Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: MaticLaunch
Platform: Polygon
Language: Solidity
Date: November 10th, 2021

Table of contents

Email: audit@EtherAuthority.io

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.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the MaticLaunch team to perform the Security audit of the MaticLaunch 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 November 10th, 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 has functionality like creating new pools, initializing new pools, stake, unstake, vote, etc.

Audit scope

Name	Code Review and Security Analysis Report for MaticLaunch Smart Contracts
Platform	**Polygon / Solidity**
File 1	MTCLFactory.sol
File 1 MD5 Hash	A181F3DF55918E90F5121B1C815C5C44
File 2	MTCLInfo.sol
File 2 MD5 Hash	BDC393A7C4FE76C8E8340023CF37BD4B
File 3	MTCLPool.sol
File 3 MD5 Hash	22E278306B0C71C07132B711C38D5B77
File 4	MTCLStaking.sol
File 4 MD5 Hash	BC873F392712904B8FAE1A6A54BFF843
Audit Date	November 10th, 2021

Claimed Smart Contracts Features

Claimed Feature Detail	Our Observation
File 1: MTCLFactory.sol • The MTCLFactory can function like creating a new pool, etc.	**YES, This is valid.**
File 2: MTCLInfo.sol • DevFeePercentage: 2% • Minimum Dev Fee In Weight: 5 ether • Minimum Investor MTCL Balance: 100 • Minimum Stake Time: 24 hours • Minimum Unstake Time: 24 hours • Pool Creator Stake: 100 • Guaranteed Allocation Time: 12 hours	**YES, This is valid.**
File 3: MTCLPool.sol • Minimum Yes Votes Threshold: 1000 • Minimum Voter MTCL Balance: 10	**YES, This is valid.**
File 4: MTCLStaking.sol • The MTCLStaking can access functions like stake, unstake, etc.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **“Secured”**. This project contract does contain owner control, which does not make it fully decentralized.

We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 1 high, 0 medium and 4 low and some very low level issues.
A high and some low level issues have been resolved / acknowledged.

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

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	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	Moderated
	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**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 4 smart contract files. Smart contracts contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in MaticLaunch 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 MaticLaunch.

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 a MaticLaunch smart contracts code in the form of a github link.The hash of that code is mentioned above in the table.

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

Use of Dependencies

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

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

AS-IS overview

MTCLFactory.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	IpV2LibPairFor	internal	Passed	No Issue
3	initializePool	internal	Passed	No Issue
4	createPool	external	Passed	No Issue

MTCLInfo.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	getMTCLFactoryAddress	external	Passed	No Issue
3	onlyFactory	modifier	Passed	No Issue
4	addPoolAddress	external	access only Factory	No Issue
5	getPoolsCount	external	Passed	No Issue
6	getPoolAddress	external	Passed	No Issue
7	getPoolCreatorMinStake	external	Passed	No Issue
8	setPoolCreatorMinStake	external	access only Owner	No Issue
9	getDevFeePercentage	external	Passed	No Issue
10	setDevFeePercentage	external	access only Owner	No Issue
11	getMinDevFeeInWei	external	Passed	No Issue
12	setMinDevFeeInWei	external	access only Owner	No Issue
13	getMinInvestorMTCLBalance	external	Passed	No Issue
14	setMinInvestorMTCLBalance	external	access only Owner	No Issue
15	getMinStakeTime	external	Passed	No Issue
16	setMinStakeTime	external	access only Owner	No Issue
17	getMinUnstakeTime	external	Passed	No Issue
18	setMinUnstakeTime	external	access only Owner	No Issue
19	getGuaranteedAllocationTime	external	Passed	No Issue
20	setGuaranteedAllocationTime	external	access only Owner	No Issue
21	getDexRouter	external	Passed	No Issue

22	setDexRouter	external	access only Owner	No Issue
23	getDexFactory	external	Passed	No Issue
24	setDexFactory	external	access only Owner	No Issue
25	getUSDT	external	Passed	No Issue
26	setUSDT	external	access only Owner	No Issue
27	owner	read	Passed	No Issue
28	onlyOwner	modifier	Passed	No Issue
29	renounceOwnership	write	access only Owner	No Issue
30	transferOwnership	write	access only Owner	No Issue
31	_transferOwnership	internal	Passed	No Issue

MTCLPool.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyMTCLOwner	modifier	Passed	No Issue
3	onlyPoolCreatorOrMTCLFactoryOrMTCLOwner	modifier	Passed	No Issue
4	onlyPoolCreatorOrMTCLOwner	modifier	Passed	No Issue
5	poolIsNotCancelled	modifier	Passed	No Issue
6	investorOnly	modifier	Passed	No Issue
7	notYetClaimedOrRefunded	modifier	Passed	No Issue
8	votesPassed	modifier	Passed	No Issue
9	setAddressInfo	external	Passed	No Issue
10	setGeneralInfo	external	Passed	No Issue
11	setDexInfo	external	Passed	No Issue
12	setStringInfo	external	Function input parameters lack of check	Refer Audit Findings
13	setMTCLInfo	external	Function input parameters lack of check	Refer Audit Findings
14	setVestingInfo	external	access only PoolCreatorOrMTCLOwner	No Issue
15	setMTCLDevFeesExempted	external	access only MTCL Owner	No Issue
16	setWhitelistedAddressesAllowed	external	access only MTCL Owner	No Issue

17	setMTCLOwnerApproved	external	access only MTCL Owner	No Issue
18	setMultiplier	external	access only MTCL Owner	No Issue
19	setminVoterMTCLBalance	external	access only MTCL Owner	No Issue
20	setMinYesVotesThreshold	external	access only MTCL Owner	No Issue
21	addWhitelistedAddresses	external	Infinite loops possibility	Refer Audit Findings
22	getGuaranteedInvestAmount	write	Passed	No Issue
23	invest	write	Passed	No Issue
24	setAllowClaim	external	access only MTCL Owner	No Issue
25	addLiquidityAndLockLPTokens	external	Passed	No Issue
26	vote	external	access only for poolIsNotCancelled	No Issue
27	claimInitial	external	access by investor Only	No Issue
28	claimVestingTokens	external	access by investor Only	No Issue
29	getRefund	external	access by investor Only	No Issue
30	cancelAndTransferTokensToPresaleCreator	external	Passed	No Issue
31	collectFundsRaised	external	access only PoolCreatorOrMTCLOwner	No Issue
32	burnUnsoldTokens	external	access only PoolCreatorOrMTCLOwner	No Issue

MTCLStaking.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	stake	write	Passed	No Issue
3	unstake	external	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.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

(1) User can unstake any time for multiple stakes: - **MTCLStaking.sol**

As the staking amount gets added but unstaketime does not change every time, the user can take any time once he does for the first stake.

Status: **Fixed**

Medium

No Medium severity vulnerabilities were found.

Low

(1) Function input parameters lack of check: - **MTCLPool.sol**

Some functions require validation before execution.

Functions are:

- setVestingInfo,
- setMTCLInfo,
- setStringInfo

Resolution: We suggest using validation like for numerical variables that should be greater than 0 and for address type check variables that are not address(0). For percentage type variables, values should have some range like minimum 0 and maximum 100.

Status: **Partially Fixed.**

(2) Division before multiplication: - **MTCLPool.sol**

In claimVestingTokens function, at line number 570 - division before multiplication has been done.

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

Resolution: Consider ordering multiplication before division.

Status: **Fixed**

(3) Function input parameters lack of check: - **MTCLStaking.sol**

In the unstake function, _burnFeePercent did not get validated and allowed to enter value > 100.

Resolution: _burnFeePercent must be validated. Token should be greater than 0 and checked before transferring to the user's wallet.

Status: **Fixed**

(4) Infinite loops possibility: - **MTCLPool.sol**

As array elements will increase, then it will cost more and more gas. And eventually, it will stop all the functionality. After several hundreds of transactions, all those functions depending on it will stop. We suggest avoiding loops. For example, use mapping to store the array index. And query that data directly, instead of looping through all the elements to find an element.

Resolution: Adjust logic to replace loops with mapping or other code structure.

Write Function:

- addWhitelistedAddresses() - _whitelistedAddresses.length

Status: **Acknowledged**

Very Low / Informational / Best practices:

(1) Missing/Ambiguous error message: - **MTCLPool.sol**

There are many places where validation has been done using require, but at some places error messages are missing and at some places ambiguous error messages given.

Resolution: We suggest setting relevant error messages to get the failure of the transaction.

Status: **Acknowledged**

(2) Owner functions:- **MTCLINFO.sol**

Many of the functions are callable from the owner only.

Resolution: Owner has to keep his private key very secure.

Status: **Acknowledged**

(3) Variable set but not used anywhere: - **MTCLStaking.sol**

LastUnstakedTimestamp is set but not used anywhere.

Resolution: We suggest removing unused variables.
Status: **Acknowledged**

Centralization

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

- setMTCLFactoryAddress: The MTCLInfo factory owner can set MTCL factory address.
- addPoolAddress: The MTCLInfo factory owner can add a pool address.
- setPoolCreatorMinStake: The MTCLInfo owner can set pool creator minimum stake.
- setDevFeePercentage: The MTCLInfo owner can set developer fee percentage.
- setMinDevFeeInWei: The MTCLInfo owner can set a minimum developer fee weight.
- setMinInvestorMTCLBalance: The MTCLInfo owner can set a minimum investor MTCL balance.
- setMinStakeTime: The MTCLInfo owner can set a minimum stake time.
- setMinUnstakeTime: The MTCLInfo owner can set a minimum unstake time.
- setGuaranteedAllocationTime: The MTCLInfo owner can set a guaranteed allocation time.
- setDexFactory: The MTCLInfo owner can set a dex factory.
- setUSDT: The MTCLInfo owner can set USDT.
- setAddressInfo: The MTCLPool PoolCreator owner can set address information.
- setGeneralInfo: The MTCLPool PoolCreator owner can set general information.
- setDexInfo: The MTCLPool PoolCreator owner can set dex information.
- setStringInfo: The MTCLPool PoolCreator owner can set string information.
- setMTCLInfo: The MTCLPool MTCL Owner can set MTCL information.
- setVestingInfo: The MTCLPool PoolCreator can set vesting information.

- setMTCLDevFeesExempted: The MTCLPool PoolCreator can set MTCL developer fees exempted.
- setWhitelistedAddressesAllowed: The MTCLPool PoolCreator owner can set whitelist addresses allowed.
- setMTCLOwnerApproved: The MTCLPool MTCLOwner owner can set MTCL owner approved.
- setMultiplier: The MTCLPool MTCLOwner owner can set MTCL multiplier.
- setminVoterMTCLBalance: The MTCLPool MTCLOwner owner can set a minimum voter MTCL balance.
- setMinYesVotesThreshold: The MTCLPool MTCLOwner owner can set a minimum voter threshold yes.
- .addWhitelistedAddresses: The MTCLPool MTCLOwner owner can add whitelist addresses.
- setAllowClaim: The MTCLPool MTCLOwner owner can set allow claims.
- claimInitial: The MTCLPool investor owner can claim initialization.
- claimVestingTokens: The MTCLPool investor owner can claim vesting tokens.
- getRefund: The MTCLPool investor owner can get a refund.
- burnUnsoldTokens: The MTCLPool PoolCreator owner can burn unsold tokens.
- collectFundsRaised: The MTCLPool PoolCreator owner can collect funds raised.

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, but they 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.

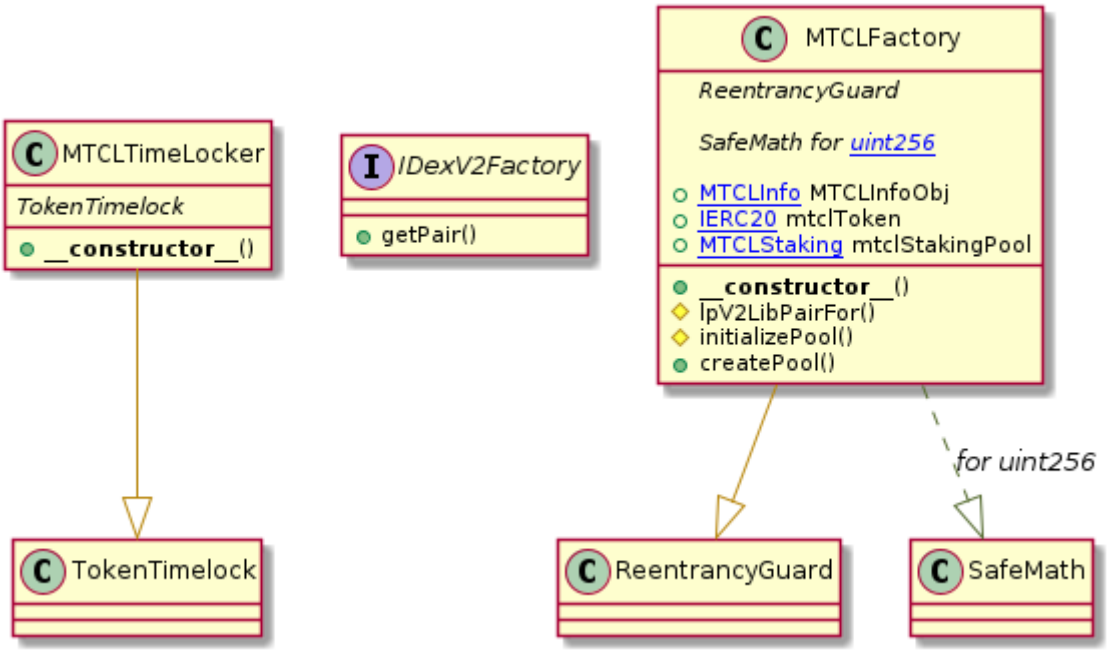
Technical Disclaimer

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.

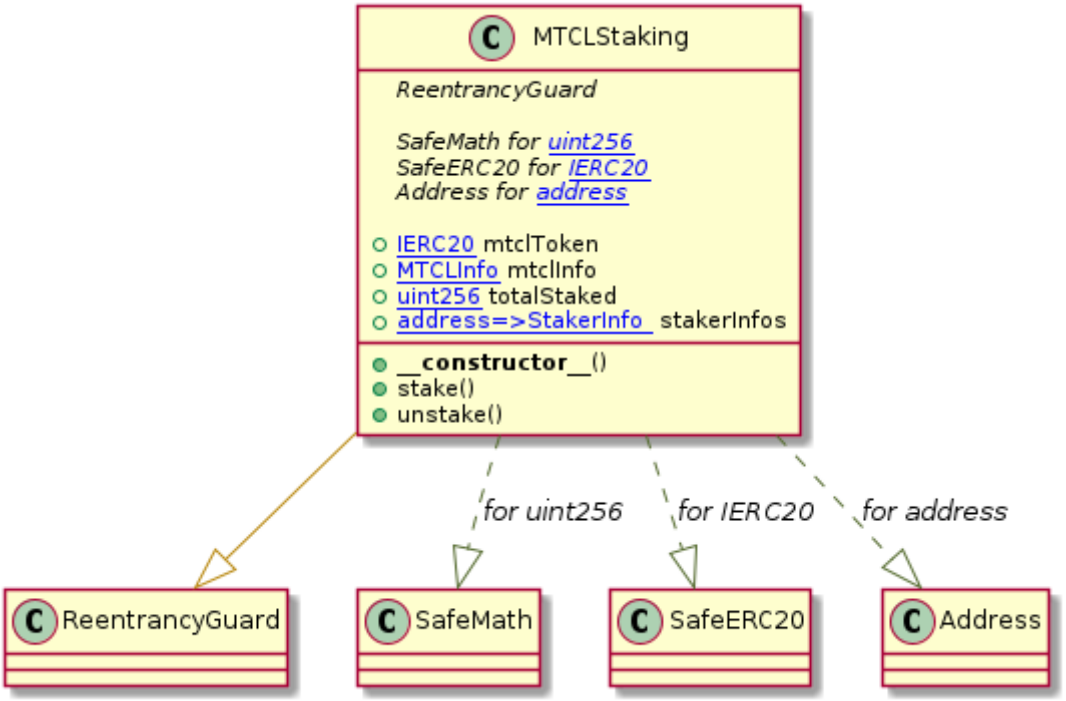
Appendix

Code Flow Diagram - MaticLaunch

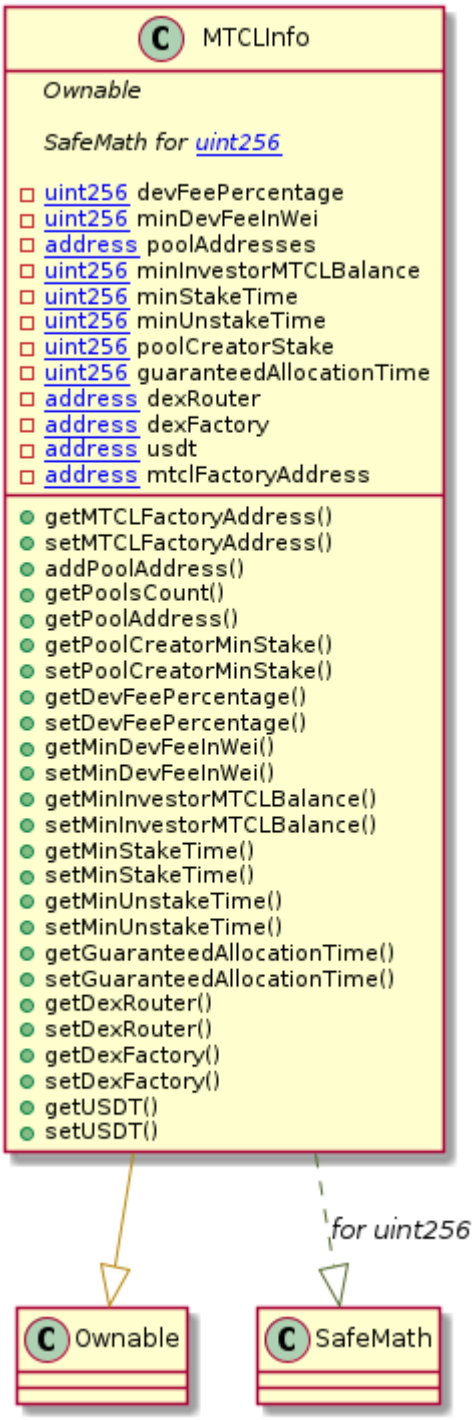
MTCLFactory Diagram



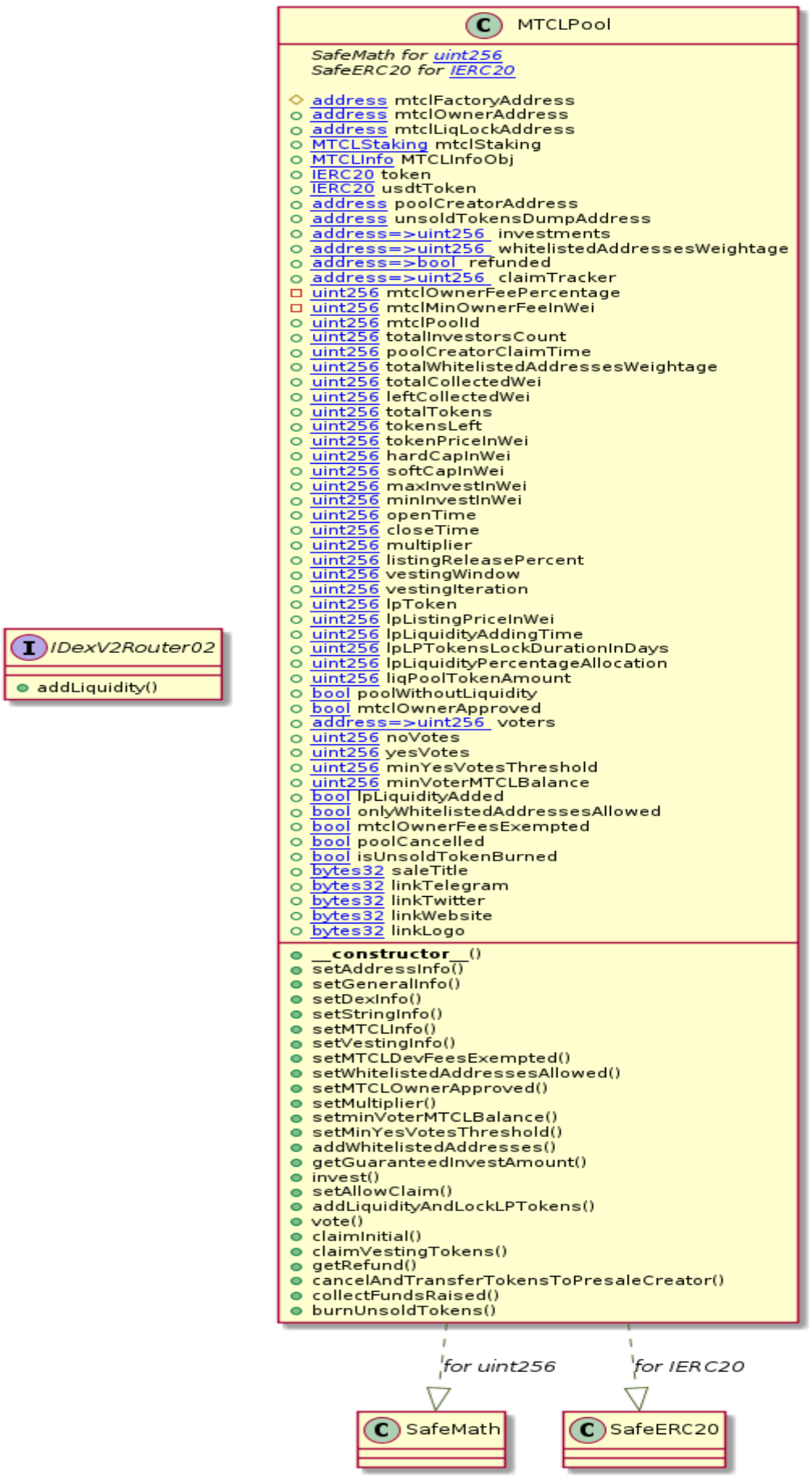
MTCLStaking Diagram



MTCLInfo Diagram



MTCLPool Diagram



Slither Results Log

Slither log >> MTCLFactory.sol

```
INFO:Detectors:
Reentrancy in MTCLFactory.createPool(MTCLFactory.PoolInfo,MTCLFactory.PoolDexInfo,MTCLFactory.PoolStringInfo,MTCLFactory.PoolVestingInfo)
 (MTCLFactory.sol#1769-1857):
        External calls:
        - require(bool)(token.transferFrom(msg.sender,address(pool),requiredTokenAmount)) (MTCLFactory.sol#1808-1810)
        - initializePool(pool,maxTokensToBeSold,maxLiqPoolTokenAmount,_info,_lpInfo,_stringInfo) (MTCLFactory.sol#1812-1819)
                - _pool.setAddressInfo(msg.sender,_info.tokenAddress,MTCLInfoObj.getUSDT(),_info.unsoldTokensDumpAddress) (MTCLFactory.so
l#1736-1741)
                - _pool.setGeneralInfo(_totalTokens,_lpToken,_info.tokenPriceInWei,_info.hardCapInWei,_info.softCapInWei,_info.maxInvestI
nWei,_info.minInvestInWei,_info.openTime,_info.closeTime) (MTCLFactory.sol#1742-1752)
                - _pool.setDexInfo(_lpInfo.listingPriceInWei,_lpInfo.liquidityAddingTime,_lpInfo.lpTokensLockDurationInDays,_lpInfo.liqui
dityPercentageAllocation,_lpInfo.poolWithoutLiquidity) (MTCLFactory.sol#1753-1759)
                - _pool.setStringInfo(_stringInfo.saleTitle,_stringInfo.linkTelegram,_stringInfo.linkTwitter,_stringInfo.linkWebsite,_str
ingInfo.linkLogo) (MTCLFactory.sol#1760-1766)
        - mtclPoolId = MTCLInfoObj.addPoolAddress(address(pool)) (MTCLFactory.sol#1841)
        - pool.setMTCLInfo(address(liquidityLock),MTCLInfoObj.getDevFeePercentage(),MTCLInfoObj.getMinDevFeeInWei(),mtclPoolId,address(mt
clStakingPool)) (MTCLFactory.sol#1842-1848)
        - pool.setVestingInfo(_vestingInfo.listingReleasePercent,_vestingInfo.vestingWindow,_vestingInfo.vestingIteration) (MTCLFactory.s
ol#1850-1854)
        Event emitted after the call(s):
        - PoolCreated(_stringInfo.saleTitle,mtclPoolId,msg.sender) (MTCLFactory.sol#1856)
Reentrancy in MTCLStaking.stake(uint256) (MTCLFactory.sol#924-939):
        External calls:
        - mtclToken.safeTransferFrom(msg.sender,address(this),_amount) (MTCLFactory.sol#930)
        Event emitted after the call(s):
        - Staked(msg.sender,_amount) (MTCLFactory.sol#938)
Reentrancy in MTCLStaking.unstake(uint256,uint256) (MTCLFactory.sol#941-977):
        External calls:
        - mtclToken.transfer(address(0x000000000000000000000000000000000000dEaD),burnAmount) (MTCLFactory.sol#969-972)
        - mtclToken.safeTransfer(msg.sender,_amount) (MTCLFactory.sol#975)
        Event emitted after the call(s):
        - Unstaked(msg.sender,_amount) (MTCLFactory.sol#976)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
TokenTimelock.constructor(IERC20,address,uint256) (MTCLFactory.sol#597-606) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(releaseTime_ > block.timestamp,TokenTimelock: release time is before current time) (MTCLFactory.sol#602)
TokenTimelock.release() (MTCLFactory.sol#632-639) uses timestamp for comparisons
```

```
Parameter MTCLPool.setMTCLInfo(address,uint256,uint256,uint256,address)._mtclMinOwnerFeeInWei (MTCLFactory.sol#1226) is not in mixedCase
Parameter MTCLPool.setMTCLInfo(address,uint256,uint256,uint256,address)._mtclPoolId (MTCLFactory.sol#1227) is not in mixedCase
Parameter MTCLPool.setMTCLInfo(address,uint256,uint256,uint256,address)._mtclStaking (MTCLFactory.sol#1228) is not in mixedCase
Parameter MTCLPool.setVestingInfo(uint256,uint256,uint256)._listingReleasePercent (MTCLFactory.sol#1238) is not in mixedCase
Parameter MTCLPool.setVestingInfo(uint256,uint256,uint256)._vestingWindow (MTCLFactory.sol#1239) is not in mixedCase
Parameter MTCLPool.setVestingInfo(uint256,uint256,uint256)._vestingIteration (MTCLFactory.sol#1240) is not in mixedCase
Parameter MTCLPool.setMTCLDevFeesExempted(bool)._mtclOwnerFeesExempted (MTCLFactory.sol#1247) is not in mixedCase
Parameter MTCLPool.setWhitelistedAddressesAllowed(bool)._onlyWhitelistedAddressesAllowed (MTCLFactory.sol#1255) is not in mixedCase
Parameter MTCLPool.setMTCLOwnerApproved(bool)._mtclOwnerApproved (MTCLFactory.sol#1260) is not in mixedCase
Parameter MTCLPool.setMultiplier(uint256)._multiplier (MTCLFactory.sol#1267) is not in mixedCase
Parameter MTCLPool.setminVoterMTCLBalance(uint256)._minVoterMTCLBalance (MTCLFactory.sol#1273) is not in mixedCase
Parameter MTCLPool.setMinYesVotesThreshold(uint256)._minYesVotesThreshold (MTCLFactory.sol#1281) is not in mixedCase
Parameter MTCLPool.addWhitelistedAddresses(address[],uint256[])._whitelistedAddresses (MTCLFactory.sol#1290) is not in mixedCase
Parameter MTCLPool.addWhitelistedAddresses(address[],uint256[])._weightage (MTCLFactory.sol#1291) is not in mixedCase
Parameter MTCLPool.invest(uint256)._investmentAmount (MTCLFactory.sol#1342) is not in mixedCase
Variable MTCLPool.MTCLInfoObj (MTCLFactory.sol#1008) is not in mixedCase
Parameter MTCLFactory.initializePool(MTCLPool,uint256,uint256,MTCLFactory.PoolInfo,MTCLFactory.PoolDexInfo,MTCLFactory.PoolStringInfo)._p
ool (MTCLFactory.sol#1729) is not in mixedCase
Parameter MTCLFactory.initializePool(MTCLPool,uint256,uint256,MTCLFactory.PoolInfo,MTCLFactory.PoolDexInfo,MTCLFactory.PoolStringInfo)._t
otalTokens (MTCLFactory.sol#1730) is not in mixedCase
Parameter MTCLFactory.initializePool(MTCLPool,uint256,uint256,MTCLFactory.PoolInfo,MTCLFactory.PoolDexInfo,MTCLFactory.PoolStringInfo)._l
pToken (MTCLFactory.sol#1731) is not in mixedCase
Parameter MTCLFactory.initializePool(MTCLPool,uint256,uint256,MTCLFactory.PoolInfo,MTCLFactory.PoolDexInfo,MTCLFactory.PoolStringInfo)._i
nfo (MTCLFactory.sol#1732) is not in mixedCase
Parameter MTCLFactory.initializePool(MTCLPool,uint256,uint256,MTCLFactory.PoolInfo,MTCLFactory.PoolDexInfo,MTCLFactory.PoolStringInfo)._l
pInfo (MTCLFactory.sol#1733) is not in mixedCase
Parameter MTCLFactory.initializePool(MTCLPool,uint256,uint256,MTCLFactory.PoolInfo,MTCLFactory.PoolDexInfo,MTCLFactory.PoolStringInfo)._s
tringInfo (MTCLFactory.sol#1734) is not in mixedCase
Parameter MTCLFactory.createPool(MTCLFactory.PoolInfo,MTCLFactory.PoolDexInfo,MTCLFactory.PoolStringInfo,MTCLFactory.PoolVestingInfo)._in
fo (MTCLFactory.sol#1770) is not in mixedCase
Parameter MTCLFactory.createPool(MTCLFactory.PoolInfo,MTCLFactory.PoolDexInfo,MTCLFactory.PoolStringInfo,MTCLFactory.PoolVestingInfo)._lp
Info (MTCLFactory.sol#1771) is not in mixedCase
```

```
Parameter MTCLFactory.createPool(MTCLFactory.PoolInfo,MTCLFactory.PoolDexInfo,MTCLFactory.PoolStringInfo,MTCLFactory.PoolVestingInfo)._lp
Info (MTCLFactory.sol#1771) is not in mixedCase
Parameter MTCLFactory.createPool(MTCLFactory.PoolInfo,MTCLFactory.PoolDexInfo,MTCLFactory.PoolStringInfo,MTCLFactory.PoolVestingInfo)._st
ringInfo (MTCLFactory.sol#1772) is not in mixedCase
Parameter MTCLFactory.createPool(MTCLFactory.PoolInfo,MTCLFactory.PoolDexInfo,MTCLFactory.PoolStringInfo,MTCLFactory.PoolVestingInfo)._ve
stingInfo (MTCLFactory.sol#1773) is not in mixedCase
Variable MTCLFactory.MTCLInfoObj (MTCLFactory.sol#1649) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable TokenTimelock._releaseTime (MTCLFactory.sol#595) is too similar to TokenTimelock.constructor(IERC20,address,uint256).releaseTime
_ (MTCLFactory.sol#600)
Variable TokenTimelock._beneficiary (MTCLFactory.sol#592) is too similar to TokenTimelock.constructor(IERC20,address,uint256).beneficiary
_ (MTCLFactory.sol#599)
Variable IDexV2Router02.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (MTCLFactory.sol#985
) is too similar to IDexV2Router02.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (MTCLFact
ory.sol#986)
Variable MTCLTimeLocker.constructor(IERC20,uint256,address)._releaseTime (MTCLFactory.sol#1632) is too similar to TokenTimelock.construct
or(IERC20,address,uint256).releaseTime_ (MTCLFactory.sol#600)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
MTCLStaking.unstake(uint256,uint256) (MTCLFactory.sol#941-977) uses literals with too many digits:
        - mtclToken.transfer(address(0x000000000000000000000000000000000000dEaD),burnAmount) (MTCLFactory.sol#969-972)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
release() should be declared external:
        - TokenTimelock.release() (MTCLFactory.sol#632-639)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (MTCLFactory.sol#685-687)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (MTCLFactory.sol#693-696)
stake(uint256) should be declared external:
        - MTCLStaking.stake(uint256) (MTCLFactory.sol#924-939)
invest(uint256) should be declared external:
        - MTCLPool.invest(uint256) (MTCLFactory.sol#1342-1392)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:MTCLFactory.sol analyzed (15 contracts with 75 detectors), 161 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> MTCLInfo.sol

```
INFO:Detectors:
MTCLInfo.setMTCLFactoryAddress(address)._newFactoryAddress (MTCLInfo.sol#686) lacks a zero-check on :
                - mtclFactoryAddress = _newFactoryAddress (MTCLInfo.sol#690)
MTCLInfo.setDexRouter(address)._dexRouter (MTCLInfo.sol#773) lacks a zero-check on :
                - dexRouter = _dexRouter (MTCLInfo.sol#777)
MTCLInfo.setDexFactory(address)._dexFactory (MTCLInfo.sol#784) lacks a zero-check on :
                - dexFactory = _dexFactory (MTCLInfo.sol#788)
MTCLInfo.setUSDT(address)._usdt (MTCLInfo.sol#795) lacks a zero-check on :
                - usdt = _usdt (MTCLInfo.sol#796)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Address.isContract(address) (MTCLInfo.sol#234-244) uses assembly
        - INLINE ASM (MTCLInfo.sol#240-242)
Address.verifyCallResult(bool,bytes,string) (MTCLInfo.sol#403-423) uses assembly
        - INLINE ASM (MTCLInfo.sol#415-418)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address.functionCall(address,bytes) (MTCLInfo.sol#287-289) is never used and should be removed
Address.functionCall(address,bytes,string) (MTCLInfo.sol#297-303) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (MTCLInfo.sol#316-322) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (MTCLInfo.sol#330-341) is never used and should be removed
Address.functionDelegateCall(address,bytes) (MTCLInfo.sol#376-378) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (MTCLInfo.sol#386-395) is never used and should be removed
Address.functionStaticCall(address,bytes) (MTCLInfo.sol#349-351) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (MTCLInfo.sol#359-368) is never used and should be removed
Address.isContract(address) (MTCLInfo.sol#234-244) is never used and should be removed
Address.sendValue(address,uint256) (MTCLInfo.sol#262-267) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (MTCLInfo.sol#403-423) is never used and should be removed
Context._msgData() (MTCLInfo.sol#590-592) is never used and should be removed
SafeERC20._callOptionalReturn(IERC20,bytes) (MTCLInfo.sol#571-581) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (MTCLInfo.sol#528-541) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (MTCLInfo.sol#552-563) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (MTCLInfo.sol#543-550) is never used and should be removed
SafeERC20.safeTransfer(IERC20,address,uint256) (MTCLInfo.sol#504-510) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (MTCLInfo.sol#512-519) is never used and should be removed
SafeMath.add(uint256,uint256) (MTCLInfo.sol#81-83) is never used and should be removed
SafeMath.div(uint256,uint256) (MTCLInfo.sol#123-125) is never used and should be removed
```

```
SafeMath.div(uint256,uint256) (MTCLInfo.sol#123-125) is never used and should be removed
SafeMath.div(uint256,uint256,string) (MTCLInfo.sol#179-188) is never used and should be removed
SafeMath.mod(uint256,uint256) (MTCLInfo.sol#139-141) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (MTCLInfo.sol#205-214) is never used and should be removed
SafeMath.mul(uint256,uint256) (MTCLInfo.sol#109-111) is never used and should be removed
SafeMath.sub(uint256,uint256) (MTCLInfo.sol#95-97) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (MTCLInfo.sol#156-165) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (MTCLInfo.sol#10-16) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (MTCLInfo.sol#52-57) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (MTCLInfo.sol#64-69) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (MTCLInfo.sol#35-45) is never used and should be removed
SafeMath.trySub(uint256,uint256) (MTCLInfo.sol#23-28) 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 (MTCLInfo.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:
Low level call in Address.sendValue(address,uint256) (MTCLInfo.sol#262-267):
        - (success) = recipient.call{value: amount}() (MTCLInfo.sol#265)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (MTCLInfo.sol#330-341):
        - (success,returndata) = target.call{value: value}(data) (MTCLInfo.sol#339)
Low level call in Address.functionStaticCall(address,bytes,string) (MTCLInfo.sol#359-368):
        - (success,returndata) = target.staticcall(data) (MTCLInfo.sol#366)
Low level call in Address.functionDelegateCall(address,bytes,string) (MTCLInfo.sol#386-395):
        - (success,returndata) = target.delegatecall(data) (MTCLInfo.sol#393)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter MTCLInfo.setMTCLFactoryAddress(address)._newFactoryAddress (MTCLInfo.sol#686) is not in mixedCase
Parameter MTCLInfo.addPoolAddress(address)._pool (MTCLInfo.sol#693) is not in mixedCase
Parameter MTCLInfo.setPoolCreatorMinStake(uint256)._minStake (MTCLInfo.sol#714) is not in mixedCase
Parameter MTCLInfo.setDevFeePercentage(uint256)._devFeePercentage (MTCLInfo.sol#722) is not in mixedCase
Parameter MTCLInfo.setMinDevFeeInWei(uint256)._minDevFeeInWei (MTCLInfo.sol#730) is not in mixedCase
Parameter MTCLInfo.setMinInvestorMTCLBalance(uint256)._minInvestorMTCLBalance (MTCLInfo.sol#738) is not in mixedCase
Parameter MTCLInfo.setMinStakeTime(uint256)._minStakeTime (MTCLInfo.sol#749) is not in mixedCase
Parameter MTCLInfo.setMinUnstakeTime(uint256)._minUnstakeTime (MTCLInfo.sol#757) is not in mixedCase
Parameter MTCLInfo.setGuaranteedAllocationTime(uint256)._guaranteedAllocationTime (MTCLInfo.sol#765) is not in mixedCase
Parameter MTCLInfo.setDexRouter(address)._dexRouter (MTCLInfo.sol#773) is not in mixedCase
```

```
Parameter MTCLInfo.setDexRouter(address)._dexRouter (MTCLInfo.sol#773) is not in mixedCase
Parameter MTCLInfo.setDexFactory(address)._dexFactory (MTCLInfo.sol#784) is not in mixedCase
Parameter MTCLInfo.setUSDT(address)._usdt (MTCLInfo.sol#795) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (MTCLInfo.sol#629-631)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (MTCLInfo.sol#637-640)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:MTCLInfo.sol analyzed (7 contracts with 75 detectors), 57 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
root@server:/chetan/gaza/mycontracts#
```

Slither log >> MTCLPool.sol

```
INFO:Detectors:
MTCLPool.setAddressInfo(address,address,address,address) (MTCLPool.sol#1076-1090) should emit an event for:
        - poolCreatorAddress = address(_poolCreator) (MTCLPool.sol#1086)
        - poolCreatorAddress = address(_poolCreator) (MTCLPool.sol#1086)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control
INFO:Detectors:
MTCLPool.setGeneralInfo(uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256) (MTCLPool.sol#1092-1128) should emit an
event for:
        - tokensLeft = _totalTokens (MTCLPool.sol#1119)
        - tokenPriceInWei = _tokenPriceInWei (MTCLPool.sol#1121)
        - hardCapInWei = _hardCapInWei (MTCLPool.sol#1122)
        - softCapInWei = _softCapInWei (MTCLPool.sol#1123)
        - maxInvestInWei = _maxInvestInWei (MTCLPool.sol#1124)
        - minInvestInWei = _minInvestInWei (MTCLPool.sol#1125)
        - openTime = _openTime (MTCLPool.sol#1126)
        - closeTime = _closeTime (MTCLPool.sol#1127)
MTCLPool.setDexInfo(uint256,uint256,uint256,uint256,bool) (MTCLPool.sol#1130-1151) should emit an event for:
        - lpLiquidityAddingTime = _lpLiquidityAddingTime (MTCLPool.sol#1147)
MTCLPool.setVestingInfo(uint256,uint256,uint256) (MTCLPool.sol#1181-1189) should emit an event for:
        - listingReleasePercent = _listingReleasePercent (MTCLPool.sol#1186)
        - vestingWindow = _vestingWindow (MTCLPool.sol#1187)
        - vestingIteration = _vestingIteration (MTCLPool.sol#1188)
MTCLPool.setMultiplier(uint256) (MTCLPool.sol#1211-1215) should emit an event for:
        - multiplier = _multiplier (MTCLPool.sol#1214)
MTCLPool.setminVoterMTCLBalance(uint256) (MTCLPool.sol#1217-1223) should emit an event for:
        - minVoterMTCLBalance = _minVoterMTCLBalance * 1e18 (MTCLPool.sol#1222)
MTCLPool.addWhitelistedAddresses(address[],uint256[]) (MTCLPool.sol#1233-1247) should emit an event for:
        - totalWhitelistedAddressesWeightage = totalWhitelistedAddressesWeightage.sub(whitelistedAddressesWeightage[_whitelistedAddresses
[i]]) (MTCLPool.sol#1239-1240)
        - totalWhitelistedAddressesWeightage = totalWhitelistedAddressesWeightage.add(_weightage[i]) (MTCLPool.sol#1244-1245)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
MTCLInfo.setMTCLFactoryAddress(address)._newFactoryAddress (MTCLPool.sol#686) lacks a zero-check on :
                - mtclFactoryAddress = _newFactoryAddress (MTCLPool.sol#690)
MTCLInfo.setDexRouter(address)._dexRouter (MTCLPool.sol#773) lacks a zero-check on :
                - dexRouter = _dexRouter (MTCLPool.sol#777)
MTCLInfo.setDexFactory(address)._dexFactory (MTCLPool.sol#784) lacks a zero-check on :
                - dexFactory = _dexFactory (MTCLPool.sol#788)
```

```
MTCLInfo.setUSDT(address)._usdt (MTCLPool.sol#795) lacks a zero-check on :
                - usdt = _usdt (MTCLPool.sol#796)
MTCLPool.setMTCLInfo(address,uint256,uint256,uint256,address)._mtclLiqLockAddress (MTCLPool.sol#1168) lacks a zero-check on :
                - mtclLiqLockAddress = _mtclLiqLockAddress (MTCLPool.sol#1174)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in MTCLPool.addLiquidityAndLockLPTokens() (MTCLPool.sol#1365-1426):
        External calls:
        - usdtToken.safeTransfer(mtclOwnerAddress,mtclDevFeeInWei) (MTCLPool.sol#1394)
        State variables written after the call(s):
        - liqPoolTokenAmount = liqPoolUSDTAmount.mul(1e18).div(lpListingPriceInWei) (MTCLPool.sol#1400-1402)
Reentrancy in MTCLPool.addLiquidityAndLockLPTokens() (MTCLPool.sol#1365-1426):
        External calls:
        - usdtToken.safeTransfer(mtclOwnerAddress,mtclDevFeeInWei) (MTCLPool.sol#1394)
        - usdtToken.approve(address(dexRouter),liqPoolUSDTAmount) (MTCLPool.sol#1408)
        - token.approve(address(dexRouter),liqPoolTokenAmount) (MTCLPool.sol#1409)
        - dexRouter.addLiquidity(address(token),address(usdtToken),liqPoolTokenAmount,liqPoolUSDTAmount,0,0,mtclLiqLockAddress,block.time
stamp.add(900)) (MTCLPool.sol#1411-1420)
        State variables written after the call(s):
        - leftCollectedWei = finalTotalCollectedWei (MTCLPool.sol#1424)
        - poolCreatorClaimTime = block.timestamp (MTCLPool.sol#1425)
Reentrancy in MTCLPool.setAllowClaim() (MTCLPool.sol#1338-1363):
        External calls:
        - usdtToken.safeTransfer(mtclOwnerAddress,mtclDevFeeInWei) (MTCLPool.sol#1358)
        State variables written after the call(s):
        - leftCollectedWei = finalTotalCollectedWei (MTCLPool.sol#1361)
        - poolCreatorClaimTime = block.timestamp (MTCLPool.sol#1362)
Reentrancy in MTCLStaking.stake(uint256) (MTCLPool.sol#868-883):
        External calls:
        - mtclToken.safeTransferFrom(msg.sender,address(this),_amount) (MTCLPool.sol#874)
        State variables written after the call(s):
        - totalStaked = totalStaked.add(_amount) (MTCLPool.sol#877)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in MTCLStaking.stake(uint256) (MTCLPool.sol#868-883):
        External calls:
        - mtclToken.safeTransferFrom(msg.sender,address(this),_amount) (MTCLPool.sol#874)
        Event emitted after the call(s):
```

```
        - Staked(msg.sender,_amount) (MTCLPool.sol#882)
Reentrancy in MTCLStaking.unstake(uint256,uint256) (MTCLPool.sol#885-921):
        External calls:
        - mtclToken.transfer(address(0x000000000000000000000000000000000000dEaD),burnAmount) (MTCLPool.sol#913-916)
        - mtclToken.safeTransfer(msg.sender,_amount) (MTCLPool.sol#919)
        Event emitted after the call(s):
        - Unstaked(msg.sender,_amount) (MTCLPool.sol#920)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
MTCLStaking.stake(uint256) (MTCLPool.sol#868-883) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(account.balance >= minStakeAmount,MTCLStaking: min stake required) (MTCLPool.sol#876)
        - account.lastUnstakedTimestamp == 0 (MTCLPool.sol#879)
MTCLStaking.unstake(uint256,uint256) (MTCLPool.sol#885-921) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(account.lastStakedTimestamp + minUnstakeTime <= block.timestamp,Invalid unstake time) (MTCLPool.sol#891-89
4)
        - require(bool,string)(account.balance > 0,Nothing to unstake) (MTCLPool.sol#895)
MTCLPool.setMultiplier(uint256) (MTCLPool.sol#1211-1215) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(block.timestamp < openTime) (MTCLPool.sol#1212)
MTCLPool.getGuaranteedInvestAmount() (MTCLPool.sol#1249-1284) uses timestamp for comparisons
        Dangerous comparisons:
        - lastStakedTimestamp + minStakeTime <= block.timestamp (MTCLPool.sol#1270)
MTCLPool.invest(uint256) (MTCLPool.sol#1286-1336) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= openTime,a) (MTCLPool.sol#1291)
        - require(bool,string)(block.timestamp < closeTime,b) (MTCLPool.sol#1292)
        - openTime.add(MTCLInfoObj.getGuaranteedAllocationTime()) > block.timestamp (MTCLPool.sol#1304)
MTCLPool.addLiquidityAndLockLPTokens() (MTCLPool.sol#1365-1426) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(block.timestamp >= lpLiquidityAddingTime) (MTCLPool.sol#1373)
MTCLPool.vote(bool) (MTCLPool.sol#1428-1451) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(block.timestamp < openTime) (MTCLPool.sol#1429)
        - lastStakedTimestamp + minStakeTime <= block.timestamp (MTCLPool.sol#1438)
MTCLPool.claimInitial() (MTCLPool.sol#1453-1464) uses timestamp for comparisons
        Dangerous comparisons:
```

```
        - require(bool)(claimTracker[msg.sender] == 0) (MTCLPool.sol#1455)
MTCLPool.claimVestingTokens() (MTCLPool.sol#1466-1496) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(claimTracker[msg.sender] >= 1,Initial claim required) (MTCLPool.sol#1469)
        - require(bool)(claimableVestCount > 0) (MTCLPool.sol#1487)
        - require(bool,string)(claimable > 0,Nothing to claim) (MTCLPool.sol#1494)
        - (claimableVestCount > vestingIteration) (MTCLPool.sol#1481-1483)
MTCLPool.getRefund() (MTCLPool.sol#1498-1520) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= openTime,Not opened) (MTCLPool.sol#1500)
        - require(bool,string)(block.timestamp >= closeTime,Not closed) (MTCLPool.sol#1501)
MTCLPool.collectFundsRaised() (MTCLPool.sol#1543-1551) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(block.timestamp >= poolCreatorClaimTime) (MTCLPool.sol#1546)
MTCLPool.burnUnsoldTokens() (MTCLPool.sol#1553-1569) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(block.timestamp >= poolCreatorClaimTime) (MTCLPool.sol#1556)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (MTCLPool.sol#235-245) uses assembly
        - INLINE ASM (MTCLPool.sol#241-243)
Address.verifyCallResult(bool,bytes,string) (MTCLPool.sol#404-424) uses assembly
        - INLINE ASM (MTCLPool.sol#416-419)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
MTCLPool.addWhitelistedAddresses(address[],uint256[]) (MTCLPool.sol#1233-1247) has costly operations inside a loop:
        - totalWhitelistedAddressesWeightage = totalWhitelistedAddressesWeightage.sub(whitelistedAddressesWeightage[_whitelistedAddresses
[i]]) (MTCLPool.sol#1239-1240)
MTCLPool.addWhitelistedAddresses(address[],uint256[]) (MTCLPool.sol#1233-1247) has costly operations inside a loop:
        - totalWhitelistedAddressesWeightage = totalWhitelistedAddressesWeightage.add(_weightage[i]) (MTCLPool.sol#1244-1245)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
Address.functionCall(address,bytes) (MTCLPool.sol#288-290) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (MTCLPool.sol#317-323) is never used and should be removed
Address.functionDelegateCall(address,bytes) (MTCLPool.sol#377-379) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (MTCLPool.sol#387-396) is never used and should be removed
Address.functionStaticCall(address,bytes) (MTCLPool.sol#350-352) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (MTCLPool.sol#360-369) is never used and should be removed
```

```
INFO:Detectors:
Pragma version^0.8.0 (MTCLPool.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:
Low level call in Address.sendValue(address,uint256) (MTCLPool.sol#263-268):
        - (success) = recipient.call{value: amount}() (MTCLPool.sol#266)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (MTCLPool.sol#331-342):
        - (success,returndata) = target.call{value: value}(data) (MTCLPool.sol#340)
Low level call in Address.functionStaticCall(address,bytes,string) (MTCLPool.sol#360-369):
        - (success,returndata) = target.staticcall(data) (MTCLPool.sol#367)
Low level call in Address.functionDelegateCall(address,bytes,string) (MTCLPool.sol#387-396):
        - (success,returndata) = target.delegatecall(data) (MTCLPool.sol#394)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter MTCLInfo.setMTCLFactoryAddress(address)._newFactoryAddress (MTCLPool.sol#686) is not in mixedCase
Parameter MTCLInfo.addPoolAddress(address)._pool (MTCLPool.sol#693) is not in mixedCase
Parameter MTCLInfo.setPoolCreatorMinStake(uint256)._minStake (MTCLPool.sol#714) is not in mixedCase
Parameter MTCLInfo.setDevFeePercentage(uint256)._devFeePercentage (MTCLPool.sol#722) is not in mixedCase
Parameter MTCLInfo.setMinDevFeeInWei(uint256)._minDevFeeInWei (MTCLPool.sol#730) is not in mixedCase
Parameter MTCLInfo.setMinInvestorMTCLBalance(uint256)._minInvestorMTCLBalance (MTCLPool.sol#738) is not in mixedCase
Parameter MTCLInfo.setMinStakeTime(uint256)._minStakeTime (MTCLPool.sol#749) is not in mixedCase
Parameter MTCLInfo.setMinUnstakeTime(uint256)._minUnstakeTime (MTCLPool.sol#757) is not in mixedCase
Parameter MTCLInfo.setGuaranteedAllocationTime(uint256)._guaranteedAllocationTime (MTCLPool.sol#765) is not in mixedCase
Parameter MTCLInfo.setDexRouter(address)._dexRouter (MTCLPool.sol#773) is not in mixedCase
Parameter MTCLInfo.setDexFactory(address)._dexFactory (MTCLPool.sol#784) is not in mixedCase
Parameter MTCLInfo.setUSDT(address)._usdt (MTCLPool.sol#795) is not in mixedCase
Parameter MTCLStaking.stake(uint256)._amount (MTCLPool.sol#868) is not in mixedCase
Parameter MTCLStaking.unstake(uint256,uint256)._amount (MTCLPool.sol#885) is not in mixedCase
Parameter MTCLStaking.unstake(uint256,uint256)._burnFeePercent (MTCLPool.sol#885) is not in mixedCase
Parameter MTCLPool.setAddressInfo(address,address,address,address)._poolCreator (MTCLPool.sol#1077) is not in mixedCase
Parameter MTCLPool.setAddressInfo(address,address,address,address)._tokenAddress (MTCLPool.sol#1078) is not in mixedCase
Parameter MTCLPool.setAddressInfo(address,address,address,address)._usdtTokenAddress (MTCLPool.sol#1079) is not in mixedCase
Parameter MTCLPool.setAddressInfo(address,address,address,address)._unsoldTokensDumpAddress (MTCLPool.sol#1080) is not in mixedCase
Parameter MTCLPool.setGeneralInfo(uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256)._totalTokens (MTCLPool.sol#109
3) is not in mixedCase
Parameter MTCLPool.setGeneralInfo(uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256)._lpToken (MTCLPool.sol#1094) i
s not in mixedCase
```

```
Parameter MTCLPool.setMTCLInfo(address,uint256,uint256,uint256,address)._mtclPoolId (MTCLPool.sol#1171) is not in mixedCase
Parameter MTCLPool.setMTCLInfo(address,uint256,uint256,uint256,address)._mtclStaking (MTCLPool.sol#1172) is not in mixedCase
Parameter MTCLPool.setVestingInfo(uint256,uint256,uint256)._listingReleasePercent (MTCLPool.sol#1182) is not in mixedCase
Parameter MTCLPool.setVestingInfo(uint256,uint256,uint256)._vestingWindow (MTCLPool.sol#1183) is not in mixedCase
Parameter MTCLPool.setVestingInfo(uint256,uint256,uint256)._vestingIteration (MTCLPool.sol#1184) is not in mixedCase
Parameter MTCLPool.setMTCLDevFeesExempted(bool)._mtclOwnerFeesExempted (MTCLPool.sol#1191) is not in mixedCase
Parameter MTCLPool.setWhitelistedAddressesAllowed(bool)._onlyWhitelistedAddressesAllowed (MTCLPool.sol#1199) is not in mixedCase
Parameter MTCLPool.setMTCLOwnerApproved(bool)._mtclOwnerApproved (MTCLPool.sol#1204) is not in mixedCase
Parameter MTCLPool.setMultiplier(uint256)._multiplier (MTCLPool.sol#1211) is not in mixedCase
Parameter MTCLPool.setminVoterMTCLBalance(uint256)._minVoterMTCLBalance (MTCLPool.sol#1217) is not in mixedCase
Parameter MTCLPool.setMinYesVotesThreshold(uint256)._minYesVotesThreshold (MTCLPool.sol#1225) is not in mixedCase
Parameter MTCLPool.addWhitelistedAddresses(address[],uint256[])._whitelistedAddresses (MTCLPool.sol#1234) is not in mixedCase
Parameter MTCLPool.addWhitelistedAddresses(address[],uint256[])._weightage (MTCLPool.sol#1235) is not in mixedCase
Parameter MTCLPool.invest(uint256)._investmentAmount (MTCLPool.sol#1286) is not in mixedCase
Variable MTCLPool.MTCLInfoObj (MTCLPool.sol#952) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable IDexV2Router02.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (MTCLPool.sol#929) i
s too similar to IDexV2Router02.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (MTCLPool.so
l#930)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
MTCLStaking.unstake(uint256,uint256) (MTCLPool.sol#885-921) uses literals with too many digits:
        - mtclToken.transfer(address(0x000000000000000000000000000000000000dEaD),burnAmount) (MTCLPool.sol#913-916)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (MTCLPool.sol#629-631)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (MTCLPool.sol#637-640)
stake(uint256) should be declared external:
        - MTCLStaking.stake(uint256) (MTCLPool.sol#868-883)
invest(uint256) should be declared external:
        - MTCLPool.invest(uint256) (MTCLPool.sol#1286-1336)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:MTCLPool.sol analyzed (11 contracts with 75 detectors), 138 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
root@server:/chetan/gaza/mycontracts#
```

Slither log >> MTCLStaking.sol

```
INFO:Detectors:
MTCLInfo.setMTCLFactoryAddress(address)._newFactoryAddress (MTCLStaking.sol#688) lacks a zero-check on :
                - mtclFactoryAddress = _newFactoryAddress (MTCLStaking.sol#692)
MTCLInfo.setDexRouter(address)._dexRouter (MTCLStaking.sol#775) lacks a zero-check on :
                - dexRouter = _dexRouter (MTCLStaking.sol#779)
MTCLInfo.setDexFactory(address)._dexFactory (MTCLStaking.sol#786) lacks a zero-check on :
                - dexFactory = _dexFactory (MTCLStaking.sol#790)
MTCLInfo.setUSDT(address)._usdt (MTCLStaking.sol#797) lacks a zero-check on :
                - usdt = _usdt (MTCLStaking.sol#798)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in MTCLStaking.stake(uint256) (MTCLStaking.sol#870-885):
        External calls:
        - mtclToken.safeTransferFrom(msg.sender,address(this),_amount) (MTCLStaking.sol#876)
        State variables written after the call(s):
        - totalStaked = totalStaked.add(_amount) (MTCLStaking.sol#879)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in MTCLStaking.stake(uint256) (MTCLStaking.sol#870-885):
        External calls:
        - mtclToken.safeTransferFrom(msg.sender,address(this),_amount) (MTCLStaking.sol#876)
        Event emitted after the call(s):
        - Staked(msg.sender,_amount) (MTCLStaking.sol#884)
Reentrancy in MTCLStaking.unstake(uint256,uint256) (MTCLStaking.sol#887-923):
        External calls:
        - mtclToken.transfer(address(0x000000000000000000000000000000000000dEaD),burnAmount) (MTCLStaking.sol#915-918)
        - mtclToken.safeTransfer(msg.sender,_amount) (MTCLStaking.sol#921)
        Event emitted after the call(s):
        - Unstaked(msg.sender,_amount) (MTCLStaking.sol#922)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
MTCLStaking.stake(uint256) (MTCLStaking.sol#870-885) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(account.balance >= minStakeAmount,MTCLStaking: min stake required) (MTCLStaking.sol#878)
        - account.lastUnstakedTimestamp == 0 (MTCLStaking.sol#881)
MTCLStaking.unstake(uint256,uint256) (MTCLStaking.sol#887-923) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(account.lastStakedTimestamp + minUnstakeTime <= block.timestamp,Invalid unstake time) (MTCLStaking.sol#893
```

```
-896)
        - require(bool,string)(account.balance > 0,Nothing to unstake) (MTCLStaking.sol#897)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (MTCLStaking.sol#237-247) uses assembly
        - INLINE ASM (MTCLStaking.sol#243-245)
Address.verifyCallResult(bool,bytes,string) (MTCLStaking.sol#406-426) uses assembly
        - INLINE ASM (MTCLStaking.sol#418-421)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address.functionCall(address,bytes) (MTCLStaking.sol#290-292) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (MTCLStaking.sol#319-325) is never used and should be removed
Address.functionDelegateCall(address,bytes) (MTCLStaking.sol#379-381) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (MTCLStaking.sol#389-398) is never used and should be removed
Address.functionStaticCall(address,bytes) (MTCLStaking.sol#352-354) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (MTCLStaking.sol#362-371) is never used and should be removed
Address.sendValue(address,uint256) (MTCLStaking.sol#265-270) is never used and should be removed
Context._msgData() (MTCLStaking.sol#592-594) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (MTCLStaking.sol#531-544) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (MTCLStaking.sol#555-566) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (MTCLStaking.sol#546-553) is never used and should be removed
SafeMath.div(uint256,uint256,string) (MTCLStaking.sol#181-190) is never used and should be removed
SafeMath.mod(uint256,uint256) (MTCLStaking.sol#141-143) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (MTCLStaking.sol#207-216) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (MTCLStaking.sol#158-167) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (MTCLStaking.sol#12-18) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (MTCLStaking.sol#54-59) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (MTCLStaking.sol#66-71) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (MTCLStaking.sol#37-47) is never used and should be removed
SafeMath.trySub(uint256,uint256) (MTCLStaking.sol#25-30) 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:
Low level call in Address.sendValue(address,uint256) (MTCLStaking.sol#265-270):
        - (success) = recipient.call{value: amount}() (MTCLStaking.sol#268)
```

```
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (MTCLStaking.sol#333-344):
        - (success,returndata) = target.call{value: value}(data) (MTCLStaking.sol#342)
Low level call in Address.functionStaticCall(address,bytes,string) (MTCLStaking.sol#362-371):
        - (success,returndata) = target.staticcall(data) (MTCLStaking.sol#369)
Low level call in Address.functionDelegateCall(address,bytes,string) (MTCLStaking.sol#389-398):
        - (success,returndata) = target.delegatecall(data) (MTCLStaking.sol#396)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter MTCLInfo.setMTCLFactoryAddress(address)._newFactoryAddress (MTCLStaking.sol#688) is not in mixedCase
Parameter MTCLInfo.addPoolAddress(address)._pool (MTCLStaking.sol#695) is not in mixedCase
Parameter MTCLInfo.setPoolCreatorMinStake(uint256)._minStake (MTCLStaking.sol#716) is not in mixedCase
Parameter MTCLInfo.setDevFeePercentage(uint256)._devFeePercentage (MTCLStaking.sol#724) is not in mixedCase
Parameter MTCLInfo.setMinDevFeeInWei(uint256)._minDevFeeInWei (MTCLStaking.sol#732) is not in mixedCase
Parameter MTCLInfo.setMinInvestorMTCLBalance(uint256)._minInvestorMTCLBalance (MTCLStaking.sol#740) is not in mixedCase
Parameter MTCLInfo.setMinStakeTime(uint256)._minStakeTime (MTCLStaking.sol#751) is not in mixedCase
Parameter MTCLInfo.setMinUnstakeTime(uint256)._minUnstakeTime (MTCLStaking.sol#759) is not in mixedCase
Parameter MTCLInfo.setGuaranteedAllocationTime(uint256)._guaranteedAllocationTime (MTCLStaking.sol#767) is not in mixedCase
Parameter MTCLInfo.setDexRouter(address)._dexRouter (MTCLStaking.sol#775) is not in mixedCase
Parameter MTCLInfo.setDexFactory(address)._dexFactory (MTCLStaking.sol#786) is not in mixedCase
Parameter MTCLInfo.setUSDT(address)._usdt (MTCLStaking.sol#797) is not in mixedCase
Parameter MTCLStaking.stake(uint256)._amount (MTCLStaking.sol#870) is not in mixedCase
Parameter MTCLStaking.unstake(uint256,uint256)._amount (MTCLStaking.sol#887) is not in mixedCase
Parameter MTCLStaking.unstake(uint256,uint256)._burnFeePercent (MTCLStaking.sol#887) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
MTCLStaking.unstake(uint256,uint256) (MTCLStaking.sol#887-923) uses literals with too many digits:
        - mtclToken.transfer(address(0x000000000000000000000000000000000000dEaD),burnAmount) (MTCLStaking.sol#915-918)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (MTCLStaking.sol#631-633)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (MTCLStaking.sol#639-642)
stake(uint256) should be declared external:
        - MTCLStaking.stake(uint256) (MTCLStaking.sol#870-885)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:MTCLStaking.sol analyzed (9 contracts with 75 detectors), 59 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

MTCLFactory.sol

Security

Check-effects-interaction:

INTERNAL ERROR in module Check-effects-interaction: GA(...) is undefined
Pos: not available

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: 240:8:

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: 1612:16:

Gas costs:

Gas requirement of function MTCLFactory.createPool 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: 1769:4:

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: 1794:12:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 1808:8:

Email: audit@EtherAuthority.io

MTCLInfo.sol

Constant/View/Pure functions:

Address.functionStaticCall(address,bytes) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 350:4:

No return:

IERC20.transfer(address,uint256): Defines a return type but never explicitly returns a value.
Pos: 445:4:

No return:

IERC20.allowance(address,address): Defines a return type but never explicitly returns a value.
Pos: 454:4:

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: 416:16:

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: 266:27:

Gas & Economy

Gas costs:

Gas requirement of function MTCLInfo.addPoolAddress 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: 694:4:

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: 679:8:

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: 187:19:

MTCLPool.sol

Security

Check-effects-interaction:

INTERNAL ERROR in module Check-effects-interaction: GA(...) is undefined
Pos: not available

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: 241:8:

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: 1212:16:

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: 340:50:

Gas costs:

Gas requirement of function MTCLPool.setAddressInfo 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: 1076:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point.
Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 1238:8:

Constant/View/Pure functions:

INTERNAL ERROR in module Constant/View/Pure functions: GA(...) is undefined
Pos: not available

Similar variable names:

MTCLPool.setGeneralInfo(uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256) : Variables have very similar names "maxInvestInWei" and "minInvestInWei". Note: Modifiers are currently not considered by this static analysis.
Pos: 1124:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code).
Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 1104:8:

MTCLStaking.sol

Check-effects-interaction:

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

more

Pos: 870:4:

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: 880:38:

Gas & Economy

Gas costs:

Gas requirement of function MTCLInfo.addPoolAddress 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: 695:4:

Miscellaneous

Constant/View/Pure functions:

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

more

Pos: 26:4:

Constant/View/Pure functions:

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

more

Pos: 695:4:

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: 680:8:

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: 625:19:

Solhint Linter

MTCLFactory.sol

```
MTCLFactory.sol:11:18: Error: Parse error: missing ';' at '{'
MTCLFactory.sol:24:18: Error: Parse error: missing ';' at '{'
MTCLFactory.sol:36:18: Error: Parse error: missing ';' at '{'
MTCLFactory.sol:53:18: Error: Parse error: missing ';' at '{'
MTCLFactory.sol:65:18: Error: Parse error: missing ';' at '{'
MTCLFactory.sol:161:18: Error: Parse error: missing ';' at '{'
MTCLFactory.sol:184:18: Error: Parse error: missing ';' at '{'
MTCLFactory.sol:210:18: Error: Parse error: missing ';' at '{'
MTCLFactory.sol:557:18: Error: Parse error: missing ';' at '{'
```

MTCLInfo.sol

```
MTCLInfo.sol:12:18: Error: Parse error: missing ';' at '{'
MTCLInfo.sol:25:18: Error: Parse error: missing ';' at '{'
MTCLInfo.sol:37:18: Error: Parse error: missing ';' at '{'
MTCLInfo.sol:54:18: Error: Parse error: missing ';' at '{'
MTCLInfo.sol:66:18: Error: Parse error: missing ';' at '{'
MTCLInfo.sol:162:18: Error: Parse error: missing ';' at '{'
MTCLInfo.sol:185:18: Error: Parse error: missing ';' at '{'
MTCLInfo.sol:211:18: Error: Parse error: missing ';' at '{'
MTCLInfo.sol:558:18: Error: Parse error: missing ';' at '{'
```

MTCLPool.sol

```
MTCLPool.sol:11:18: Error: Parse error: missing ';' at '{'
MTCLPool.sol:24:18: Error: Parse error: missing ';' at '{'
MTCLPool.sol:36:18: Error: Parse error: missing ';' at '{'
MTCLPool.sol:53:18: Error: Parse error: missing ';' at '{'
MTCLPool.sol:65:18: Error: Parse error: missing ';' at '{'
MTCLPool.sol:161:18: Error: Parse error: missing ';' at '{'
MTCLPool.sol:184:18: Error: Parse error: missing ';' at '{'
MTCLPool.sol:210:18: Error: Parse error: missing ';' at '{'
MTCLPool.sol:558:18: Error: Parse error: missing ';' at '{'
```

MTCLStaking.sol

```
MTCLStaking.sol:349:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:450:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:463:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:475:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:492:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:504:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:600:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:623:18: Error: Parse error: missing ';' at '{'
MTCLStaking.sol:649:18: 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.

Email: audit@EtherAuthority.io