Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Mida Token
Platform: Ethereum
Language: Solidity
Date: April 27th, 2023

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 Mida to perform the Security audit of the Mida token smart contracts code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on April 27th, 2023.

The purpose of this audit was to address the following:

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

Project Background

- Mida Token is a NFT smart contract which has mint, bulkBurn, burn, enterMine, exitMine functionalities.
- There are 4 smart contracts, which were included in the audit scope. And there were some standard library code such as OpenZepelin, which were excluded. Because those standard library code is considered as time tested and community audited, so we can safely ignore them.

Email: audit@EtherAuthority.io

Audit scope

Name	Code Review and Security Analysis Report for Mida token Smart Contracts
Platform	**Ethereum / Solidity**
File 1	Mida.sol
File 1 MD5 Hash	06C988460522AC81D1416FD71666F3B1
Updated File 1 MD5 Hash	5A2BAF6378EF48FC48164EE1F10AA4CE
File 1 Online Code Link	0x57d8dc5eF3762395AB1E842473354CEc9Ab14f5B
File 2	MTM.sol
File 2 MD5 Hash	CF1B00D238BB75868C83F99C7D297F45
Updated File 2 MD5 Hash	2A91F1A05962CE3E93C74162EE528027
File 2 Online Code Link	0xF9ea27d6248D2c0B0b064c19E7b532BBb9fEC50a
File 3	Mineable.sol
File 3 MD5 Hash	512FB7E9C79BD90841886E612D6F9F86
File 4	MShareCalculable.sol
File 4 MD5 Hash	D98CACCD509173E821FEE96477FEAD1A
Updated File 4 MD5 Hash	76E948A6F361018251A174FD200F68BB
Audit Date	April 27th, 2023
Revised Audit Date	May 4th, 2023

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 Mida.sol • Name: Mida • Symbol: MIDA • Decimals: 6 • Total Supply: 42 trillion • Total Mineable Supply: 21 trillion • 1 MShare: 694,200	**YES, This is valid.**
File 2 MTM.sol • Name: Mida Token Miner • Symbol: MTM **Owner Specifications:** • MTM token removed from total supply by the owner. • Mint this MTM back to the owner.	**YES, This is valid.**
File 3 Mineable.sol • Init Reload Period: 5 days • Mining Duration: 30 days • Reload Period: 2 days	**YES, This is valid.**
File 4 MShareCalculable.sol • Mshare Decimals: 6	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. Also, these contracts does not contain owner control, which makes them fully decentralized.

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

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

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

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	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	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

Code Quality

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

The libraries in the Mida Protocol 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 Mida Protocol.

The Mida token team has provided unit test scripts, which 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 an Mida Protocol smart contract code in the form of a file. The hash of that code is mentioned above in the table.

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

Use of Dependencies

As per our observation, the libraries are used in this smart contracts 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

Mida.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	totalSupply	read	Passed	No Issue
3	balanceOf	read	Passed	No Issue
4	transfer	write	Passed	No Issue
5	allowance	read	Passed	No Issue
6	approve	write	Passed	No Issue
7	transferFrom	write	Passed	No Issue
8	increaseAllowance	write	Passed	No Issue
9	decreaseAllowance	write	Passed	No Issue
10	_transfer	internal	Passed	No Issue
11	_mint	internal	Passed	No Issue
12	_burn	internal	Passed	No Issue
13	_approve	internal	Passed	No Issue
14	_spendAllowance	internal	Passed	No Issue
15	_beforeTokenTransfer	internal	Passed	No Issue
16	_afterTokenTransfer	internal	Passed	No Issue
17	nonReentrant	modifier	Passed	No Issue
18	_nonReentrantBefore	write	Passed	No Issue
19	_nonReentrantAfter	write	Passed	No Issue
20	_reentrancyGuardEntered	internal	Passed	No Issue
21	minerStart	write	Passed	No Issue
21	minerEnd	write	Passed	No Issue
22	_minerStart	write	Passed	No Issue
23	_minerEnd	write	Passed	No Issue
24	_verifyMinerStart	read	Passed	No Issue
25	_verifyMinerEnd	read	Passed	No Issue
26	shouldStartMiner	read	Passed	No Issue
27	shouldEndMiner	read	Passed	No Issue
28	_timeToChangeStatus	read	Passed	No Issue
29	notMining	modifier	Passed	No Issue
30	minerStartable	modifier	Passed	No Issue
31	minerEndable	modifier	Passed	No Issue
32	onlyReload	modifier	Passed	No Issue
33	maxSupplyNotReached	modifier	Passed	No Issue
34	maxSupply	write	Passed	No Issue
35	mineableSupply	write	Passed	No Issue
36	enterMine	external	Passed	No Issue
37	continueMining	external	Passed	No Issue
38	exitMine	external	Passed	No Issue
39	_enterMine	internal	Passed	No Issue

40	_reEnterMine	internal	Passed	No Issue
41	_claimRewards	internal	Passed	No Issue
42	_retreiveMtms	internal	Passed	No Issue
43	_verifyRewards	internal	Passed	No Issue

MTM.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	nonReentrant	modifier	Passed	No Issue
3	_nonReentrantBefore	write	Passed	No Issue
4	_nonReentrantAfter	write	Passed	No Issue
5	_reentrancyGuardEntered	internal	Passed	No Issue
6	supportsInterface	read	Passed	No Issue
7	balanceOf	read	Passed	No Issue
8	ownerOf	read	Passed	No Issue
9	name	read	Passed	No Issue
10	symbol	read	Passed	No Issue
11	tokenURI	read	Passed	No Issue
12	_baseURI	internal	Passed	No Issue
13	approve	write	Passed	No Issue
14	getApproved	read	Passed	No Issue
15	setApprovalForAll	write	Passed	No Issue
16	isApprovedForAll	read	Passed	No Issue
17	transferFrom	write	Passed	No Issue
18	safeTransferFrom	write	Passed	No Issue
19	safeTransferFrom	write	Passed	No Issue
20	_safeTransfer	internal	Passed	No Issue
21	_ownerOf	internal	Passed	No Issue
21	_exists	internal	Passed	No Issue
22	_isApprovedOrOwner	internal	Passed	No Issue
23	_safeMint	internal	Passed	No Issue
24	_safeMint	internal	Passed	No Issue
25	_mint	internal	Passed	No Issue
26	_burn	internal	Passed	No Issue
27	_transfer	internal	Passed	No Issue
28	_approve	internal	Passed	No Issue
29	_setApprovalForAll	internal	Passed	No Issue
30	_requireMinted	internal	Passed	No Issue
31	_checkOnERC721Received	write	Passed	No Issue
32	_beforeTokenTransfer	internal	Passed	No Issue
33	_afterTokenTransfer	internal	Passed	No Issue
34	__unsafe_increaseBalance	internal	Passed	No Issue
35	isTokenAvailableForMTM	read	Passed	No Issue
36	_tokenToUSDPrice	internal	Passed	No Issue

37	_getTokenAddressFromPair	read	Passed	No Issue
38	_getUSDPrice	read	Passed	No Issue
39	onlyOwner	modifier	Passed	No Issue
40	owner	external	Passed	No Issue
41	calcTokenAmt	read	Function input parameters lack of check	Refer Audit Findings
42	calcMTMPoints	read	Function input parameters lack of check	Refer Audit Findings
43	mint	external	Passed	No Issue
44	_mintSingle	write	Passed	No Issue
45	bulkBurn	external	Passed	No Issue
46	burn	write	Passed	No Issue
47	enterMine	external	access only Owner	No Issue
48	exitMine	external	access only Owner	No Issue
49	_mtmMintIsOver	internal	Passed	No Issue
50	tokenURI	read	Passed	No Issue
51	_substr	write	Passed	No Issue

MShareCalculable.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	isTokenAvailableForMTM	read	Passed	No Issue
3	_tokenToUSDPrice	internal	Passed	No Issue
4	_getTokenAddressFromPair	read	Passed	No Issue
5	_getUSDPrice	read	Passed	No Issue
6	getV2TokenPrice	read	Passed	No Issue
7	ethV2Price	internal	Passed	No Issue
8	getV2TokenAddressFromPair	internal	Passed	No Issue
9	getV3TokenPrice	internal	Passed	No Issue
10	etherV3Price	internal	Passed	No Issue
11	getV3TokenAddressFromPair	internal	Passed	No Issue
12	getSqrtTwapX96	internal	Passed	No Issue
13	getPriceX96FromSqrtPriceX96	internal	Passed	No Issue

Mineable.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	nonReentrant	modifier	Passed	No Issue
3	_nonReentrantBefore	write	Passed	No Issue
4	_nonReentrantAfter	write	Passed	No Issue
5	_reentrancyGuardEntered	internal	Passed	No Issue
6	minerStart	write	Passed	No Issue
7	minerEnd	write	Passed	No Issue
8	_minerStart	write	Passed	No Issue
9	_minerEnd	write	Passed	No Issue
10	_verifyMinerStart	read	Passed	No Issue
11	_verifyMinerEnd	read	Passed	No Issue
12	shouldStartMiner	read	Passed	No Issue
13	shouldEndMiner	read	Passed	No Issue
14	_timeToChangeStatus	read	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 in the revised contract code.

High Severity

No high severity vulnerabilities were found in the revised contract code.

Medium

No medium severity vulnerabilities were found in the revised contract code.

Low

(1) In the _claimRewards() function, "_ownerMiner.rewards" should be stored in a local variable, and do "_ownerMiner.rewards = uint128(0);" before "_mint(_msgSender(), _ownerMiner.rewards);" for good practice. **- Mida.sol**

Function: _claimRewards()

```
function _claimRewards() internal {
  OwnerMiner storage _ownerMiner = ownerMiner[_msgSender()];

  if(_ownerMiner.rewards == 0) {
  }

  _mint(_msgSender(), _ownerMiner.rewards);
  pendingMidaMined -= _ownerMiner.rewards;
  _ownerMiner.rewards = uint128(0);
  _ownerMiner.periodId = uint56(0);
}
```

Status: Fixed

(2) "_verifyRewards()" requires null validation for "uint128 midaMining" to save gas consumption.**- Mida.sol**

Status: Fixed

(3) Emit Events for all functions.**- Mida.sol, MTM.sol**

Status: Fixed. Events have been emitted for Mida contract only.

(4) Multiplication before division: **MShareCalculable.sol**

In the "_tokenToUSDPrice()", line 65 should do multiplication before division.

Status: Fixed

(5) Possible gas consuming loop: **- Mida.sol**

Function: _enterMine()Status: Fixed

```
function _enterMine(uint[] memory mtmIds) internal {
  OwnerMiner storage _ownerMiner = ownerMiner[_msgSender()];
  uint128 totalMPoints;
  for(uint i; i < mtmIds.length;) {
    if(_mtm.ownerOf(mtmIds[i]) != _msgSender()) {
    }
    (, uint128 mPoints, ) = _mtm.mtmStorage(mtmIds[i]);
    totalMPoints += uint128(mPoints);
    _mtm.enterMine(mtmIds[i]);
    _ownerMiner.mtmIds.push(mtmIds[i]);
    unchecked {
      i++;
    }
  }
  uint128 midaMining = _verifyRewards(totalMPoints * MSHARE_RATE);
  _ownerMiner.rewards += midaMining;
  _ownerMiner.periodId = currentPeriod.periodId;
}
```

In the "_enterMine()" function, "mtmIds[i]" can be stored in a local variable and used multiple times to save gas consumption.

Status: Fixed

(6) Function input parameters lack of check: **- MTM.sol**

"calcTokenAmt()" and calcMTMPoints() functions check requires input parameters.

Status: Acknowledged

Very Low / Informational / Best practices:

No Informational severity vulnerabilities were found in the revised contract code.

Centralization

MTM smart contract is owned by Mida and Mida does not have any owner functions

So there is no centralization issue.

Conclusion

We were given a contract code in the form of a file. And we have used all possible tests based on given objects as files. We had observed 6 low severity issues in the smart contracts. All the issues have been resolved / acknowledged in the revised code. **So, the smart contracts are ready for the mainnet deployment**.

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

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

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

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

Our Methodology

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

Manual Code Review:

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

Vulnerability Analysis:

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

Documenting Results:

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

Suggested Solutions:

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

Disclaimers

EtherAuthority.io Disclaimer

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

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

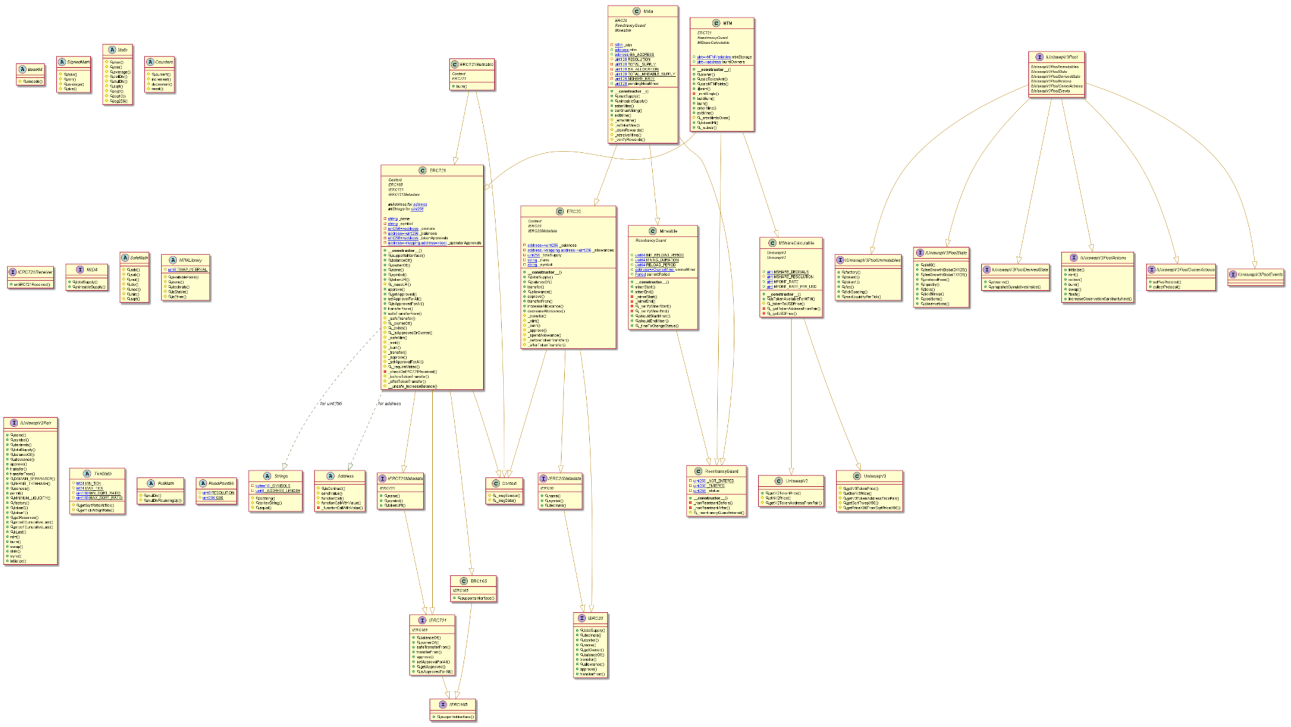
Technical Disclaimer

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

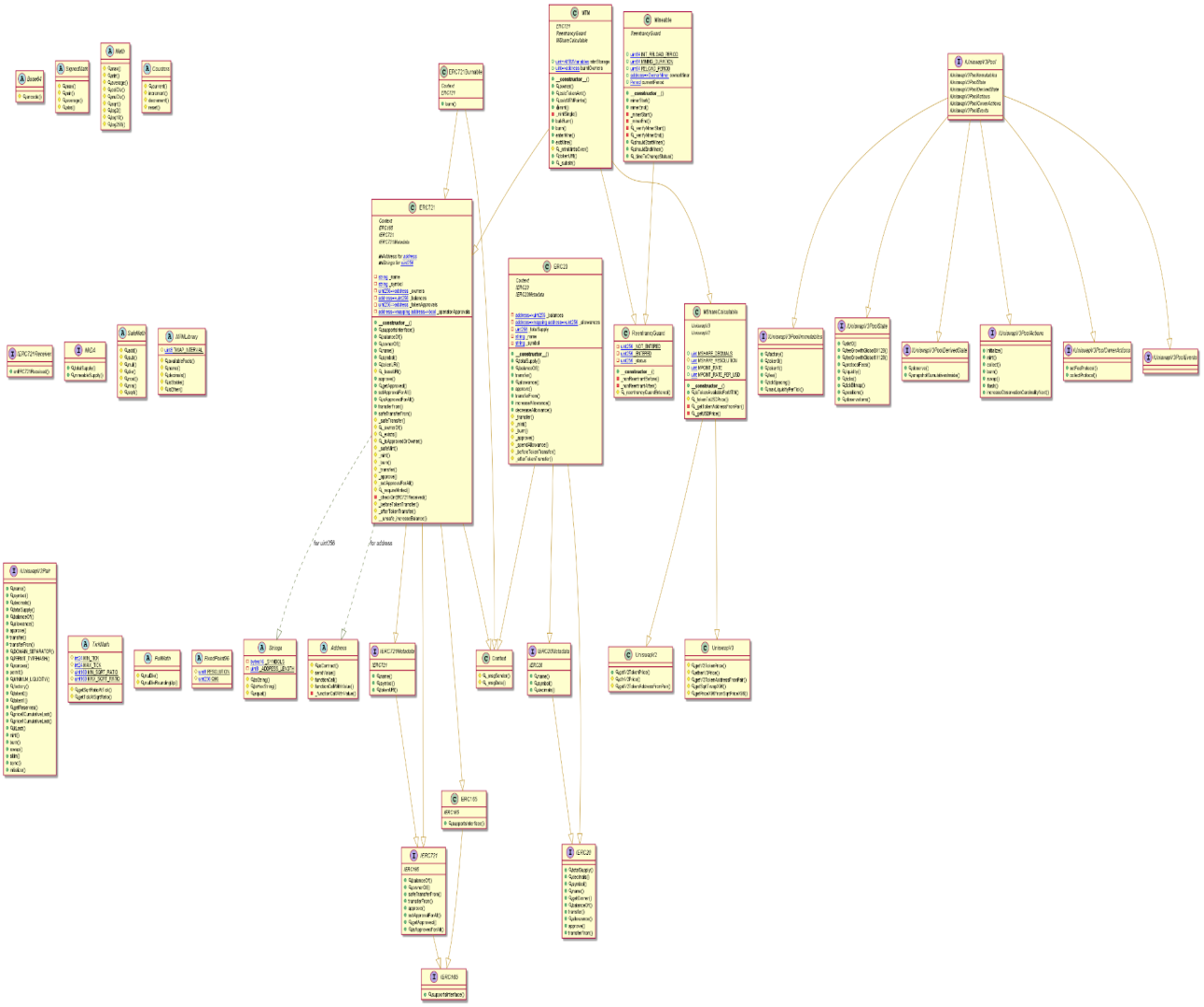
Appendix

Code Flow Diagram - Mida token

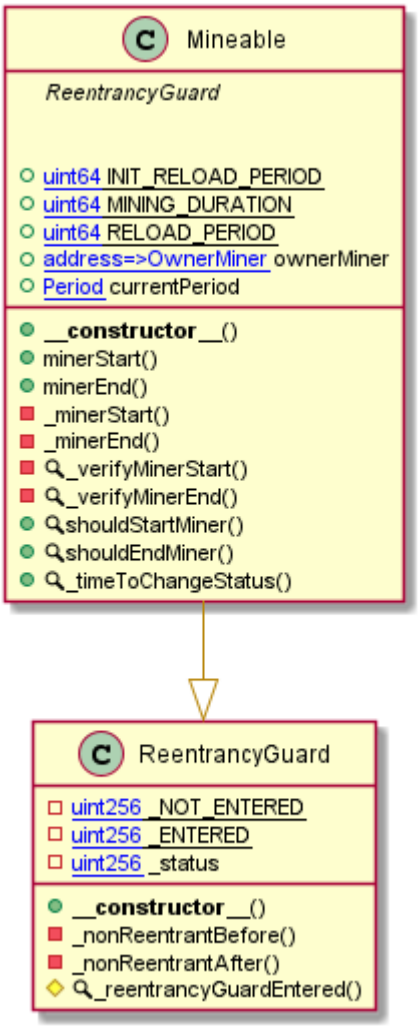
Mida Diagram



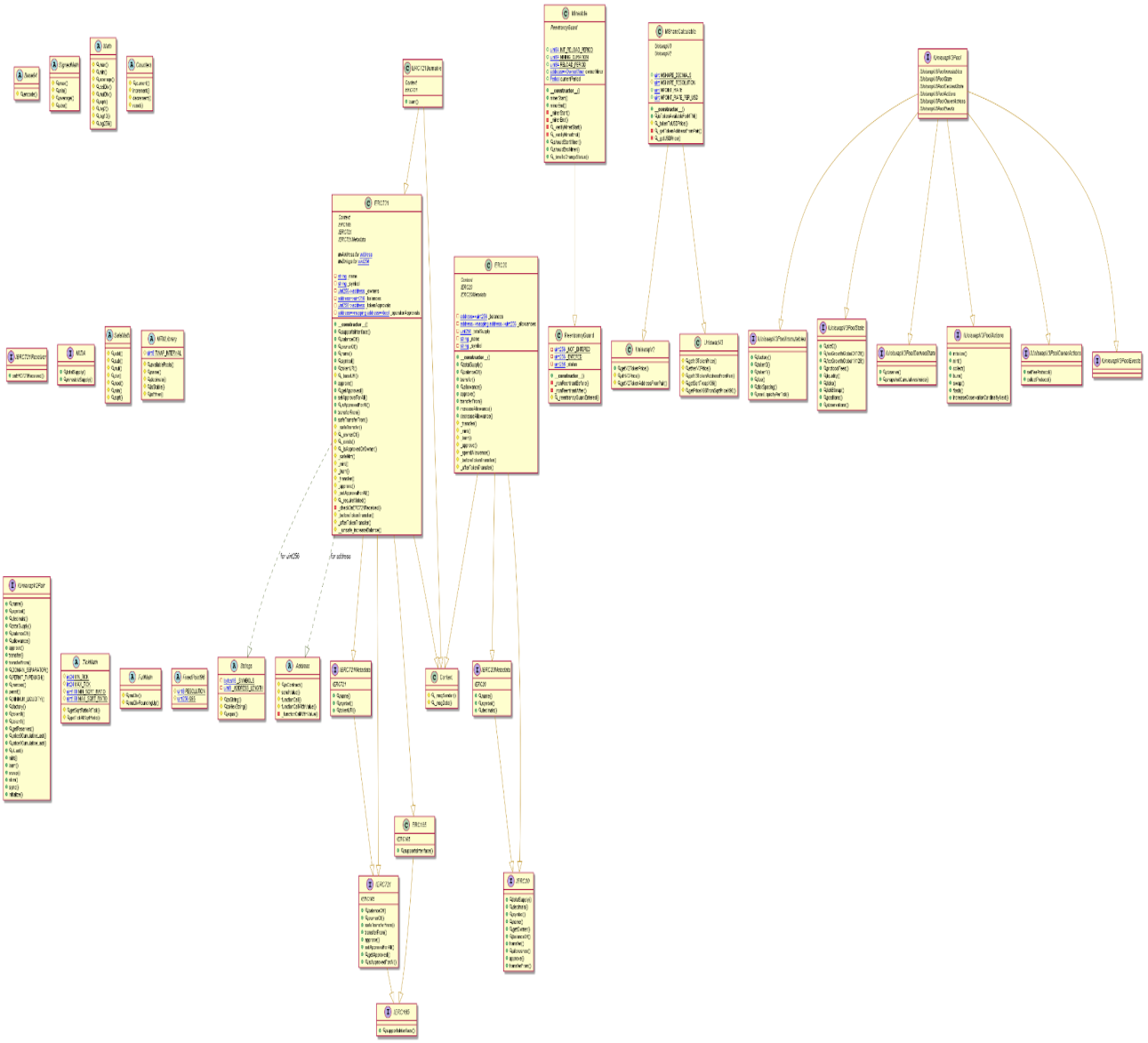
MTM Diagram



Mineable Diagram



MShareCalculable Diagram



Slither Results Log

Slither log >> Mida.sol

```
MTM._mtmMintIsOver() (Mida.sol#2192-2194) has external calls inside a loop: _mida.totalSupply() < (_mida.mineableSupply() / 2)
 (Mida.sol#2193)
UniswapV3.getSqrtTwapX96(address) (Mida.sol#1723-1735) has external calls inside a loop: (tickCumulatives) = IUniswapV3Pool(un
iswapV3Pool).observe(secondsAgos) (Mida.sol#1728)
UniswapV2.getV2TokenPrice(address,uint256) (Mida.sol#1310-1355) has external calls inside a loop: (reserves0,reserves1) = pool
.getReserves() (Mida.sol#1315)
UniswapV2.getV2TokenPrice(address,uint256) (Mida.sol#1310-1355) has external calls inside a loop: pool.token1().isEther() || p
ool.token1().isStable() (Mida.sol#1320)
UniswapV2.getV2TokenPrice(address,uint256) (Mida.sol#1310-1355) has external calls inside a loop: token = pool.token0() (Mida.
sol#1322)
UniswapV2.getV2TokenPrice(address,uint256) (Mida.sol#1310-1355) has external calls inside a loop: stableOrWeth = pool.token1()
 (Mida.sol#1323)
UniswapV2.ethV2Price() (Mida.sol#1357-1361) has external calls inside a loop: (reserves0,reserves1) = pool.getReserves() (Mida
.sol#1359)
UniswapV2.getV2TokenPrice(address,uint256) (Mida.sol#1310-1355) has external calls inside a loop: pool.token0().isEther() || p
ool.token0().isStable() (Mida.sol#1327)
UniswapV2.getV2TokenPrice(address,uint256) (Mida.sol#1310-1355) has external calls inside a loop: token = pool.token1() (Mida.
sol#1329)
UniswapV2.getV2TokenPrice(address,uint256) (Mida.sol#1310-1355) has external calls inside a loop: stableOrWeth = pool.token0()
 (Mida.sol#1330)
MTMLibrary.decimals(address) (Mida.sol#1225-1231) has external calls inside a loop: ERC20(token).decimals() (Mida.sol#1229)
UniswapV3.getV3TokenPrice(address,uint256) (Mida.sol#1654-1706) has external calls inside a loop: pool.token1().isEther() || p
ool.token1().isStable() (Mida.sol#1664)
UniswapV3.getV3TokenPrice(address,uint256) (Mida.sol#1654-1706) has external calls inside a loop: token = pool.token0() (Mida.
sol#1666)
UniswapV3.getV3TokenPrice(address,uint256) (Mida.sol#1654-1706) has external calls inside a loop: stableOrWeth = pool.token1()
 (Mida.sol#1667)
UniswapV3.getV3TokenPrice(address,uint256) (Mida.sol#1654-1706) has external calls inside a loop: pool.token0().isEther() || p
ool.token0().isStable() (Mida.sol#1671)
UniswapV3.getV3TokenPrice(address,uint256) (Mida.sol#1654-1706) has external calls inside a loop: token = pool.token1() (Mida.
sol#1673)
UniswapV3.getV3TokenPrice(address,uint256) (Mida.sol#1654-1706) has external calls inside a loop: stableOrWeth = pool.token0()
 (Mida.sol#1674)
MTM._mintSingle(MTM._MTMVariables) (Mida.sol#2118-2157) has external calls inside a loop: (sent) = address(BA_ADDRESS).call{va
lue: tokenAmount}() (Mida.sol#2138)
```

```
Reentrancy in Mida._enterMine(uint256[]) (Mida.sol#2399-2426):
        External calls:
        - _mtm.enterMine(mtmIds[i]) (Mida.sol#2412)
        State variables written after the call(s):
        - midaMining = _verifyRewards(totalMPoints * MSHARE_RATE) (Mida.sol#2421)
                - currentPeriod.totalMidaMined += midaMining (Mida.sol#2494)
        - midaMining = _verifyRewards(totalMPoints * MSHARE_RATE) (Mida.sol#2421)
                - pendingMidaMined += midaMining (Mida.sol#2495)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Mineable.shouldStartMiner() (Mida.sol#919-922) uses timestamp for comparisons
        Dangerous comparisons:
        - _timeToChangeStatus() && currentPeriod.status == PeriodStatus.Reload && currentPeriod.totalMidaMined > 0 (Mida.sol#9
20-921)
Mineable.shouldEndMiner() (Mida.sol#924-926) uses timestamp for comparisons
        Dangerous comparisons:
        - _timeToChangeStatus() && currentPeriod.status == PeriodStatus.Mining (Mida.sol#925)
Mineable._timeToChangeStatus() (Mida.sol#928-930) uses timestamp for comparisons
        Dangerous comparisons:
        - currentPeriod.timeToChangeStatus <= block.timestamp (Mida.sol#929)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
Pragma version^0.8.17 (Mida.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.1
6
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (Mida.sol#430-435):
        - (success) = recipient.call{value: amount}() (Mida.sol#433)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (Mida.sol#467-489):
        - (success,returndata) = target.call{value: weiValue}(data) (Mida.sol#475)
Low level call in MTM._mintSingle(MTM._MTMVariables) (Mida.sol#2118-2157):
        - (sent) = address(BA_ADDRESS).call{value: tokenAmount}() (Mida.sol#2138)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Mida (Mida.sol#2290-2500) should inherit from MIDA (Mida.sol#940-943)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance
```

```
Function ERC721.__unsafe_increaseBalance(address,uint256) (Mida.sol#800-802) is not in mixedCase
Function Mineable._timeToChangeStatus() (Mida.sol#928-930) is not in mixedCase
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (Mida.sol#1269) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (Mida.sol#1270) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (Mida.sol#1287) is not in mixedCase
Variable MShareCalculable.MSHARE_RESOLUTION (Mida.sol#1964) is not in mixedCase
Variable MShareCalculable.MPOINT_RATE (Mida.sol#1965) is not in mixedCase
Variable MShareCalculable.MPOINT_RATE_PER_USD (Mida.sol#1966) is not in mixedCase
Struct MTM._MTMVariables (Mida.sol#2055-2058) is not in CapWords
Parameter MTM.mint(MTM._MTMVariables[])._mtmVariables (Mida.sol#2106) is not in mixedCase
Function MTM._substr(string,uint256,uint256) (Mida.sol#2279-2286) is not in mixedCase
Variable MTM.OWNER (Mida.sol#2049) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (Mida.sol#559)" inContext (Mida.sol#554-562)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

```
Variable MTMLibrary.GUSD_ADDRESS (Mida.sol#1186) is too similar to MTMLibrary.USDD_ADDRESS (Mida.sol#1191)
Variable MTMLibrary.GUSD_ADDRESS (Mida.sol#1186) is too similar to MTMLibrary.USDP_ADDRESS (Mida.sol#1188)
Variable MTMLibrary.GUSD_ADDRESS (Mida.sol#1186) is too similar to MTMLibrary.LUSD_ADDRESS (Mida.sol#1185)
Variable MTMLibrary.LUSD_ADDRESS (Mida.sol#1185) is too similar to MTMLibrary.USDD_ADDRESS (Mida.sol#1191)
Variable MTMLibrary.LUSD_ADDRESS (Mida.sol#1185) is too similar to MTMLibrary.USDP_ADDRESS (Mida.sol#1188)
Variable MTMLibrary.GUSD_ADDRESS (Mida.sol#1186) is too similar to MTMLibrary.USDC_ADDRESS (Mida.sol#1184)
Variable MTMLibrary.LUSD_ADDRESS (Mida.sol#1185) is too similar to MTMLibrary.USDC_ADDRESS (Mida.sol#1184)
Variable MTMLibrary.USDC_ADDRESS (Mida.sol#1184) is too similar to MTMLibrary.USDD_ADDRESS (Mida.sol#1191)
Variable MTMLibrary.USDC_ADDRESS (Mida.sol#1184) is too similar to MTMLibrary.USDP_ADDRESS (Mida.sol#1188)
Variable MTMLibrary.USDD_ADDRESS (Mida.sol#1191) is too similar to MTMLibrary.USDP_ADDRESS (Mida.sol#1188)
Variable MTMLibrary.GUSD_ADDRESS (Mida.sol#1186) is too similar to MTMLibrary.USDT_ADDRESS (Mida.sol#1183)
Variable MTMLibrary.LUSD_ADDRESS (Mida.sol#1185) is too similar to MTMLibrary.USDT_ADDRESS (Mida.sol#1183)
Variable MTMLibrary.USDC_ADDRESS (Mida.sol#1184) is too similar to MTMLibrary.USDT_ADDRESS (Mida.sol#1183)
Variable MTMLibrary.USDD_ADDRESS (Mida.sol#1191) is too similar to MTMLibrary.USDT_ADDRESS (Mida.sol#1183)
Variable MTMLibrary.USDP_ADDRESS (Mida.sol#1188) is too similar to MTMLibrary.USDT_ADDRESS (Mida.sol#1183)
Variable MTMLibrary.WETH_USDC_POOL (Mida.sol#1195) is too similar to MTMLibrary.WETH_USDT_POOL (Mida.sol#1196)
Variable IUniswapV3PoolState.positions(bytes32).feeGrowthInside0LastX128 (Mida.sol#1802) is too similar to IUniswapV3PoolState
.positions(bytes32).feeGrowthInside1LastX128 (Mida.sol#1803)
```

Slither log >> MTM.sol

```
TickMath.getTickAtSqrtRatio(uint160) (MTM.sol#1313-1456) uses assembly
        - INLINE ASM (MTM.sol#1320-1324)
        - INLINE ASM (MTM.sol#1325-1329)
        - INLINE ASM (MTM.sol#1330-1334)
        - INLINE ASM (MTM.sol#1335-1339)
        - INLINE ASM (MTM.sol#1340-1344)
        - INLINE ASM (MTM.sol#1345-1349)
        - INLINE ASM (MTM.sol#1350-1354)
        - INLINE ASM (MTM.sol#1355-1358)
        - INLINE ASM (MTM.sol#1365-1370)
        - INLINE ASM (MTM.sol#1371-1376)
        - INLINE ASM (MTM.sol#1377-1382)
        - INLINE ASM (MTM.sol#1383-1388)
        - INLINE ASM (MTM.sol#1389-1394)
        - INLINE ASM (MTM.sol#1395-1400)
        - INLINE ASM (MTM.sol#1401-1406)
        - INLINE ASM (MTM.sol#1407-1412)
        - INLINE ASM (MTM.sol#1413-1418)
        - INLINE ASM (MTM.sol#1419-1424)
        - INLINE ASM (MTM.sol#1425-1430)
        - INLINE ASM (MTM.sol#1431-1436)
        - INLINE ASM (MTM.sol#1437-1442)
        - INLINE ASM (MTM.sol#1443-1447)
FullMath.mulDiv(uint256,uint256,uint256) (MTM.sol#1461-1519) uses assembly
        - INLINE ASM (MTM.sol#1469-1473)
        - INLINE ASM (MTM.sol#1477-1479)
        - INLINE ASM (MTM.sol#1487-1489)
        - INLINE ASM (MTM.sol#1490-1493)
        - INLINE ASM (MTM.sol#1496-1498)
        - INLINE ASM (MTM.sol#1500-1502)
        - INLINE ASM (MTM.sol#1503-1505)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
MTM.burn(uint256) (MTM.sol#2063-2069) has costly operations inside a loop:
        - delete mtmStorage[tokenId] (MTM.sol#2066)
ERC721._burn(uint256) (MTM.sol#716-733) has costly operations inside a loop:
        - delete _tokenApprovals[tokenId] (MTM.sol#723)
ERC721._burn(uint256) (MTM.sol#716-733) has costly operations inside a loop:
        - delete _owners[tokenId] (MTM.sol#728)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```

```
TickMath.getSqrtRatioAtTick(int24) (MTM.sol#1280-1311) uses literals with too many digits:
        - ratio = 0x100000000000000000000000000000000 (MTM.sol#1284-1286)
FixedPoint96.slitherConstructorConstantVariables() (MTM.sol#1537-1540) uses literals with too many digits:
        - Q96 = 0x1000000000000000000000000 (MTM.sol#1539)
MTM.tokenURI(uint256) (MTM.sol#2089-2170) uses literals with too many digits:
        - nftUri = ipfs://bafybeiauxp4zmkjq5bq27esclclnbxcujaldl5vlpsphkz546pkmesp3pu/100000.png (MTM.sol#2097)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

ERC20 (MTM.sol#935-1070) does not implement functions:
        - IERC20Metadata.decimals() (MTM.sol#932)
        - IERC20.getOwner() (MTM.sol#397)
        - IERC20Metadata.name() (MTM.sol#928)
        - IERC20Metadata.symbol() (MTM.sol#930)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

MTMLibrary.WETH_USDC_POOL (MTM.sol#1091) is never used in MTMLibrary (MTM.sol#1072-1147)
MTMLibrary.WETH_USDT_POOL (MTM.sol#1092) is never used in MTMLibrary (MTM.sol#1072-1147)
MTMLibrary.TWAP_INTERVAL (MTM.sol#1094) is never used in MTMLibrary (MTM.sol#1072-1147)
TickMath.MAX_TICK (MTM.sol#1275) is never used in TickMath (MTM.sol#1272-1457)
TickMath.MIN_SQRT_RATIO (MTM.sol#1277) is never used in TickMath (MTM.sol#1272-1457)
TickMath.MAX_SQRT_RATIO (MTM.sol#1278) is never used in TickMath (MTM.sol#1272-1457)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

ERC20._name (MTM.sol#942) should be immutable
ERC20._symbol (MTM.sol#943) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
MTM.sol analyzed (36 contracts with 84 detectors), 176 result(s) found
```

Slither log >> Mineable.sol

```
Mineable.shouldStartMiner() (Mineable.sol#157-160) uses timestamp for comparisons
        Dangerous comparisons:
        - _timeToChangeStatus() && currentPeriod.status == PeriodStatus.Reload && currentPeriod.totalMidaMined > 0 (Mineable.s
ol#158-159)
Mineable.shouldEndMiner() (Mineable.sol#162-164) uses timestamp for comparisons
        Dangerous comparisons:
        - _timeToChangeStatus() && currentPeriod.status == PeriodStatus.Mining (Mineable.sol#163)
Mineable._timeToChangeStatus() (Mineable.sol#166-168) uses timestamp for comparisons
        Dangerous comparisons:
        - currentPeriod.timeToChangeStatus <= block.timestamp (Mineable.sol#167)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

ReentrancyGuard._nonReentrantAfter() (Mineable.sol#46-50) is never used and should be removed
ReentrancyGuard._nonReentrantBefore() (Mineable.sol#38-44) is never used and should be removed
ReentrancyGuard._reentrancyGuardEntered() (Mineable.sol#56-58) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.17 (Mineable.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0
.8.16
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Function Mineable._timeToChangeStatus() (Mineable.sol#166-168) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
Mineable.sol analyzed (2 contracts with 84 detectors), 9 result(s) found
```

Slither log >> MShareCalculable.sol

```
Pragma version^0.8.17 (MShareCalculable.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12
/0.7.6/0.8.16
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (MShareCalculable.sol#94-99):
        - (success) = recipient.call{value: amount}() (MShareCalculable.sol#97)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (MShareCalculable.sol#131-153):
        - (success,returndata) = target.call{value: weiValue}(data) (MShareCalculable.sol#139)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function IUniswapV2Pair.DOMAIN_SEPARATOR() (MShareCalculable.sol#433) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (MShareCalculable.sol#434) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (MShareCalculable.sol#451) is not in mixedCase
Variable MShareCalculable.MSHARE_RESOLUTION (MShareCalculable.sol#1131) is not in mixedCase
Variable MShareCalculable.MPOINT_RATE (MShareCalculable.sol#1132) is not in mixedCase
Variable MShareCalculable.MPOINT_RATE_PER_USD (MShareCalculable.sol#1133) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
ERC20 (MShareCalculable.sol#203-338) does not implement functions:
        - IERC20Metadata.decimals() (MShareCalculable.sol#200)
        - IERC20.getOwner() (MShareCalculable.sol#164)
        - IERC20Metadata.name() (MShareCalculable.sol#196)
        - IERC20Metadata.symbol() (MShareCalculable.sol#198)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

MTMLibrary.WETH_USDC_POOL (MShareCalculable.sol#359) is never used in MTMLibrary (MShareCalculable.sol#340-415)
MTMLibrary.WETH_USDT_POOL (MShareCalculable.sol#360) is never used in MTMLibrary (MShareCalculable.sol#340-415)
MTMLibrary.TWAP_INTERVAL (MShareCalculable.sol#362) is never used in MTMLibrary (MShareCalculable.sol#340-415)
TickMath.MAX_TICK (MShareCalculable.sol#544) is never used in TickMath (MShareCalculable.sol#540-728)
TickMath.MIN_SQRT_RATIO (MShareCalculable.sol#546) is never used in TickMath (MShareCalculable.sol#540-728)
TickMath.MAX_SQRT_RATIO (MShareCalculable.sol#547) is never used in TickMath (MShareCalculable.sol#540-728)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

ERC20._name (MShareCalculable.sol#210) should be immutable
ERC20._symbol (MShareCalculable.sol#211) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
MShareCalculable.sol analyzed (21 contracts with 84 detectors), 108 result(s) found
```

Solidity Static Analysis

Mida.sol

Security

Check-effects-interaction:

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

more

Pos: 124:2:

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: 116:47:

Low level calls:

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

more

Pos: 147:45:

Gas & Economy

Delete dynamic array:

The "delete" operation when applied to a dynamically sized array in Solidity generates code to delete each of the elements contained. If the array is large, this operation can surpass the block gas limit and raise an OOG exception. Also nested dynamically sized objects can produce the same results.

more

Pos: 250:4:

Email: audit@EtherAuthority.io

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: 161:4:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type

more

Pos: 9:4:

Miscellaneous

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: 214:6:

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: 39:4:

MTM.sol

Security

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
more
Pos: 147:45:

Gas & Economy

Gas costs:

Gas requirement of function MTM.mint 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: 110:2:

Gas costs:

Gas requirement of function MTM.bulkBurn 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: 169:2:

Miscellaneous

Constant/View/Pure functions:

MTM.calcTokenAmt(address,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 89:2:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
more
Pos: 185:6:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
more
Pos: 204:8:

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: 233:8:

Mineable.sol

Security

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: 85:70:

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: 116:47:

Miscellaneous

Constant/View/Pure functions:

Mineable._verifyMinerStart() : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 77:2:

MShareCalculable.sol

Security

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: 200:6:

Miscellaneous

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: 28:26:

Solhint Linter

Mida.sol

```
Mida.sol:49:23: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mida.sol:50:30: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mida.sol:61:26: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mida.sol:85:46: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mida.sol:93:38: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mida.sol:164:30: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mida.sol:175:16: Error: Parse error: missing ';' at '{'
Mida.sol:199:30: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mida.sol:206:16: Error: Parse error: missing ';' at '{'
Mida.sol:225:29: Error: Parse error: mismatched input '(' expecting
{';', '='}
Mida.sol:245:16: Error: Parse error: missing ';' at '{'
```

MTM.sol

```
MTM.sol:38:8: Error: Parse error: extraneous input '{' expecting
{'from', 'calldata', 'callback', 'leave', 'payable', 'receive',
Identifier}
MTM.sol:38:62: Error: Parse error: mismatched input '.' expecting
{';', '='}
MTM.sol:38:68: Error: Parse error: mismatched input '}' expecting
{'from', 'calldata', 'callback', 'override', 'constant', 'immutable',
'leave', 'internal', 'payable', 'private', 'public', 'receive',
Identifier}
MTM.sol:38:70: Error: Parse error: extraneous input 'for' expecting
{<EOF>, 'pragma', 'import', 'from', 'abstract', 'contract',
'interface', 'library', 'struct', 'function', 'enum', 'address',
'mapping', 'calldata', 'var', 'bool', 'string', 'byte', 'callback',
Int, Uint, Byte, Fixed, Ufixed, 'leave', 'payable', 'constructor',
'fallback', 'receive', Identifier}
MTM.sol:38:81: Error: Parse error: mismatched input ';' expecting
'constant'
MTM.sol:62:45: Error: Parse error: mismatched input ';' expecting
'constant'
MTM.sol:64:15: Error: Parse error: mismatched input '(' expecting
'constant'
MTM.sol:64:21: Error: Parse error: missing 'constant' at
'mtmVariables'
MTM.sol:64:33: Error: Parse error: mismatched input ',' expecting '='
MTM.sol:66:8: Error: Parse error: missing 'constant' at
```

```
'MTMMintIsOver'
MTM.sol:66:21: Error: Parse error: missing '=' at '('
MTM.sol:67:8: Error: Parse error: missing 'constant' at
'InsuffucentAllowanceForTokenPOP'
MTM.sol:73:13: Error: Parse error: missing 'constant' at 'NotOwner'
MTM.sol:73:21: Error: Parse error: missing '=' at '('
MTM.sol:74:4: Error: Parse error: extraneous input '}' expecting
{<EOF>, 'pragma', 'import', 'from', 'abstract', 'contract',
'interface', 'library', 'struct', 'function', 'enum', 'address',
'mapping', 'calldata', 'var', 'bool', 'string', 'byte', 'callback',
Int, Uint, Byte, Fixed, Ufixed, 'leave', 'payable', 'constructor',
'fallback', 'receive', Identifier}
MTM.sol:75:5: Error: Parse error: mismatched input ';' expecting
'constant'
MTM.sol:117:16: Error: Parse error: missing ';' at '{'
MTM.sol:129:26: Error: Parse error: mismatched input '(' expecting
{';', '='}
MTM.sol:174:16: Error: Parse error: missing ';' at '{'
MTM.sol:305:0: Error: Parse error: extraneous input '}' expecting
{<EOF>, 'pragma', 'import', 'from', 'abstract', 'contract',
'interface', 'library', 'struct', 'function', 'enum', 'address',
'mapping', 'calldata', 'var', 'bool', 'string', 'byte', 'callback',
Int, Uint, Byte, Fixed, Ufixed, 'leave', 'payable', 'constructor',
'fallback', 'receive', Identifier}
```

Mineable.sol

```
Mineable.sol:30:22: Error: Parse error: mismatched input '('
expecting {';', '='}
Mineable.sol:80:32: Error: Parse error: mismatched input '('
expecting {';', '='}
Mineable.sol:85:35: Error: Parse error: mismatched input '('
expecting {';', '='}
Mineable.sol:90:27: Error: Parse error: mismatched input '('
expecting {';', '='}
Mineable.sol:97:30: Error: Parse error: mismatched input '('
expecting {';', '='}
Mineable.sol:102:33: Error: Parse error: mismatched input '('
expecting {';', '='}
```

MShareCalculable.sol

```
MShareCalculable.sol:11:8: Error: Parse error: extraneous input '{'
expecting {'from', 'calldata', 'callback', 'leave', 'payable',
'receive', Identifier}
MShareCalculable.sol:11:20: Error: Parse error: mismatched input '.'
expecting 'for'
MShareCalculable.sol:11:29: Error: Parse error: extraneous input ','
expecting {'from', 'calldata', 'callback', 'override', 'constant',
'immutable', 'leave', 'internal', 'payable', 'private', 'public',
'receive', Identifier}
MShareCalculable.sol:11:41: Error: Parse error: mismatched input '.'
```

```
expecting {';', '='}
MShareCalculable.sol:11:49: Error: Parse error: extraneous input ','
expecting {'from', 'calldata', 'callback', 'override', 'constant',
'immutable', 'leave', 'internal', 'payable', 'private', 'public',
'receive', Identifier}
MShareCalculable.sol:11:61: Error: Parse error: mismatched input '.'
expecting {';', '='}
MShareCalculable.sol:19:47: Error: Parse error: mismatched input ')'
expecting '='
MShareCalculable.sol:40:16: Error: Parse error: missing ';' at '{'
MShareCalculable.sol:99:0: Error: Parse error: extraneous input '}'
expecting {<EOF>, 'pragma', 'import', 'from', 'abstract', 'contract',
'interface', 'library', 'struct', 'function', 'enum', 'address',
'mapping', 'calldata', 'var', 'bool', 'string', 'byte', 'callback',
Int, Uint, Byte, Fixed, Ufixed, 'leave', 'payable', 'constructor',
'fallback', 'receive', Identifier}
```

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