MahaDAO

Security Assessment

January 21th, 2021

For :

MahaDAO

By :

Zach Zhou @ CertiK

jun.zhou@certik.org

Rudolph Wang @ CertiK

rudolph.wang@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	MahaDAO
Description	Non-Depreciating Currency
Platform	Solidity
Codebase	GitHub Repository
Commit	03eb502cf6f4255b901216a7113627cf2d5b49a7

Audit Summary

Delivery Date	Jan. 21, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Jan. 11, 2021 - Jan. 21, 2021

Vulnerability Summary

Total Issues	6
Total Critical	0
Total Major	1
Total Minor	0
Total Informational	5

Executive Summary

This report has been prepared for **MahaDAO** protocol to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

This audit is based on a premise that all the external methods cited are legally implemented. All calculation models of arthcoin v2 refers to MahaDAO's whitepaper.

Additionally, to bridge the trust gap between operator and users, operator needs to express a sincere attitude with the consideration of the operator team's anonymousness. The operator has the responsibility to notify users with the following capability of the operator:

- Operator can transfer assets in this contract under unpredicted cases via migrate method
- Owner have privilege to change current operator method

The advantage of migrate method in the protocol is that the operator reserves the ability to migrate the assets in this contract under unexpected cases. It is also worthy of note the downside of migrate method, where the treasury in this contract can be migrated to any addresses.

To improve the trustworthiness of the project, any dynamic runtime changes on the protocol should be notified to stakeholders.

For calling the migrate method, highly recommend moving to the execution queue of Timelock, and also emit events. This will increase your community awareness before the migration.

The community governance voting feature is not completed during this auditing period. However, **MahaDAO** mentioned the community voting feature will be completed before deployment.

This protocol has an external dependency. User deposits is transferred to a third-party service (like UniswapV2 Oracle), the price of LP Tokens in this system are determined by UniswapV2 Price Oracle. So the system should only be used if the service is appropriately trusted.

File in Scope

ID	Contract	SHA-256 Checksum
TR	Treasury.sol	5040aaede21eba0b2221fb7aef4714d93786c809b76c32651f33539dbc6131b6
BD	Boardroom.sol	95b379f61e7288e625e408875cb3b8c9ea05fffc2f7c3af0b7c74f597e042a6f
SF	Safe112.sol	a3d164b6520e14492ced61968311e453b1e1d6f6449f5ba8e6c1c5a4611b968c
OP	Operator.sol	bfc480fe8b8869468fba00a89ee5a836184037a2963892ae4945306d652b7b92
CG	ContractGuard.sol	1cd5c14c17d00067029e7de6d57996528435d868b79d54e241cacfceb88052d3
BA	IBasisAsset.sol	d7a8d5fa7fe782a9b8e4a9094c10e5c451b17518f9882afe8565662c067e52cb
ST	StakingTimelock.sol	5046fa0a1609bc42a19a401603612b7ea1a91501feebd85a35e4d6840febb462
CE	ICustomERC20.sol	c03cc733d8a1a7129796006ffa3208d0c11d8118b1aeb4d99ce40989b272e3de
UF	IUniswapV2Factory.sol	e5ed0d1a728bf038af0481fb8ba1f4d8dc303f4674a16b5a712af7f190a9189a

Findings

ID	Title	Type	Severity	Resolved
TR-01	Old Compiler Version Declaration	Optimization	Informational	✓
TR-02	Potentially excessive permissions	Permission	Major	✓
TR-03	Approve after transfer	Optimization	Informational	✓
BD-04	Missing zero address validation	Optimization	Informational	✓
ST-05	Use storage to save gas	Gas Optimization	Informational	✓
TR-06	Confused calculation	Optimization	Informational	!

Exhibit-01: Old Compiler Version Declaration

Type	Severity	Location
Optimization	Informational	Boardroom.sol L1, Treasury.sol L1

Description:

`solc` frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks.

Recommendation:

Deploy with any of the following Solidity versions:

- 0.5.11 - 0.5.13,
- 0.5.15 - 0.5.17,
- 0.6.8,
- 0.6.10 - 0.6.11. Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

Alleviation:

Resolved

Exhibit-02: Potentially excessive permissions

Type	Severity	Location
Permission	Major	Treasury.sol L92

Description:

In `migrate` function, the operator could transfers all assets without authorization without delay.

```
Operator(cash).transferOperator(target);
Operator(cash).transferOwnership(target);
ICustomERC20(cash).transfer(
    target,
    ICustomERC20(cash).balanceOf(address(this))
);
// bond
Operator(bond).transferOperator(target);
Operator(bond).transferOwnership(target);
ICustomERC20(bond).transfer(
    target,
    ICustomERC20(bond).balanceOf(address(this))
);
// share - disabled ownership and operator functions as MAHA tokens
don't have these
ICustomERC20(share).transfer(
    target,
    ICustomERC20(share).balanceOf(address(this))
);
```

Recommendation:

Add timelock and community vote for the `migrate` function.

Alleviation:

MahaDAO responses that this is actually also fixed because we have a timelock.sol contract.

CertiK requests: this operation is sensitive, it's better to add community voting before migrate.

MahaDAO responses the community voting will be completed before deployment.

Exhibit-03: Approve after transfer

Type	Severity	Location
Optimization	Informational	Treasury.sol 165

Description:

It's better to set the approve 0 after transfer

```
    // 2. Approve dai for trade on uniswap
    ICustomERC20(dai).safeApprove(uniswapRouter, amountInDai);

    // 3. Swap dai for ARTH from uniswap and send the ARTH to the sender
    // we send the ARTH back to the sender just in case there is some
    slippage
    // in our calculations and we end up with more ARTH than what is needed.
    uint256[] memory output =
        IUniswapV2Router02(uniswapRouter).swapExactTokensForTokens(
            amountInDai,
            expectedCashAmount,
            path,
            msg.sender,
            block.timestamp
        );
```

Recommendation:

It's better to set approve 0 after swap dai for ARTH from uniswap.

```
    ICustomERC20(dai).safeApprove(uniswapRouter, 0);
```

Alleviation:

Resolved

Exhibit-04: Missing zero address validation

Type	Severity	Location
Optimization	Informational	Treasury.sol L92, Boardroom.sol L108 L130 L138 L152

Description:

Detect missing zero address validation.

Recommendation:

Consider adding validation for address.

For example:

```
function migrate(address target) public onlyOperator checkOperator {
    require(target != address(0), "ERC20: migrate to the zero address")
    require(!migrated, 'Treasury: migrated');
```

Alleviation:

Resolved

Exhibit-05: Use storage to save gas

Type	Severity	Location
Gas Optimization	Informational	StakingTimelock.sol L26

Description:

Use `storage` to save gas.

```
StakingDetails memory _stakerDetails = _stakingDetails[msg.sender];
```

Recommendation:

Consider adding validation for address.

For example:

```
StakingDetails storage _stakerDetails = _stakingDetails[msg.sender];
```

Alleviation:

Resolved

Exhibit-06: Confused calcluation

Type	Severity	Location
Optimization	Informational	Treasury.sol L191

Description:

It is confused that `dondsToIssue` contains `bondDiscount` but `accumulatedBonds` doesn't contains it according to the formula.

```
accumulatedBonds = accumulatedBonds.add(cashToConvert);
```

Recommendation:

```
accumulatedBonds = accumulatedBonds.add(bondToIssue);
```

Alleviation:

No allevation

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

✓ : Issue resolved

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.