Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	LAToken
Website:	latoken.com
Platform:	Ethereum
Language:	Solidity
Date:	May 23rd, 2024

Table of contents

Introduction ………………………………………………………………………………… 4
Project Background ……………………………………………………………………………4
Audit Scope ………………………………………………………………………………… 4
Claimed Smart Contract Features …………………………………………………………….. 5
Audit Summary ……………..…………………………………………………………………….6
Technical Quick Stats …..…………………………………………………………………… 7
Business Risk Analysis …..………………………………………………………………… 8
Code Quality ………………………………………………………………………………… 9
Documentation ……………………………………………………………………………….. 9
Use of Dependencies …………………………………………………………………………9
AS-IS overview ……………………………………………………………………………….. 10
Severity Definitions ……………………………………………………………………………12
Audit Findings ……………………………………………………………………………… 13
Conclusion ………………………………………………………………………………… 17
Our Methodology ……………………………………………………………………………... 18
Disclaimers ………………………………………………………………………………… 20
Appendix

- Code Flow Diagram ……………………………………………………………….... 21
- Slither Results Log ……………………………………………………………………. 22
- Solidity static analysis ……………………………………………………………….. 25

`

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the LAToken smart contract from latoken.com was audited extensively. 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 May 23rd, 2024.

The purpose of this audit was to address the following:

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

Project Background

- This smart contract suite is designed to implement a token called `LAToken` which follows the ERC-20 standard and includes additional functionality for token issuance, burning, and exchanging old tokens for new ones. The suite also includes a `LATokenMinter` contract to manage the distribution of tokens to the team over time.
- This smart contract suite is a comprehensive implementation for managing a token lifecycle, including creation, distribution, migration, and burning, with robust access controls and safety mechanisms.
- The contract is without any other custom functionality and without any ownership control, which makes it truly decentralized.

Audit scope

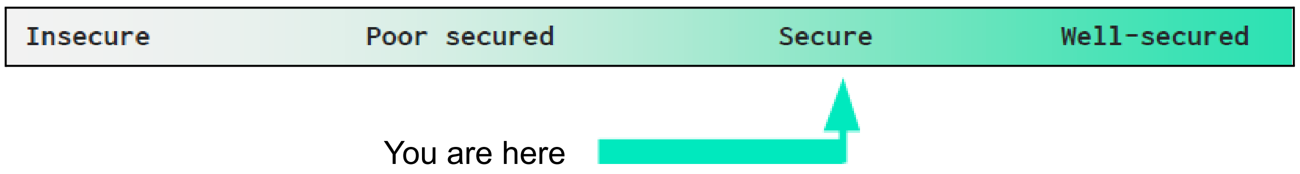
Name	**Code Review and Security Analysis Report for LAToken Smart Contract**
Platform	**Ethereum**
Language	**Solidity**
File	LATToken.sol
Smart Contract Code	0xe50365f5d679cb98a1dd62d6f6e58e59321bcddf
Audit Date	May 23rd, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Name: LAToken ● Symbol: LAToken ● Decimals: 18 ● Total Supply: 400 million	**YES, This is valid.**
Ownership Control: ● There are no owner functions, which makes it 100% decentralized.	**YES, This is valid.**
Founder Control: ● The founder can send 400 million tokens to the team pool at the token sale end. ● The current Founder can set a new Founder address. ● The founder can update a new helper address, team pool address, and frozen address. ● Update a new conversion rate.	**YES, This is valid.**
Helper Control: ● The harvest method which will be called each day after 5 years to get unfrozen tokens can be called by the Helper.	**YES, This is valid.**
Minter And Exchanger Control: ● Minter And Exchanger can burn tokens.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **"Secured"**. This token contract does not have any ownership control, hence it is **100% decentralized**.



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

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

We found 0 critical, 0 high, 0 medium, 2 low, and 6 very low level issues.

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

Technical Quick Stats

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

Overall Audit Result: PASSED

Business Risk Analysis

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

Overall Audit Result: PASSED

Code Quality

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

The libraries in LAToken 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 LAToken.

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

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

Documentation

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

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

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure 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

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Function with the same name as the contract	Refer Audit Findings
2	onlyFounder	modifier	Passed	No Issue
3	onlyHelper	modifier	Passed	No Issue
4	fundTeamInstant	external	Missing required error message	Refer Audit Findings
5	changeTokenAddress	external	Critical operation lacks event log, Missing zero address validation	Refer Audit Findings
6	changeFounder	external	Critical operation lacks event log, Missing zero address validation	Refer Audit Findings
7	changeHelper	external	Critical operation lacks event log, Missing zero address validation	Refer Audit Findings
8	changeTeamPoolInstant	external	access only Founder	No Issue
9	changeTeamPoolForFrozen Tokens	external	access only Founder	No Issue
10	harvest	external	Missing required error message	Refer Audit Findings
11	LATokenMinter	write	Missing zero address validation	Refer Audit Findings
12	getBlockTimestamp	write	Passed	No Issue
13	onlyFounder	modifier	Passed	No Issue
14	onlyPreviousToken	modifier	Passed	No Issue
15	changeCourse	write	access only Founder	No Issue
16	exchange	write	access only Previous Token	No Issue
17	changeFounder	external	Critical operation lacks an event log,	Refer Audit Findings
18	ExchangeContract	write	Missing zero address validation	Refer Audit Findings
19	onlyFounder	modifier	Passed	No Issue
20	onlyMinterAndExchanger	modifier	Passed	No Issue
21	transfer	write	Passed	No Issue
22	issueTokens	external	access only Minter And Exchanger	No Issue
23	burnTokens	external	Critical operation lacks event log	Refer Audit Findings

24	changeMinter	write	Critical operation lacks event log, Missing zero address validation	Refer Audit Findings
25	changeFounder	write	Critical operation lacks event log, Missing zero address validation	Refer Audit Findings
26	changeExchanger	write	Critical operation lacks event log, Missing zero address validation	Refer Audit Findings
27	LATToken	write	Passed	No Issue
28	transfer	write	Passed	No Issue
29	transferFrom	write	Passed	No Issue
30	balanceOf	write	Passed	No Issue
31	approve	write	Passed	No Issue
32	approveAndCall	write	Passed	No Issue
33	allowance	write	Passed	No Issue
34	balanceOf	write	Passed	No Issue
35	transfer	write	Passed	No Issue
36	transferFrom	write	Passed	No Issue
37	approve	write	Passed	No Issue
38	approveAndCall	write	Passed	No Issue
39	allowance	write	Passed	No Issue

Severity Definitions

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

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium Severity vulnerabilities were found.

Low

(1) Critical operation lacks event log:

Missing event log for:

- burnTokens
- changeMinter
- changeFounder
- changeExchanger
- changeFounder
- changeTokenAddress
- changeHelper

Resolution: Please write an event log for the listed events.

(2) Missing zero address validation:

Detect missing zero address validation:

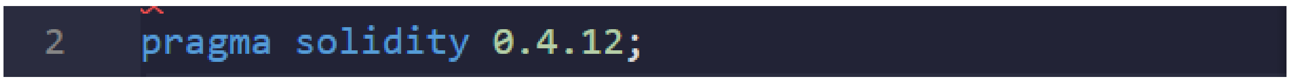
- changeMinter
- changeFounder
- changeExchanger
- changeFounder
- changeHelper
- ExchangeContract

- changeTokenAddress
- LATokenMinter

Resolution: We suggest adding zero address validation for the above function.

Very Low / Informational / Best practices:

(1) Use the latest solidity version:

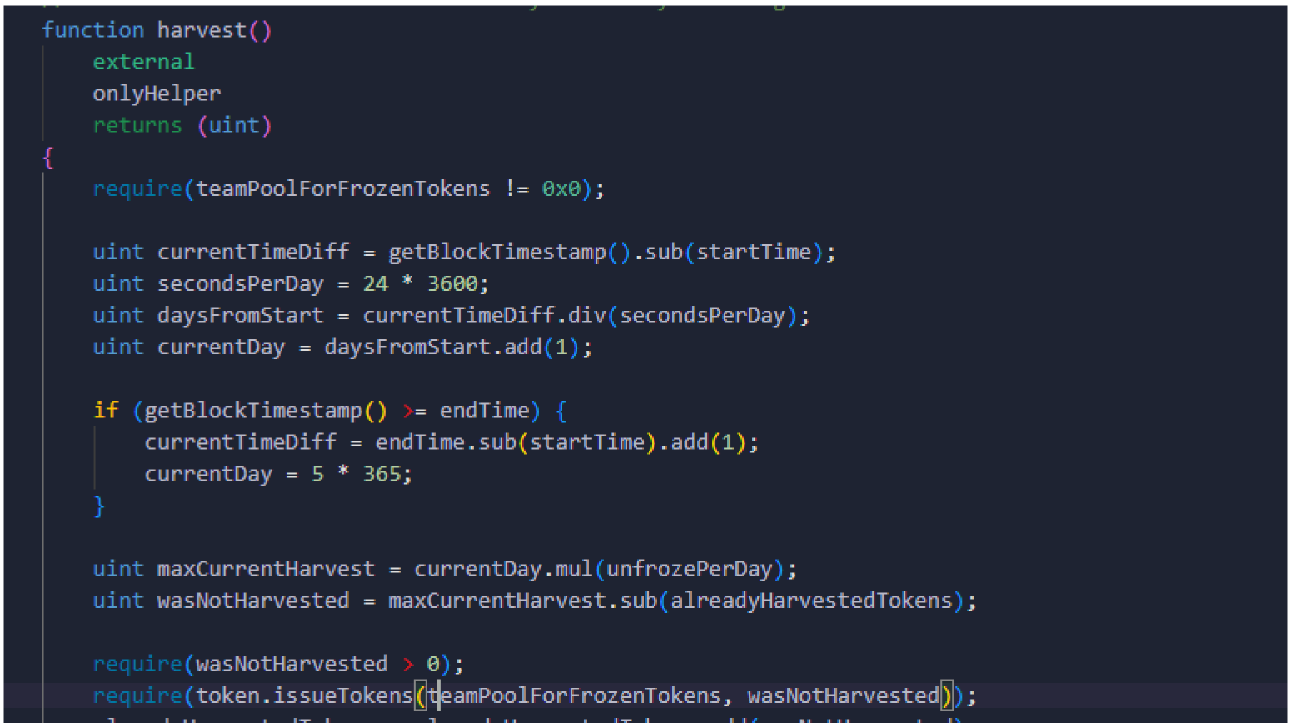


```
pragma solidity 0.4.12;
```

Using the latest solidity will prevent any compiler-level bugs.

Resolution: We suggest using the latest solidity compiler version.

(2) Missing required error message:



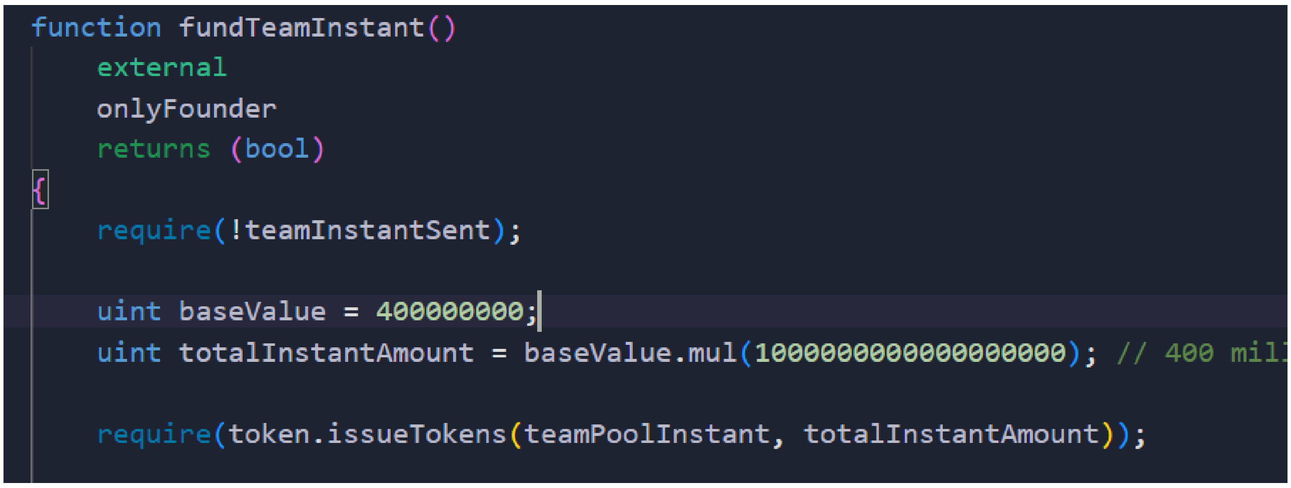
```
function harvest()
    external
    onlyHelper
    returns (uint)
{
    require(teamPoolForFrozenTokens != 0x0);

    uint currentTimeDiff = getBlockTimestamp().sub(startTime);
    uint secondsPerDay = 24 * 3600;
    uint daysFromStart = currentTimeDiff.div(secondsPerDay);
    uint currentDay = daysFromStart.add(1);

    if (getBlockTimestamp() >= endTime) {
        currentTimeDiff = endTime.sub(startTime).add(1);
        currentDay = 5 * 365;
    }

    uint maxCurrentHarvest = currentDay.mul(unfrozePerDay);
    uint wasNotHarvested = maxCurrentHarvest.sub(alreadyHarvestedTokens);

    require(wasNotHarvested > 0);
    require(token.issueTokens(teamPoolForFrozenTokens, wasNotHarvested));
```



```
function fundTeamInstant()
    external
    onlyFounder
    returns (bool)
{
    require(!teamInstantSent);

    uint baseValue = 400000000;
    uint totalInstantAmount = baseValue.mul(1000000000000000000); // 400 mil

    require(token.issueTokens(teamPoolInstant, totalInstantAmount));
```

Detecting missing errors in the whole require statement.

Resolution: We suggest providing a custom error message that will be displayed if the condition is not met. This can be helpful for debugging and understanding why a transaction failed.

(3) Function with the same name as the contract:

Defining constructors as functions with the same name as the contract is deprecated.

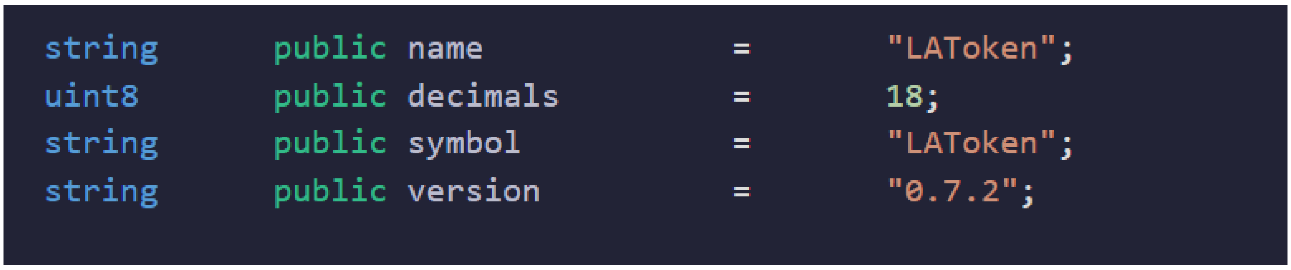
Resolution: Use "constructor(...) { ... }" instead.

(4) No visibility specified:

No visibility was specified. Defaulting to "public".

Resolution: Use public keywords as default.

(5) Declare variables constant:



```
string      public name         =       "LAToken";
uint8       public decimals     =       18;
string      public symbol       =       "LAToken";
string      public version      =       "0.7.2";
```

These variables' values will remain unchanged. so, we suggest making them constant. It is best practice and it also saves some gas. Just add a constant keyword.

Resolution: Please suggest making variables constant.

(6) Visibility can be external over the public:
Any functions which are not called internally should be declared as external. This saves some gas and is considered a good practice.
https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

Centralization Risk

The LAToken smart contract does not have any ownership control, **hence it is 100% decentralized.**

Therefore, there is **no** centralization risk.

Conclusion

We were given a contract code in the form of Etherscan web links. And we have used all possible tests based on given objects as files. We observed 2 low and 6 Informational issues in the smart contracts. but those are not critical. So, **it's good to go for the production**.

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

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

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

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

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the 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 white box penetration testing. We look at the project's website 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, and 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).

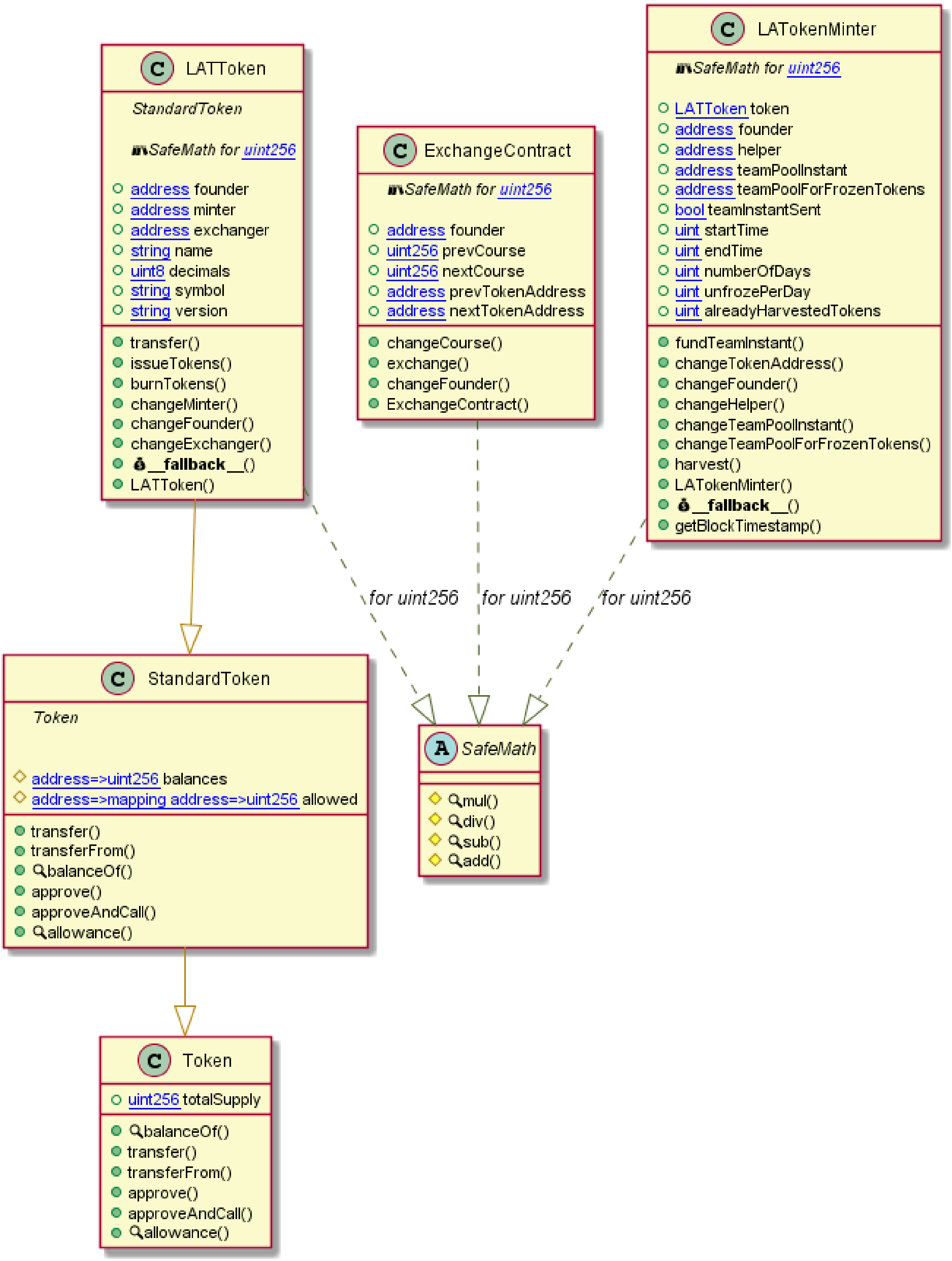
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While 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 the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - LAToken

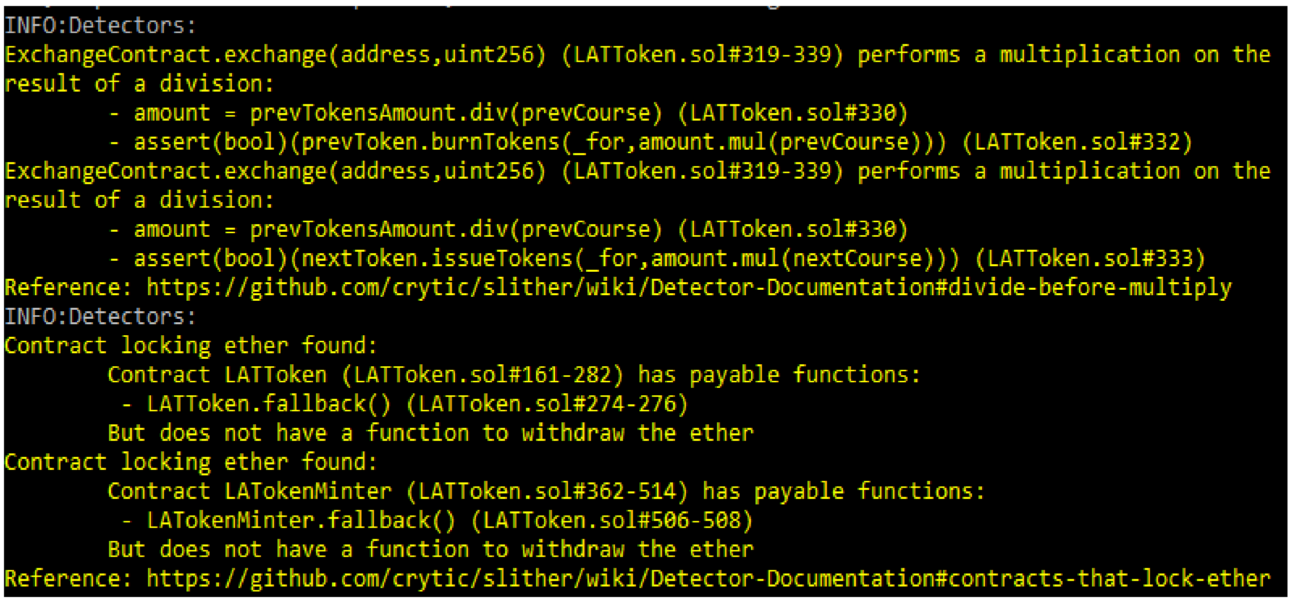


Slither Results Log

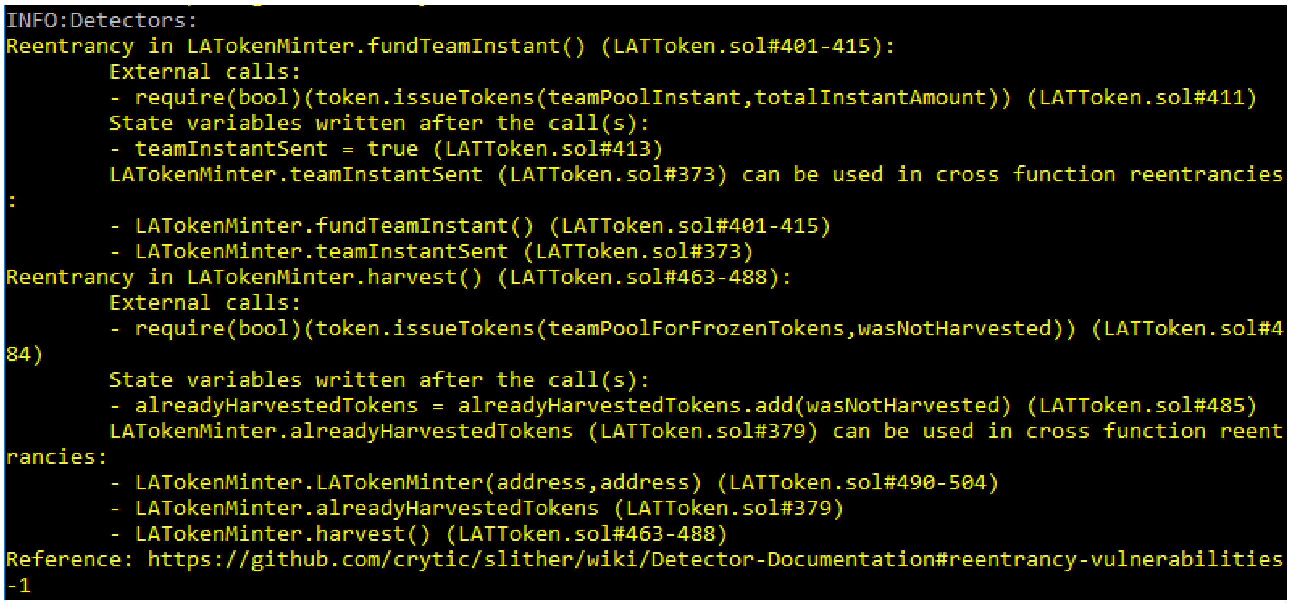
Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report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

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

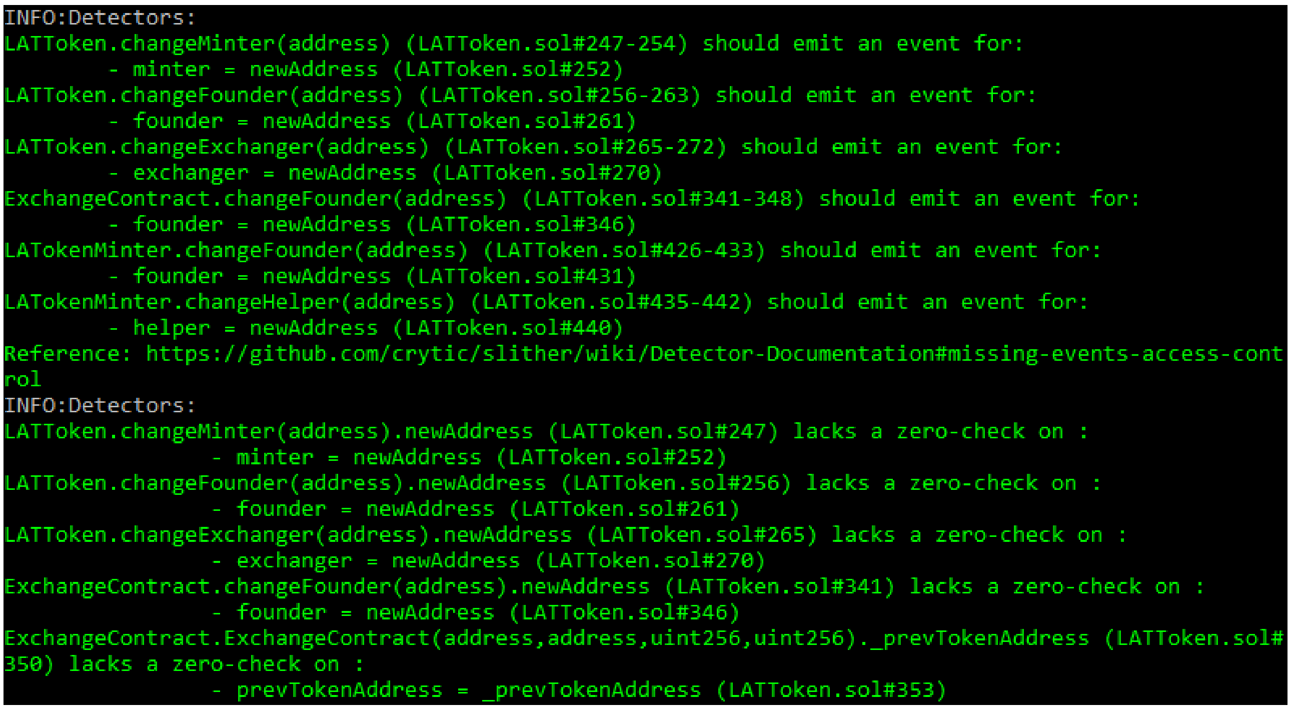
Slither Log >> LATToken.sol



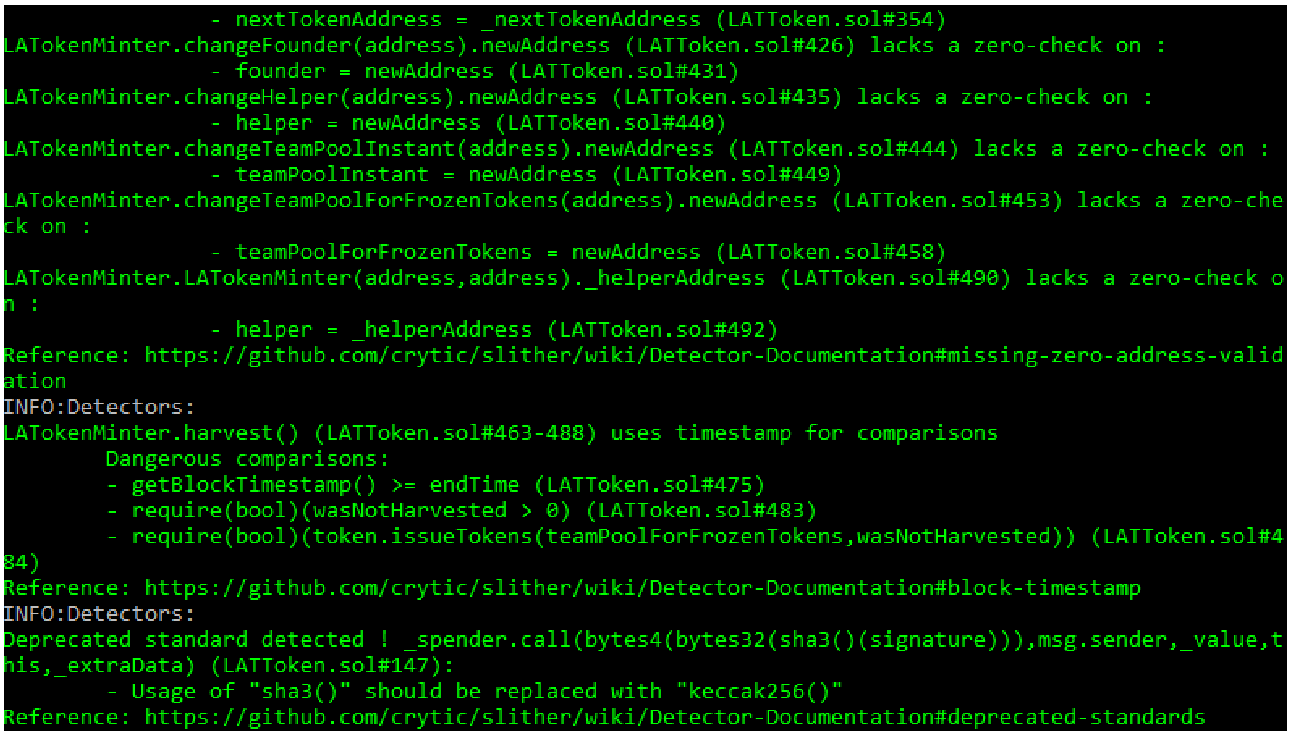
```
INFO:Detectors:
ExchangeContract.exchange(address,uint256) (LATToken.sol#319-339) performs a multiplication on the
result of a division:
        - amount = prevTokensAmount.div(prevCourse) (LATToken.sol#330)
        - assert(bool)(prevToken.burnTokens(_for,amount.mul(prevCourse))) (LATToken.sol#332)
ExchangeContract.exchange(address,uint256) (LATToken.sol#319-339) performs a multiplication on the
result of a division:
        - amount = prevTokensAmount.div(prevCourse) (LATToken.sol#330)
        - assert(bool)(nextToken.issueTokens(_for,amount.mul(nextCourse))) (LATToken.sol#333)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Contract locking ether found:
        Contract LATToken (LATToken.sol#161-282) has payable functions:
         - LATToken.fallback() (LATToken.sol#274-276)
        But does not have a function to withdraw the ether
Contract locking ether found:
        Contract LATokenMinter (LATToken.sol#362-514) has payable functions:
         - LATokenMinter.fallback() (LATToken.sol#506-508)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
```



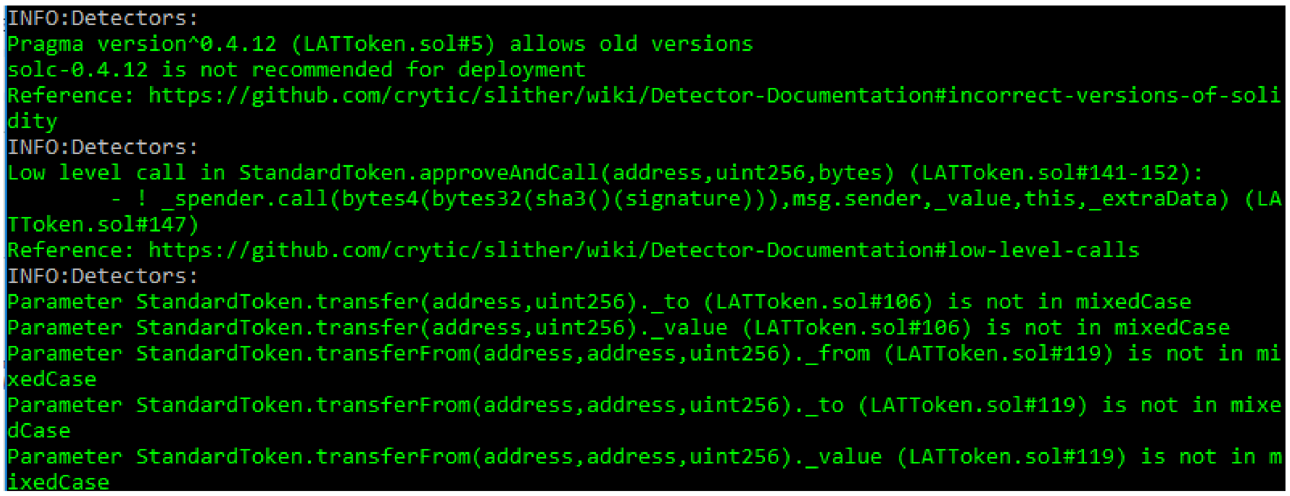
```
INFO:Detectors:
Reentrancy in LATokenMinter.fundTeamInstant() (LATToken.sol#401-415):
        External calls:
        - require(bool)(token.issueTokens(teamPoolInstant,totalInstantAmount)) (LATToken.sol#411)
        State variables written after the call(s):
        - teamInstantSent = true (LATToken.sol#413)
        LATokenMinter.teamInstantSent (LATToken.sol#373) can be used in cross function reentrancies
:
        - LATokenMinter.fundTeamInstant() (LATToken.sol#401-415)
        - LATokenMinter.teamInstantSent (LATToken.sol#373)
Reentrancy in LATokenMinter.harvest() (LATToken.sol#463-488):
        External calls:
        - require(bool)(token.issueTokens(teamPoolForFrozenTokens,wasNotHarvested)) (LATToken.sol#4
84)
        State variables written after the call(s):
        - alreadyHarvestedTokens = alreadyHarvestedTokens.add(wasNotHarvested) (LATToken.sol#485)
        LATokenMinter.alreadyHarvestedTokens (LATToken.sol#379) can be used in cross function reent
rancies:
        - LATokenMinter.LATokenMinter(address,address) (LATToken.sol#490-504)
        - LATokenMinter.alreadyHarvestedTokens (LATToken.sol#379)
        - LATokenMinter.harvest() (LATToken.sol#463-488)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
-1
```



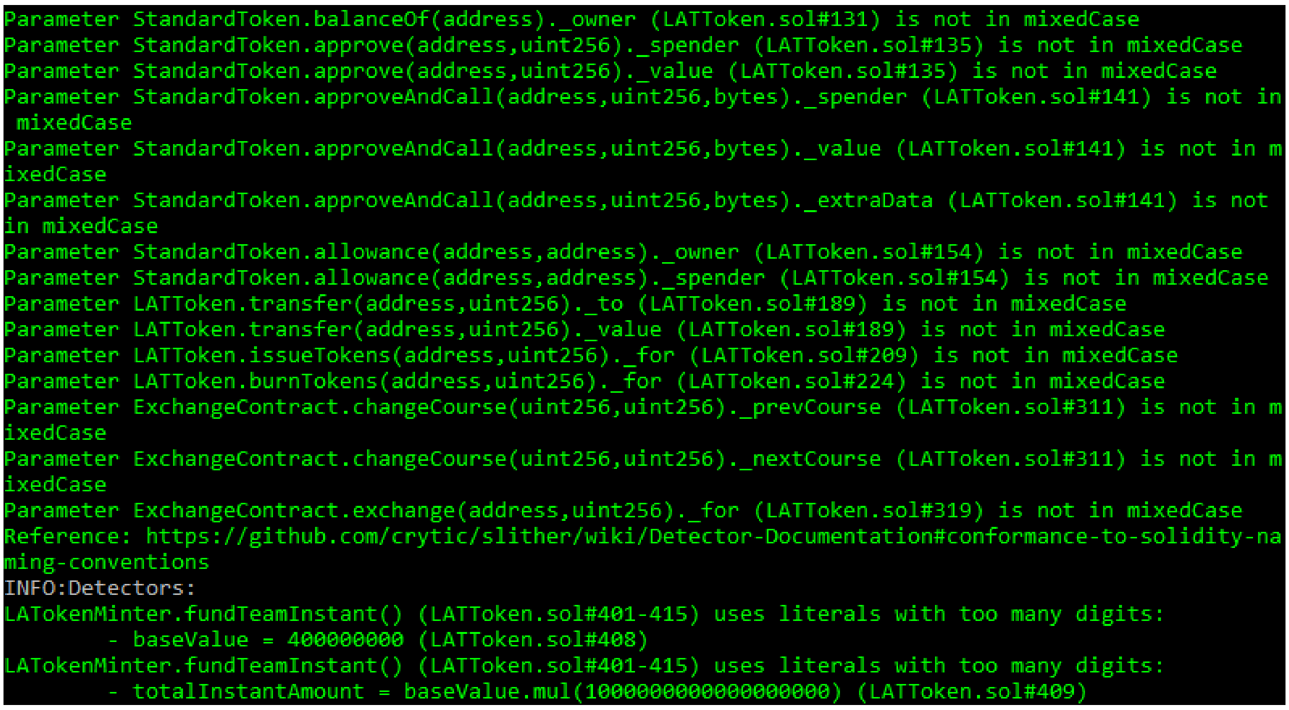
```
INFO:Detectors:
LATToken.changeMinter(address) (LATToken.sol#247-254) should emit an event for:
        - minter = newAddress (LATToken.sol#252)
LATToken.changeFounder(address) (LATToken.sol#256-263) should emit an event for:
        - founder = newAddress (LATToken.sol#261)
LATToken.changeExchanger(address) (LATToken.sol#265-272) should emit an event for:
        - exchanger = newAddress (LATToken.sol#270)
ExchangeContract.changeFounder(address) (LATToken.sol#341-348) should emit an event for:
        - founder = newAddress (LATToken.sol#346)
LATokenMinter.changeFounder(address) (LATToken.sol#426-433) should emit an event for:
        - founder = newAddress (LATToken.sol#431)
LATokenMinter.changeHelper(address) (LATToken.sol#435-442) should emit an event for:
        - helper = newAddress (LATToken.sol#440)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-cont
rol
INFO:Detectors:
LATToken.changeMinter(address).newAddress (LATToken.sol#247) lacks a zero-check on :
                - minter = newAddress (LATToken.sol#252)
LATToken.changeFounder(address).newAddress (LATToken.sol#256) lacks a zero-check on :
                - founder = newAddress (LATToken.sol#261)
LATToken.changeExchanger(address).newAddress (LATToken.sol#265) lacks a zero-check on :
                - exchanger = newAddress (LATToken.sol#270)
ExchangeContract.changeFounder(address).newAddress (LATToken.sol#341) lacks a zero-check on :
                - founder = newAddress (LATToken.sol#346)
ExchangeContract.ExchangeContract(address,address,uint256,uint256)._prevTokenAddress (LATToken.sol#
350) lacks a zero-check on :
                - prevTokenAddress = _prevTokenAddress (LATToken.sol#353)
```



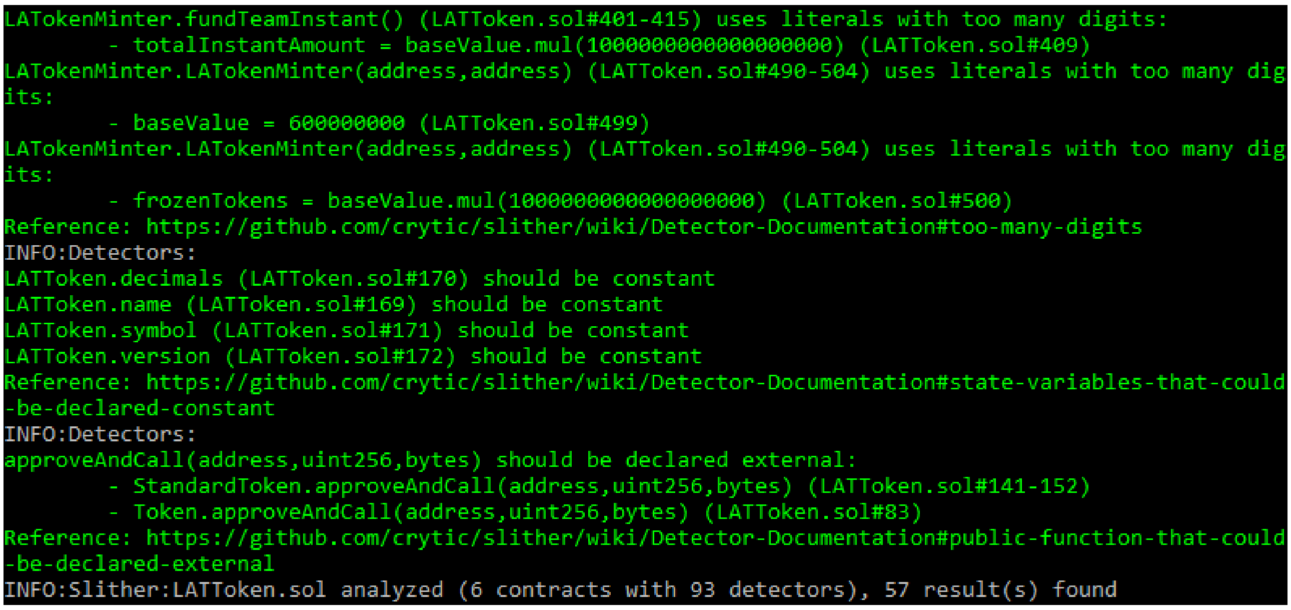
```
                - nextTokenAddress = _nextTokenAddress (LATToken.sol#354)
LATokenMinter.changeFounder(address).newAddress (LATToken.sol#426) lacks a zero-check on :
                - founder = newAddress (LATToken.sol#431)
LATokenMinter.changeHelper(address).newAddress (LATToken.sol#435) lacks a zero-check on :
                - helper = newAddress (LATToken.sol#440)
LATokenMinter.changeTeamPoolInstant(address).newAddress (LATToken.sol#444) lacks a zero-check on :
                - teamPoolInstant = newAddress (LATToken.sol#449)
LATokenMinter.changeTeamPoolForFrozenTokens(address).newAddress (LATToken.sol#453) lacks a zero-che
ck on :
                - teamPoolForFrozenTokens = newAddress (LATToken.sol#458)
LATokenMinter.LATokenMinter(address,address)._helperAddress (LATToken.sol#490) lacks a zero-check o
n :
                - helper = _helperAddress (LATToken.sol#492)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-valid
ation
INFO:Detectors:
LATokenMinter.harvest() (LATToken.sol#463-488) uses timestamp for comparisons
        Dangerous comparisons:
        - getBlockTimestamp() >= endTime (LATToken.sol#475)
        - require(bool)(wasNotHarvested > 0) (LATToken.sol#483)
        - require(bool)(token.issueTokens(teamPoolForFrozenTokens,wasNotHarvested)) (LATToken.sol#4
84)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Deprecated standard detected ! _spender.call(bytes4(bytes32(sha3()(signature))),msg.sender,_value,t
his,_extraData) (LATToken.sol#147):
        - Usage of "sha3()" should be replaced with "keccak256()"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#deprecated-standards
```



```
INFO:Detectors:
Pragma version^0.4.12 (LATToken.sol#5) allows old versions
solc-0.4.12 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-soli
dity
INFO:Detectors:
Low level call in StandardToken.approveAndCall(address,uint256,bytes) (LATToken.sol#141-152):
        - ! _spender.call(bytes4(bytes32(sha3()(signature))),msg.sender,_value,this,_extraData) (LA
TToken.sol#147)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter StandardToken.transfer(address,uint256)._to (LATToken.sol#106) is not in mixedCase
Parameter StandardToken.transfer(address,uint256)._value (LATToken.sol#106) is not in mixedCase
Parameter StandardToken.transferFrom(address,address,uint256)._from (LATToken.sol#119) is not in mi
xedCase
Parameter StandardToken.transferFrom(address,address,uint256)._to (LATToken.sol#119) is not in mixe
dCase
Parameter StandardToken.transferFrom(address,address,uint256)._value (LATToken.sol#119) is not in m
ixedCase
```



```
Parameter StandardToken.balanceOf(address)._owner (LATToken.sol#131) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._spender (LATToken.sol#135) is not in mixedCase
Parameter StandardToken.approve(address,uint256)._value (LATToken.sol#135) is not in mixedCase
Parameter StandardToken.approveAndCall(address,uint256,bytes)._spender (LATToken.sol#141) is not in
 mixedCase
Parameter StandardToken.approveAndCall(address,uint256,bytes)._value (LATToken.sol#141) is not in m
ixedCase
Parameter StandardToken.approveAndCall(address,uint256,bytes)._extraData (LATToken.sol#141) is not
in mixedCase
Parameter StandardToken.allowance(address,address)._owner (LATToken.sol#154) is not in mixedCase
Parameter StandardToken.allowance(address,address)._spender (LATToken.sol#154) is not in mixedCase
Parameter LATToken.transfer(address,uint256)._to (LATToken.sol#189) is not in mixedCase
Parameter LATToken.transfer(address,uint256)._value (LATToken.sol#189) is not in mixedCase
Parameter LATToken.issueTokens(address,uint256)._for (LATToken.sol#209) is not in mixedCase
Parameter LATToken.burnTokens(address,uint256)._for (LATToken.sol#224) is not in mixedCase
Parameter ExchangeContract.changeCourse(uint256,uint256)._prevCourse (LATToken.sol#311) is not in m
ixedCase
Parameter ExchangeContract.changeCourse(uint256,uint256)._nextCourse (LATToken.sol#311) is not in m
ixedCase
Parameter ExchangeContract.exchange(address,uint256)._for (LATToken.sol#319) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-na
ming-conventions
INFO:Detectors:
LATokenMinter.fundTeamInstant() (LATToken.sol#401-415) uses literals with too many digits:
        - baseValue = 400000000 (LATToken.sol#408)
LATokenMinter.fundTeamInstant() (LATToken.sol#401-415) uses literals with too many digits:
        - totalInstantAmount = baseValue.mul(1000000000000000000) (LATToken.sol#409)
```

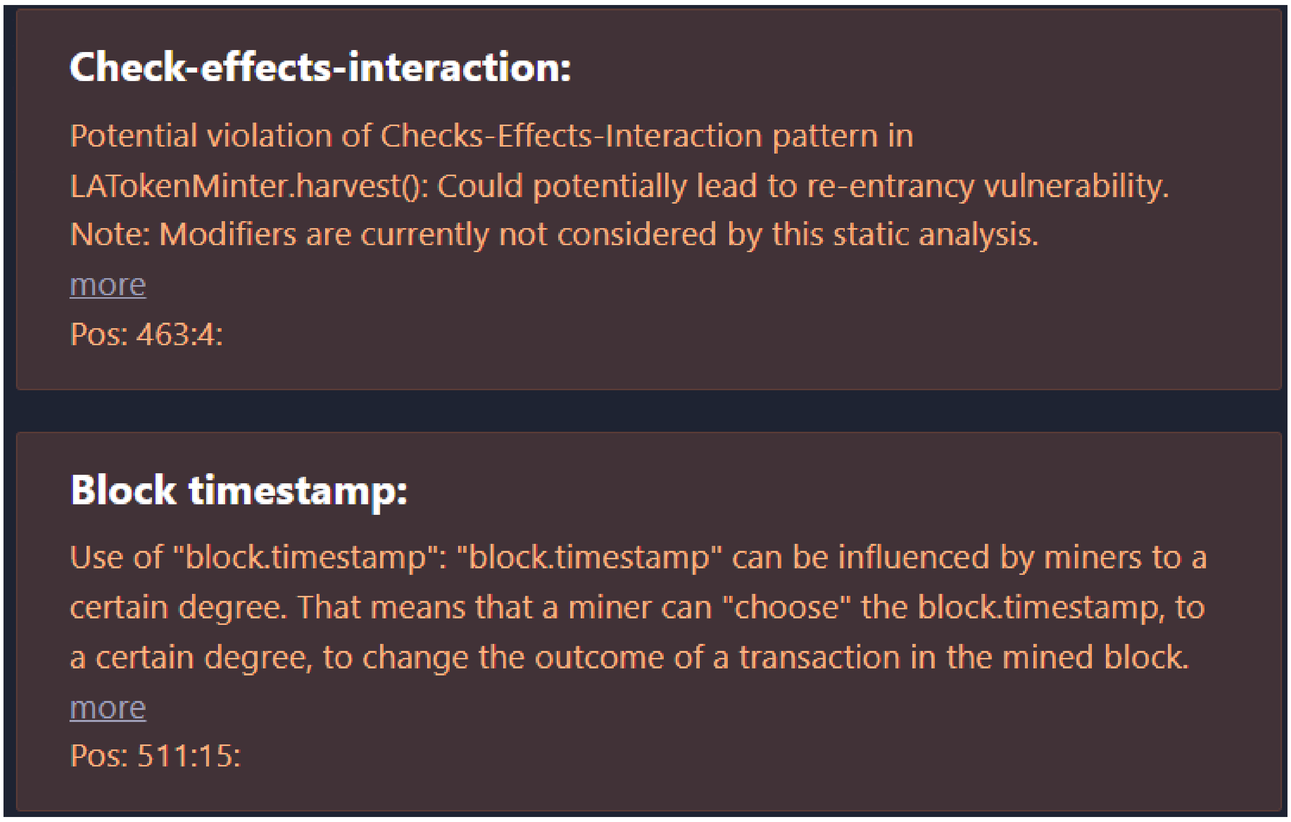


```
LATokenMinter.fundTeamInstant() (LATToken.sol#401-415) uses literals with too many digits:
        - totalInstantAmount = baseValue.mul(1000000000000000000) (LATToken.sol#409)
LATokenMinter.LATokenMinter(address,address) (LATToken.sol#490-504) uses literals with too many dig
its:
        - baseValue = 600000000 (LATToken.sol#499)
LATokenMinter.LATokenMinter(address,address) (LATToken.sol#490-504) uses literals with too many dig
its:
        - frozenTokens = baseValue.mul(1000000000000000000) (LATToken.sol#500)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
LATToken.decimals (LATToken.sol#170) should be constant
LATToken.name (LATToken.sol#169) should be constant
LATToken.symbol (LATToken.sol#171) should be constant
LATToken.version (LATToken.sol#172) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could
-be-declared-constant
INFO:Detectors:
approveAndCall(address,uint256,bytes) should be declared external:
        - StandardToken.approveAndCall(address,uint256,bytes) (LATToken.sol#141-152)
        - Token.approveAndCall(address,uint256,bytes) (LATToken.sol#83)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could
-be-declared-external
INFO:Slither:LATToken.sol analyzed (6 contracts with 93 detectors), 57 result(s) found
```

Solidity Static Analysis

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

LATToken.sol

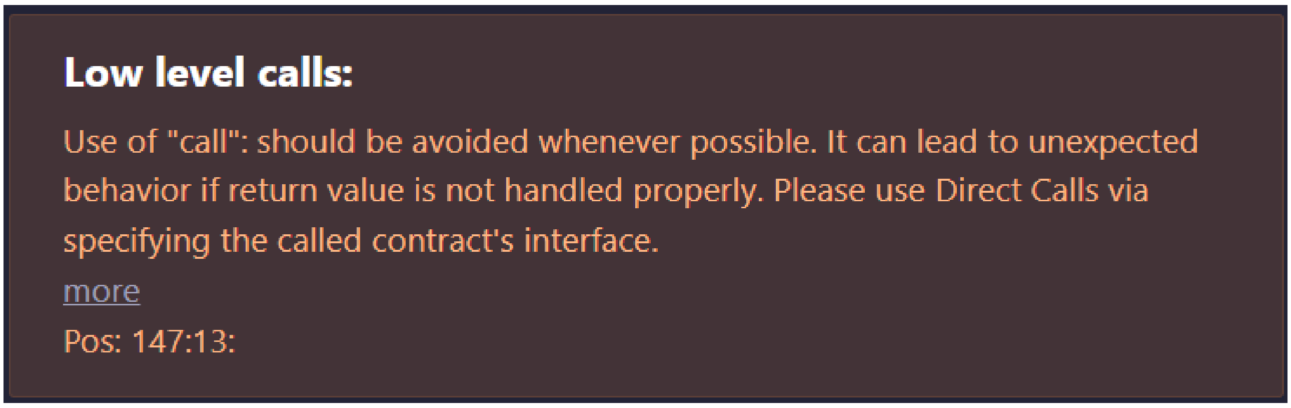


Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in LATokenMinter.harvest(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 463:4:

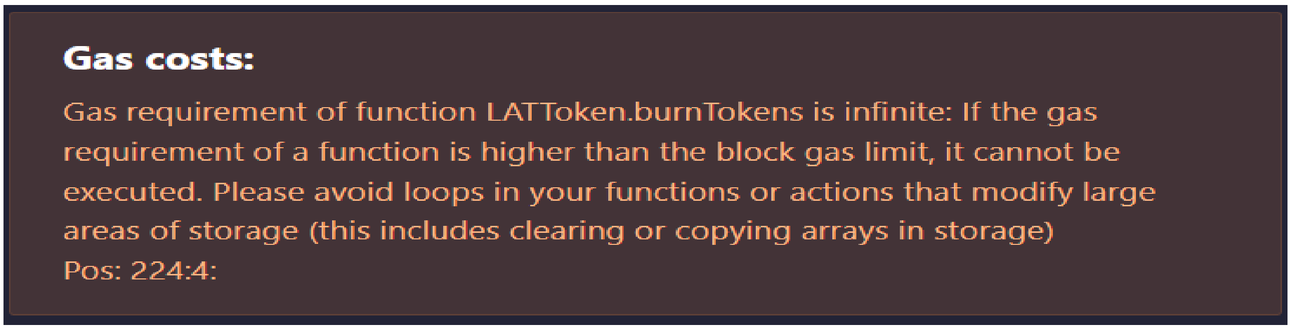
Block timestamp:

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: 511:15:



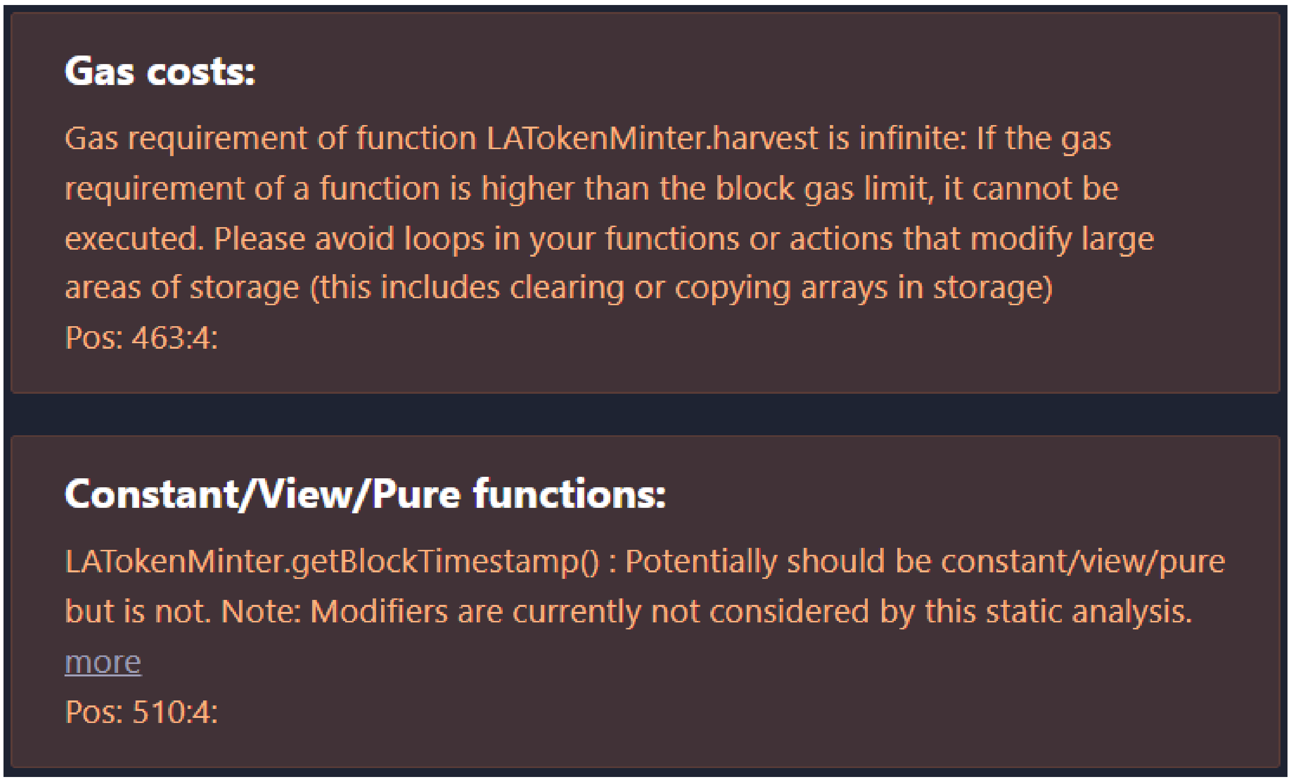
Low level calls:

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: 147:13:



Gas costs:

Gas requirement of function LATToken.burnTokens 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: 224:4:

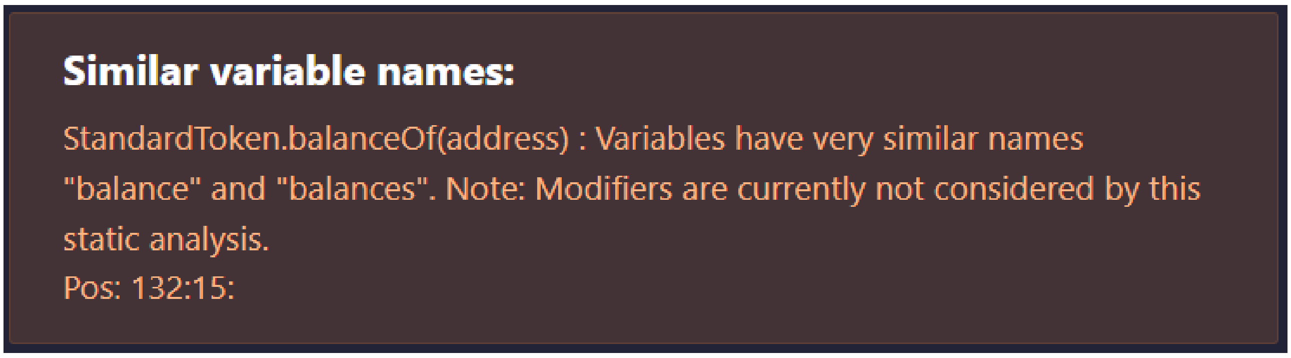


Gas costs:

Gas requirement of function LATokenMinter.harvest 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: 463:4:

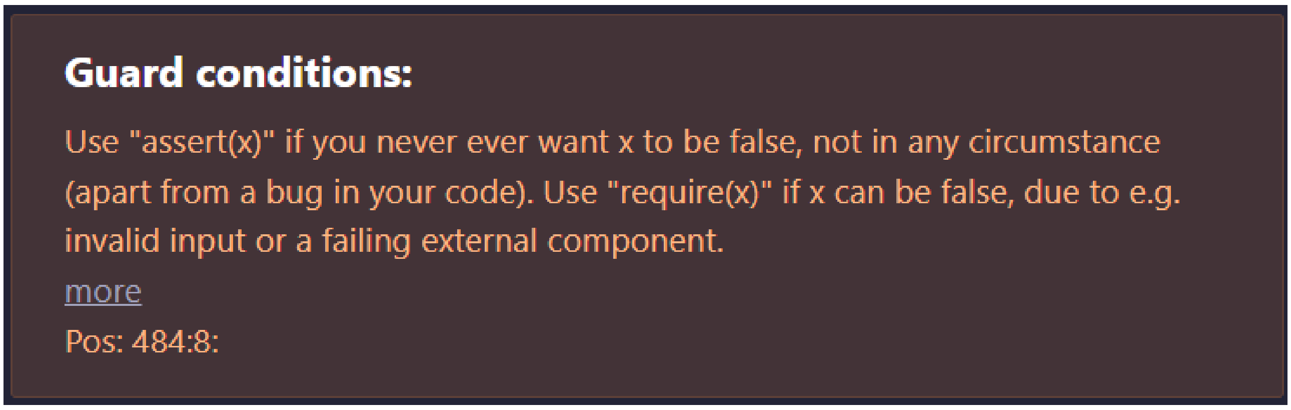
Constant/View/Pure functions:

LATokenMinter.getBlockTimestamp() : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 510:4:



Similar variable names:

StandardToken.balanceOf(address) : Variables have very similar names "balance" and "balances". Note: Modifiers are currently not considered by this static analysis.
Pos: 132:15:



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: 484:8:

Software analysis result:

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