MahaDAO

Smart Contract Audit Report

ArthPool.sol

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/

IMMUNE BYTES

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: ArthPool.sol
- Languages: Solidity(Smart contract)
- Github commit hash for audit:**6bac5389fbe25316614ceb10ee230aabb4dd07ff**
- GitHub link: https://github.com/MahaDAO/arthcoin-v2/blob/master/contracts/Arth/Pools/ArthPool.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	2	4
Closed	-	-	-

High severity issues

1. **"_AMO_ROLE" is initialized but never assigned to any address**
   _ Functions with onlyAMOS modifier are completely inaccessible
   Line no: 81, 134-137
   Description:
   The ArthPool contract uses **AccessControl** to assign specific roles in the contract to particular addresses.
   At Line 89, it initializes the **AMO_ROLE** along with other roles.

```
81      bytes32 private constant _AMO_ROLE = keccak256('AMO_ROLE');
82      bytes32 private constant _MINT_PAUSER = keccak256('MINT_PAUSER');
83      bytes32 private constant _REDEEM_PAUSER = keccak256('REDEEM_PAUSER');
84      bytes32 private constant _BUYBACK_PAUSER = keccak256('BUYBACK_PAUSER');
85
```

   However, while assigning these roles to a specific address using **grantRole() function** in the constructor (Line 182-187), no address is assigned for the **_AMO_ROLE.**

```
182             grantRole(_MINT_PAUSER, _timelockAddress);
183             grantRole(_REDEEM_PAUSER, _timelockAddress);
184             grantRole(_BUYBACK_PAUSER, _timelockAddress);
185             grantRole(_RECOLLATERALIZE_PAUSER, _timelockAddress);
186             grantRole(_COLLATERAL_PRICE_PAUSER, _timelockAddress);
187         }
```

   Moreover, the **AMO_ROLE** is used in the modifier **onlyAMOS** which is assigned to the following the functions in the contract:

   - **borrow** at **Line 338**
   - **repay** at **Line 350**

```
134     modifier onlyAMOS {
135         require(hasRole(_AMO_ROLE, _msgSender()), 'ArthPool: forbidden');
136         _;
137     }
```

   This **onlyAMOS** modifier ensures that only those addresses that have been assigned the **AMO_ROLE** should be able to call the function.
   However, since the AMO_ROLE is **not assigned** to any address in the constructor, this will lead to an extremely critical issue where the function **borrow** and **repay** will become completely inaccessible and can never be executed as they have the **onlyAMOS modifier** attached to them.

   Recommendation:
   The **_AMO_ROLE** must be assigned to a particular address in the constructor in order to avoid the above-mentioned scenario.

Medium severity issues

1. **Contract State Variables are being updated after External Calls.**
   Line no - 344-345, 360-362
   Description:
   The **ArthPool** contract includes quite a few functions that update some of the very imperative state variables of the contract after the external calls are being made. According to the Check_Effects_Interaction Pattern in Solidity, external calls should be made at the very end of the function and event emission, as well as any state variable modification, must be done before the external call is made.

   Therefore, as per the Solidity Guidelines, any modification of the state variables in the base contract must be performed before executing the external call.
   Updating state variables after an external call might lead to a potential re-entrancy scenario. Although the call is made to the Collateral address itself, the check-effects-interaction pattern must not be violated.

   The following functions in the contract updates the state variables and emits events after making an external call:
   - **borrow() function** at Line **344-345**
   - **repay() function** at Line **360-362**

```
338     function borrow(uint256 _amount) external override onlyAMOS {
339         require(
340             _COLLATERAL.balanceOf(address(this)) > _amount,
341             'ArthPool: Insufficent funds in the pool'
342         );
343 
344         _COLLATERAL.transfer(msg.sender, _amount);
345         borrowedCollateral[msg.sender] += _amount;
```

   Recommendation:
   Modification of any State Variables must be performed before making an external call. Check Effects Interaction Pattern must be followed while implementing external calls in a function.

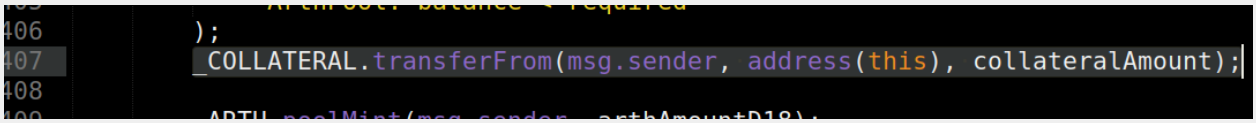
2. **Multiplication is being performed on the result of Division**
   Line no - 562-584, 626-632, 821-829
   Description:
   During the automated testing of the **ArthPool** contract, it was found that some of the functions in the contract are performing multiplication on the result of a Division. Integer 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:
   - **redeemFractionalARTH** at Line 562-584
   - **getCollateralGMUBalance** at Line 626-632
   - **estimateStabilityFeeInMAHA** at Line 821-829

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

Low severity issues

1. **Return Value of an External Call is never used Effectively**
   Line no -344, 360, 407, 489, 683, 685, 732, 775
   Description:
   The external calls made in the above-mentioned lines return a boolean value that indicates whether or not the external call made was successful.
   These boolean return values can be used in the function as a check to ensure that the execution of an external call was successful.
   However, the **ArthPool** contract does not use these return values throughout the contract.



   Recommendation:
   Effective use of all the return values from external calls must be ensured within the contract.

2. **Modifier onlyAdmin never used in the ArthPool Contract**
   Line no - 116
   Description:
   The ArthPool contract includes the **onlyAdmin** modifier at Line 116 but never uses it throughout the contract.

```
116     modifier onlyAdmin() {
117         require(
118             hasRole(DEFAULT_ADMIN_ROLE, _msgSender()),
119             'ArthPool: You are not the admin'
120         );
121         _;
122     }
```

   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 the **onlyAdmin** modifier holds no significance it should be removed from the contract.

3. **Comparison to boolean Constant**
   Line no: 299, 683,684,697, 749, 140, 144
   Description:
   Boolean constants can directly be used in conditional statements or require statements.

   Therefore, it's not considered a better practice to explicitly use **TRUE or FALSE** in the **require** or **IF-Else** statements.

```
748     {
749         require(buyBackPaused == false, 'Buyback is paused');
750
```

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

4. **External Visibility should be preferred**
   Description:
   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:

- **getCollateralGMUBalance** at Line 801

Recommendations:
If the public visibility of the above-mentioned is not intended, the function should be assigned an external keyword.

Recommendations

1. **Coding Style Issues in the Contract**
   Description:
   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.

```
Parameter ArthPool.setCollatETHOracle(address,address)._collateralWETHOracleAddress (contracts/Arth/flat_ArthPoo
Parameter ArthPool.setCollatETHOracle(address,address).__wethAddress (contracts/Arth/flat_ArthPool.sol#1564) is
Parameter ArthPool.setTimelock(address).new_timelock (contracts/Arth/flat_ArthPool.sol#1618) is not in mixedCase
Parameter ArthPool.setOwner(address).__ownerAddress (contracts/Arth/flat_ArthPool.sol#1626) is not in mixedCase
Parameter ArthPool.borrow(uint256)._amount (contracts/Arth/flat_ArthPool.sol#1634) is not in mixedCase
Variable ArthPool._ARTH (contracts/Arth/flat_ArthPool.sol#1329) is not in mixedCase
Variable ArthPool._ARTHX (contracts/Arth/flat_ArthPool.sol#1330) is not in mixedCase
Variable ArthPool._COLLATERAL (contracts/Arth/flat_ArthPool.sol#1331) is not in mixedCase
Variable ArthPool._MAHA (contracts/Arth/flat_ArthPool.sol#1332) is not in mixedCase
Variable ArthPool._ARTHMAHAOracle (contracts/Arth/flat_ArthPool.sol#1333) is not in mixedCase
```

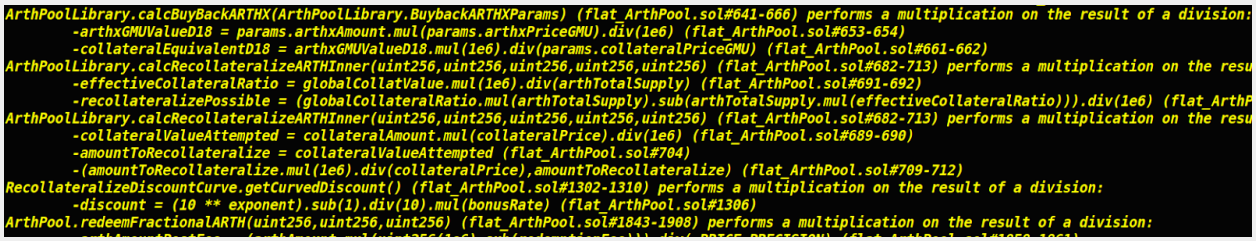
   During the automated testing, it was found that the ArthPool contract had quite a few code style issues.

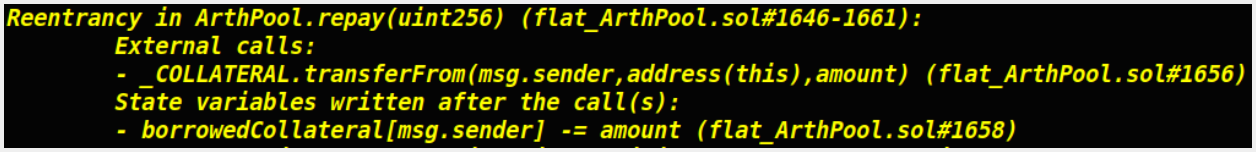
   Recommendation:
   Therefore, it is highly recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

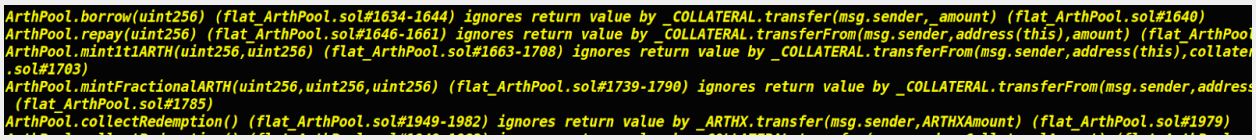
2. **NatSpec Annotations must be included**
   Description:
   The smart contracts do not include the NatSpec annotations adequately.

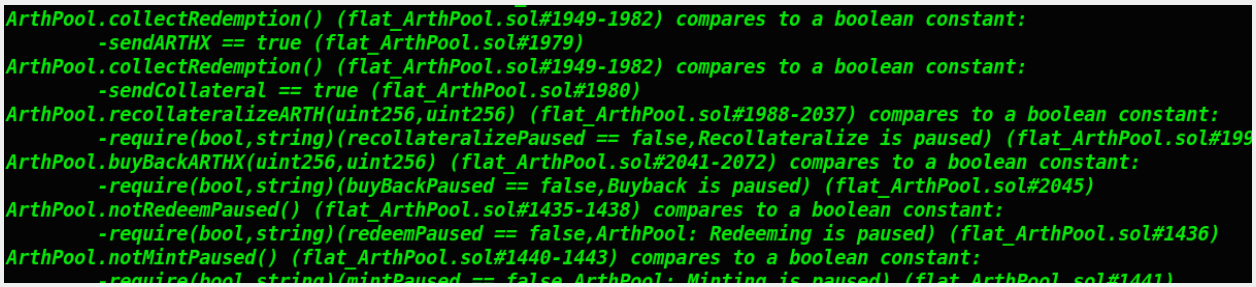
   Recommendation:
   Cover by NatSpec all Contract methods.

Automated Test Result









Concluding Remarks

While conducting the audits of MahaDAO smart contract - ArthPool.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.