MahaDAO

Smart Contract Audit Report

Arth.sol

May 02, 2021

Introduction 3
About MahaDAO 3
About ImmuneBytes 3
Documentation Details 3
Audit Process & Methodology 4
Audit Details 4
Audit Goals 5
Security Level References 5
High severity issues 6
Medium severity issues 6
Low severity issues 6
Recommendations 7
Automated Test Result 7
Concluding Remarks 8
Disclaimer 8

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: Arth.Sol
- Languages: Solidity(Smart contract)
- Github commit hash for audit:**932a78780b49e20bf5d69b6f7537be3e4ce963b9**
- GitHub Link:https://github.com/MahaDAO/arthcoin-v2/blob/master/contracts/Arth/Arth.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	-	-	2
Closed	-	-	-

High severity issues

No issues found.

Medium severity issues

No issues found.

Low severity issues

1. **Comparison to boolean Constant**
   Line no: 31, 66, 76
   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 statements.

```
71      function removePool(address pool)
72          external
73          override
74          onlyByOwnerOrGovernance
75      {
76          require(pools[pool] == true, "pool doesn't exist");
77          delete pools[pool];
78      }
```

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

2. **setGovernance function lacks a Zero Address Check.**
   Description:
   The **setGovernance** function doesn't validate the **_governance** address passed as a parameter.
   It is considered a better and secure practice in solidity to ensure that the address arguments passed to a function are not Zero Addresses.

```
80      function setGovernance(address _governance) external override onlyOwner {
81          governance = _governance;
82      }
```

   Recommendation:
   The governance address argument must be checked with a require statement.

require(_governance != address(0),"Invalid address passed");

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 ARTHStablecoin contract had a few code style issues.

```
Parameter ARTHStablecoin.setGovernance(address)._governance (flat_Arth.sol#1881) is not in mixedCase
Parameter ARTHStablecoin.setIncentiveController(IIncentiveController)._incentiveController (flat_Arth.sol#1885) is not in mixedCase
Constant ARTHStablecoin.genesisSupply (flat_Arth.sol#1824) 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:
   Smart contract does not include the NatSpec annotations adequately.
   The Coding style issues in a Smart Contract highly influences its code readability and in some cases may lead to bugs in future.

   Recommendation:
   It's recommended to use the Solidity Style Guide to fix all the issues. Moreover, the smart contract should be covered with NatSpec Annotations.

Automated Test Result

```
Parameter ARTHStablecoin.setGovernance(address)._governance (flat_Arth.sol#1881) is not in mixedCase
Parameter ARTHStablecoin.setIncentiveController(IIncentiveController)._incentiveController (flat_Arth.sol#1885) is not in mixedCase
Constant ARTHStablecoin.genesisSupply (flat_Arth.sol#1824) is not in UPPER_CASE_WITH_UNDERSCORES
```

```
ARTHStablecoin.addPool(address) (flat_Arth.sol#1866-1869) compares to a boolean constant:
        -require(bool,string)(pools[pool] == false,pool exists) (flat_Arth.sol#1867)
ARTHStablecoin.removePool(address) (flat_Arth.sol#1872-1879) compares to a boolean constant:
        -require(bool,string)(pools[pool] == true,pool doesn't exist) (flat_Arth.sol#1877)
ARTHStablecoin.onlyPools() (flat_Arth.sol#1831-1834) compares to a boolean constant:
        -require(bool,string)(pools[msg.sender] == true,ARTH: not pool) (flat_Arth.sol#1832)
```

```
ARTHStablecoin (flat_Arth.sol#1815-1908) does not implement functions:
        - Ownable.owner() (flat_Arth.sol#734-736)
        - Ownable.renounceOwnership() (flat_Arth.sol#753-756)
        - Ownable.transferOwnership(address) (flat_Arth.sol#762-769)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions
```

Concluding Remarks

While conducting the audits of MahaDAO smart contract - Arth.Sol, it was observed that the contracts contain only Low severity issues, along with a few areas of recommendations.

Our auditors suggest that 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.