MahaDAO

Smart Contract Audit Report

ArthController.sol

IMMUNE BYTES

Audits

MAHADAO

May 02, 2021

Introduction

1. About MahaDAO

ARTH is a new type of currency designed to not be pegged to government-owned currencies (like US Dollar, Euro, or Chinese Yuan), but still remain relatively stable (unlike Gold and Bitcoin).

Without being influenced by government-owned currencies, ARTH will be immune to inflation. Through stability, ARTH also becomes a superior choice of currency for means of trade. This is unlike Gold or Bitcoin, which are used more as a store of value rather than a medium of exchange.

Visit http://mahadao.com/ to learn more about.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The MahaDAO team has provided documentation for the purpose of conducting the audit. The documents are:

1. https://docs.arthcoin.com/

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: **ARTH v2**
- Contract Name: ArthController.sol
- Languages: Solidity(Smart contract)
- Github commit hash for audit:**d4d445c8e8fe9708ef04a94c09be2e961aa48105**
- GitHub link: https://github.com/MahaDAO/arthcoin-v2/blob/develop/contracts/Arth/ArthController.sol
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

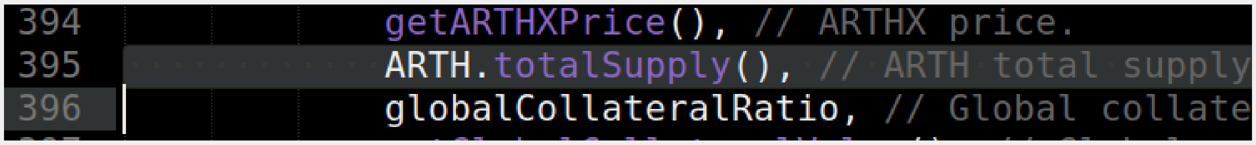
Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	1	3	5
Closed	-	-	-

High severity issues

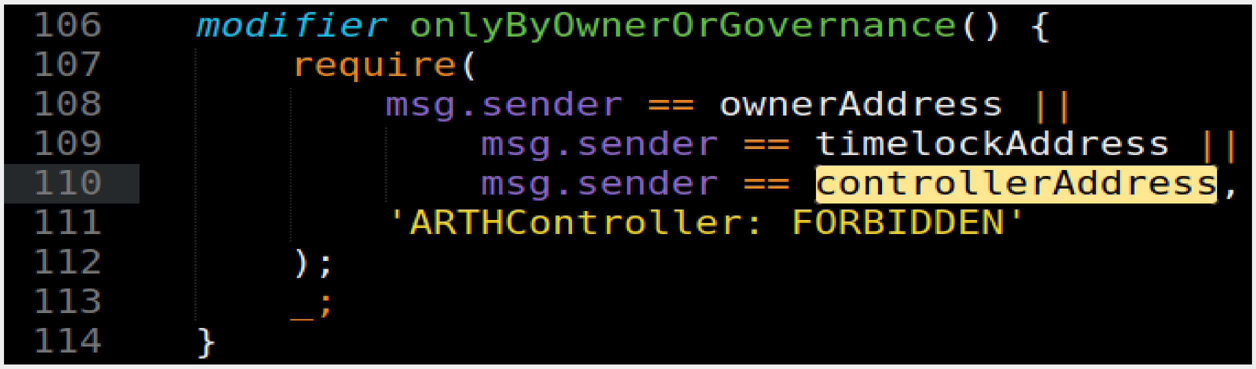
1. **Contract uses Uninitialized State Variables**
   Line - 32, 43, 110, 395
   Description:
   The ArthController contract accesses some imperative State Variables that have never been initialized throughout the contract.

   Mentioned below are those State Variables and the specific parts in the contract where they are accessed:
   - **ARTH** (Line 32) is being accessed in the **getARTHInfo** function at Line 395 but was never initialized.



```
394                 getARTHXPrice(), // ARTHX price.
395                 ARTH.totalSupply(), // ARTH total supply
396                 globalCollateralRatio, // Global collate
```

   - **controllerAddress** (Line 43) is being used in the **onlyByOwnerOrGovernance** modifier at Line 110 but was never initialized.



```
106     modifier onlyByOwnerOrGovernance() {
107         require(
108             msg.sender == ownerAddress ||
109                 msg.sender == timelockAddress ||
110                 msg.sender == controllerAddress,
111             'ARTHController: FORBIDDEN'
112         );
113         _;
114     }
```

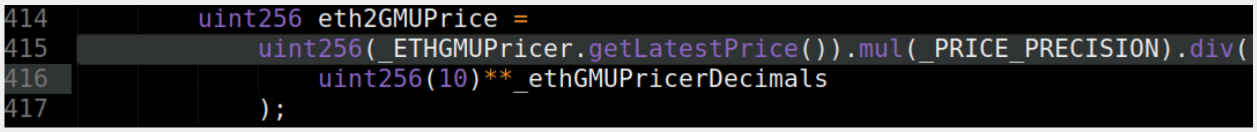
   Recommendation:
   The above-mentioned state variables must be initialized within the constructor before being used in any particular functions, to avoid any unwanted scenario during function executions.

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Medium severity issues

1. **Multiplication is being performed on the result of Division**
   Line no - 414-417
   Explanation:
   During the automated testing of the ArthController.sol contract, it was found that one of the functions in the contract is performing multiplication on the result of a Division. Integer Divisions in Solidity might truncate. Moreover, performing division before multiplication might lead to loss of precision.

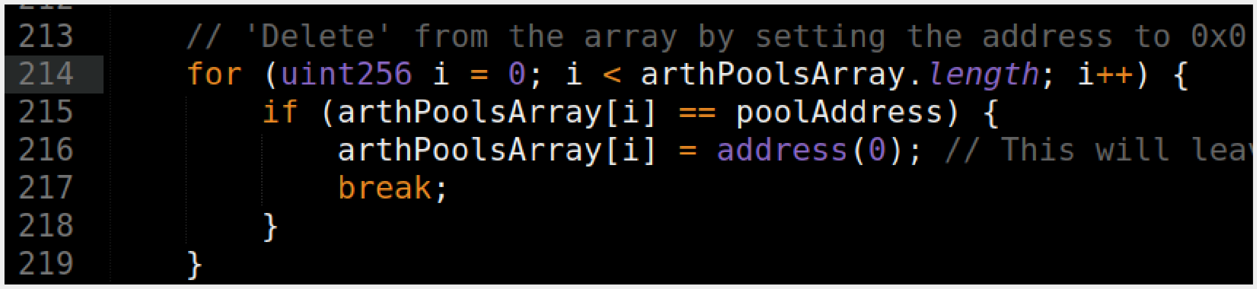
   The following functions involve division before multiplication in the mentioned lines:
   - **_getOraclePrice**



```
414         uint256 eth2GMUPrice =
415             uint256(_ETHGMUPricer.getLatestPrice()).mul(_PRICE_PRECISION).div(
416                 uint256(10)**_ethGMUPricerDecimals
417             );
```

   Recommendation:
   Solidity doesn't encourage arithmetic operations that involve division before multiplication. Therefore the above-mentioned function should be checked once and redesigned if it does not lead to expected results.

2. **Functions includes Costly Loops**
   Line no - 214, 365
   Description:
   The **ArthController** contract has **for loops** in some functions that includes state variables like **.length** of a non-memory array in the condition of the for loops.



```
213     // 'Delete' from the array by setting the address to 0x0
214     for (uint256 i = 0; i < arthPoolsArray.length; i++) {
215         if (arthPoolsArray[i] == poolAddress) {
216             arthPoolsArray[i] = address(0); // This will lea
217             break;
218         }
219     }
```

   As a result, these state variables consume a lot more extra gas for every iteration of the loop.
   The following functions in the contract includes such loops at the specified Line numbers:

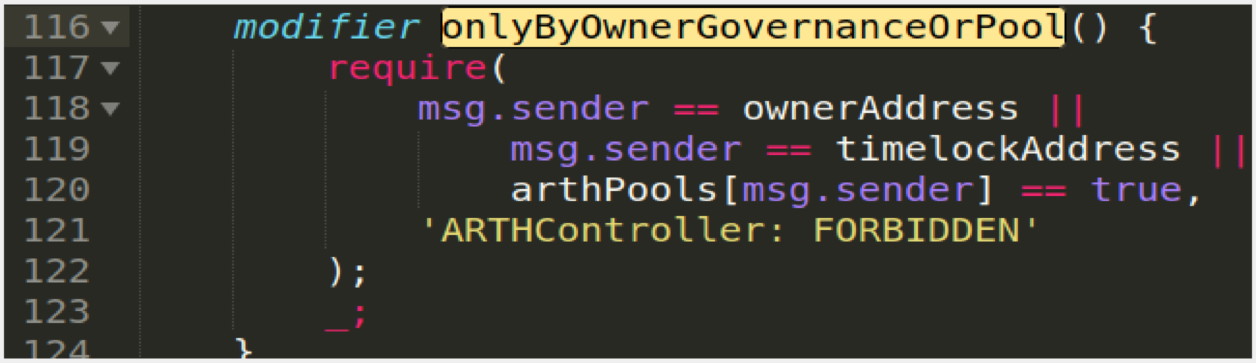
- **removePool** at Line 214
- **getGlobalCollateralValue** at Line 365

Recommendation:
A better and effective approach would be to use a local variable instead of a state variable like **.length** in a loop.

For instance,

```
local_variable = arthPoolsArray.length;
for (uint256 i = 0; i < local_variable; i++) {
        if (arthPoolsArray[i] == poolAddress) {
            arthPoolsArray[i] = address(0); // This will leave a null in the array and keep
the indices the same.
            break;
        }
```

3. **Modifiers created but never used**
   Line no - 93, 116
   Description:
   The ArthController contract includes some modifiers, at the above-mentioned lines, that have been created but never used throughout the contract.



```
116      modifier onlyByOwnerGovernanceOrPool() {
117          require(
118              msg.sender == ownerAddress ||
119                  msg.sender == timelockAddress ||
120                  arthPools[msg.sender] == true,
121              'ARTHController: FORBIDDEN'
122          );
123          _;
124      }
```

   While this consumes additional space in the contract, it also adversely affects the gas optimization as well as the readability of the smart contract code.

   Recommendation:
   Adequate use of all State Variable, modifiers, mappings etc, must be ensured in the contract. If a particular modifier holds no significance it should be removed from the contract.

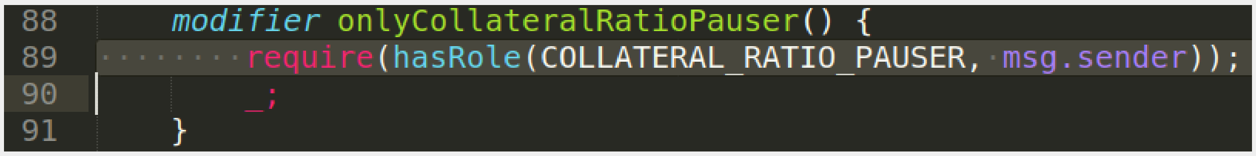
Low severity issues

1. **External Visibility should be preferred**
   Desciption:
   Those functions that are never called throughout the contract should be marked as **external** visibility instead of **public** visibility.
   This will effectively result in Gas Optimization as well.

   Therefore, the following function must be marked as **external** within the contract:
   - **setARTHXAddress**
   - **setPriceTarget**
   - **setRefreshCooldown**
   - **setETHGMUOracle**
   - **setARTHXETHOracle**
   - **setARTHETHOracle**
   - **toggleCollateralRatio**
   - **setMintingFee**
   - **setArthStep**
   - **setRedemptionFee**
   - **setOwner**
   - **setPriceBand**
   - **setTimelock**
   - **getGlobalCollateralRatio**
   - **getARTHInfo**

   Recommendation:
   If the public visibility of these functions is not intended, the visibility keyword must be modified to external.

2. **Absence of Error messages in Require Statements**
   Line no - 89
   Description:
   In the ArthController contract, the **modifier onlyCollateralRatioPauser** includes a **require** statement that does not include an error message.

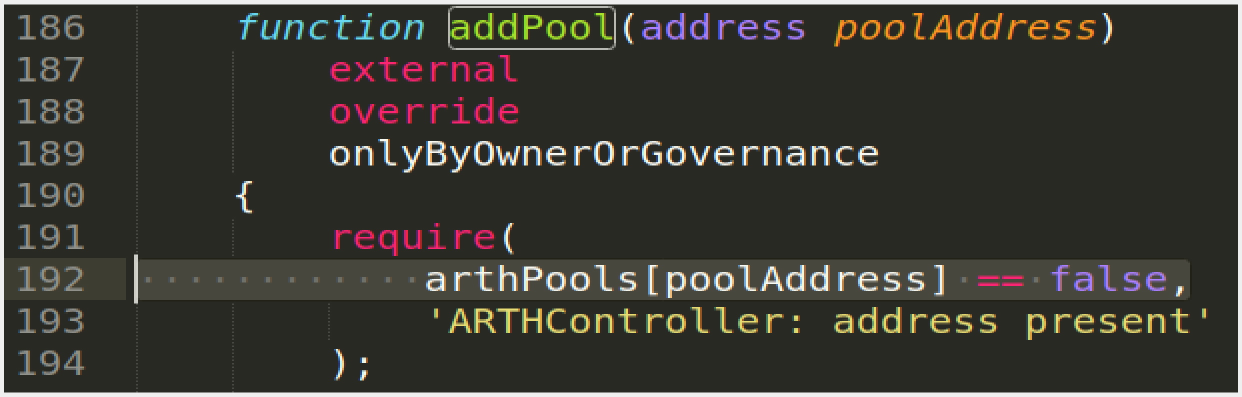


```
88     modifier onlyCollateralRatioPauser() {
89         require(hasRole(COLLATERAL_RATIO_PAUSER, msg.sender));
90         _;
91     }
```

While this makes it troublesome to detect the reason behind a particular function revert, it also reduces the readability of the code.

Recommendation:
Error Messages must be included in every require statement in the contract

3. **Comparison to boolean Constant**
   Line no- 94, 120, 154,192, 206,
   Description:
   Boolean constants can directly be used in conditional statements or require statements. Therefore, it's not considered a better practise to explicitly use **TRUE or FALS**E in the **require** statements.



```
186     function addPool(address poolAddress)
187         external
188         override
189         onlyByOwnerOrGovernance
190     {
191         require(
192             arthPools[poolAddress] == false,
193             'ARTHController: address present'
194         );
```

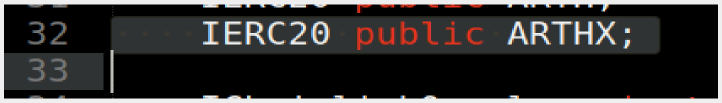
   Recommendation:
   The equality to boolean constants must be removed from the above-mentioned line.

4. **No Event emission for crucial State Variable modification**
   Line no - 242, 291, 299, 323, 307
   Description:
   Functions that modify an imperative arithmetic state variable contract should emit an event after the modification.
   However, during the automated testing it was found that the following functions modify some crucial arithmetic parameters like **priceTarget, mintingFee, redemptionFee, priceBand** etc, but doesn't emit any event afterwards:
   - **setPriceBand**
   - **setRedemptionFee**
   - **setArthStep**
   - **mintingFee**
   - **setPriceTarget**

Since there is no event emitted on updating these variables, it might be difficult to track it off-chain.

Recommendation:
An event should be fired after changing crucial arithmetic state variables.

5. **State Variable never used in the Contract**
   Line no - 32
   Description:
   ARTHX state variable has been initialized in the contract at Line 32 but never used throughout the contract.



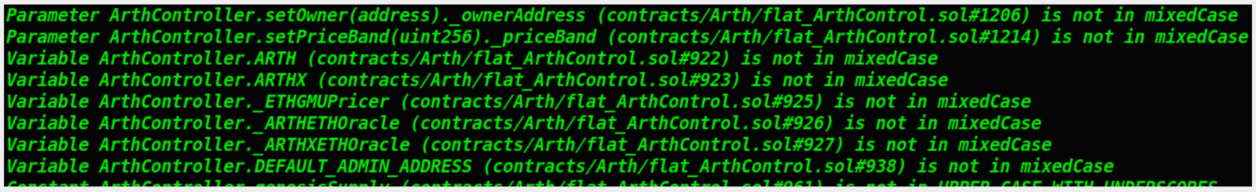
```
32      IERC20 public ARTHX;
33
```

   Recommendation:
   Effective use of all State Variables must be ensured in the contract. Unused variables should be removed from the contract.

Recommendations

1. **Coding Style Issues in the Contract**
   Explanation:
   Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

   During the automated testing, it was found that the ArthController contract had quite a few code style issues.



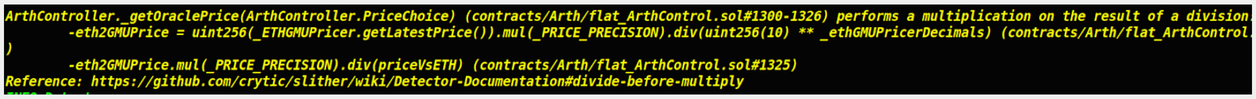
```
Parameter ArthController.setOwner(address)._ownerAddress (contracts/Arth/flat_ArthControl.sol#1206) is not in mixedCase
Parameter ArthController.setPriceBand(uint256)._priceBand (contracts/Arth/flat_ArthControl.sol#1214) is not in mixedCase
Variable ArthController.ARTH (contracts/Arth/flat_ArthControl.sol#922) is not in mixedCase
Variable ArthController.ARTHX (contracts/Arth/flat_ArthControl.sol#923) is not in mixedCase
Variable ArthController._ETHGMUPricer (contracts/Arth/flat_ArthControl.sol#925) is not in mixedCase
Variable ArthController._ARTHETHOracle (contracts/Arth/flat_ArthControl.sol#926) is not in mixedCase
Variable ArthController._ARTHXETHOracle (contracts/Arth/flat_ArthControl.sol#927) is not in mixedCase
Variable ArthController.DEFAULT_ADMIN_ADDRESS (contracts/Arth/flat_ArthControl.sol#938) is not in mixedCase
Constant ArthController.genesisSupply (contracts/Arth/flat_ArthControl.sol#961) is not in UPPER_CASE_WITH_UNDERSCORES
```

   Recommendation:
   Therefore, it is highly recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

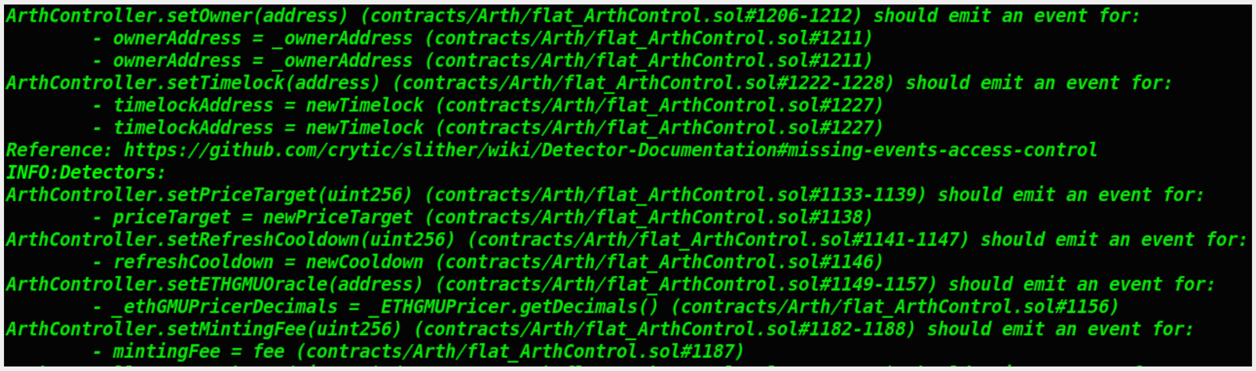
2. **NatSpec Annotations must be included**
   Description:
   The smart contracts do not include the NatSpec annotations adequately.

   Recommendation:
   Cover by NatSpec all Contract methods.

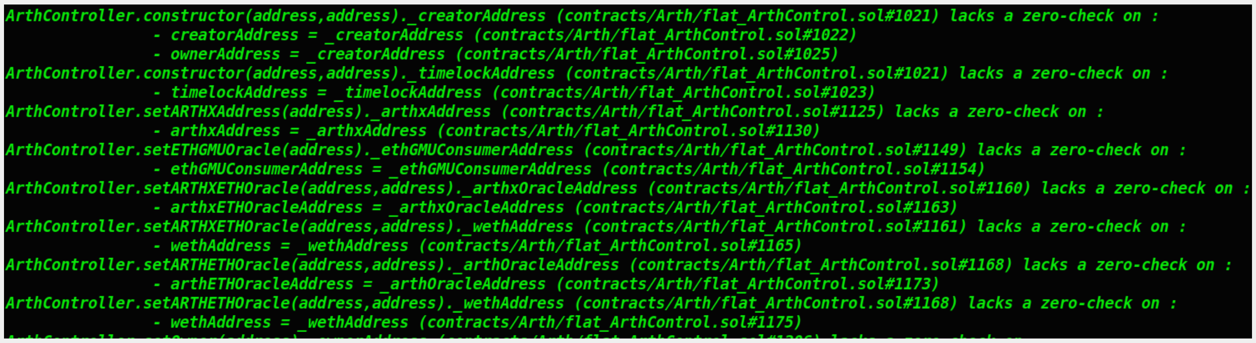
Automated Test Result



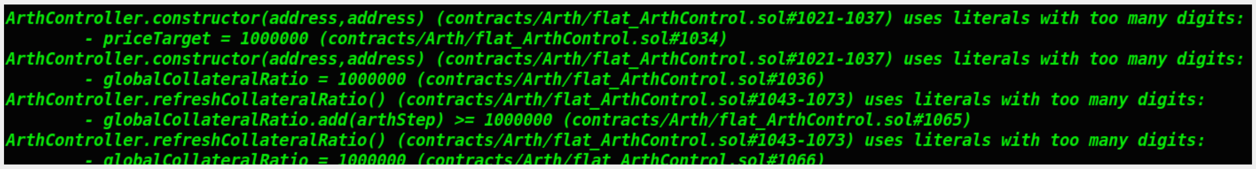
```
ArthController._getOraclePrice(ArthController.PriceChoice) (contracts/Arth/flat_ArthControl.sol#1300-1326) performs a multiplication on the result of a division.
        -eth2GMUPrice = uint256(_ETHGMUPricer.getLatestPrice()).mul(_PRICE_PRECISION).div(uint256(10) ** _ethGMUPricerDecimals) (contracts/Arth/flat_ArthControl.
)
        -eth2GMUPrice.mul(_PRICE_PRECISION).div(priceVsETH) (contracts/Arth/flat_ArthControl.sol#1325)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```



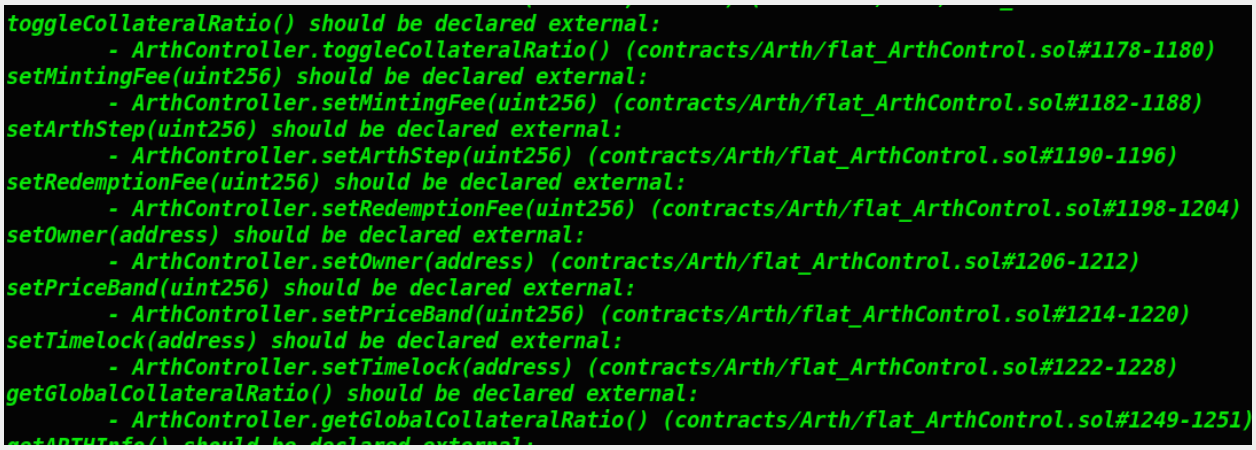
```
ArthController.setOwner(address) (contracts/Arth/flat_ArthControl.sol#1206-1212) should emit an event for:
        - ownerAddress = _ownerAddress (contracts/Arth/flat_ArthControl.sol#1211)
        - ownerAddress = _ownerAddress (contracts/Arth/flat_ArthControl.sol#1211)
ArthController.setTimelock(address) (contracts/Arth/flat_ArthControl.sol#1222-1228) should emit an event for:
        - timelockAddress = newTimelock (contracts/Arth/flat_ArthControl.sol#1227)
        - timelockAddress = newTimelock (contracts/Arth/flat_ArthControl.sol#1227)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control
INFO:Detectors:
ArthController.setPriceTarget(uint256) (contracts/Arth/flat_ArthControl.sol#1133-1139) should emit an event for:
        - priceTarget = newPriceTarget (contracts/Arth/flat_ArthControl.sol#1138)
ArthController.setRefreshCooldown(uint256) (contracts/Arth/flat_ArthControl.sol#1141-1147) should emit an event for:
        - refreshCooldown = newCooldown (contracts/Arth/flat_ArthControl.sol#1146)
ArthController.setETHGMUOracle(address) (contracts/Arth/flat_ArthControl.sol#1149-1157) should emit an event for:
        - _ethGMUPricerDecimals = _ETHGMUPricer.getDecimals() (contracts/Arth/flat_ArthControl.sol#1156)
ArthController.setMintingFee(uint256) (contracts/Arth/flat_ArthControl.sol#1182-1188) should emit an event for:
        - mintingFee = fee (contracts/Arth/flat_ArthControl.sol#1187)
```



```
ArthController.constructor(address,address)._creatorAddress (contracts/Arth/flat_ArthControl.sol#1021) lacks a zero-check on :
                - creatorAddress = _creatorAddress (contracts/Arth/flat_ArthControl.sol#1022)
                - ownerAddress = _creatorAddress (contracts/Arth/flat_ArthControl.sol#1025)
ArthController.constructor(address,address)._timelockAddress (contracts/Arth/flat_ArthControl.sol#1021) lacks a zero-check on :
                - timelockAddress = _timelockAddress (contracts/Arth/flat_ArthControl.sol#1023)
ArthController.setARTHXAddress(address)._arthxAddress (contracts/Arth/flat_ArthControl.sol#1125) lacks a zero-check on :
                - arthxAddress = _arthxAddress (contracts/Arth/flat_ArthControl.sol#1130)
ArthController.setETHGMUOracle(address)._ethGMUConsumerAddress (contracts/Arth/flat_ArthControl.sol#1149) lacks a zero-check on :
                - ethGMUConsumerAddress = _ethGMUConsumerAddress (contracts/Arth/flat_ArthControl.sol#1154)
ArthController.setARTHXETHOracle(address,address)._arthxOracleAddress (contracts/Arth/flat_ArthControl.sol#1160) lacks a zero-check on :
                - arthxETHOracleAddress = _arthxOracleAddress (contracts/Arth/flat_ArthControl.sol#1163)
ArthController.setARTHXETHOracle(address,address)._wethAddress (contracts/Arth/flat_ArthControl.sol#1161) lacks a zero-check on :
                - wethAddress = _wethAddress (contracts/Arth/flat_ArthControl.sol#1165)
ArthController.setARTHETHOracle(address,address)._arthOracleAddress (contracts/Arth/flat_ArthControl.sol#1168) lacks a zero-check on :
                - arthETHOracleAddress = _arthOracleAddress (contracts/Arth/flat_ArthControl.sol#1173)
ArthController.setARTHETHOracle(address,address)._wethAddress (contracts/Arth/flat_ArthControl.sol#1168) lacks a zero-check on :
                - wethAddress = _wethAddress (contracts/Arth/flat_ArthControl.sol#1175)
```



```
ArthController.constructor(address,address) (contracts/Arth/flat_ArthControl.sol#1021-1037) uses literals with too many digits:
        - priceTarget = 1000000 (contracts/Arth/flat_ArthControl.sol#1034)
ArthController.constructor(address,address) (contracts/Arth/flat_ArthControl.sol#1021-1037) uses literals with too many digits:
        - globalCollateralRatio = 1000000 (contracts/Arth/flat_ArthControl.sol#1036)
ArthController.refreshCollateralRatio() (contracts/Arth/flat_ArthControl.sol#1043-1073) uses literals with too many digits:
        - globalCollateralRatio.add(arthStep) >= 1000000 (contracts/Arth/flat_ArthControl.sol#1065)
ArthController.refreshCollateralRatio() (contracts/Arth/flat_ArthControl.sol#1043-1073) uses literals with too many digits:
        - globalCollateralRatio = 1000000 (contracts/Arth/flat_ArthControl.sol#1066)
```



```
toggleCollateralRatio() should be declared external:
        - ArthController.toggleCollateralRatio() (contracts/Arth/flat_ArthControl.sol#1178-1180)
setMintingFee(uint256) should be declared external:
        - ArthController.setMintingFee(uint256) (contracts/Arth/flat_ArthControl.sol#1182-1188)
setArthStep(uint256) should be declared external:
        - ArthController.setArthStep(uint256) (contracts/Arth/flat_ArthControl.sol#1190-1196)
setRedemptionFee(uint256) should be declared external:
        - ArthController.setRedemptionFee(uint256) (contracts/Arth/flat_ArthControl.sol#1198-1204)
setOwner(address) should be declared external:
        - ArthController.setOwner(address) (contracts/Arth/flat_ArthControl.sol#1206-1212)
setPriceBand(uint256) should be declared external:
        - ArthController.setPriceBand(uint256) (contracts/Arth/flat_ArthControl.sol#1214-1220)
setTimelock(address) should be declared external:
        - ArthController.setTimelock(address) (contracts/Arth/flat_ArthControl.sol#1222-1228)
getGlobalCollateralRatio() should be declared external:
        - ArthController.getGlobalCollateralRatio() (contracts/Arth/flat_ArthControl.sol#1249-1251)
```

Concluding Remarks

While conducting the audits of MahaDAO smart contract - ArthController.sol, it was observed that the contracts contain High, Medium and Low severity issues, along with a few areas of recommendations.

Our auditors suggest that High, Medium and Low severity issues should be resolved by MahaDAO developers. Resolving the areas of recommendations are up to the team's discretion. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the MahaDAO platform or its product neither this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes Pvt Ltd.