EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	WP Smart Contracts
Platform:	Multiple Blockchain
Language:	Solidity
Date:	September 23rd, 2023

Table of contents

Introduction

EtherAuthority was contracted by the WP Smart Contracts team to perform the Security audit of the WP Smart Contracts smart contracts 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 23rd, 2023.

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

- WP Smart Contracts is a contract that can be divided into multiples, each with unique functionalities:
  - **Bubblegum:** BubblegumCrowdsale is an extension of Crowdsale where tokens are held by a wallet, which approves an allowance for the crowdsale.
  - **Tiramisu:** Tiramisu is a whitelisted airdrop that allows recipients to receive a fixed amount at once.
  - **Guava:** Guava is an airdrop system that allows any recipient to receive a fixed amount..
  - **Coconut:** The smart contract is designed for secure storage and management of native coins, ERC-20 tokens, ERC-721 NFTs, and ERC-1155 NFTs, offering deposit, withdrawal, and emergency account management features.
- WP Smart Contracts are a set of blockchain networks including Ethereum, Arbitrum, Binance Smart Chain, Polygon, Avalanche, and Fantom.

Audit scope

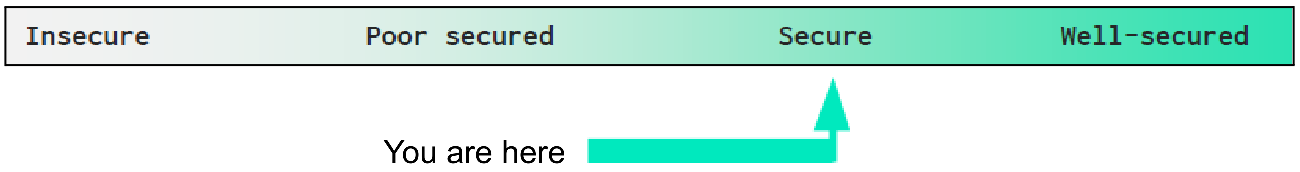
Name	**Code Review and Security Analysis Report for WP Smart Contracts Smart Contracts**
Platform	**Multiple Blockchain / Solidity**
File 1	Bubblegum.sol
File 2	Tiramisu.sol
File 3	Guava.sol
File 4	Coconut.sol
Audit Date	September 23rd, 2023
Revised Audit Date	September 25th, 2023

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 Bubblegum.sol **<u>The Owner has control over the following functions:</u>** ● Set the rate. ● Set the wallet. ● Set the token. ● Pause / Unpause the ICO activity.	**YES, This is valid.**
File 2 Tiramisu.sol **<u>The Owner has control over the following functions:</u>** ● Allows the contract owner to bulk load the addresses and their corresponding total token amounts for airdrop. ● Allows the contract owner to load a single beneficiary's address and the total token amount to airdrop to their account. ● Update the address of the ERC20 token contract. ● Update the wallet address. ● Pause / Unpause the airdrop activity.	**YES, This is valid.**
File 3 Guava.sol **<u>The Owner has control over the following functions:</u>** ● Set the amount of tokens to airdrop. ● Update the address of the ERC20 token. ● Set the wallet address. ● Pause / Unpause the airdrop activity.	**YES, This is valid.**
File 4 Coconut.sol **<u>The Owner has control over the following functions:</u>** ● Set the expiration time. ● Set the last activity timestamp. ● Add or update a trusted account.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. Also, these contracts contain owner control, which does not make them 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, 0 low and 2 very low level issues.

We confirm that all issues are fixed in the revised smart contract code.

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

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	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	Passed
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 4 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in WP Smart Contracts 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 WP Smart Contracts Protocol.

The WP Smart Contracts team has not provided unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on smart contracts.

Documentation

We were given a WP Smart Contracts smart contract code in the form of a https://gist.github.com web link. The hash of that code is mentioned above in the table.

As mentioned above, code parts are well commented on. And the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contracts infrastructure that are based on well known industry standard open source projects.

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

AS-IS overview

Bubblegum.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	tokenWallet	external	Passed	No Issue
3	remainingTokens	external	Passed	No Issue
4	_deliverTokens	internal	Passed	No Issue
5	setRate	write	access only Owner	No Issue
6	setWallet	write	access only Owner	No Issue
	_isContract	internal	Passed	No Issue
7	setToken	write	access only Owner	No Issue
8	receive	external	Passed	No Issue
9	token	read	Passed	No Issue
10	wallet	external	Passed	No Issue
11	rate	external	Passed	No Issue
12	weiRaised	read	Passed	No Issue
13	buyTokens	write	Passed	No Issue
14	_preValidatePurchase	internal	Passed	No Issue
15	_postValidatePurchase	internal	Removed	-
16	_deliverTokens	internal	Passed	No Issue
17	_processPurchase	internal	Passed	No Issue
18	_getTokenAmount	internal	Passed	No Issue
19	_updatePurchasingState	internal	Removed	-
20	_forwardFunds	internal	Passed	No Issue
21	pause	external	access only Owner	No Issue
22	unpause	external	access only Owner	No Issue

Guava.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	set	external	access only Owner	No Issue
3	claim	external	Passed	No Issue
4	changeToken	external	access only Owner	No Issue
5	changeWallet	external	access only Owner	No Issue
6	pause	external	access only Owner	No Issue
7	unpause	external	access only Owner	No Issue
8	receive	external	Passed	No Issue
9	owner	read	Passed	No Issue
10	onlyOwner	modifier	Passed	No Issue
11	renounceOwnership	write	access only Owner	No Issue
12	transferOwnership	write	access only Owner	No Issue

13	_setOwner	write	Passed	No Issue
14	nonReentrant	modifier	Passed	No Issue
15	paused	read	Passed	No Issue
16	whenNotPaused	modifier	Passed	No Issue
17	whenPaused	modifier	Passed	No Issue
18	_pause	internal	Passed	No Issue
19	_unpause	internal	Passed	No Issue

Tiramisu.sol

Functions

Sl.	**Functions**	**Type**	**Observation**	**Conclusion**
1	constructor	write	Passed	No Issue
2	load	external	access only Owner	No Issue
3	loadSingle	external	access only Owner	No Issue
4	claim	external	Passed	No Issue
5	changeToken	external	access only Owner	No Issue
6	changeWallet	external	access only Owner	No Issue
7	pause	external	access only Owner	No Issue
8	unpause	external	access only Owner	No Issue
9	receive	external	Passed	No Issue
10	owner	read	Passed	No Issue
11	onlyOwner	modifier	Passed	No Issue
12	renounceOwnership	write	access only Owner	No Issue
13	transferOwnership	write	access only Owner	No Issue
14	_setOwner	write	Passed	No Issue
15	nonReentrant	modifier	Passed	No Issue
16	paused	read	Passed	No Issue
17	whenNotPaused	modifier	Passed	No Issue
18	whenPaused	modifier	Passed	No Issue
19	_pause	internal	Passed	No Issue
20	_unpause	internal	Passed	No Issue

CoconutVault.sol

Functions

Sl.	**Functions**	**Type**	**Observation**	**Conclusion**
1	constructor	write	Passed	No Issue
2	setExpiration	external	access only Owner	No Issue
3	keepAlive	external	access only Owner	No Issue
4	deposit	write	Passed	No Issue
5	receive	external	Passed	No Issue
6	tokenDeposit	external	Passed	No Issue
7	erc721Deposit	external	Passed	No Issue
8	erc1155Deposit	external	Passed	No Issue
9	withdraw	external	can Withdraw	No Issue

10	tokenWithdraw	external	can Withdraw	No Issue
11	erc721Withdraw	external	can Withdraw	No Issue
12	erc1155Withdraw	external	can Withdraw	No Issue
13	changeTrustedAccount	external	access only Owner	No Issue
14	_updateActivity	internal	Passed	No Issue
15	canWithdraw	modifier	Passed	No Issue
16	onERC721Received	write	Passed	No Issue
17	onERC1155Received	write	Passed	No Issue
18	onERC1155BatchReceived	write	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
23	_setOwner	write	Passed	No Issue
24	nonReentrant	modifier	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have 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 in the contract code.

High Severity

No high severity vulnerabilities were found in the contract code.

Medium

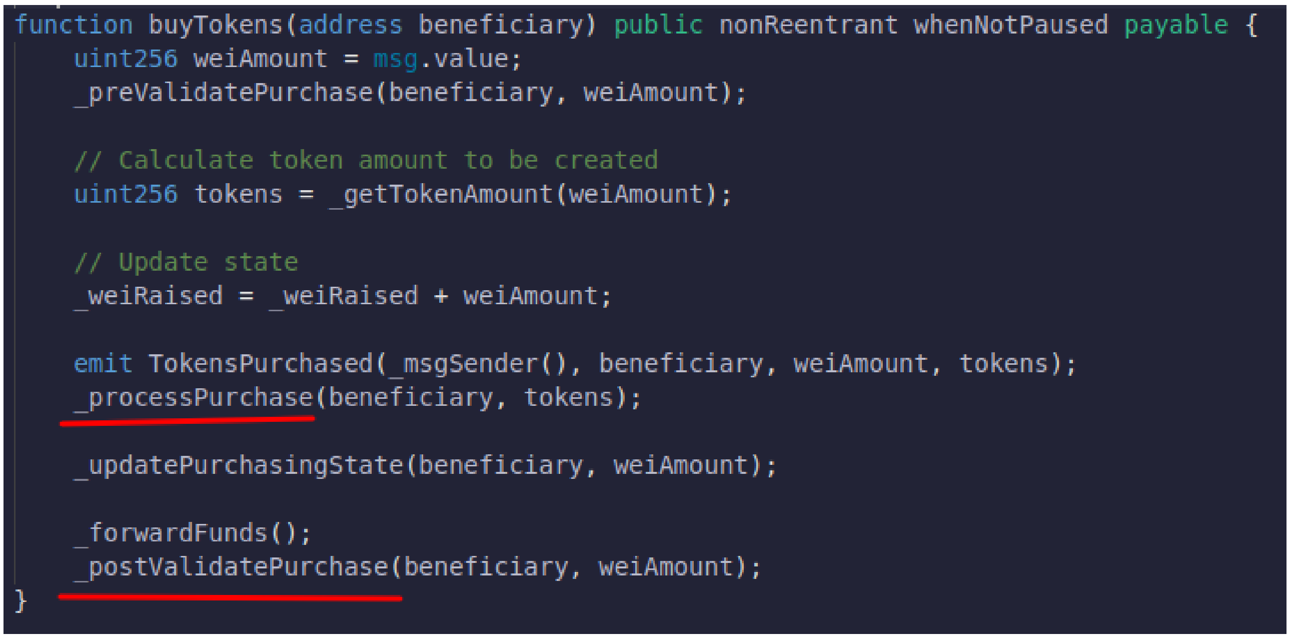
No medium severity vulnerabilities were found in the contract code.

Low

No low severity vulnerabilities were found in the contract code.

Very Low / Informational / Best practices:

(1) Blank function use: **Bubblegum.sol**

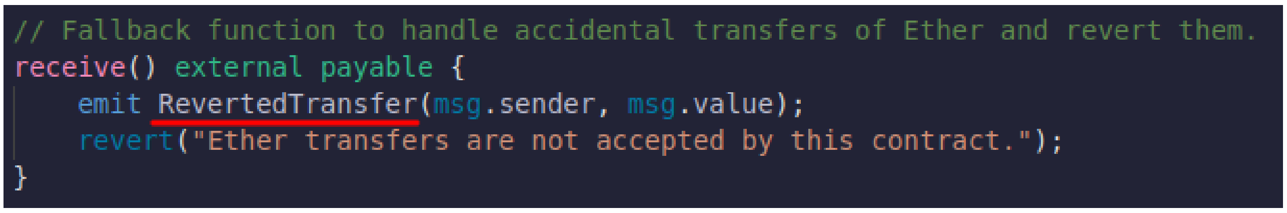


```
function buyTokens(address beneficiary) public nonReentrant whenNotPaused payable {
    uint256 weiAmount = msg.value;
    _preValidatePurchase(beneficiary, weiAmount);

    // Calculate token amount to be created
    uint256 tokens = _getTokenAmount(weiAmount);

    // Update state
    _weiRaised = _weiRaised + weiAmount;

    emit TokensPurchased(_msgSender(), beneficiary, weiAmount, tokens);
    _processPurchase(beneficiary, tokens);

    _updatePurchasingState(beneficiary, weiAmount);

    _forwardFunds();
    _postValidatePurchase(beneficiary, weiAmount);
}
```

The buyTokens() function has _postValidatePurchase() and _processPurchase() internal functions used, but these functions do not have any functionality, and the function body is empty. So these function calls are useless.

Resolution: We suggest either removing these functions or adding some functionality for these functions.
Status: Fixed; removed these functions from the smart contract.

(2) Unused event: **Tiramisu.sol and Guava.sol**



```
// Fallback function to handle accidental transfers of Ether and revert them.
receive() external payable {
    emit RevertedTransfer(msg.sender, msg.value);
    revert("Ether transfers are not accepted by this contract.");
}
```

Here the event used in the receive function will never be logged as the receive function always reverts.

Resolution: We suggest removing unused event definition and use which is never going to be logged.
Status: Fixed; removed an unused event from the smart contract.

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:

Guava.sol

- set: The amount of tokens to airdrop can be set by the contract owner.
- changeToken: The address of the ERC20 token contract can be updated by the owner.
- changeWallet: The wallet address can be updated by the owner.
- pause: Pause the airdrop activity by the owner.
- unpause: Resume the airdrop activity by the owner.

Crowdsale.sol

- setRate: The owner can change the rate.
- setWallet: The owner can change the wallet.
- setToken: The owner can change the token if the crowdsale hasn't started yet.
- pause: Pauses the ICO activity by the owner.
- unpause: Resumes the ICO activity by the owner.

Tiramisu.sol

- load: Allows the contract owner to bulk load the addresses and their corresponding total token amounts for airdrop.
- loadSingle: Allows the contract owner to load a single beneficiary's address and the total token amount to airdrop to their account.
- changeToken: The address of the ERC20 token contract can be updated by the owner.
- changeWallet: The wallet address can be updated by the owner.
- pause: Pause the airdrop activity by the owner.
- unpause: Resume the airdrop activity by the owner.

CoconutVault.sol

- setExpiration: The expiration time can be set by the owner.
- keepAlive: The last activity timestamp can be set by the owner.
- changeTrustedAccount: Add or update a trusted account by the owner.

Ownable.sol

- renounce Ownership: Deleting ownership will leave the contract without an owner, removing any owner-only functionality.
- transfer ownership: The current owner can transfer ownership of the contract to a new account.

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

Conclusion

We were given a contract code in the form of a https://gist.github.com web link. And we have used all possible tests based on given objects as files. We had observed 2 Informational severity issues in the smart contracts. We confirm that all the issues are fixed in the revised smart contract code. **So, the smart contracts are ready for the mainnet deployment.**

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 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.

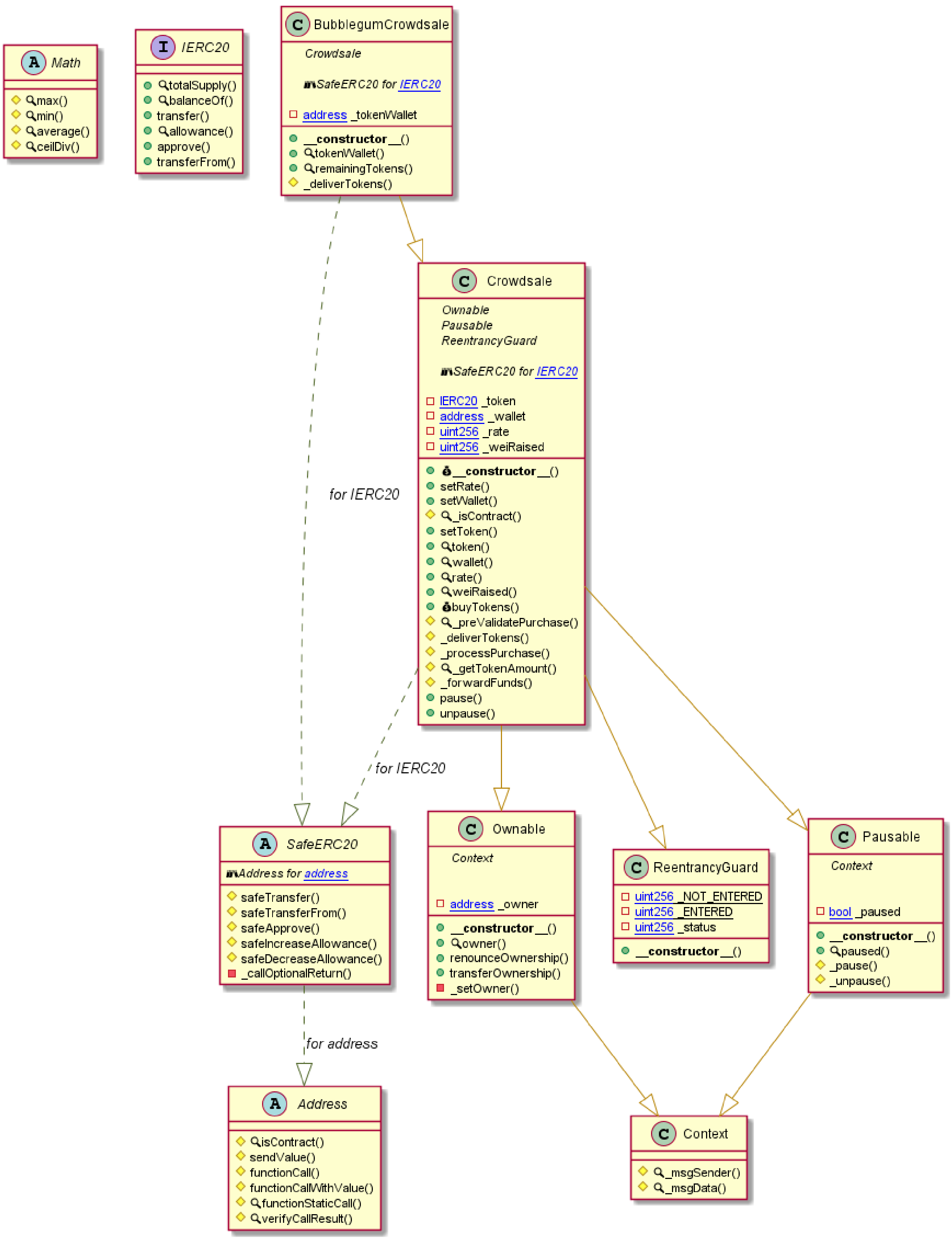
Technical Disclaimer

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.

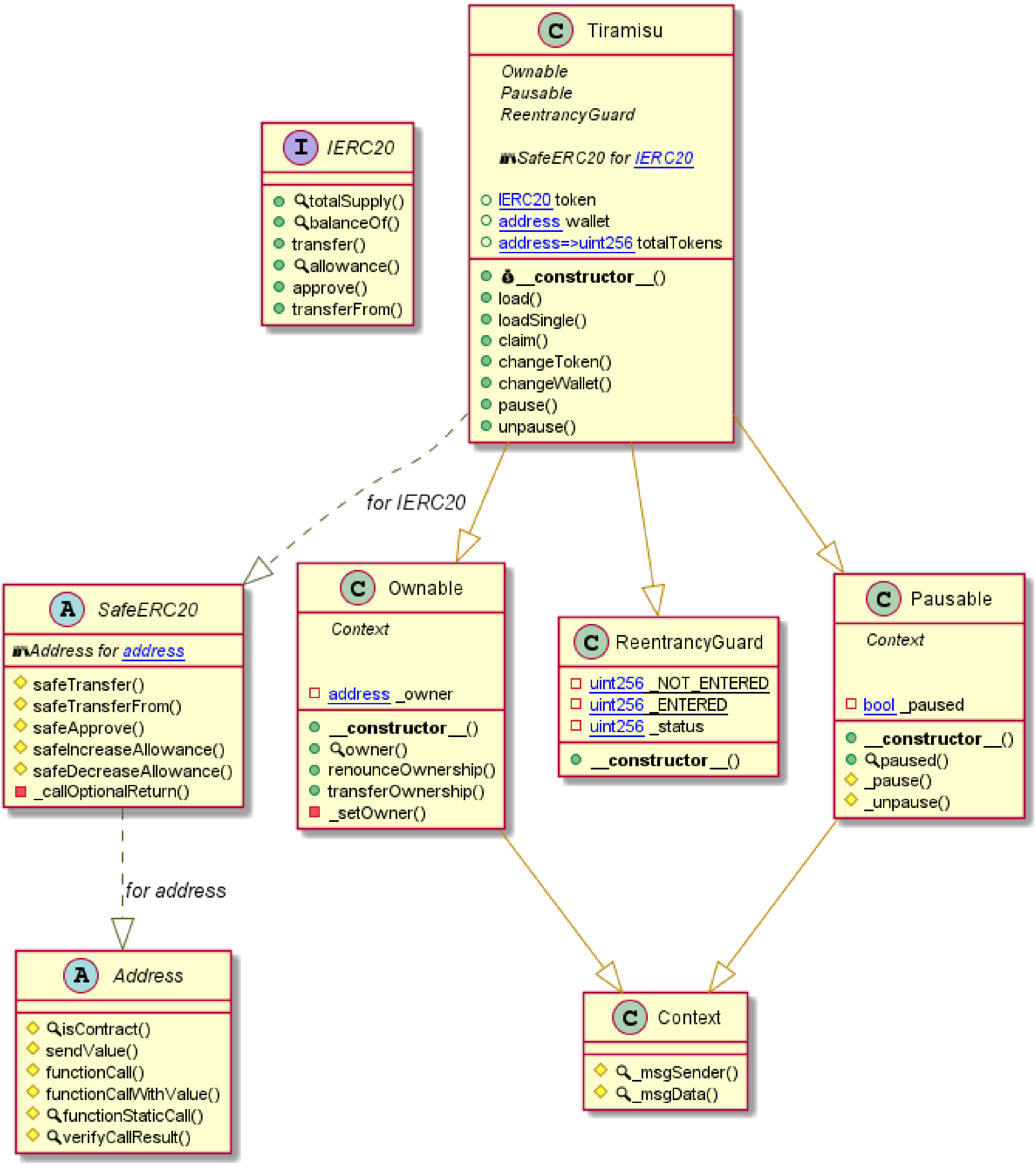
Appendix

Code Flow Diagram - WP Smart Contracts Protocol

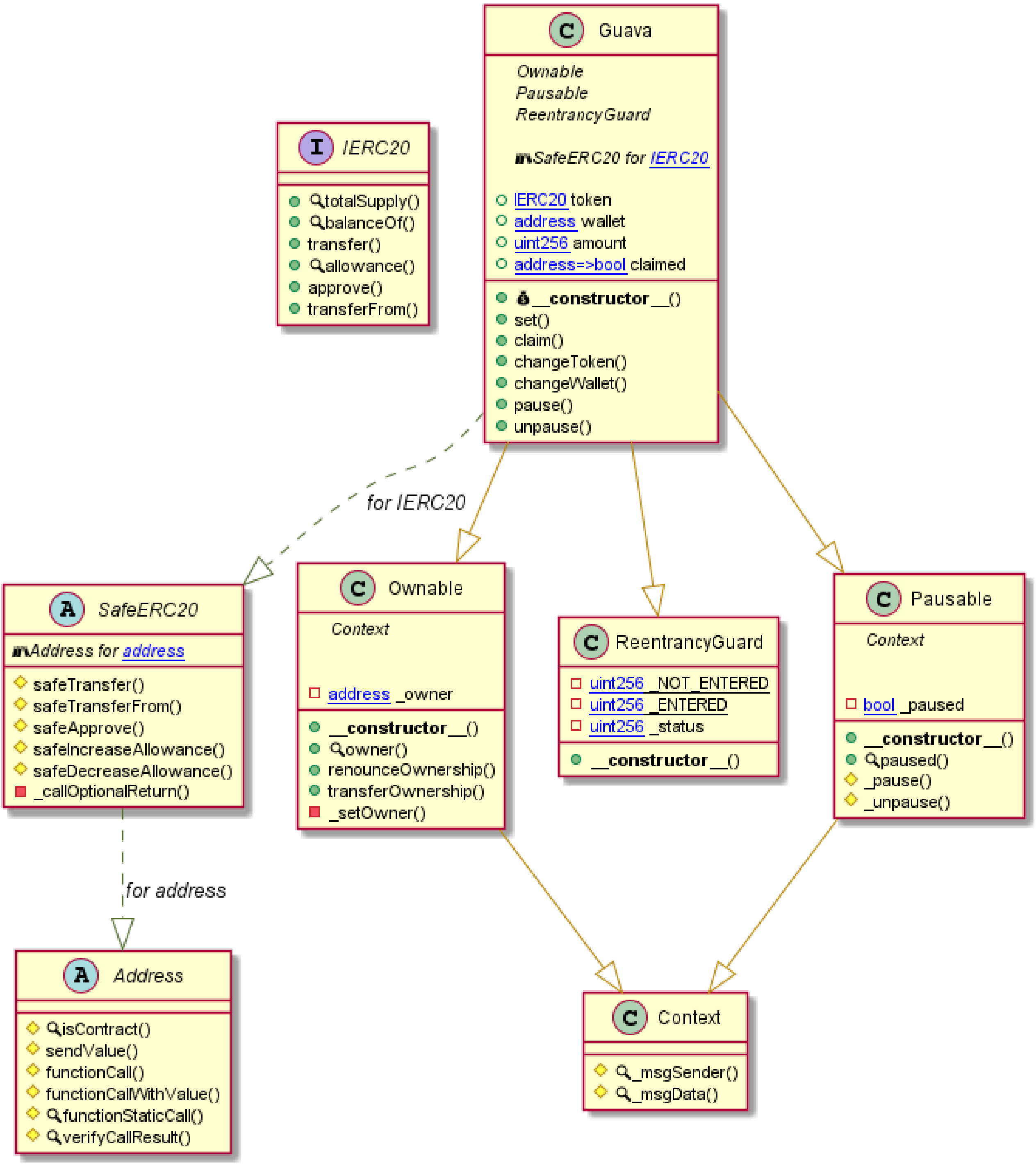
Bubblegum Diagram



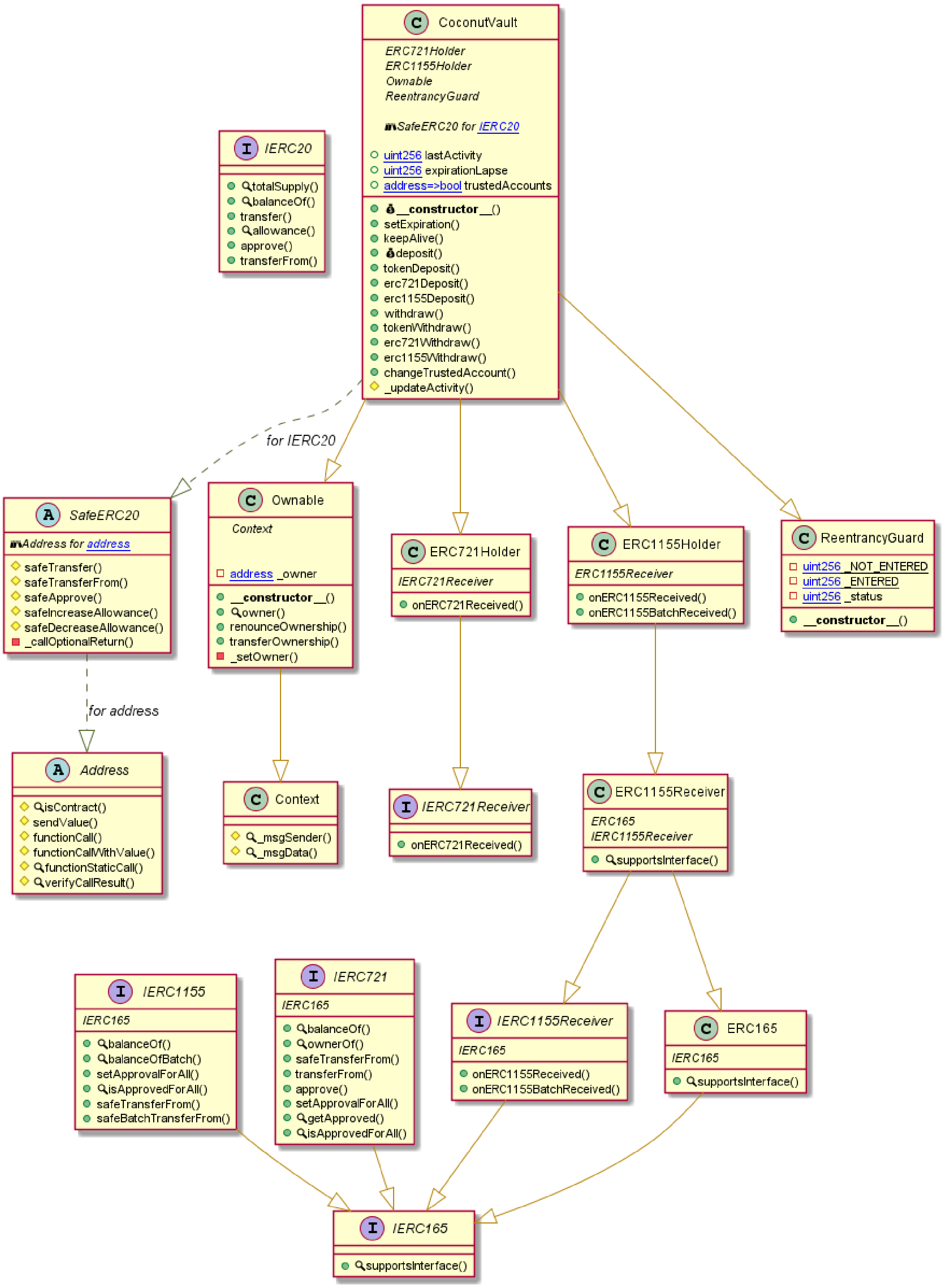
Tiramisu Diagram



Guava Diagram



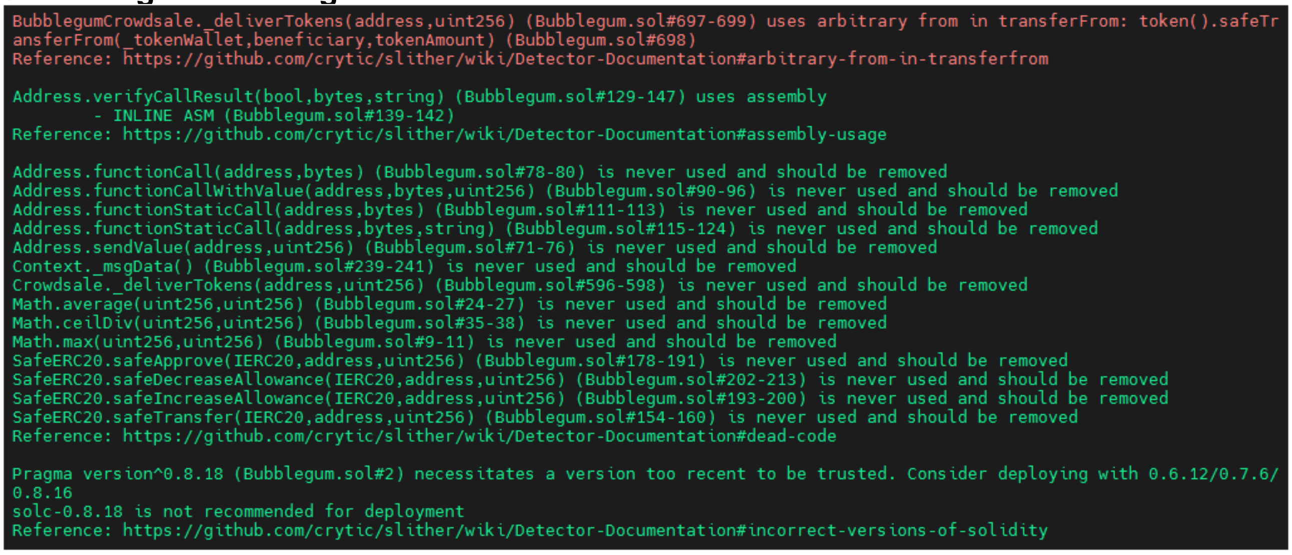
Coconut Diagram



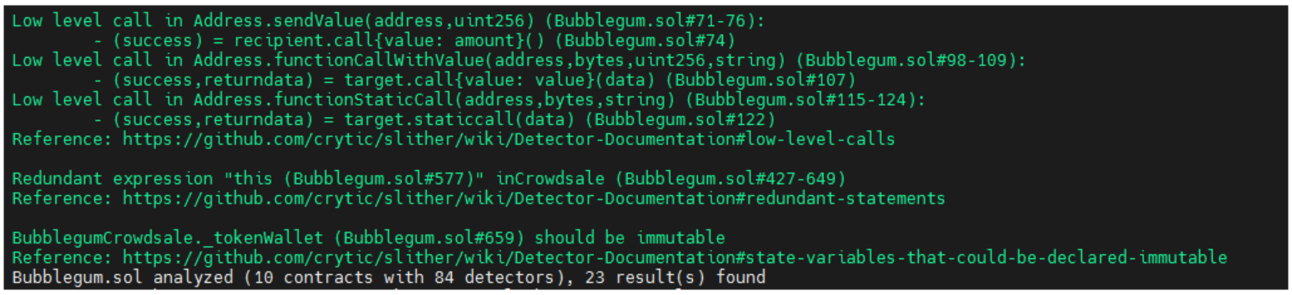
Slither Results Log

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

Slither log >> Bubblegum.sol

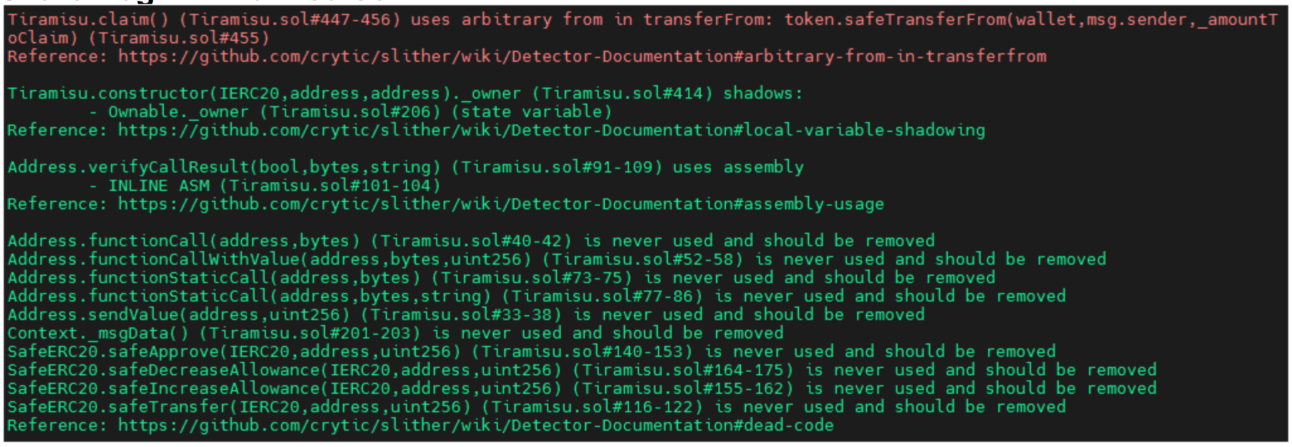


```
BubblegumCrowdsale._deliverTokens(address,uint256) (Bubblegum.sol#697-699) uses arbitrary from in transferFrom: token().safeTr
ansferFrom(_tokenWallet,beneficiary,tokenAmount) (Bubblegum.sol#698)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom

Address.verifyCallResult(bool,bytes,string) (Bubblegum.sol#129-147) uses assembly
        - INLINE ASM (Bubblegum.sol#139-142)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address.functionCall(address,bytes) (Bubblegum.sol#78-80) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Bubblegum.sol#90-96) is never used and should be removed
Address.functionStaticCall(address,bytes) (Bubblegum.sol#111-113) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (Bubblegum.sol#115-124) is never used and should be removed
Address.sendValue(address,uint256) (Bubblegum.sol#71-76) is never used and should be removed
Context._msgData() (Bubblegum.sol#239-241) is never used and should be removed
Crowdsale._deliverTokens(address,uint256) (Bubblegum.sol#596-598) is never used and should be removed
Math.average(uint256,uint256) (Bubblegum.sol#24-27) is never used and should be removed
Math.ceilDiv(uint256,uint256) (Bubblegum.sol#35-38) is never used and should be removed
Math.max(uint256,uint256) (Bubblegum.sol#9-11) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (Bubblegum.sol#178-191) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (Bubblegum.sol#202-213) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (Bubblegum.sol#193-200) is never used and should be removed
SafeERC20.safeTransfer(IERC20,address,uint256) (Bubblegum.sol#154-160) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.18 (Bubblegum.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/
0.8.16
solc-0.8.18 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

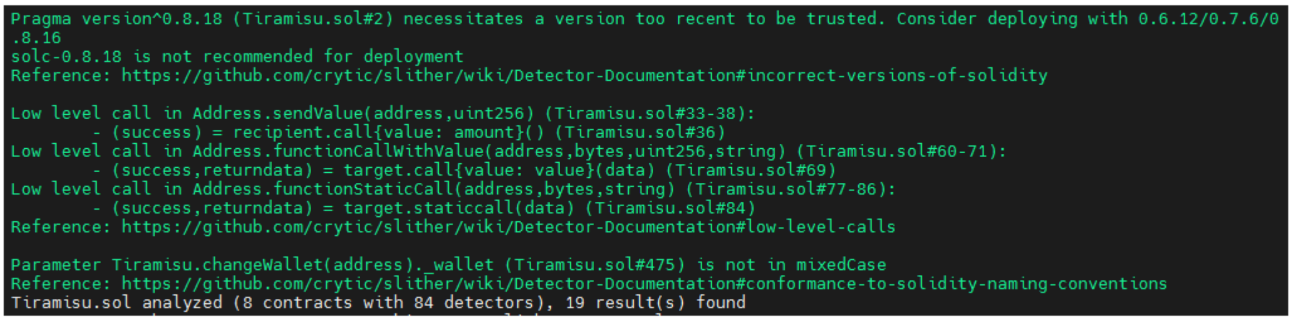


```
Low level call in Address.sendValue(address,uint256) (Bubblegum.sol#71-76):
        - (success) = recipient.call{value: amount}() (Bubblegum.sol#74)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Bubblegum.sol#98-109):
        - (success,returndata) = target.call{value: value}(data) (Bubblegum.sol#107)
Low level call in Address.functionStaticCall(address,bytes,string) (Bubblegum.sol#115-124):
        - (success,returndata) = target.staticcall(data) (Bubblegum.sol#122)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Redundant expression "this (Bubblegum.sol#577)" inCrowdsale (Bubblegum.sol#427-649)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

BubblegumCrowdsale._tokenWallet (Bubblegum.sol#659) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
Bubblegum.sol analyzed (10 contracts with 84 detectors), 23 result(s) found
```

Slither log >> Tiramisu.sol

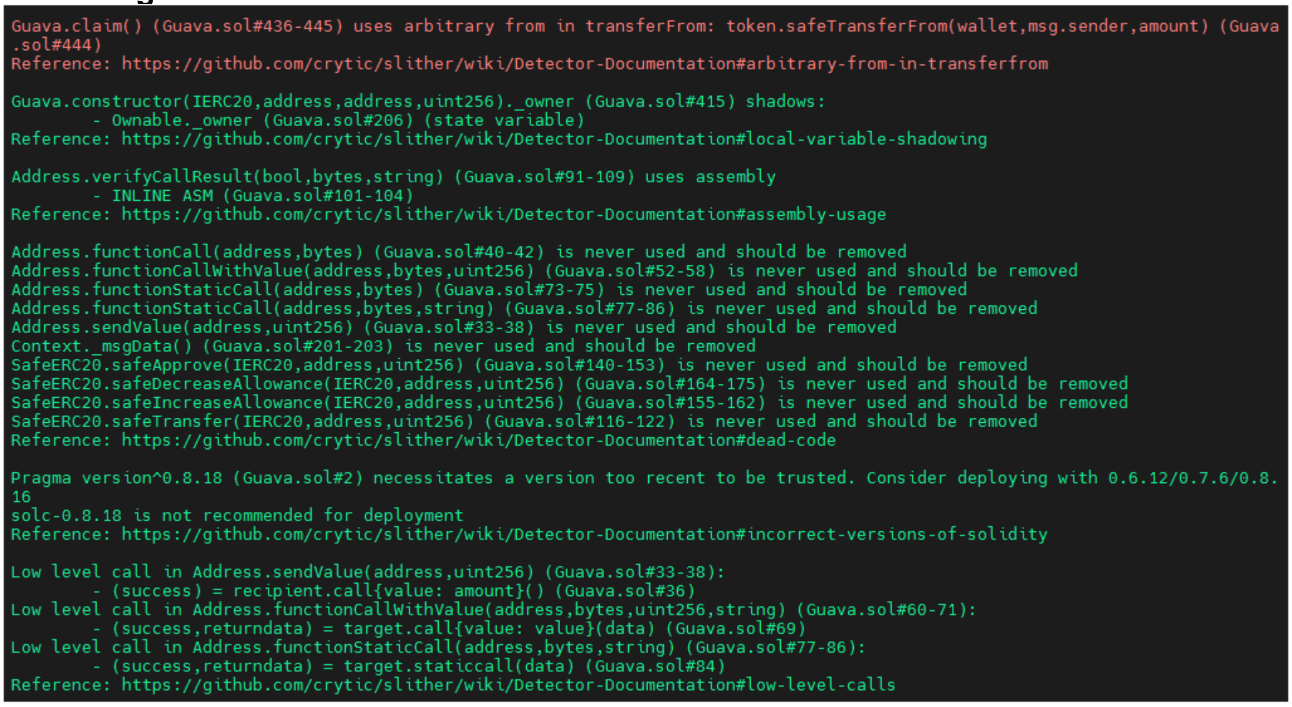


```
Tiramisu.claim() (Tiramisu.sol#447-456) uses arbitrary from in transferFrom: token.safeTransferFrom(wallet,msg.sender,_amountT
oClaim) (Tiramisu.sol#455)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom

Tiramisu.constructor(IERC20,address,address)._owner (Tiramisu.sol#414) shadows:
        - Ownable._owner (Tiramisu.sol#206) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Address.verifyCallResult(bool,bytes,string) (Tiramisu.sol#91-109) uses assembly
        - INLINE ASM (Tiramisu.sol#101-104)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address.functionCall(address,bytes) (Tiramisu.sol#40-42) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Tiramisu.sol#52-58) is never used and should be removed
Address.functionStaticCall(address,bytes) (Tiramisu.sol#73-75) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (Tiramisu.sol#77-86) is never used and should be removed
Address.sendValue(address,uint256) (Tiramisu.sol#33-38) is never used and should be removed
Context._msgData() (Tiramisu.sol#201-203) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (Tiramisu.sol#140-153) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (Tiramisu.sol#164-175) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (Tiramisu.sol#155-162) is never used and should be removed
SafeERC20.safeTransfer(IERC20,address,uint256) (Tiramisu.sol#116-122) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

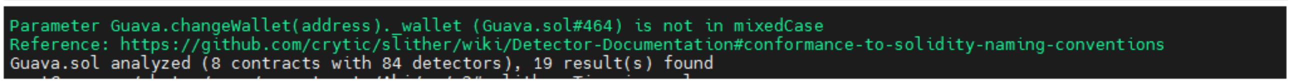


```
Pragma version^0.8.18 (Tiramisu.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0
.8.16
solc-0.8.18 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (Tiramisu.sol#33-38):
        - (success) = recipient.call{value: amount}() (Tiramisu.sol#36)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Tiramisu.sol#60-71):
        - (success,returndata) = target.call{value: value}(data) (Tiramisu.sol#69)
Low level call in Address.functionStaticCall(address,bytes,string) (Tiramisu.sol#77-86):
        - (success,returndata) = target.staticcall(data) (Tiramisu.sol#84)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter Tiramisu.changeWallet(address)._wallet (Tiramisu.sol#475) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
Tiramisu.sol analyzed (8 contracts with 84 detectors), 19 result(s) found
```

Slither log >> Guava.sol

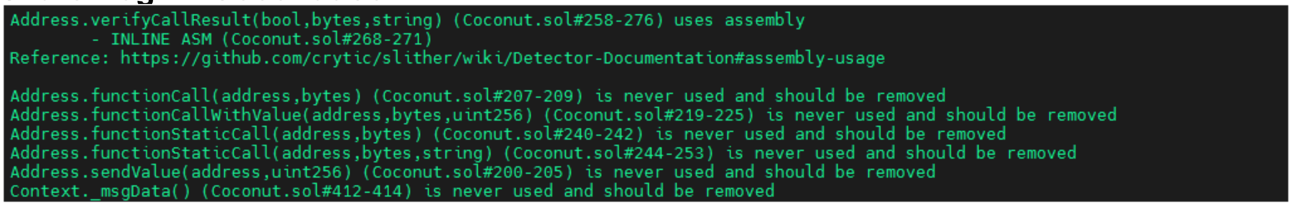


```
Guava.claim() (Guava.sol#436-445) uses arbitrary from in transferFrom: token.safeTransferFrom(wallet,msg.sender,amount) (Guava
.sol#444)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom

Guava.constructor(IERC20,address,address,uint256)._owner (Guava.sol#415) shadows:
        - Ownable._owner (Guava.sol#206) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Address.verifyCallResult(bool,bytes,string) (Guava.sol#91-109) uses assembly
        - INLINE ASM (Guava.sol#101-104)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address.functionCall(address,bytes) (Guava.sol#40-42) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Guava.sol#52-58) is never used and should be removed
Address.functionStaticCall(address,bytes) (Guava.sol#73-75) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (Guava.sol#77-86) is never used and should be removed
Address.sendValue(address,uint256) (Guava.sol#33-38) is never used and should be removed
Context._msgData() (Guava.sol#201-203) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (Guava.sol#140-153) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (Guava.sol#164-175) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (Guava.sol#155-162) is never used and should be removed
SafeERC20.safeTransfer(IERC20,address,uint256) (Guava.sol#116-122) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.18 (Guava.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.
16
solc-0.8.18 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (Guava.sol#33-38):
        - (success) = recipient.call{value: amount}() (Guava.sol#36)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Guava.sol#60-71):
        - (success,returndata) = target.call{value: value}(data) (Guava.sol#69)
Low level call in Address.functionStaticCall(address,bytes,string) (Guava.sol#77-86):
        - (success,returndata) = target.staticcall(data) (Guava.sol#84)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

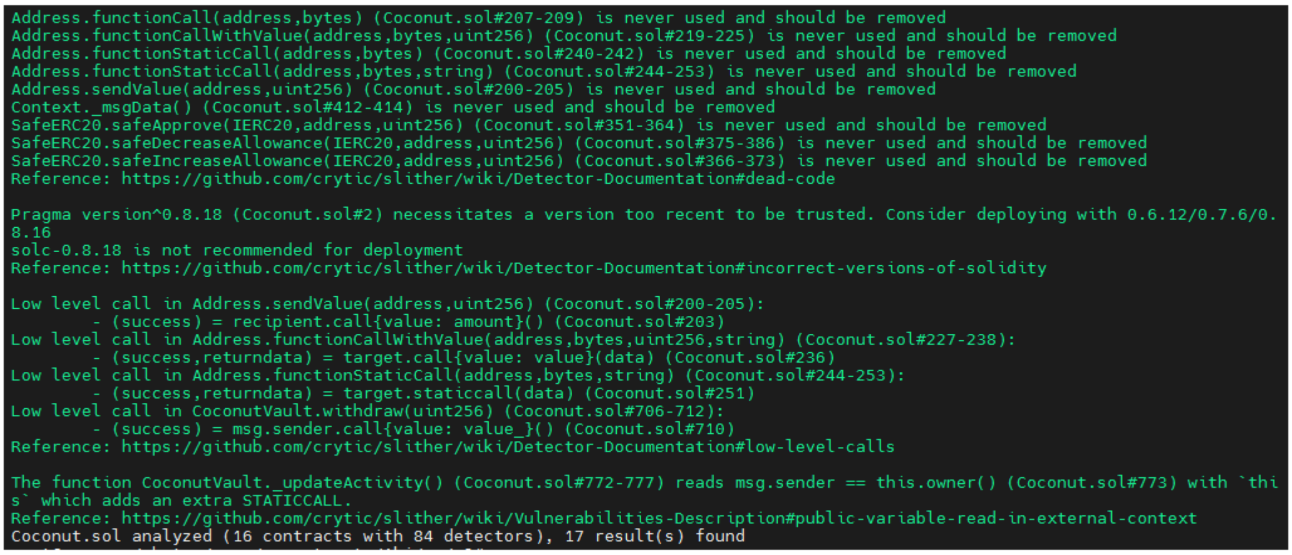


```
Parameter Guava.changeWallet(address)._wallet (Guava.sol#464) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
Guava.sol analyzed (8 contracts with 84 detectors), 19 result(s) found
```

Slither log >> Coconut.sol



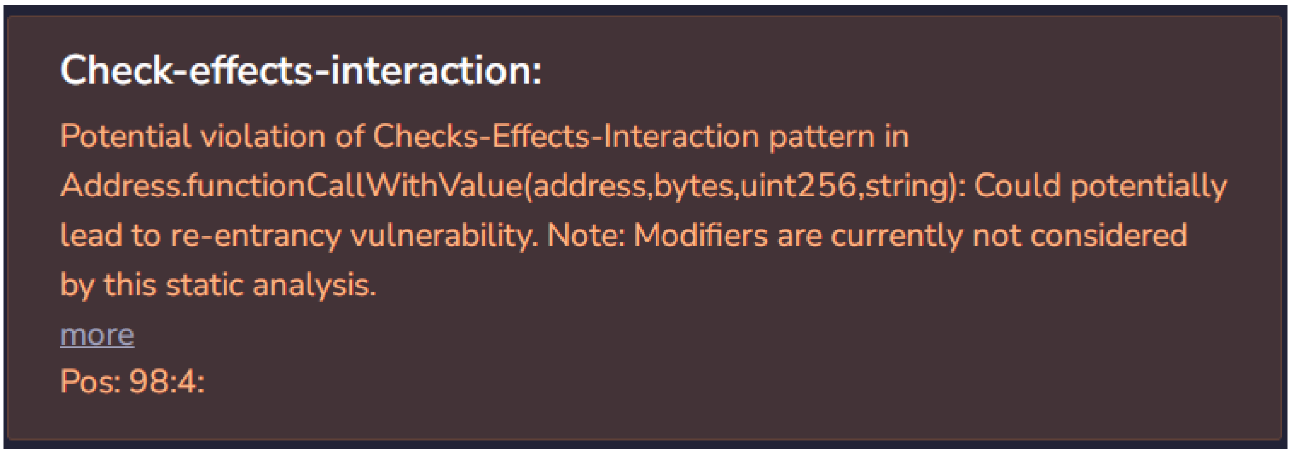
```
Address.verifyCallResult(bool,bytes,string) (Coconut.sol#258-276) uses assembly
        - INLINE ASM (Coconut.sol#268-271)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address.functionCall(address,bytes) (Coconut.sol#207-209) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Coconut.sol#219-225) is never used and should be removed
Address.functionStaticCall(address,bytes) (Coconut.sol#240-242) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (Coconut.sol#244-253) is never used and should be removed
Address.sendValue(address,uint256) (Coconut.sol#200-205) is never used and should be removed
Context._msgData() (Coconut.sol#412-414) is never used and should be removed
```



```
Address.functionCall(address,bytes) (Coconut.sol#207-209) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Coconut.sol#219-225) is never used and should be removed
Address.functionStaticCall(address,bytes) (Coconut.sol#240-242) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (Coconut.sol#244-253) is never used and should be removed
Address.sendValue(address,uint256) (Coconut.sol#200-205) is never used and should be removed
Context._msgData() (Coconut.sol#412-414) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (Coconut.sol#351-364) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (Coconut.sol#375-386) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (Coconut.sol#366-373) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.18 (Coconut.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.
8.16
solc-0.8.18 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (Coconut.sol#200-205):
        - (success) = recipient.call{value: amount}() (Coconut.sol#203)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Coconut.sol#227-238):
        - (success,returndata) = target.call{value: value}(data) (Coconut.sol#236)
Low level call in Address.functionStaticCall(address,bytes,string) (Coconut.sol#244-253):
        - (success,returndata) = target.staticcall(data) (Coconut.sol#251)
Low level call in CoconutVault.withdraw(uint256) (Coconut.sol#706-712):
        - (success) = msg.sender.call{value: value_}() (Coconut.sol#710)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

The function CoconutVault._updateActivity() (Coconut.sol#772-777) reads msg.sender == this.owner() (Coconut.sol#773) with `thi
s` which adds an extra STATICCALL.
Reference: https://github.com/crytic/slither/wiki/Vulnerabilities-Description#public-variable-read-in-external-context
Coconut.sol analyzed (16 contracts with 84 detectors), 17 result(s) found
```

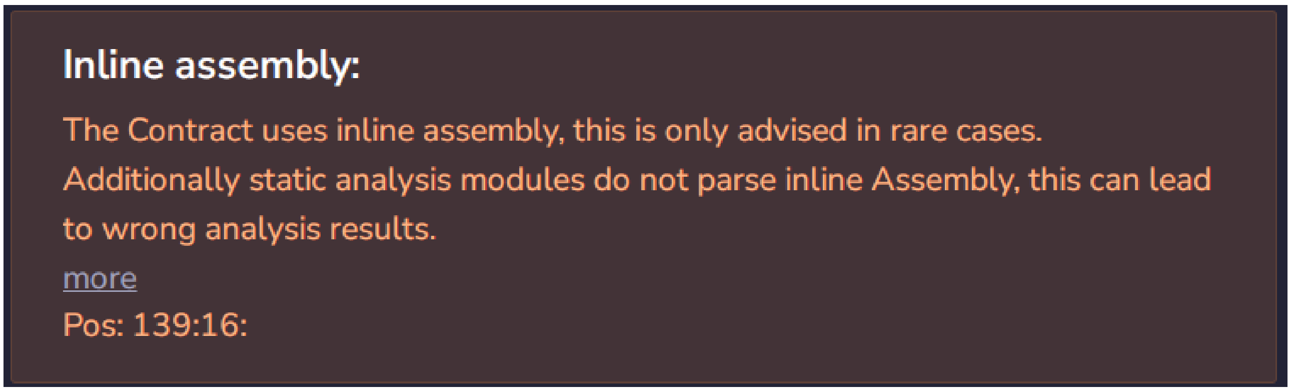
Solidity Static Analysis

Bubblegum.sol



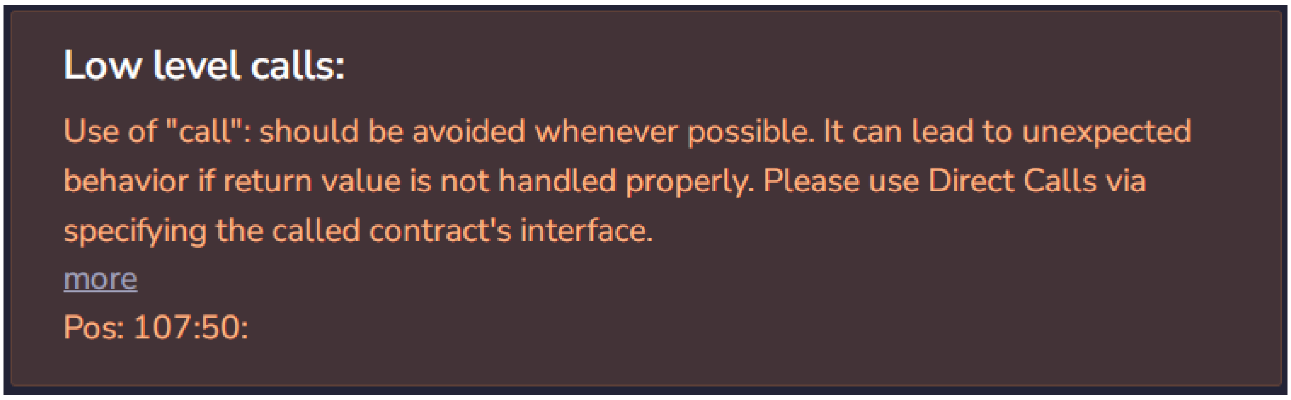
Check-effects-interaction:

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: 98:4:



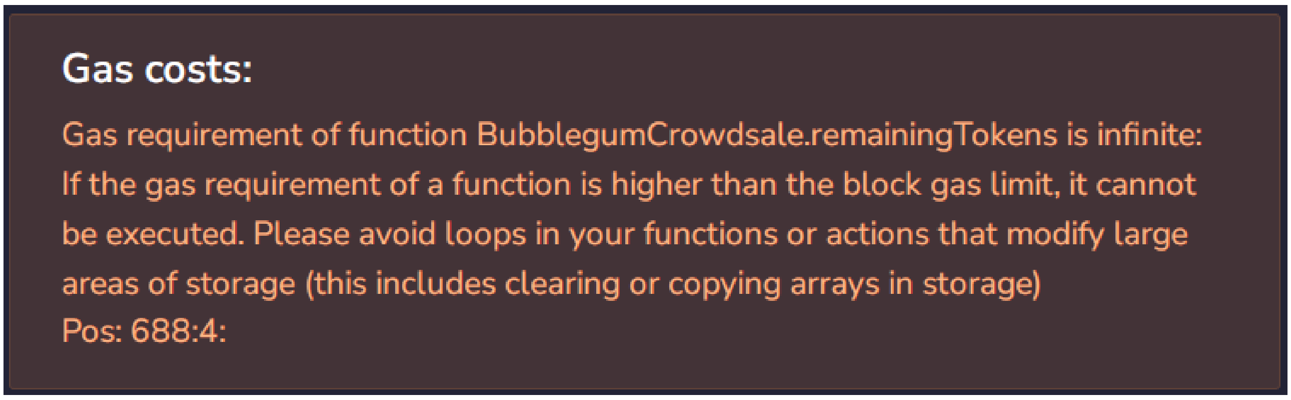
Inline assembly:

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: 139:16:



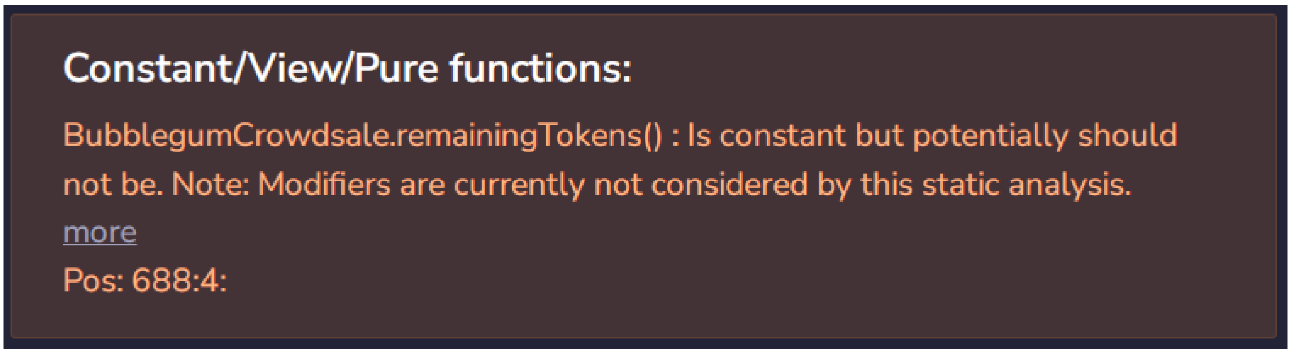
Low level calls:

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: 107:50:

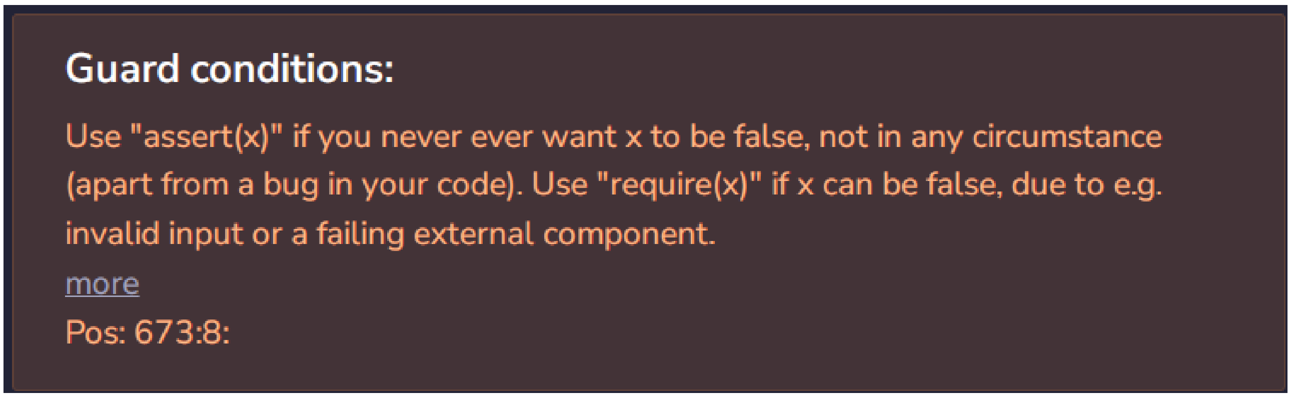


Gas costs:

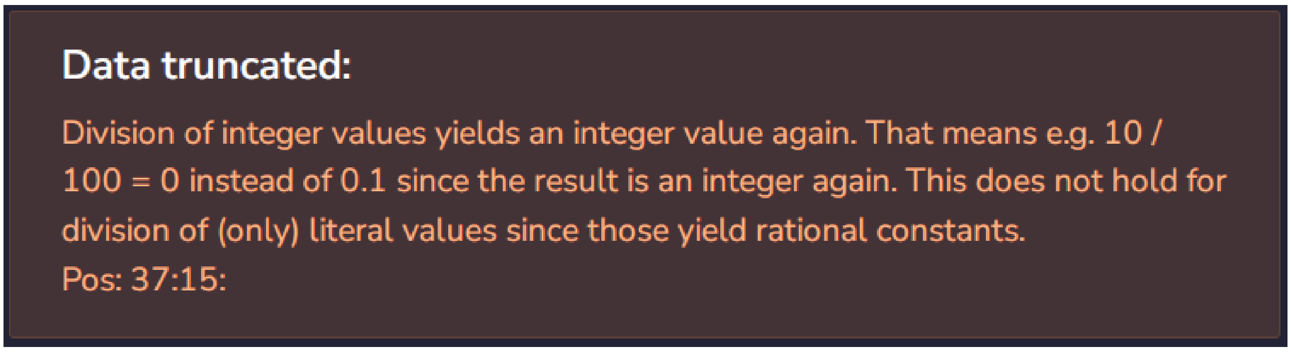
Gas requirement of function BubblegumCrowdsale.remainingTokens is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 688:4:


Constant/View/Pure functions:

BubblegumCrowdsale.remainingTokens() : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 688:4:

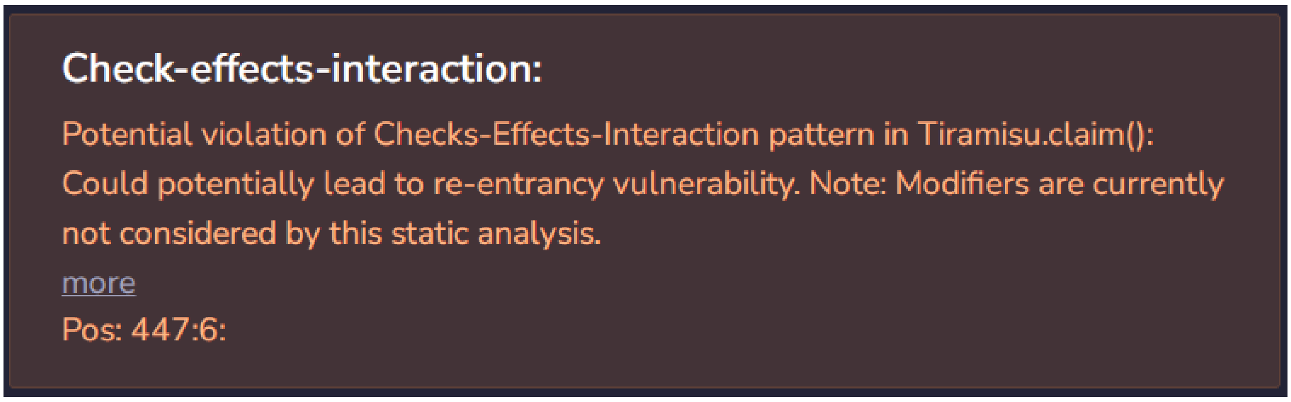

Guard conditions:

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: 673:8:

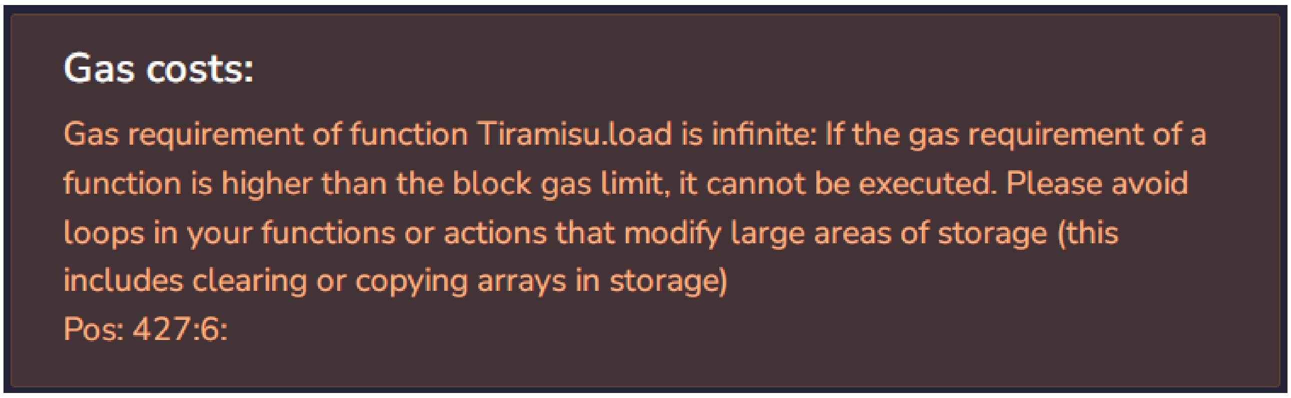

Data truncated:

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: 37:15:

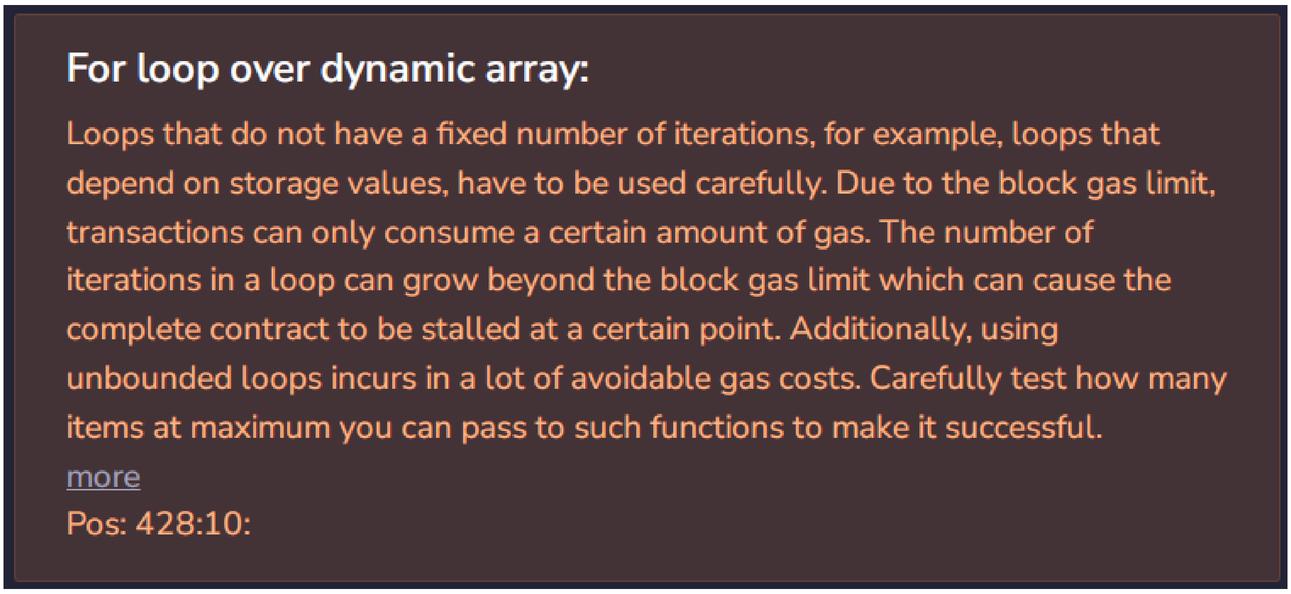
Tiramisu.sol


Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Tiramisu.claim(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 447:6:


Gas costs:

Gas requirement of function Tiramisu.load 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: 427:6:

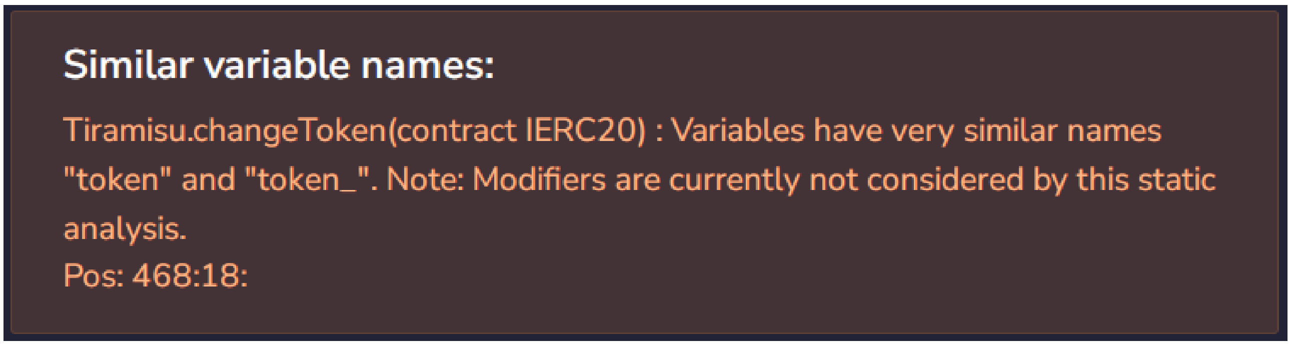


For loop over dynamic array:

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

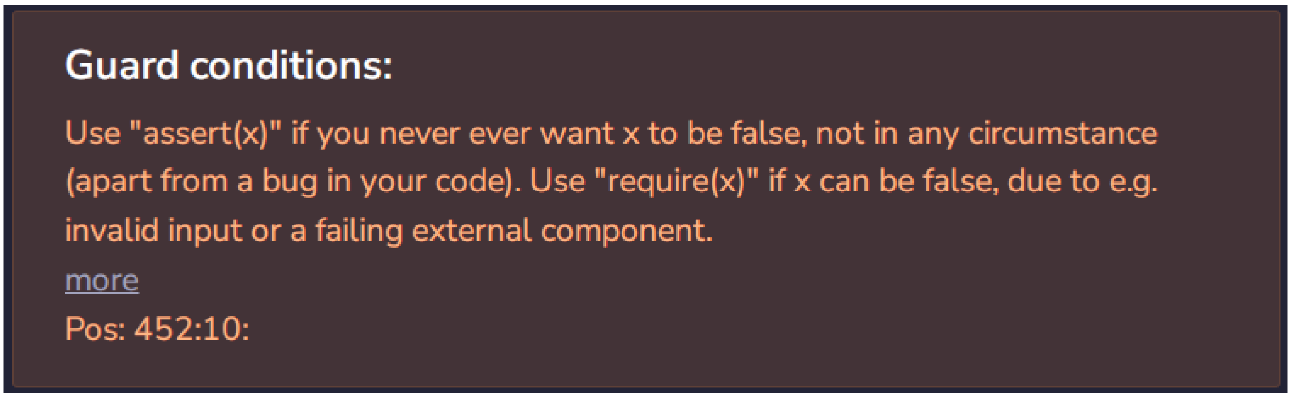
Pos: 428:10:



Similar variable names:

Tiramisu.changeToken(contract IERC20) : Variables have very similar names "token" and "token_". Note: Modifiers are currently not considered by this static analysis.

Pos: 468:18:



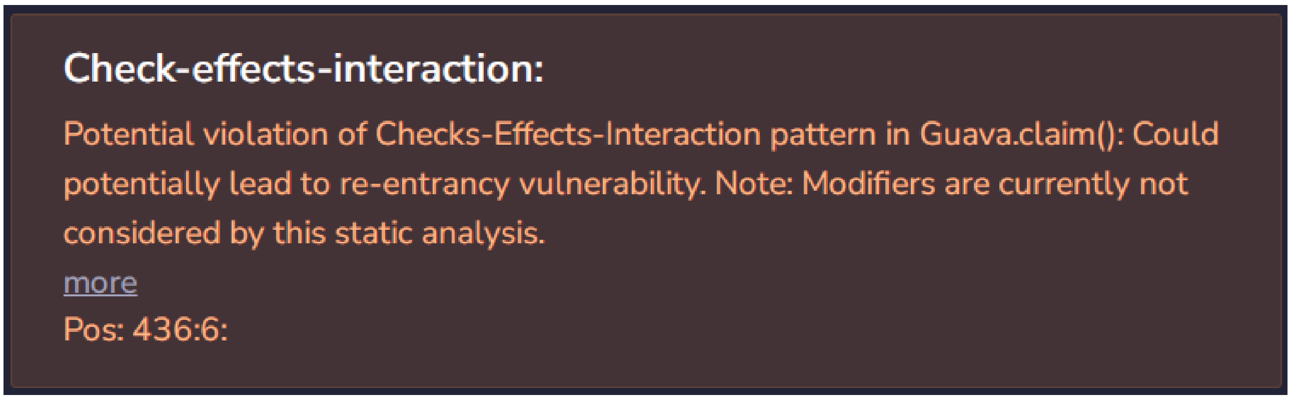
Guard conditions:

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: 452:10:

Guava.sol

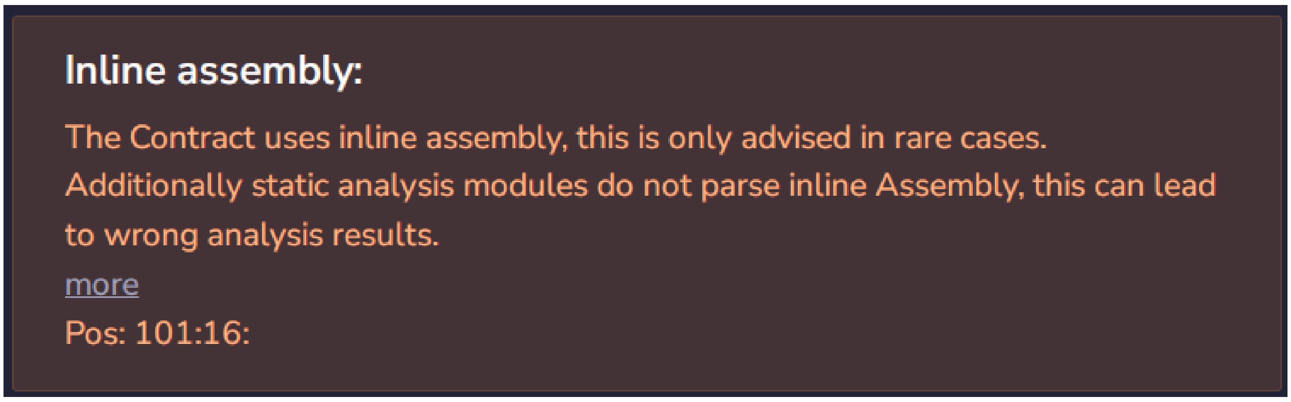


Check-effects-interaction:

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

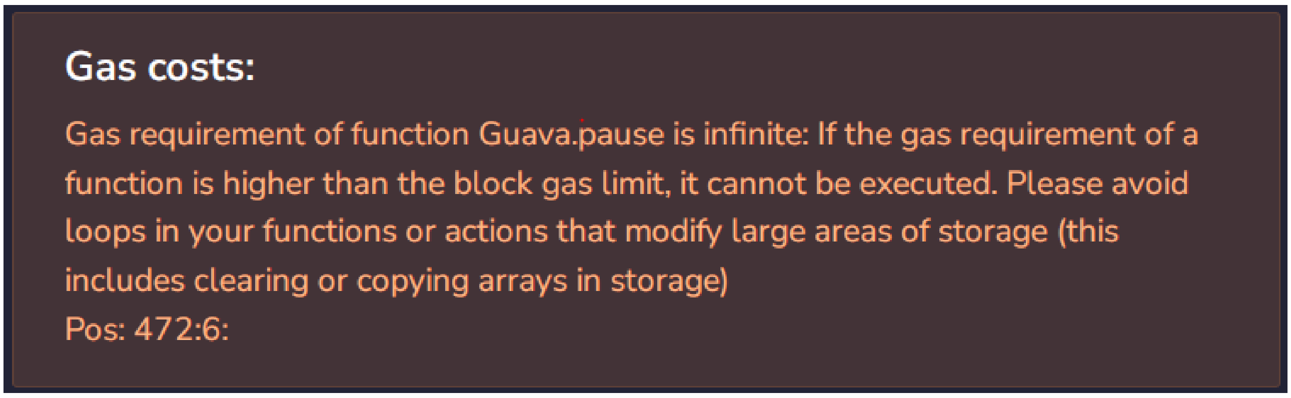
more

Pos: 436:6:



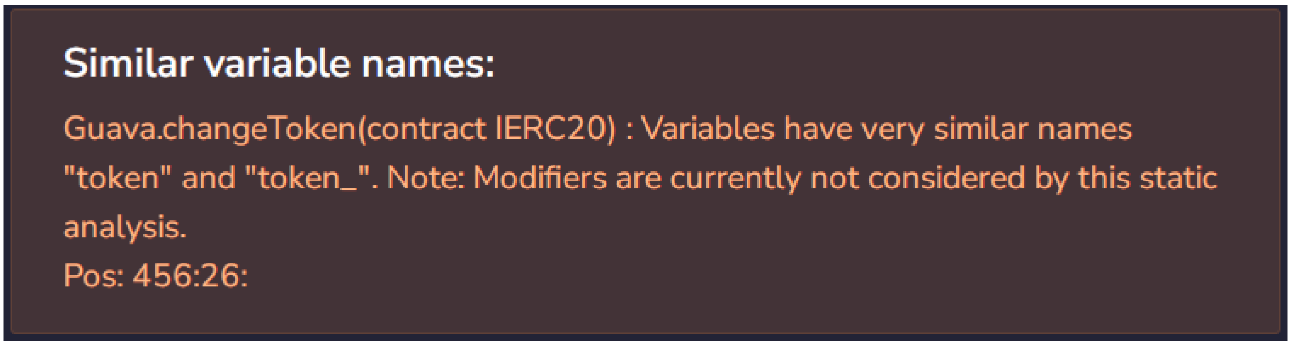
Inline assembly:

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: 101:16:



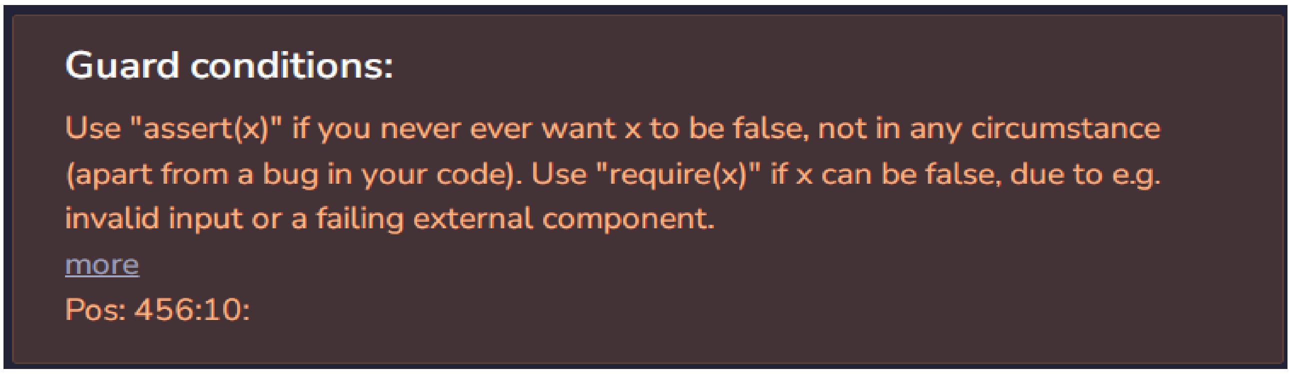
Gas costs:

Gas requirement of function Guava.pause 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: 472:6:



Similar variable names:

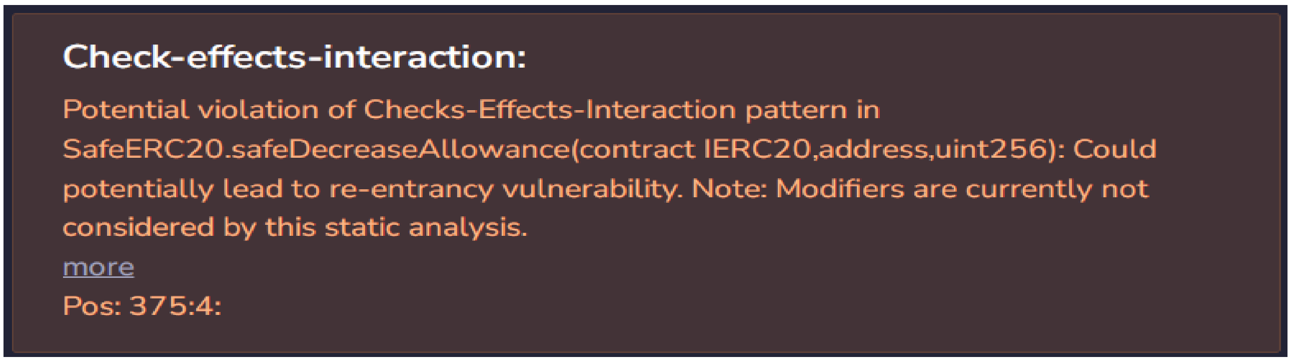
Guava.changeToken(contract IERC20) : Variables have very similar names "token" and "token_". Note: Modifiers are currently not considered by this static analysis.
Pos: 456:26:



Guard conditions:

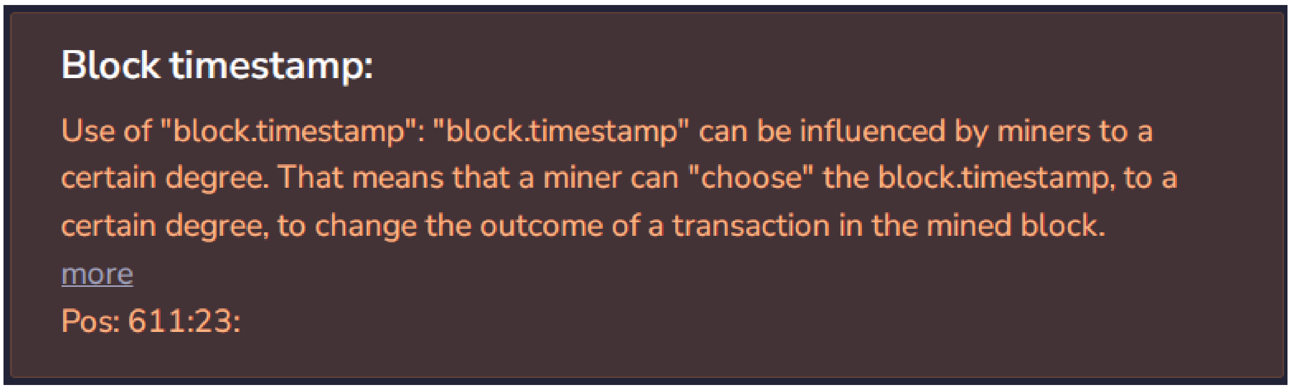
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 456:10:

Coconut.sol



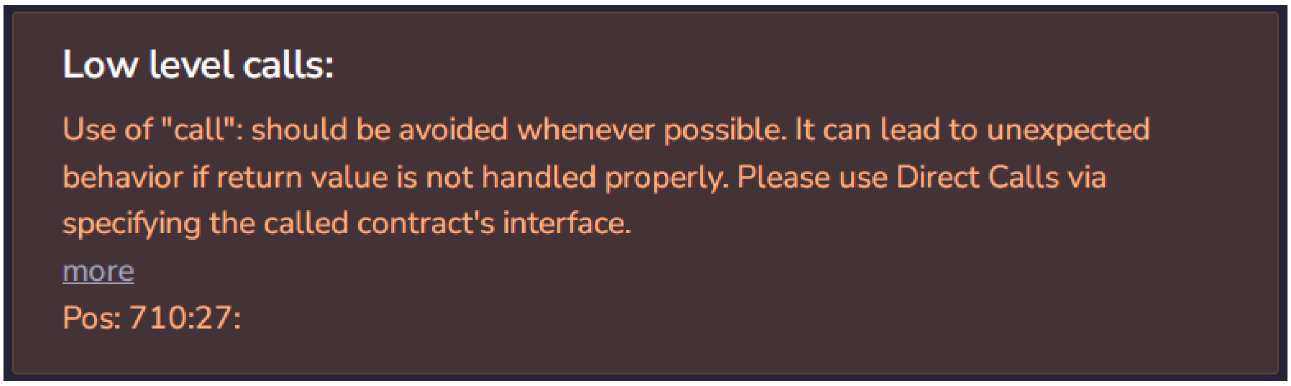
Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in SafeERC20.safeDecreaseAllowance(contract IERC20,address,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 375:4:



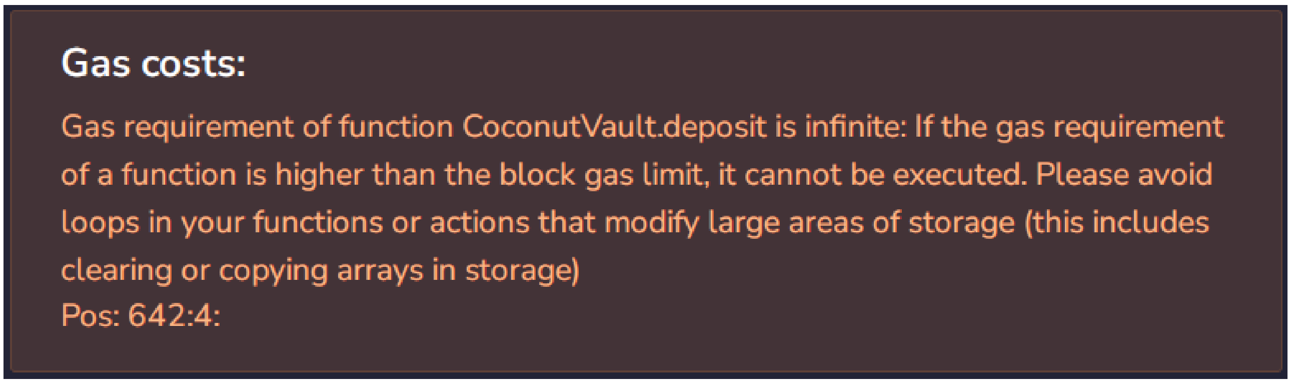
Block timestamp:

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: 611:23:



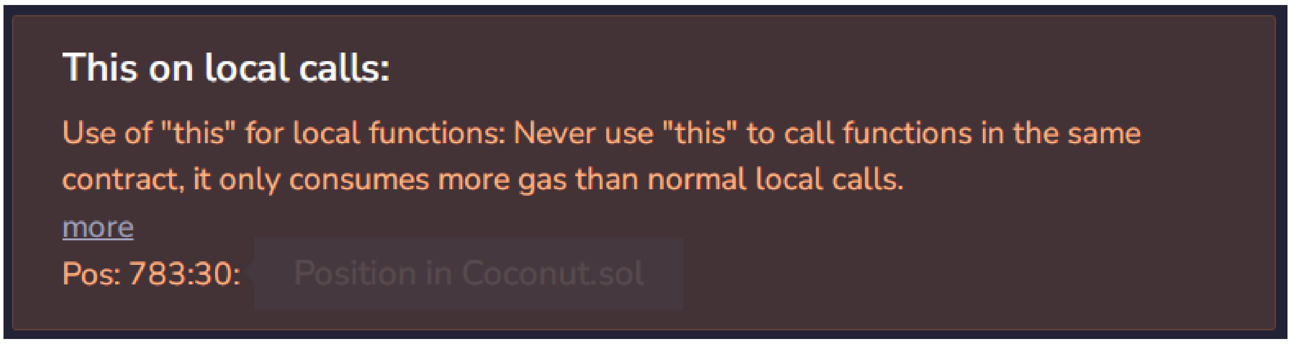
Low level calls:

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: 710:27:



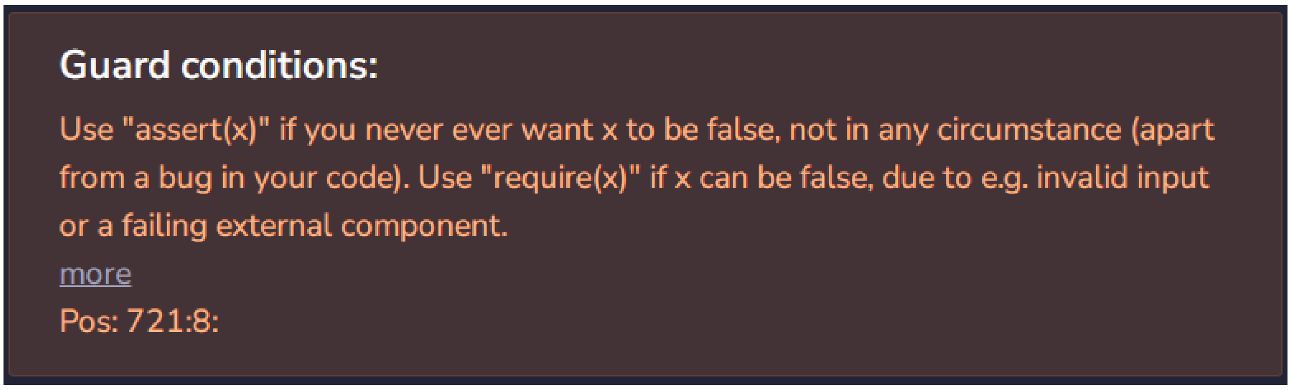
Gas costs:

Gas requirement of function CoconutVault.deposit 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: 642:4:



This on local calls:

Use of "this" for local functions: Never use "this" to call functions in the same contract, it only consumes more gas than normal local calls.
more
Pos: 783:30:

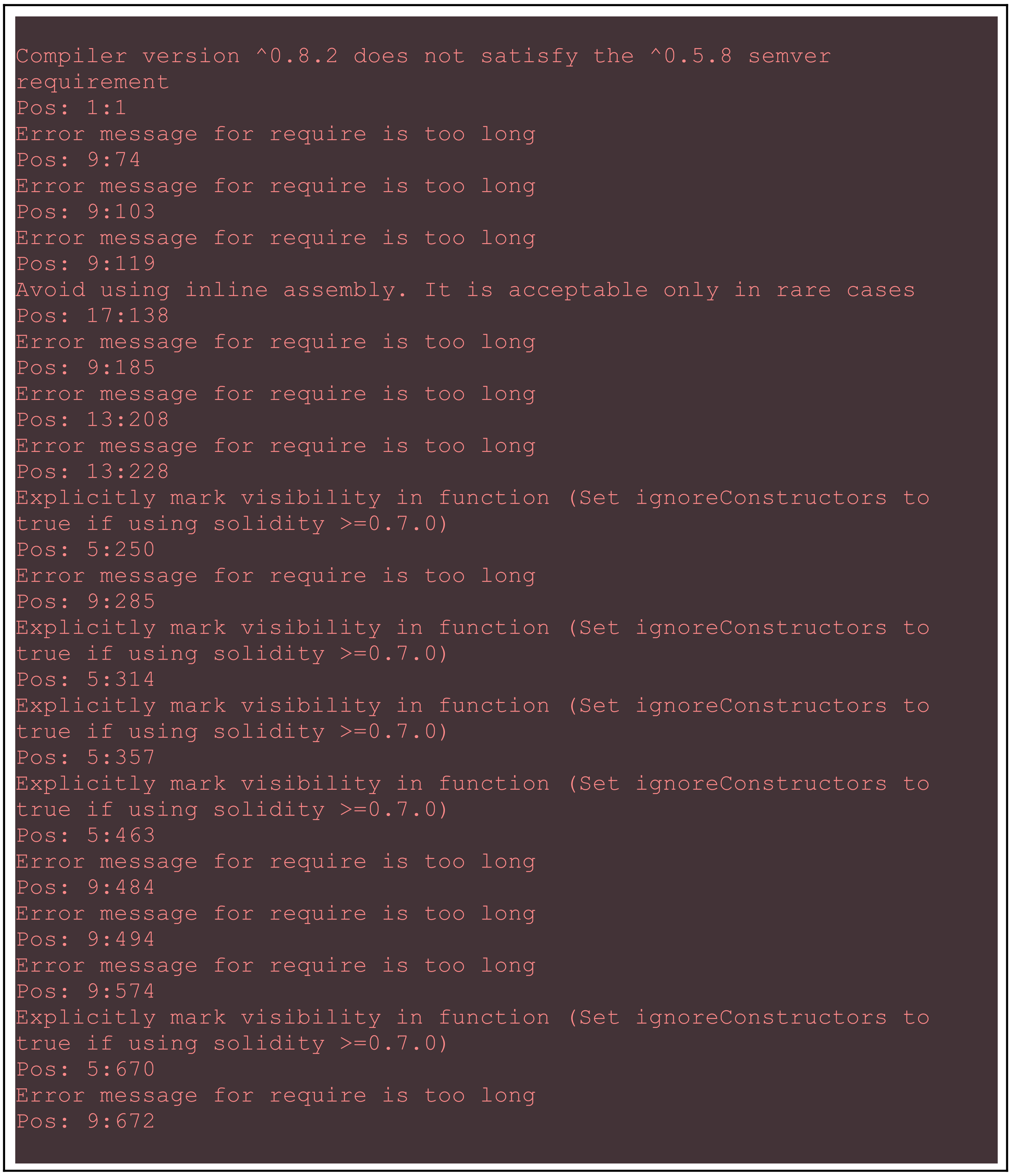


Guard conditions:

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: 721:8:

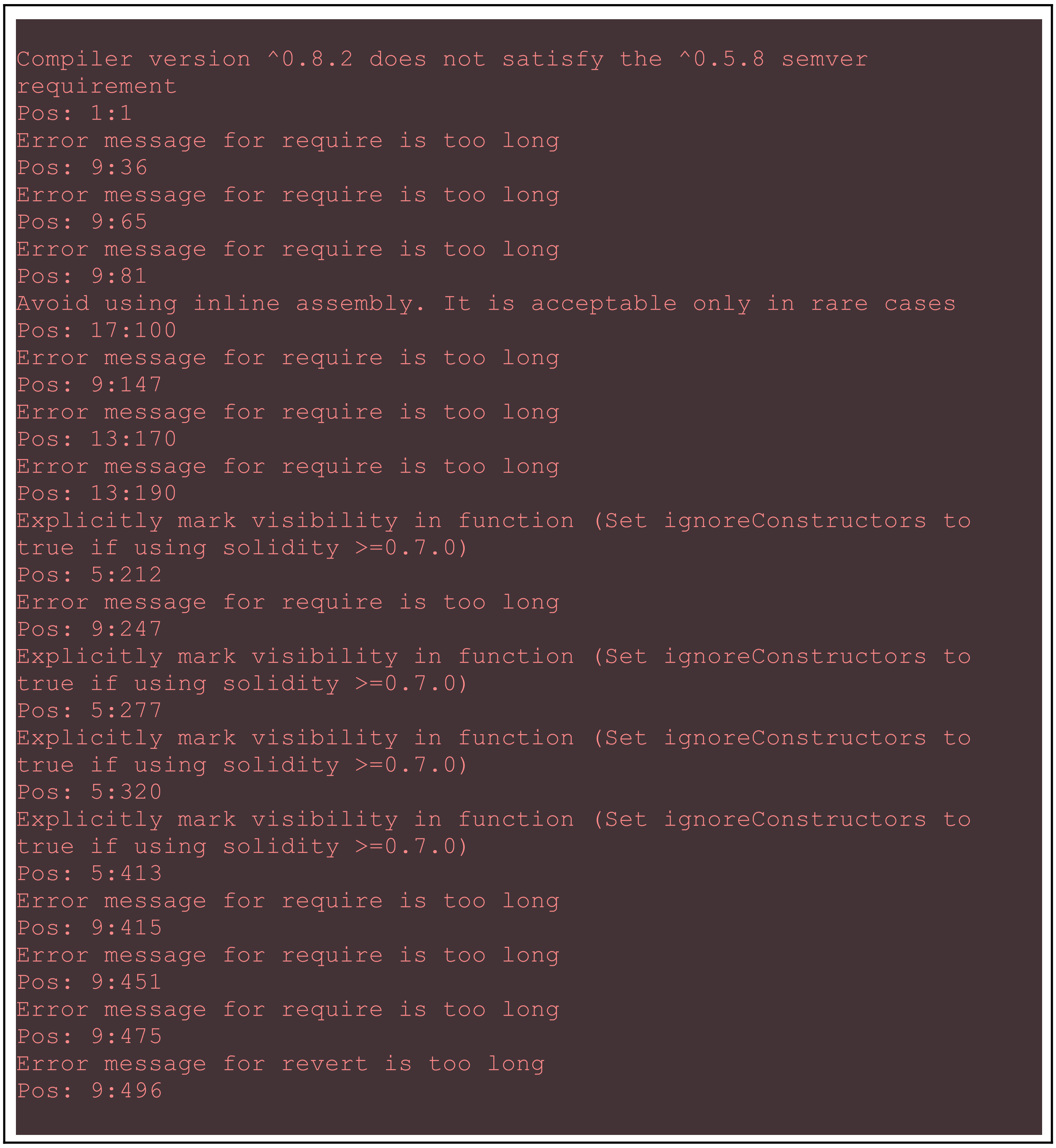
Solhint Linter

Bubblegum.sol



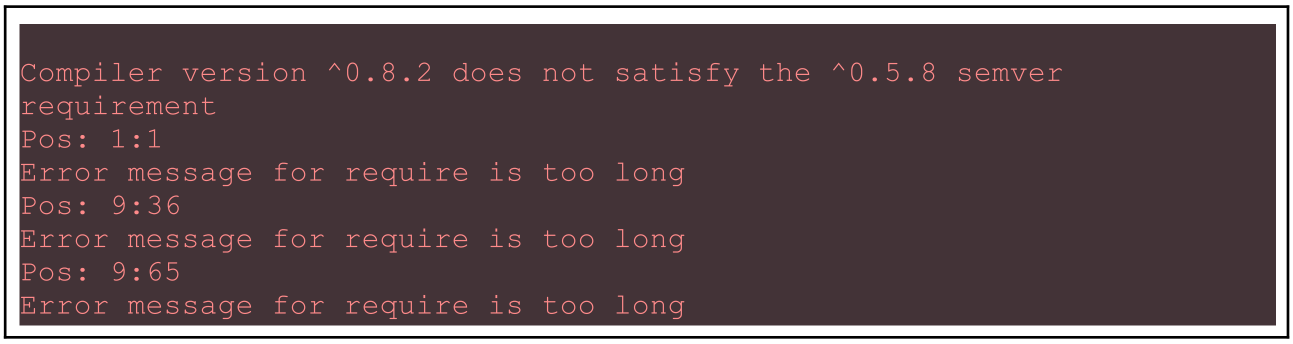
```
Compiler version ^0.8.2 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Error message for require is too long
Pos: 9:74
Error message for require is too long
Pos: 9:103
Error message for require is too long
Pos: 9:119
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 17:138
Error message for require is too long
Pos: 9:185
Error message for require is too long
Pos: 13:208
Error message for require is too long
Pos: 13:228
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:250
Error message for require is too long
Pos: 9:285
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:314
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:357
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:463
Error message for require is too long
Pos: 9:484
Error message for require is too long
Pos: 9:494
Error message for require is too long
Pos: 9:574
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:670
Error message for require is too long
Pos: 9:672
```

Tiramisu.sol

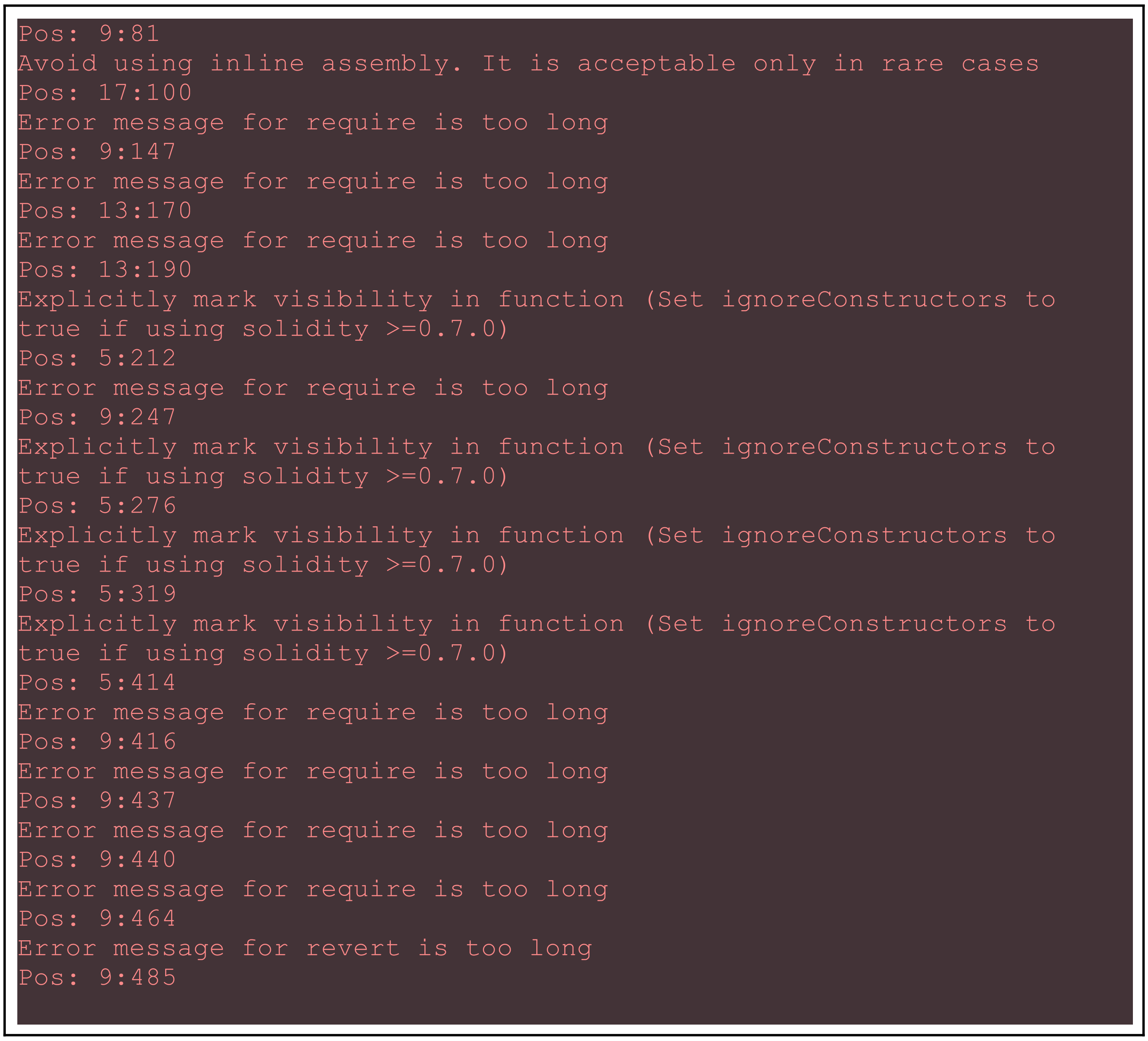


```
Compiler version ^0.8.2 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Error message for require is too long
Pos: 9:36
Error message for require is too long
Pos: 9:65
Error message for require is too long
Pos: 9:81
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 17:100
Error message for require is too long
Pos: 9:147
Error message for require is too long
Pos: 13:170
Error message for require is too long
Pos: 13:190
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:212
Error message for require is too long
Pos: 9:247
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:277
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:320
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:413
Error message for require is too long
Pos: 9:415
Error message for require is too long
Pos: 9:451
Error message for require is too long
Pos: 9:475
Error message for revert is too long
Pos: 9:496
```

Guava.sol

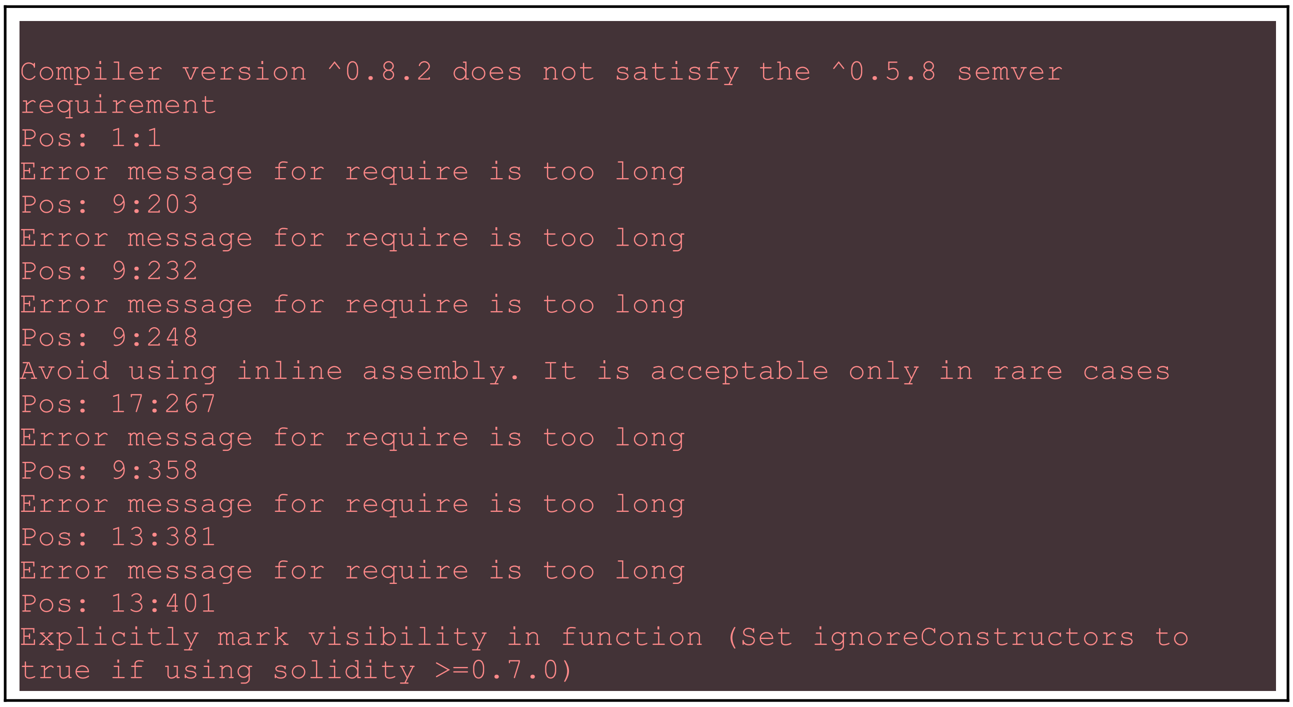


```
Compiler version ^0.8.2 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Error message for require is too long
Pos: 9:36
Error message for require is too long
Pos: 9:65
Error message for require is too long
```

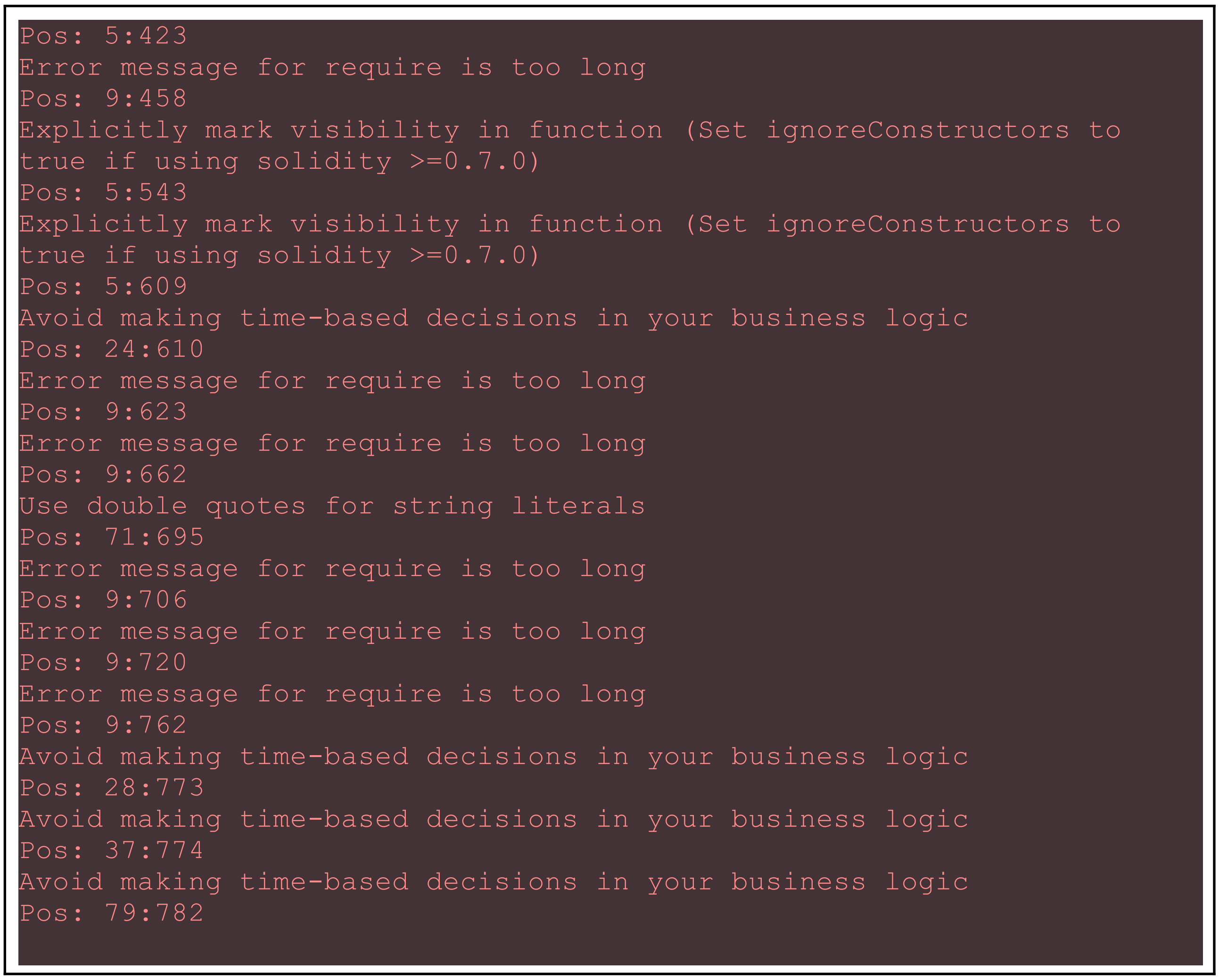


```
Pos: 9:81
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 17:100
Error message for require is too long
Pos: 9:147
Error message for require is too long
Pos: 13:170
Error message for require is too long
Pos: 13:190
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:212
Error message for require is too long
Pos: 9:247
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:276
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:319
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:414
Error message for require is too long
Pos: 9:416
Error message for require is too long
Pos: 9:437
Error message for require is too long
Pos: 9:440
Error message for require is too long
Pos: 9:464
Error message for revert is too long
Pos: 9:485
```

Coconut.sol



```
Compiler version ^0.8.2 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
Error message for require is too long
Pos: 9:203
Error message for require is too long
Pos: 9:232
Error message for require is too long
Pos: 9:248
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 17:267
Error message for require is too long
Pos: 9:358
Error message for require is too long
Pos: 13:381
Error message for require is too long
Pos: 13:401
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
```


```
Pos: 5:423
Error message for require is too long
Pos: 9:458
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:543
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:609
Avoid making time-based decisions in your business logic
Pos: 24:610
Error message for require is too long
Pos: 9:623
Error message for require is too long
Pos: 9:662
Use double quotes for string literals
Pos: 71:695
Error message for require is too long
Pos: 9:706
Error message for require is too long
Pos: 9:720
Error message for require is too long
Pos: 9:762
Avoid making time-based decisions in your business logic
Pos: 28:773
Avoid making time-based decisions in your business logic
Pos: 37:774
Avoid making time-based decisions in your business logic
Pos: 79:782
```

Software analysis result:

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