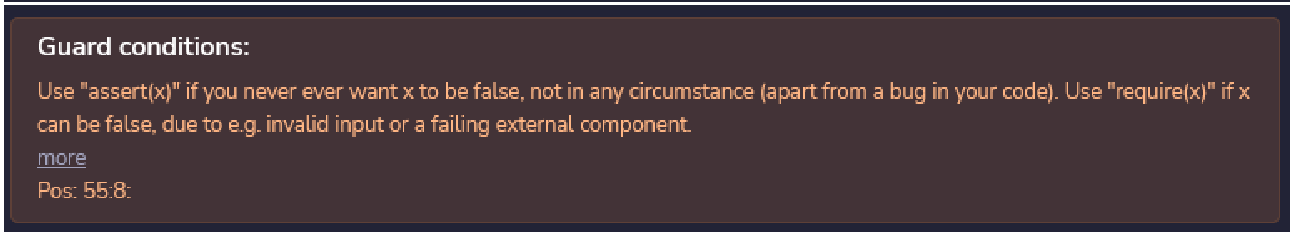
Router.addLiquidityETH(address,uint256,uint256,uint256,address,uint256): Defines a return type but never explicitly returns a value.
Pos: 324:4:

No return:

Factory.getPair(address,address): Defines a return type but never explicitly returns a value.
Pos: 335:4:

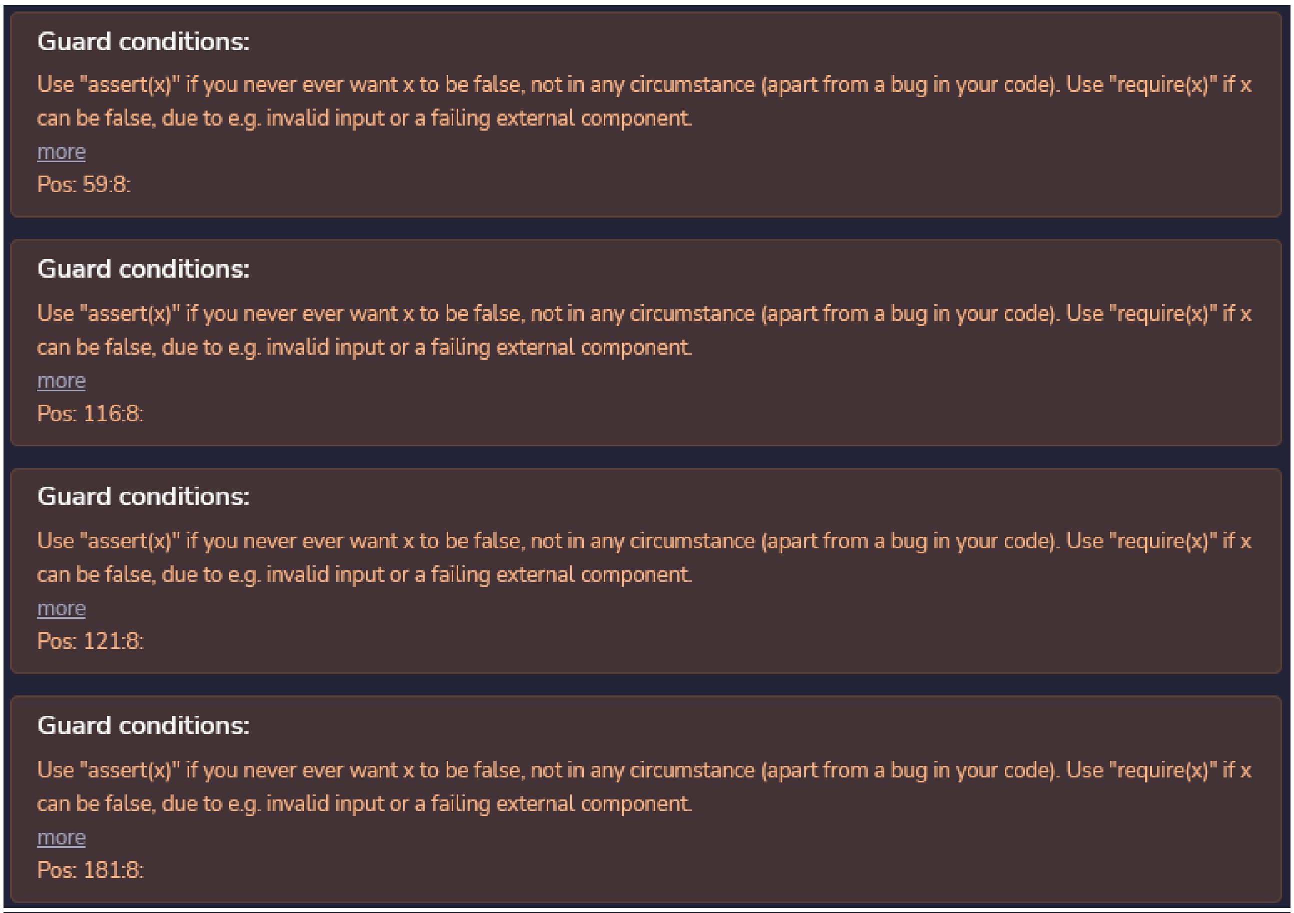
No return:

Factory.createPair(address,address): Defines a return type but never explicitly returns a value.
Pos: 336:4:



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: 55:8:



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: 59:8:

Guard conditions:

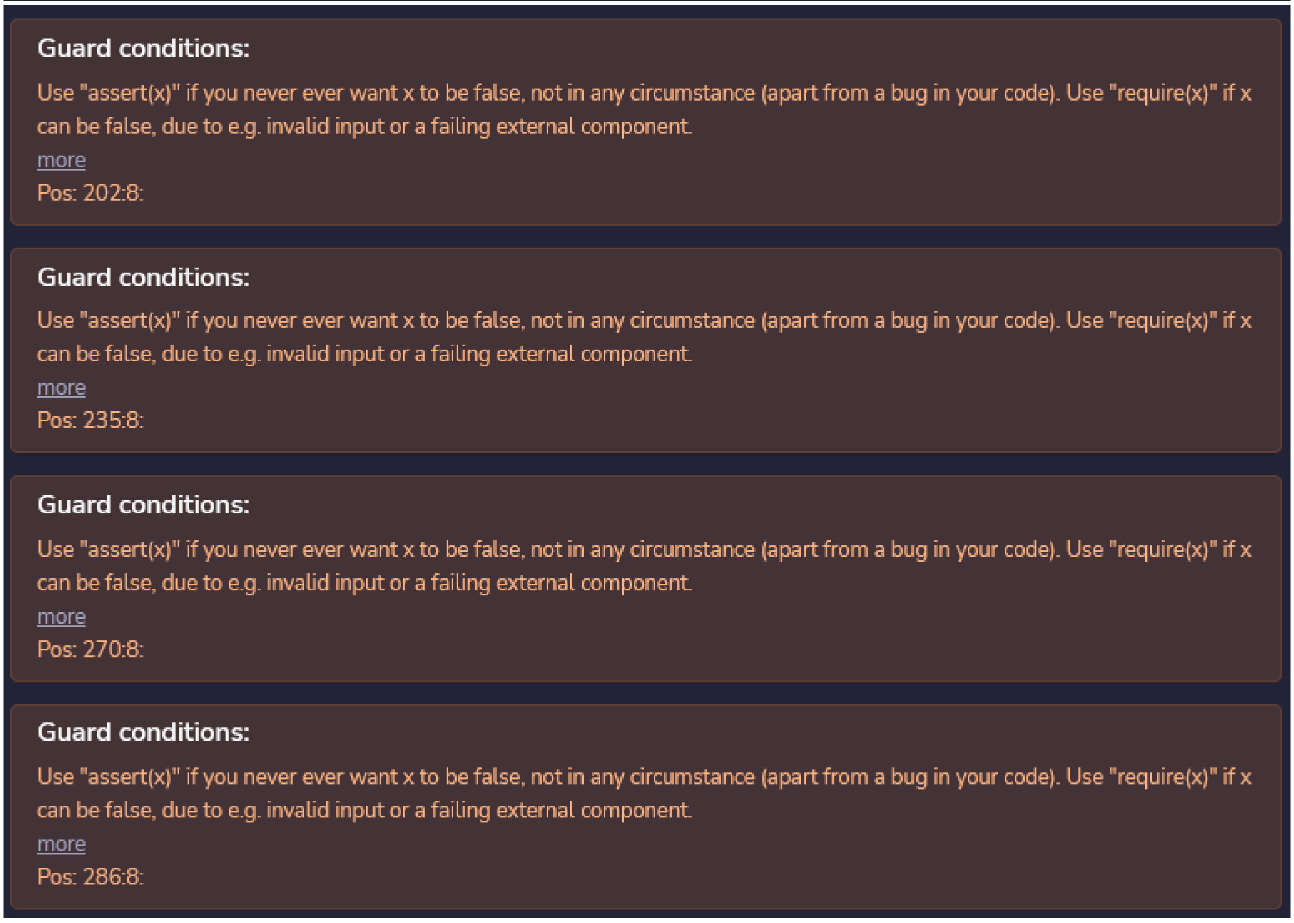
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 116:8:

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: 121:8:

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: 181:8:



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: 202:8:

Guard conditions:

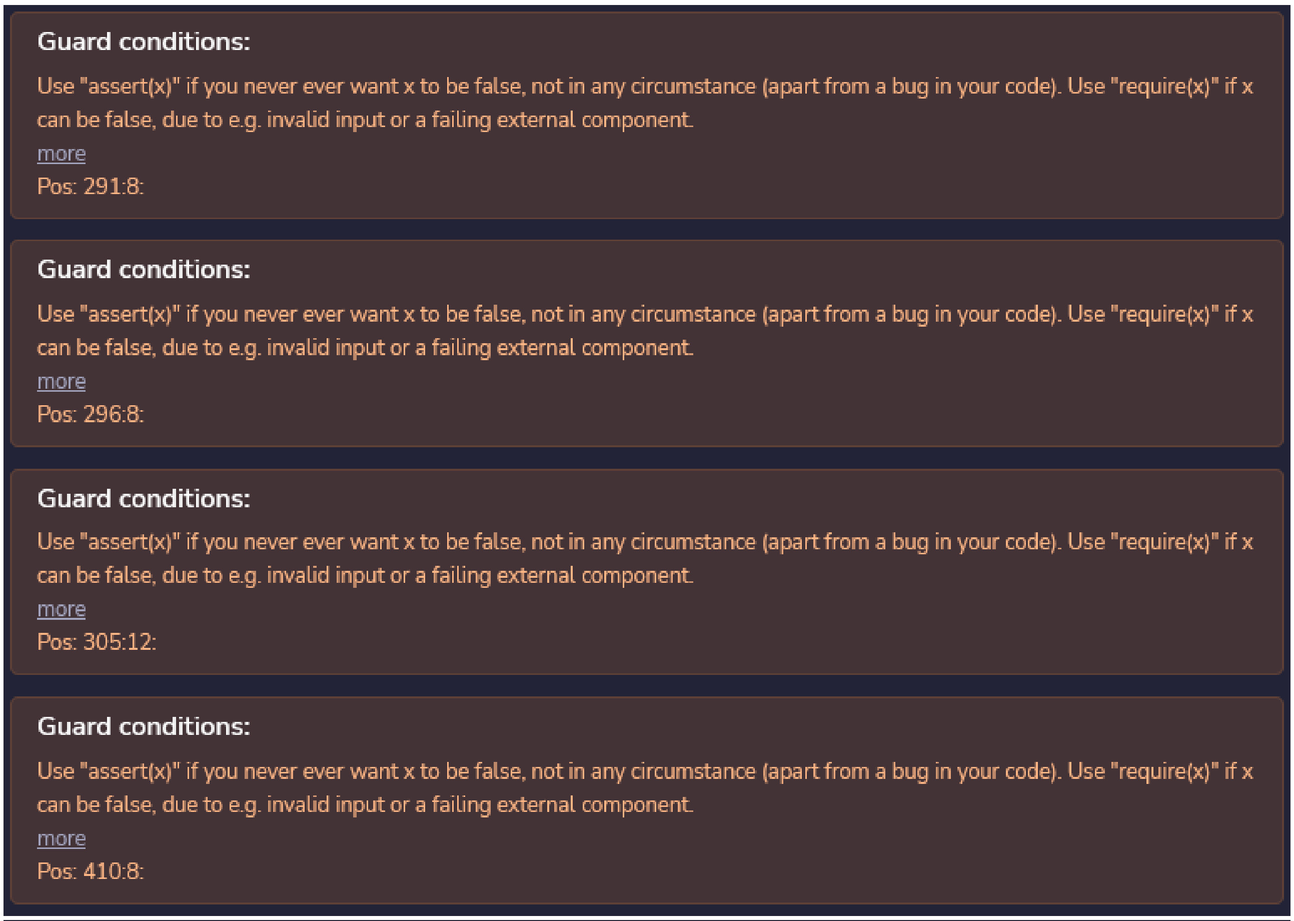
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 235:8:

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: 270:8:

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: 286:8:


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: 291:8:

Guard conditions:

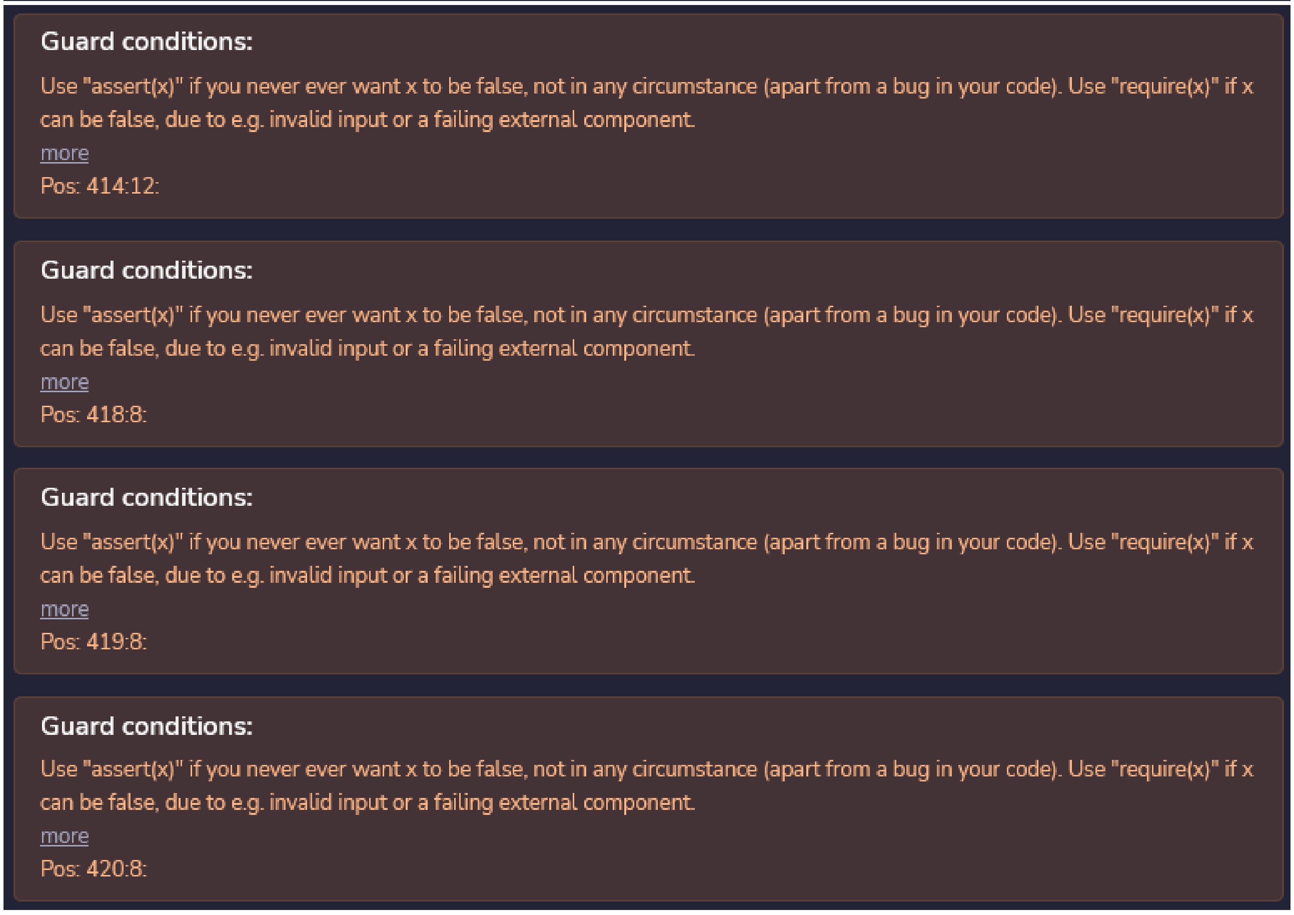
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 296:8:

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: 305:12:

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: 410:8:


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: 414:12:

Guard conditions:

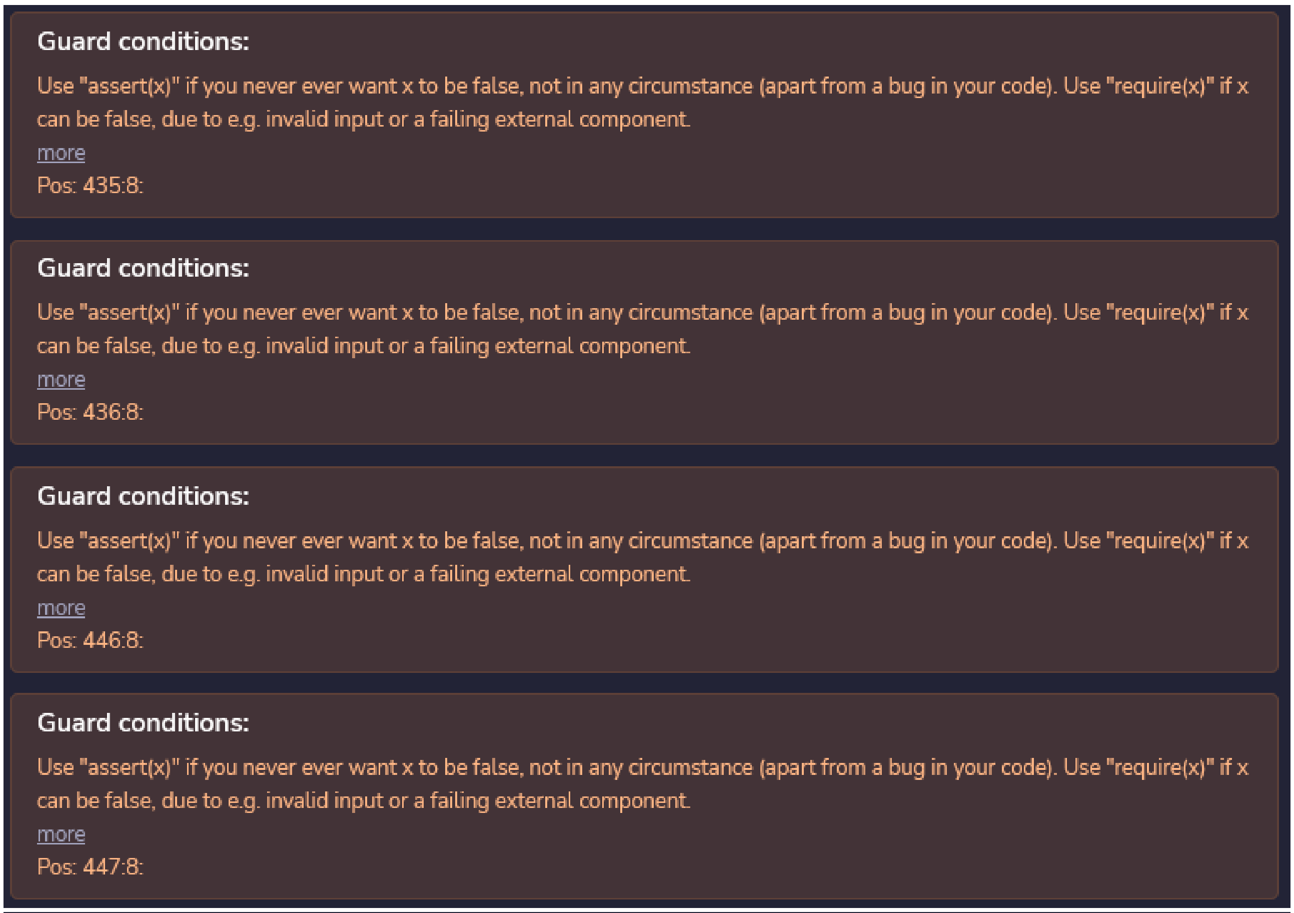
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 418:8:

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: 419:8:

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: 420:8:



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: 435:8:

Guard conditions:

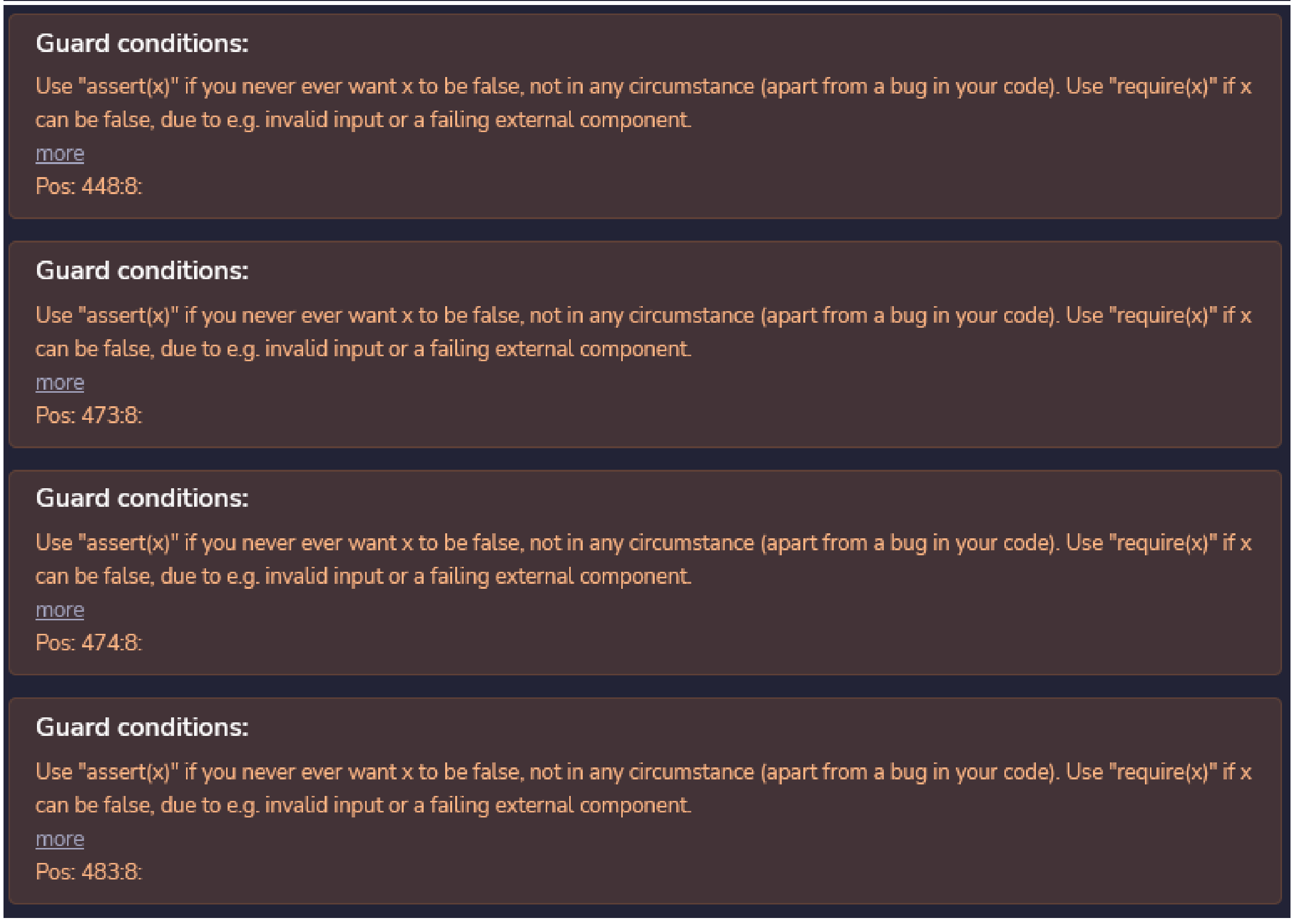
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 436:8:

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: 446:8:

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: 447:8:



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: 448:8:

Guard conditions:

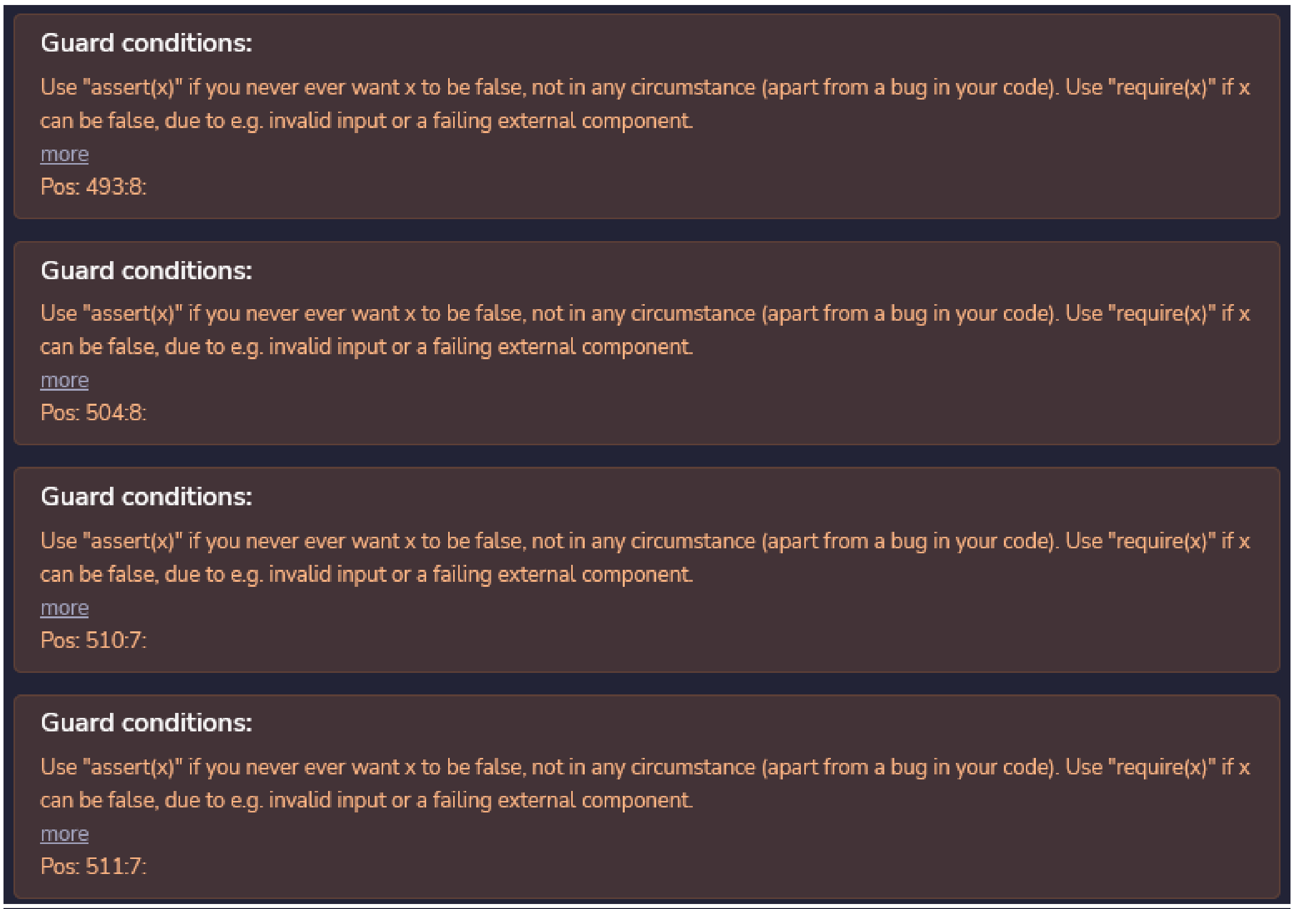
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 473:8:

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: 474:8:

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: 483:8:



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: 493:8:

Guard conditions:

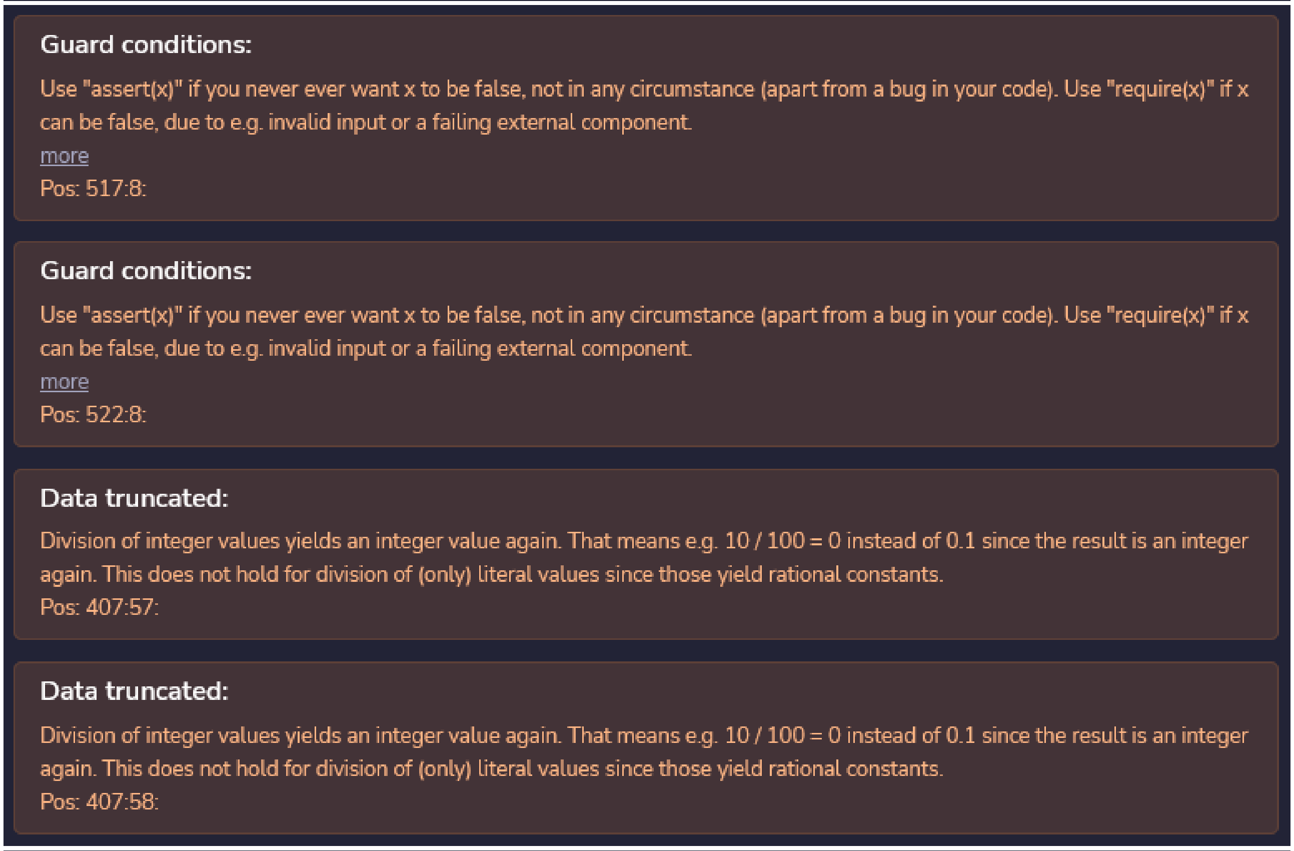
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 504:8:

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: 510:7:

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: 511:7:



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: 517:8:

Guard conditions:

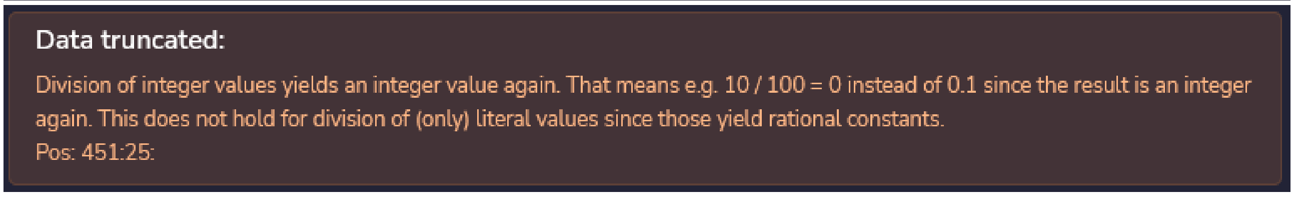
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 522:8:

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: 407:57:

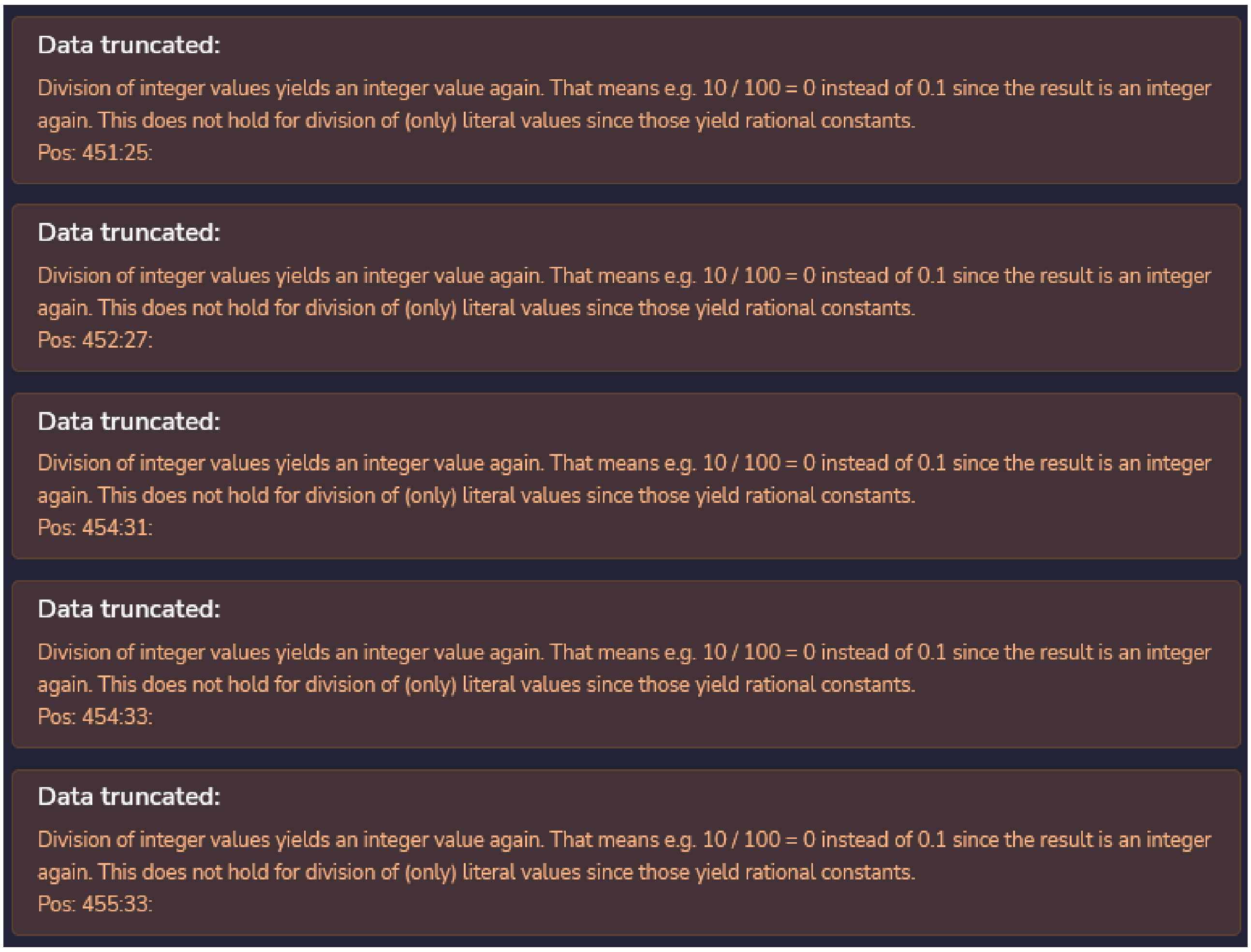
Data truncated:

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: 407:58:



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: 451:25:



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: 451:25:

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: 452:27:

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: 454:31:

Data truncated:

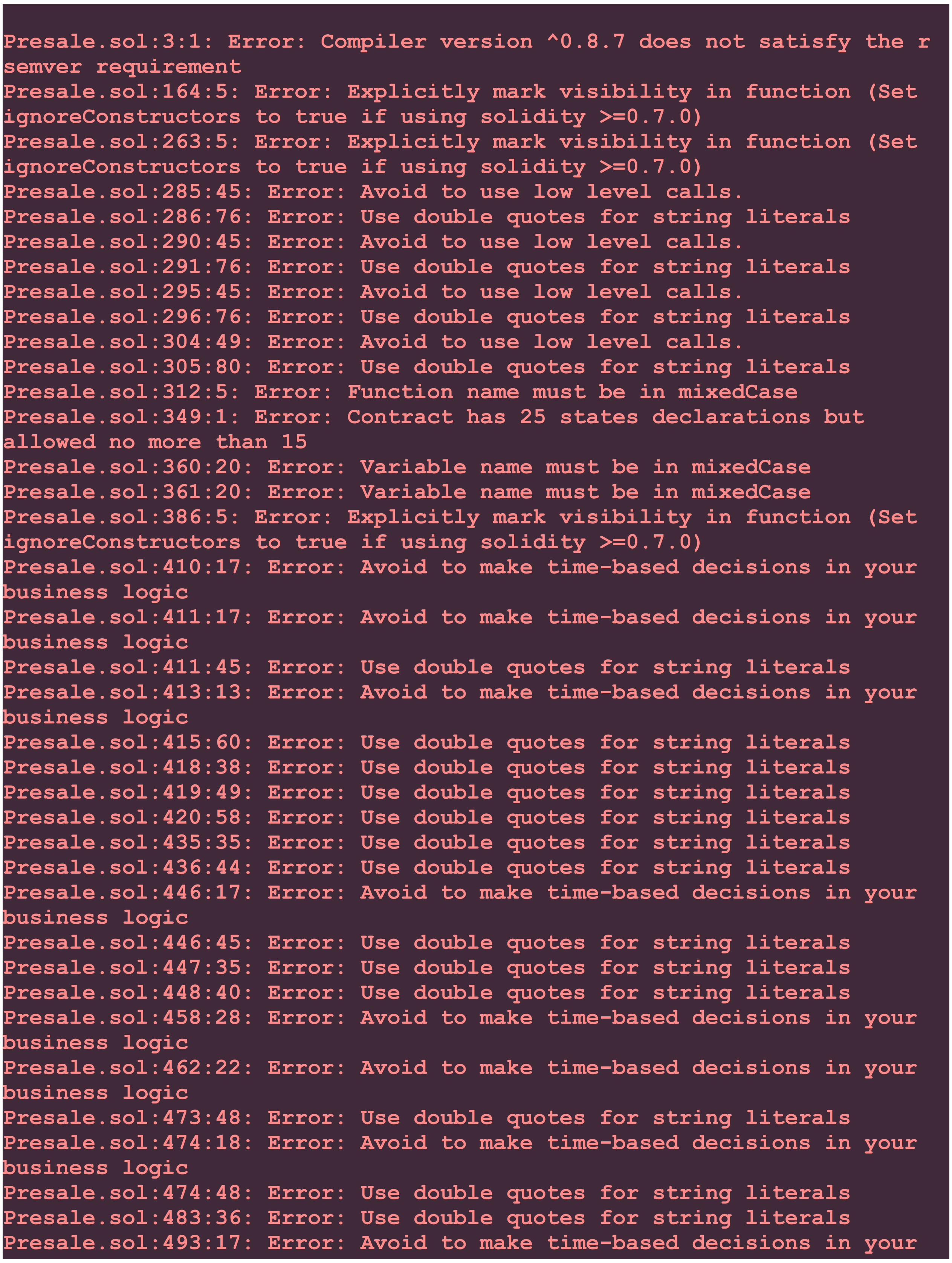
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 454:33:

Data truncated:

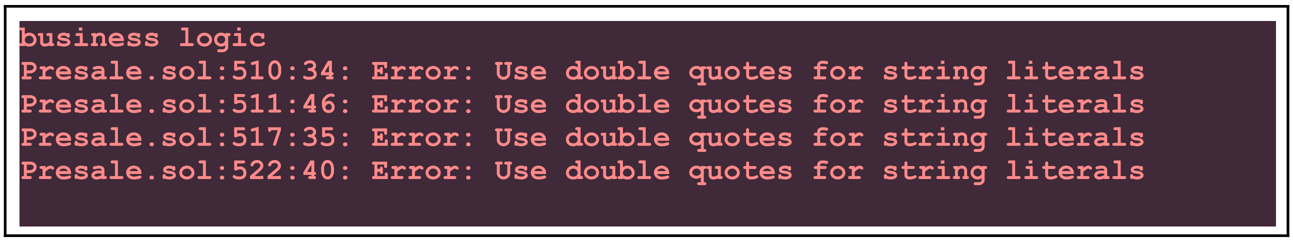
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 455:33:

Solhint Linter

presalev2.sol



```
Presale.sol:3:1: Error: Compiler version ^0.8.7 does not satisfy the r
semver requirement
Presale.sol:164:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
Presale.sol:263:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
Presale.sol:285:45: Error: Avoid to use low level calls.
Presale.sol:286:76: Error: Use double quotes for string literals
Presale.sol:290:45: Error: Avoid to use low level calls.
Presale.sol:291:76: Error: Use double quotes for string literals
Presale.sol:295:45: Error: Avoid to use low level calls.
Presale.sol:296:76: Error: Use double quotes for string literals
Presale.sol:304:49: Error: Avoid to use low level calls.
Presale.sol:305:80: Error: Use double quotes for string literals
Presale.sol:312:5: Error: Function name must be in mixedCase
Presale.sol:349:1: Error: Contract has 25 states declarations but
allowed no more than 15
Presale.sol:360:20: Error: Variable name must be in mixedCase
Presale.sol:361:20: Error: Variable name must be in mixedCase
Presale.sol:386:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
Presale.sol:410:17: Error: Avoid to make time-based decisions in your
business logic
Presale.sol:411:17: Error: Avoid to make time-based decisions in your
business logic
Presale.sol:411:45: Error: Use double quotes for string literals
Presale.sol:413:13: Error: Avoid to make time-based decisions in your
business logic
Presale.sol:415:60: Error: Use double quotes for string literals
Presale.sol:418:38: Error: Use double quotes for string literals
Presale.sol:419:49: Error: Use double quotes for string literals
Presale.sol:420:58: Error: Use double quotes for string literals
Presale.sol:435:35: Error: Use double quotes for string literals
Presale.sol:436:44: Error: Use double quotes for string literals
Presale.sol:446:17: Error: Avoid to make time-based decisions in your
business logic
Presale.sol:446:45: Error: Use double quotes for string literals
Presale.sol:447:35: Error: Use double quotes for string literals
Presale.sol:448:40: Error: Use double quotes for string literals
Presale.sol:458:28: Error: Avoid to make time-based decisions in your
business logic
Presale.sol:462:22: Error: Avoid to make time-based decisions in your
business logic
Presale.sol:473:48: Error: Use double quotes for string literals
Presale.sol:474:18: Error: Avoid to make time-based decisions in your
business logic
Presale.sol:474:48: Error: Use double quotes for string literals
Presale.sol:483:36: Error: Use double quotes for string literals
Presale.sol:493:17: Error: Avoid to make time-based decisions in your
```



```
business logic
Presale.sol:510:34: Error: Use double quotes for string literals
Presale.sol:511:46: Error: Use double quotes for string literals
Presale.sol:517:35: Error: Use double quotes for string literals
Presale.sol:522:40: Error: Use double quotes for string literals
```

Software analysis result:

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