www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	MaticLaunch
Website:	maticlaunch.org
Platform:	Polygon (Matic)
Language:	Solidity
Date:	September 12th, 2021

Table of contents

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Introduction

EtherAuthority was contracted by the MaticLaunch team to perform the Security audit of MTCL Staking 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 12th, 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

MaticLaunch is a decentralized crowdfunding platform for the next generation of ideas built on Polygon. MTCL is a core element to fuel ecosystem and will play a key role in MaticLaunch's decentralized fundraising platform. MTCL Staking rewards users based on their staking volume.

Audit scope

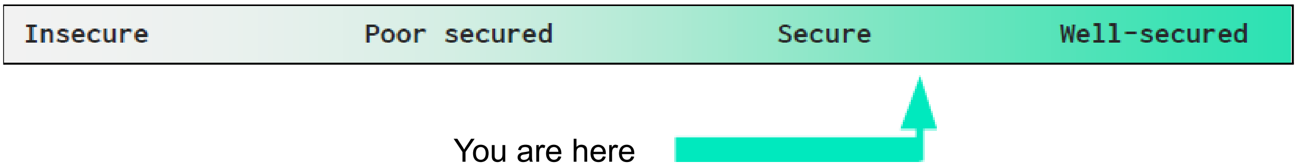
Name	**Code Review and Security Analysis Report for MTCL Staking Smart Contract**
Platform	**Polygon / Solidity**
File	MTCLStaking.sol
Online code	https://github.com/maticlaunch/MTCLStaking/blob/master/contracts/MTCLStaking.sol
Github Commit	0de172e4c9f92fee0300eb92a5fc83b3bd81b7ce
Audit Date	September 12th, 2021

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
MTCL Staking is a flexible staking contract for ERC20 token.	**YES, This is valid.**
Constant guaranteed time based return. For example 10% return per month. Payout rewards are calculated based on mining time.	**YES, This is valid.**
Reward distribution after maturity. Staking contract can have a maturity date and a fixed reward amount which will be distributed between amounts that were not withdrawn until maturity time.	**YES, This is valid.**
Cap staking amount and time-boxed staking period.	**YES, This is valid.**
Prevent withdrawal before a given withdrawal start date.	**YES, This is valid.**
No compounded return. Returns are linearly distributed.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contract is **"Secured"**. This token contract does not contain any owner control, making 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 1 low and some very low level issues. These issues do not raise major vulnerable scenarios. And they are acknowledged by the MaticLaunch team.

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	Moderated
	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	Passed
	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 1 smart contract file. Smart contract also contains a Library and Interface. These are compact and well written contracts.

The library in MTCL Staking is 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.

The MaticLaunch 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 MTCL Staking smart contracts code in the form of github repository URL. The commit 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.

Another source of information was its official website http://maticlaunch.org which provided rich information about the project architecture and tokenomics.

Use of Dependencies

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

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

AS-IS overview

(1) Interface

(a) ERC20

(2) Usages

(a) using SafeMath for uint256;

(3) Events

(a) event Staked(address indexed token, address indexed staker_, uint256 requestedAmount_, uint256 stakedAmount_);

(b) event PaidOut(address indexed token, address indexed staker_, uint256 amount_, uint256 reward_);

(c) event Refunded(address indexed token, address indexed staker_, uint256 amount_);

(4) Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	addReward	write	Passed	No Issue
3	stakeOf	read	Passed	No Issue
4	stake	write	Passed	No Issue
5	withdraw	write	Passed	No Issue
6	_withdrawEarly	write	Passed	No Issue
7	_withdrawAfterClose	write	Passed	No Issue
8	_stake	write	Passed	No Issue
9	_payMe	write	Passed	No Issue
10	_payTo	write	Passed	No Issue
11	_payDirect	write	Passed	No Issue
12	_positive	modifier	Passed	No Issue
13	_realAddress	modifier	Passed	No Issue
14	_after	modifier	Passed	No Issue
15	_before	modifier	Passed	No Issue
16	_hasAllowance	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

No Critical severity vulnerabilities were found.

High

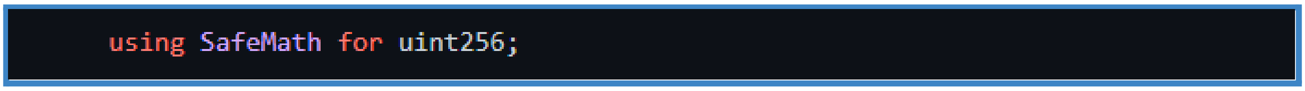
No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) SafeMath library is not needed



```
    using SafeMath for uint256;
```

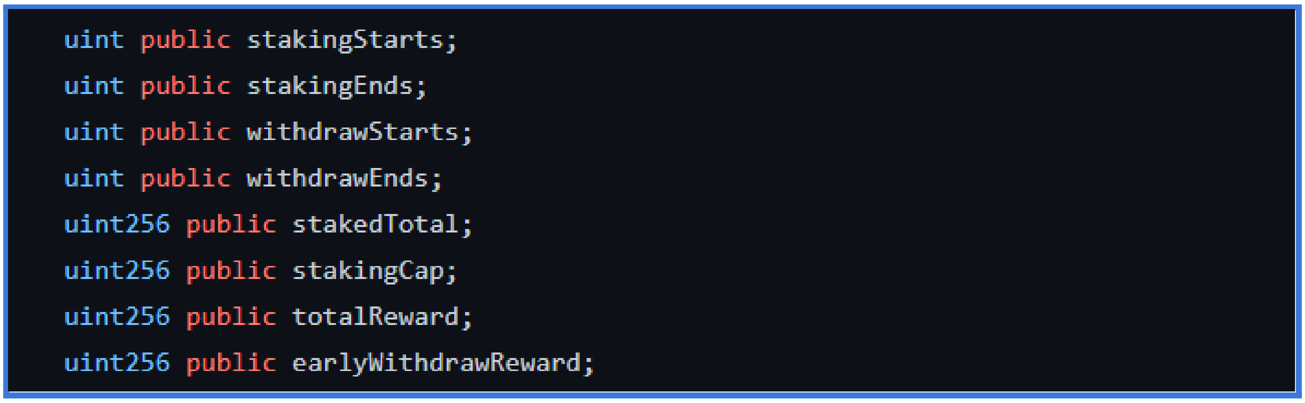
Solidity version 0.8.0 and bove has in-built support for the overflow and underflow prevention. Therefore, all the arithmetic operations can be performed without any wrapper library.

Resolution: We recommend removing the safemath library. As it will save gas on all arithmetic operations.

status: **Acknowledged**

Very Low / Informational / Best practices:

(1) Declare variables uniformly.



```
  uint public stakingStarts;
  uint public stakingEnds;
  uint public withdrawStarts;
  uint public withdrawEnds;
  uint256 public stakedTotal;
  uint256 public stakingCap;
  uint256 public totalReward;
  uint256 public earlyWithdrawReward;
```

Although uint and uint256 are the same, it is best practice to keep the variable declaration uniform.

Resolution: We suggest using uint256 everywhere.

status: **Acknowledged**

(2) All functions which are not called internally, must be declared as external. It is more efficient as sometimes it saves some gas.

https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

status: **Acknowledged**

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).

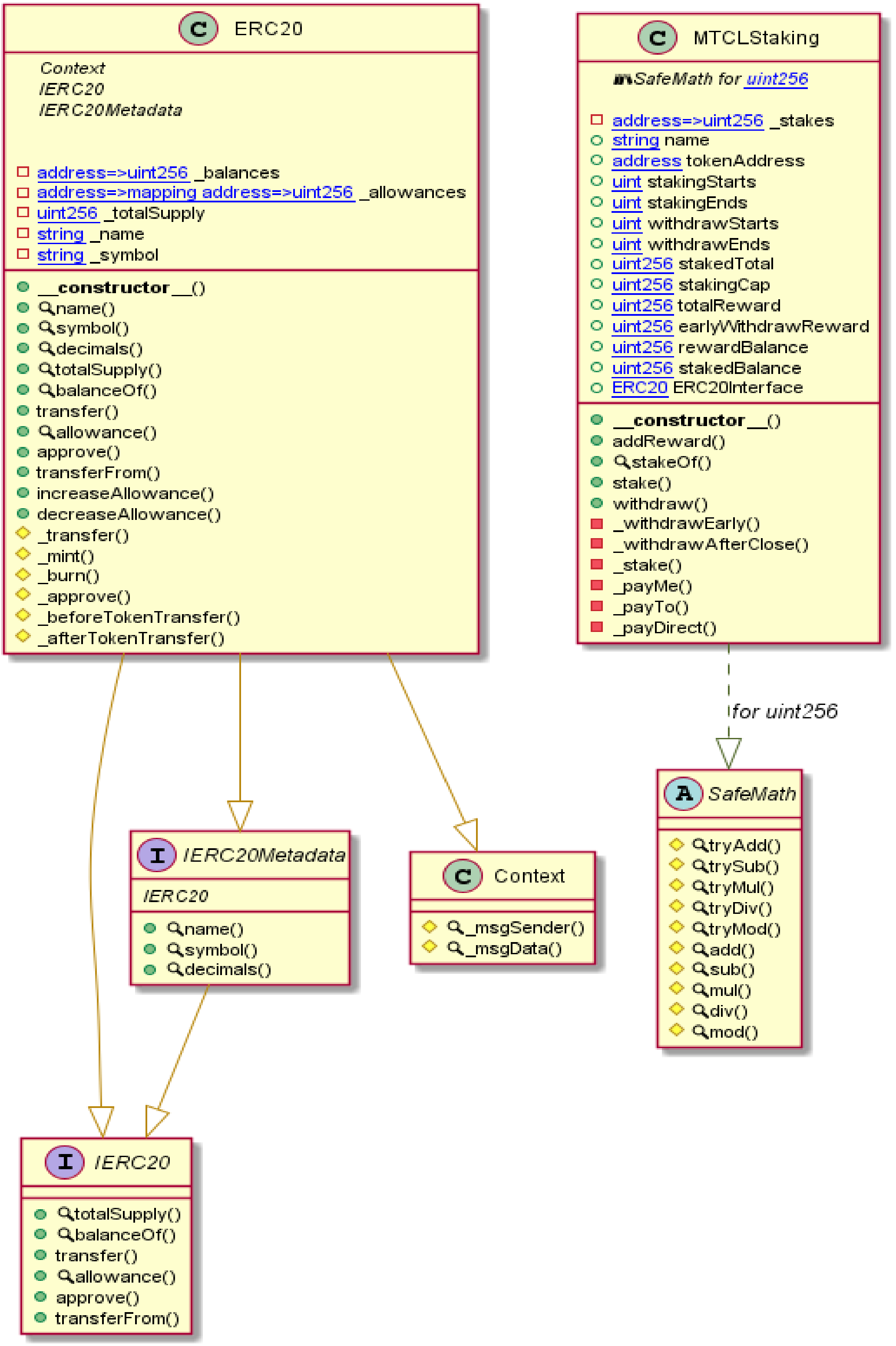
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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

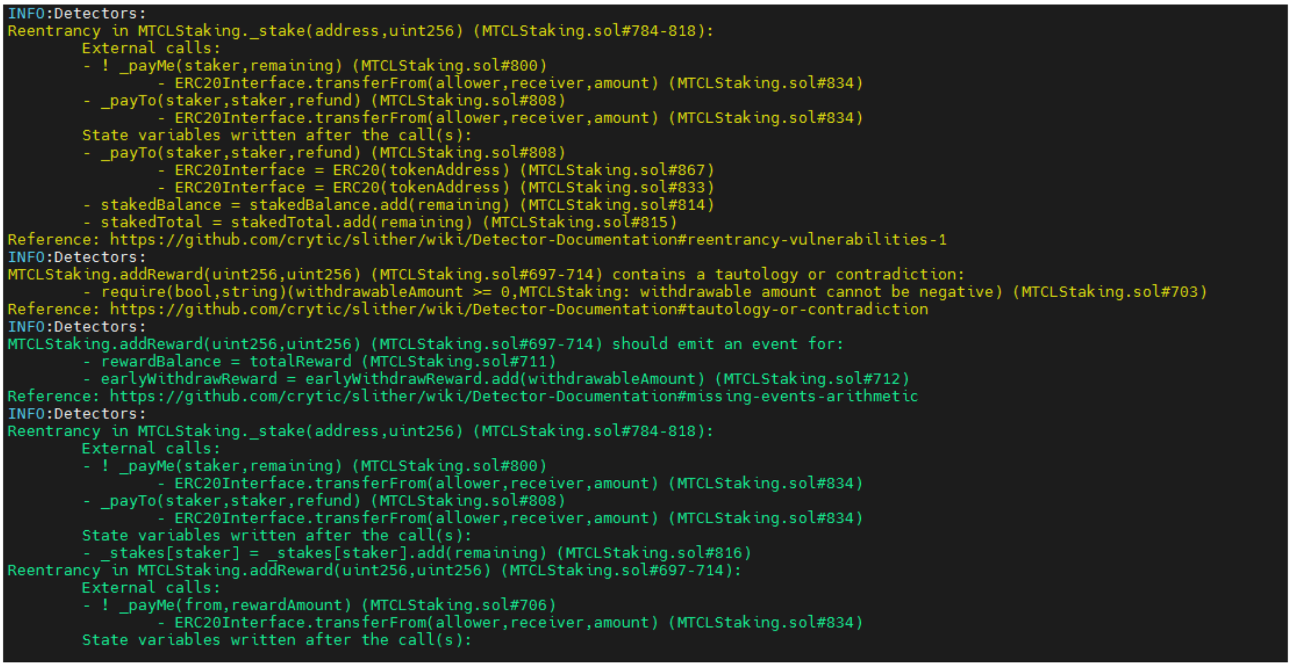
Appendix

Code Flow Diagram - MTCL Staking

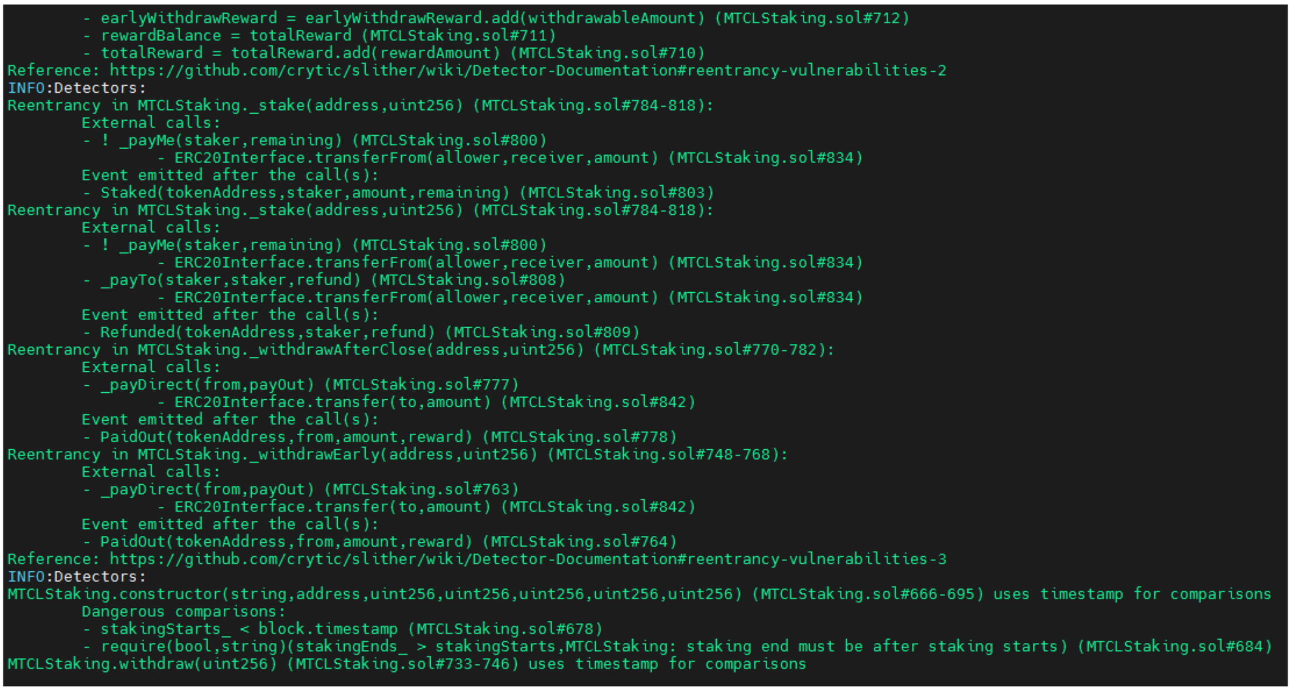


Slither Results Log

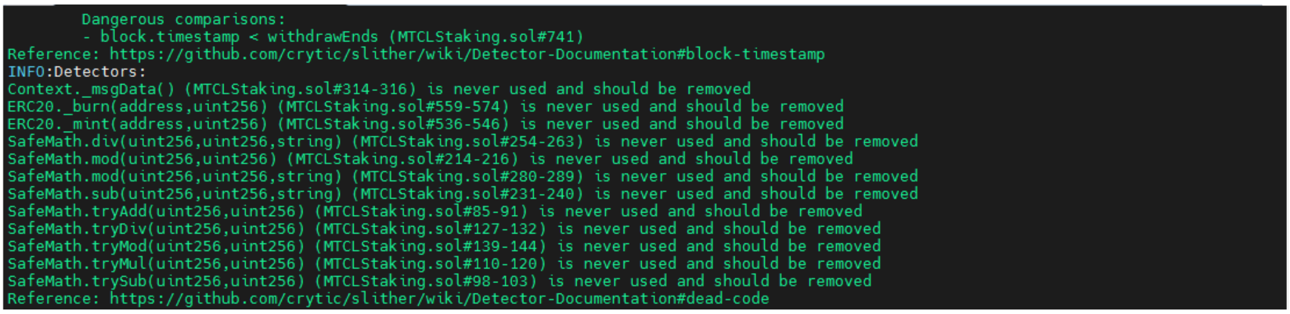
Slither log >> MTCLStaking.sol



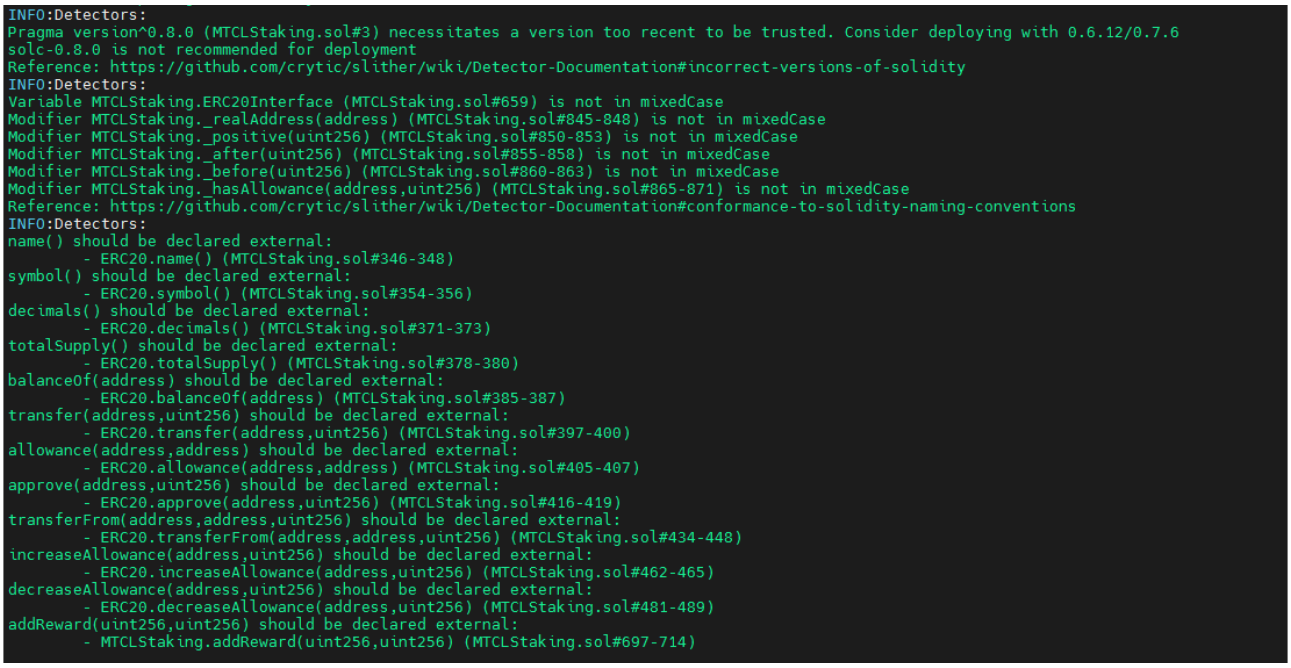
```
INFO:Detectors:
Reentrancy in MTCLStaking._stake(address,uint256) (MTCLStaking.sol#784-818):
        External calls:
        - ! _payMe(staker,remaining) (MTCLStaking.sol#800)
                - ERC20Interface.transferFrom(allower,receiver,amount) (MTCLStaking.sol#834)
        - _payTo(staker,staker,refund) (MTCLStaking.sol#808)
                - ERC20Interface.transferFrom(allower,receiver,amount) (MTCLStaking.sol#834)
        State variables written after the call(s):
        - _payTo(staker,staker,refund) (MTCLStaking.sol#808)
                - ERC20Interface = ERC20(tokenAddress) (MTCLStaking.sol#867)
                - ERC20Interface = ERC20(tokenAddress) (MTCLStaking.sol#833)
        - stakedBalance = stakedBalance.add(remaining) (MTCLStaking.sol#814)
        - stakedTotal = stakedTotal.add(remaining) (MTCLStaking.sol#815)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
MTCLStaking.addReward(uint256,uint256) (MTCLStaking.sol#697-714) contains a tautology or contradiction:
        - require(bool,string)(withdrawableAmount >= 0,MTCLStaking: withdrawable amount cannot be negative) (MTCLStaking.sol#703)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction
INFO:Detectors:
MTCLStaking.addReward(uint256,uint256) (MTCLStaking.sol#697-714) should emit an event for:
        - rewardBalance = totalReward (MTCLStaking.sol#711)
        - earlyWithdrawReward = earlyWithdrawReward.add(withdrawableAmount) (MTCLStaking.sol#712)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
Reentrancy in MTCLStaking._stake(address,uint256) (MTCLStaking.sol#784-818):
        External calls:
        - ! _payMe(staker,remaining) (MTCLStaking.sol#800)
                - ERC20Interface.transferFrom(allower,receiver,amount) (MTCLStaking.sol#834)
        - _payTo(staker,staker,refund) (MTCLStaking.sol#808)
                - ERC20Interface.transferFrom(allower,receiver,amount) (MTCLStaking.sol#834)
        State variables written after the call(s):
        - _stakes[staker] = _stakes[staker].add(remaining) (MTCLStaking.sol#816)
Reentrancy in MTCLStaking.addReward(uint256,uint256) (MTCLStaking.sol#697-714):
        External calls:
        - ! _payMe(from,rewardAmount) (MTCLStaking.sol#706)
                - ERC20Interface.transferFrom(allower,receiver,amount) (MTCLStaking.sol#834)
        State variables written after the call(s):
```



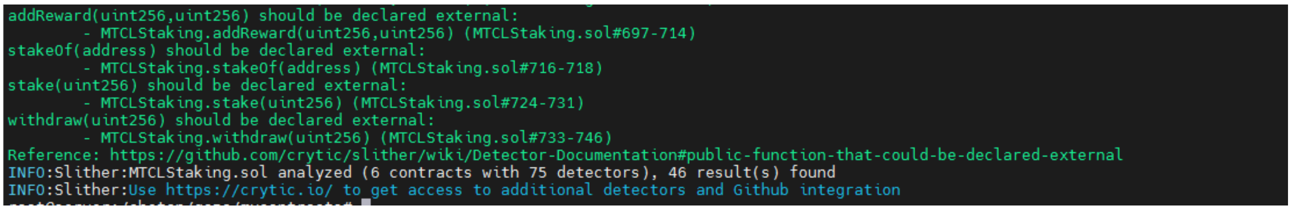
```
        - earlyWithdrawReward = earlyWithdrawReward.add(withdrawableAmount) (MTCLStaking.sol#712)
        - rewardBalance = totalReward (MTCLStaking.sol#711)
        - totalReward = totalReward.add(rewardAmount) (MTCLStaking.sol#710)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in MTCLStaking._stake(address,uint256) (MTCLStaking.sol#784-818):
        External calls:
        - ! _payMe(staker,remaining) (MTCLStaking.sol#800)
                - ERC20Interface.transferFrom(allower,receiver,amount) (MTCLStaking.sol#834)
        Event emitted after the call(s):
        - Staked(tokenAddress,staker,amount,remaining) (MTCLStaking.sol#803)
Reentrancy in MTCLStaking._stake(address,uint256) (MTCLStaking.sol#784-818):
        External calls:
        - ! _payMe(staker,remaining) (MTCLStaking.sol#800)
                - ERC20Interface.transferFrom(allower,receiver,amount) (MTCLStaking.sol#834)
        - _payTo(staker,staker,refund) (MTCLStaking.sol#808)
                - ERC20Interface.transferFrom(allower,receiver,amount) (MTCLStaking.sol#834)
        Event emitted after the call(s):
        - Refunded(tokenAddress,staker,refund) (MTCLStaking.sol#809)
Reentrancy in MTCLStaking._withdrawAfterClose(address,uint256) (MTCLStaking.sol#770-782):
        External calls:
        - _payDirect(from,payOut) (MTCLStaking.sol#777)
                - ERC20Interface.transfer(to,amount) (MTCLStaking.sol#842)
        Event emitted after the call(s):
        - PaidOut(tokenAddress,from,amount,reward) (MTCLStaking.sol#778)
Reentrancy in MTCLStaking._withdrawEarly(address,uint256) (MTCLStaking.sol#748-768):
        External calls:
        - _payDirect(from,payOut) (MTCLStaking.sol#763)
                - ERC20Interface.transfer(to,amount) (MTCLStaking.sol#842)
        Event emitted after the call(s):
        - PaidOut(tokenAddress,from,amount,reward) (MTCLStaking.sol#764)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
MTCLStaking.constructor(string,address,uint256,uint256,uint256,uint256,uint256) (MTCLStaking.sol#666-695) uses timestamp for comparisons
        Dangerous comparisons:
        - stakingStarts_ < block.timestamp (MTCLStaking.sol#678)
        - require(bool,string)(stakingEnds_ > stakingStarts,MTCLStaking: staking end must be after staking starts) (MTCLStaking.sol#684)
MTCLStaking.withdraw(uint256) (MTCLStaking.sol#733-746) uses timestamp for comparisons
```



```
        Dangerous comparisons:
        - block.timestamp < withdrawEnds (MTCLStaking.sol#741)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Context._msgData() (MTCLStaking.sol#314-316) is never used and should be removed
ERC20._burn(address,uint256) (MTCLStaking.sol#559-574) is never used and should be removed
ERC20._mint(address,uint256) (MTCLStaking.sol#536-546) is never used and should be removed
SafeMath.div(uint256,uint256,string) (MTCLStaking.sol#254-263) is never used and should be removed
SafeMath.mod(uint256,uint256) (MTCLStaking.sol#214-216) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (MTCLStaking.sol#280-289) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (MTCLStaking.sol#231-240) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (MTCLStaking.sol#85-91) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (MTCLStaking.sol#127-132) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (MTCLStaking.sol#139-144) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (MTCLStaking.sol#110-120) is never used and should be removed
SafeMath.trySub(uint256,uint256) (MTCLStaking.sol#98-103) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```



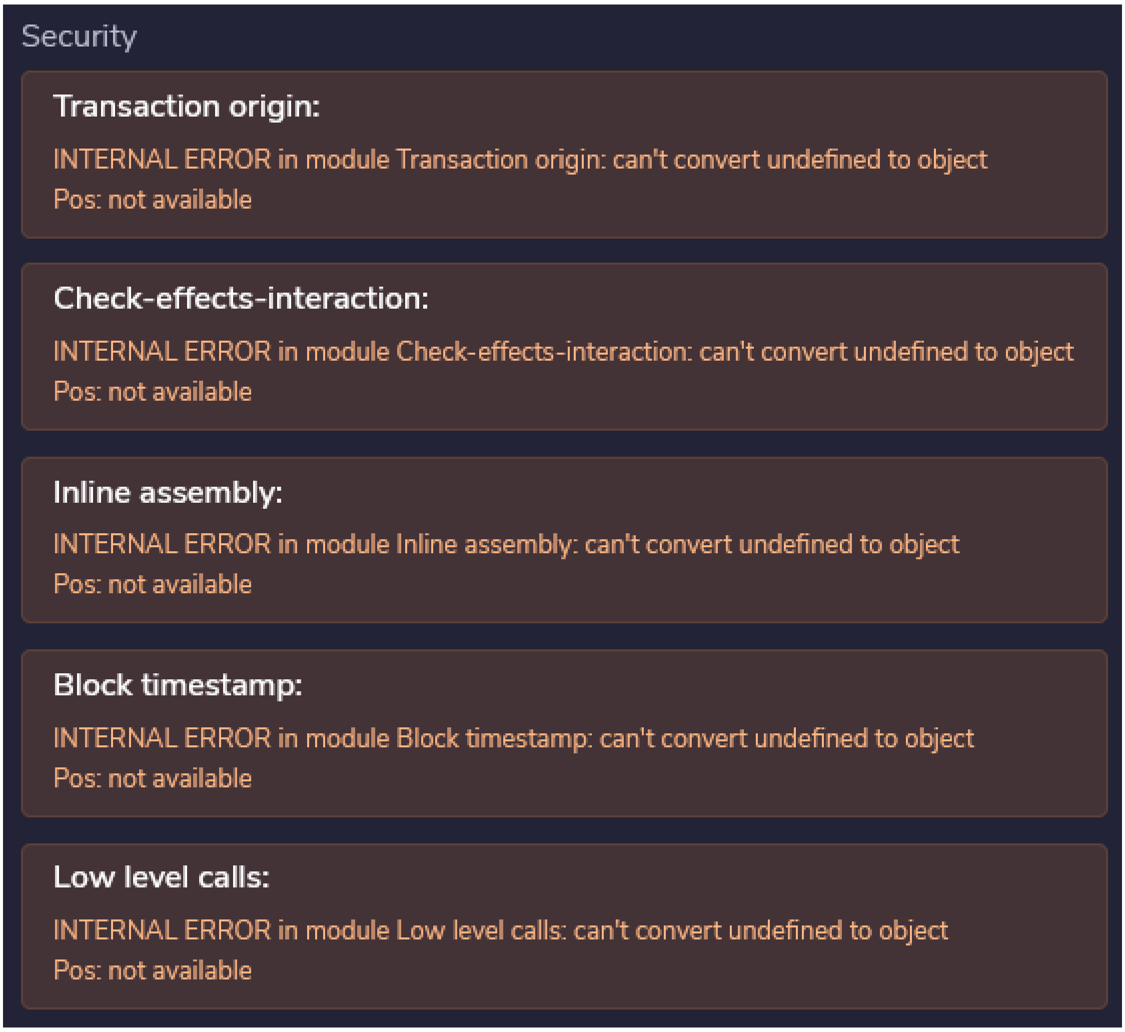
```
INFO:Detectors:
Pragma version^0.8.0 (MTCLStaking.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable MTCLStaking.ERC20Interface (MTCLStaking.sol#659) is not in mixedCase
Modifier MTCLStaking._realAddress(address) (MTCLStaking.sol#845-848) is not in mixedCase
Modifier MTCLStaking._positive(uint256) (MTCLStaking.sol#850-853) is not in mixedCase
Modifier MTCLStaking._after(uint256) (MTCLStaking.sol#855-858) is not in mixedCase
Modifier MTCLStaking._before(uint256) (MTCLStaking.sol#860-863) is not in mixedCase
Modifier MTCLStaking._hasAllowance(address,uint256) (MTCLStaking.sol#865-871) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
name() should be declared external:
        - ERC20.name() (MTCLStaking.sol#346-348)
symbol() should be declared external:
        - ERC20.symbol() (MTCLStaking.sol#354-356)
decimals() should be declared external:
        - ERC20.decimals() (MTCLStaking.sol#371-373)
totalSupply() should be declared external:
        - ERC20.totalSupply() (MTCLStaking.sol#378-380)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (MTCLStaking.sol#385-387)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (MTCLStaking.sol#397-400)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (MTCLStaking.sol#405-407)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (MTCLStaking.sol#416-419)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (MTCLStaking.sol#434-448)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (MTCLStaking.sol#462-465)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (MTCLStaking.sol#481-489)
addReward(uint256,uint256) should be declared external:
        - MTCLStaking.addReward(uint256,uint256) (MTCLStaking.sol#697-714)
```



```
addReward(uint256,uint256) should be declared external:
        - MTCLStaking.addReward(uint256,uint256) (MTCLStaking.sol#697-714)
stakeOf(address) should be declared external:
        - MTCLStaking.stakeOf(address) (MTCLStaking.sol#716-718)
stake(uint256) should be declared external:
        - MTCLStaking.stake(uint256) (MTCLStaking.sol#724-731)
withdraw(uint256) should be declared external:
        - MTCLStaking.withdraw(uint256) (MTCLStaking.sol#733-746)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:MTCLStaking.sol analyzed (6 contracts with 75 detectors), 46 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

MTCLStaking.sol



Security

Transaction origin:

INTERNAL ERROR in module Transaction origin: can't convert undefined to object
Pos: not available

Check-effects-interaction:

INTERNAL ERROR in module Check-effects-interaction: can't convert undefined to object
Pos: not available

Inline assembly:

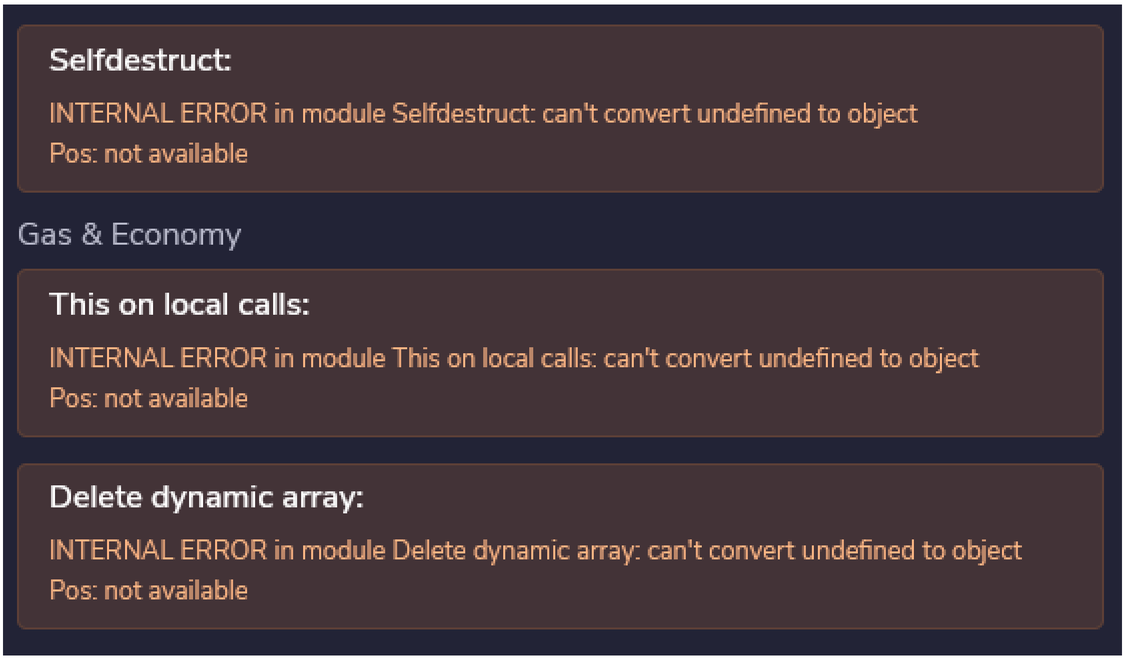
INTERNAL ERROR in module Inline assembly: can't convert undefined to object
Pos: not available

Block timestamp:

INTERNAL ERROR in module Block timestamp: can't convert undefined to object
Pos: not available

Low level calls:

INTERNAL ERROR in module Low level calls: can't convert undefined to object
Pos: not available



Selfdestruct:

INTERNAL ERROR in module Selfdestruct: can't convert undefined to object
Pos: not available

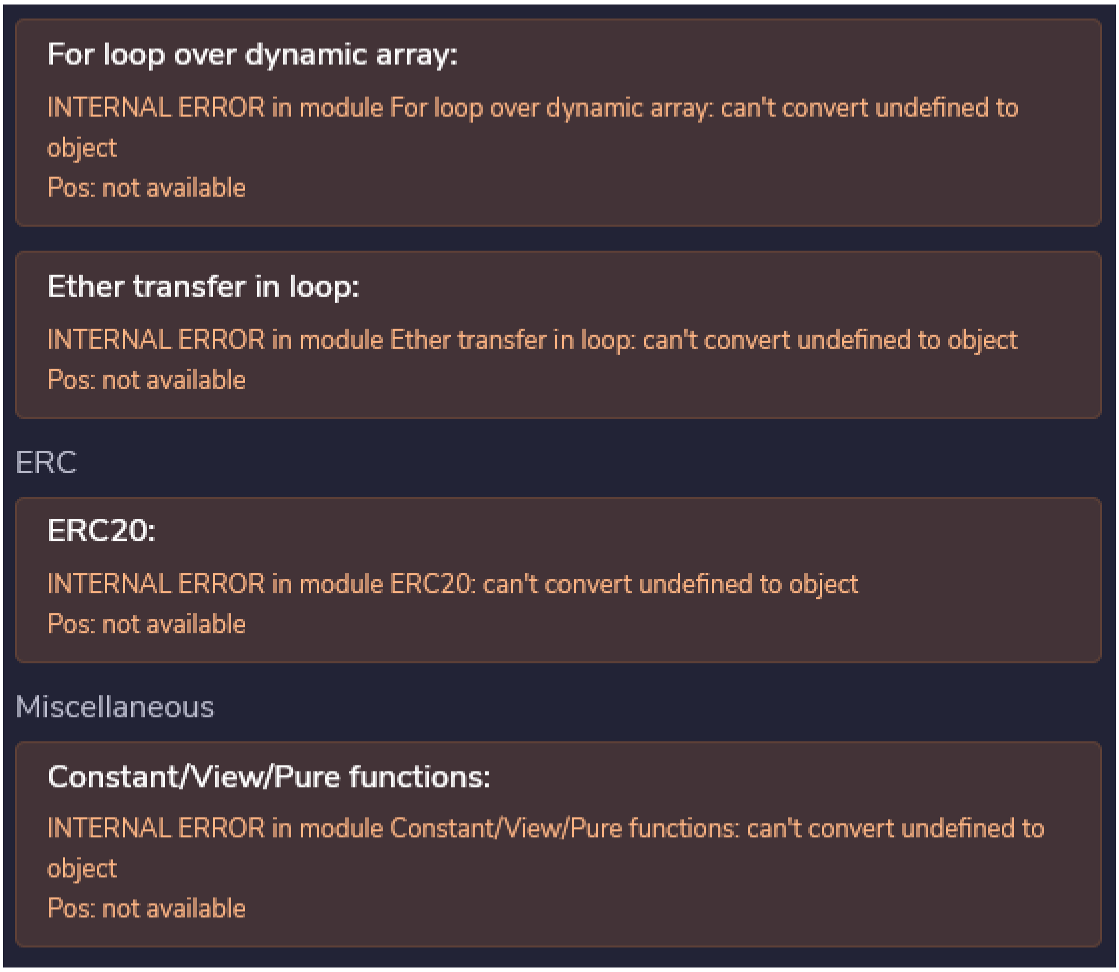
Gas & Economy

This on local calls:

INTERNAL ERROR in module This on local calls: can't convert undefined to object
Pos: not available

Delete dynamic array:

INTERNAL ERROR in module Delete dynamic array: can't convert undefined to object
Pos: not available



For loop over dynamic array:

INTERNAL ERROR in module For loop over dynamic array: can't convert undefined to object
Pos: not available

Ether transfer in loop:

INTERNAL ERROR in module Ether transfer in loop: can't convert undefined to object
Pos: not available

ERC

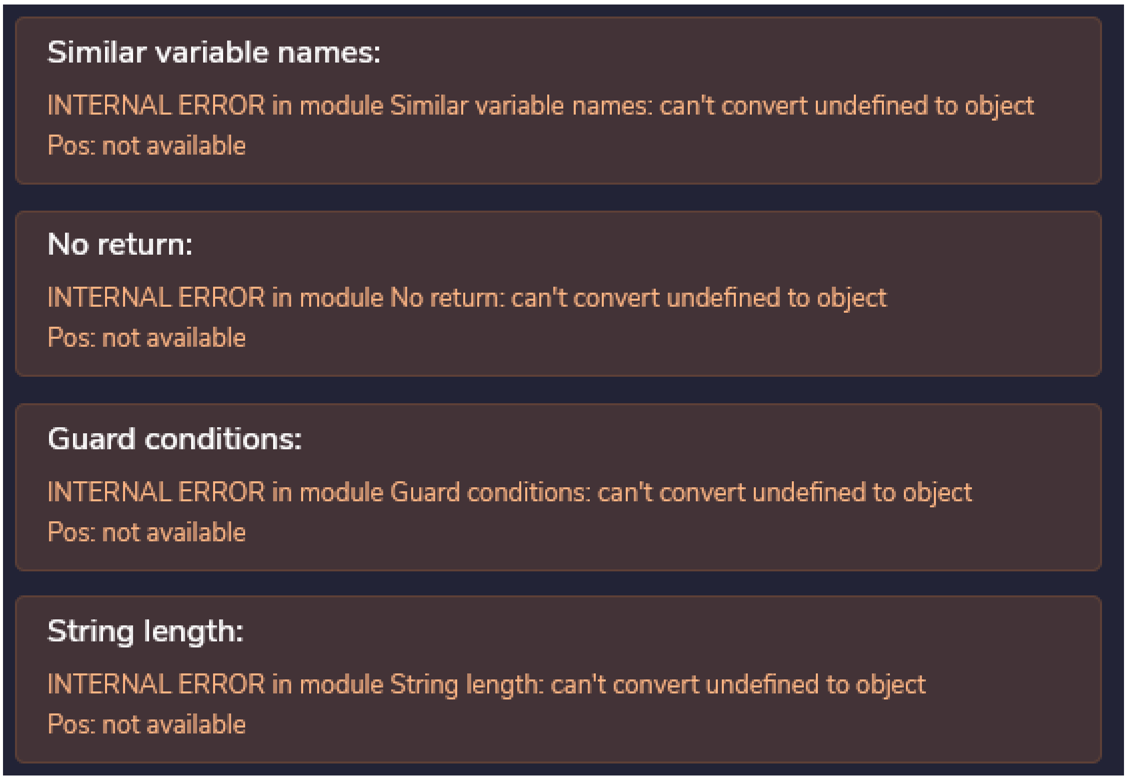
ERC20:

INTERNAL ERROR in module ERC20: can't convert undefined to object
Pos: not available

Miscellaneous

Constant/View/Pure functions:

INTERNAL ERROR in module Constant/View/Pure functions: can't convert undefined to object
Pos: not available



Similar variable names:

INTERNAL ERROR in module Similar variable names: can't convert undefined to object
Pos: not available

No return:

INTERNAL ERROR in module No return: can't convert undefined to object
Pos: not available

Guard conditions:

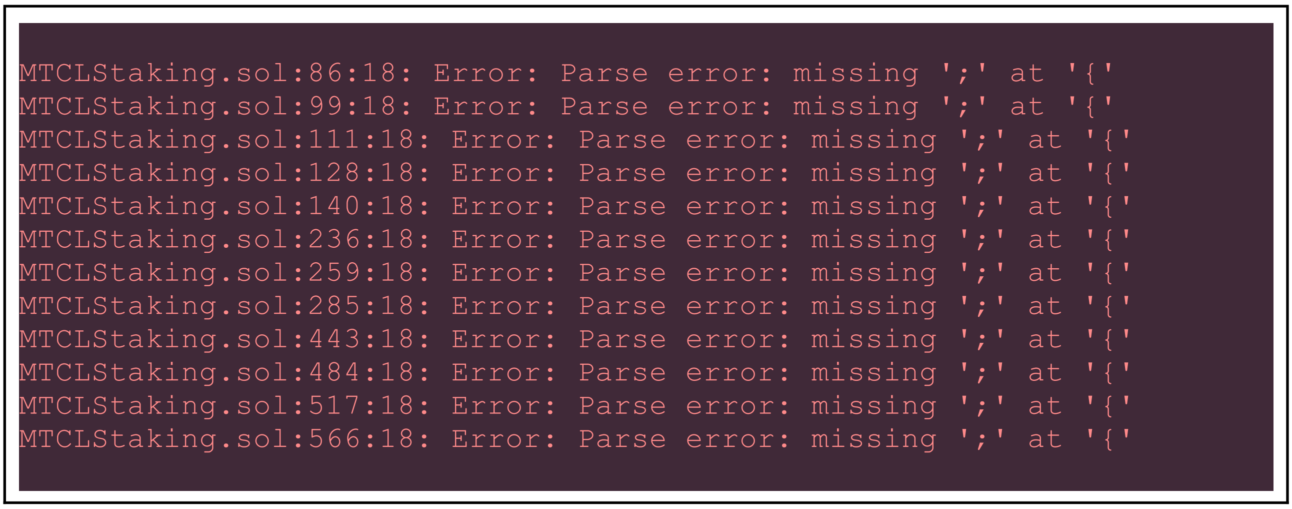
INTERNAL ERROR in module Guard conditions: can't convert undefined to object
Pos: not available

String length:

INTERNAL ERROR in module String length: can't convert undefined to object
Pos: not available

Solhint Linter

MTCLStaking



```
MTCLStaking.sol:86:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:99:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:111:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:128:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:140:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:236:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:259:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:285:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:443:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:484:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:517:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:566:18: Error: Parse error: missing ';' at '{'
```

Software analysis result:

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

EtherAuthority

Email: audit@EtherAuthority.io