Security Assessment

Meshcoin

May 21st, 2021

Table of Contents

Summary

This report has been prepared for Meshcoin smart contracts, 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 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.

The security assessment resulted in 17 findings that ranged from major to discussion. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Meshcoin
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/meshcoin-project/contract
Commits	0d22d00d1ebcd587d59368b5eb4e338d87d6ca30

Audit Summary

Delivery Date	May 21, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Total Issues	17
Critical	0
Major	1
Medium	0
Minor	2
Informational	14
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
MMC	Meshcoin.sol	2b063f273abd1fdee3d6c38f9cc3104bd0af5c859be8236aeb4f59ede7223231
MPM	MeshcoinPools.sol	f2141b66197a590ff9cc182b16a36ce2b3dc299f318dbc58b6335a1acf084aa0

Certralization Roles

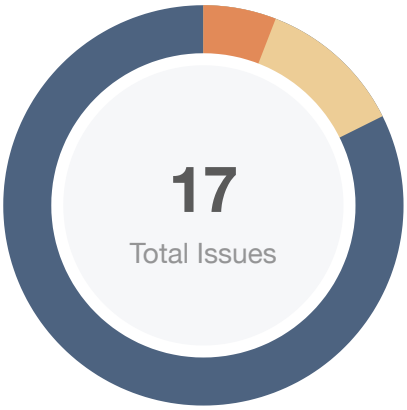
The Meshcoin smart contract introduces an authorization.

Owner:

[Meshcoin.sol]:
setpool(): set the value of `mscpools`;
mint(): mint tokens;

[MeshcoinPools.sol]:
add(): add a new lp token to the pool;
setRewardPerBlock(): set the number of `MSC` produced by each block;
setRewardDistributionFactor(): set the value of `rewardDistributionFactor`;
setAllocPoint(): update the given pool's `MSC` allocation point;
setPoolType(): set pool display type on frontend;
setReductionArgs(): set the value of `reductionBlockPeriod` and `maxReductionCount`;

Findings



Critical	0 (0.00%)
Major	1 (5.88%)
Medium	0 (0.00%)
Minor	2 (11.76%)
Informational	14 (82.35%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
MMC-01	Missing Zero Address Validation	Logical Issue	Informational	Resolved
MMC-02	Proper Usage of `public` And `external` Type	Gas Optimization	Informational	Resolved
MMC-03	Integer Overflow Risk	Mathematical Operations	Informational	Resolved
MPM-01	Missing Zero Address Validation	Logical Issue	Informational	Resolved
MPM-02	Check Effect Interaction Pattern Violated	Logical Issue	Informational	Resolved
MPM-03	Missing Emit Event	Coding Style	Informational	Declined
MPM-04	Proper Usage of `public` And `external` Type	Gas Optimization	Informational	Resolved
MPM-05	add() Function Not Restricted	Volatile Code	Major	Resolved
MPM-06	Recommended Explicit Pool Validity Checks	Logical Issue	Informational	Resolved
MPM-07	Discussion For `MeshCoin` Contract	Logical Issue	Informational	Resolved
MPM-08	Lack of Conditional Judgement	Logical Issue	Informational	Acknowledged
MPM-09	Lack Of Operations On `rewardDept` And `rewardRemain`	Logical Issue	Minor	Resolved
MPM-10	Incorrect Calculation For `value`	Logical Issue	Minor	Declined
MPM-11	Check Effect Interaction Pattern Violated	Logical Issue	Informational	Resolved

ID	Title	Category	Severity	Status
MPM-12	Discussion For The `getBlockReward` Function	Logical Issue	Informational	Acknowledged
MPM-13	Check Effect Interaction Pattern Violated	Logical Issue	Informational	Resolved
MPM-14	Meaningless Operation On `user.rewardDept`	Logical Issue	Informational	Resolved

MMC-01 | Missing Zero Address Validation

Category	Severity	Location	Status
Logical Issue	Informational	Meshcoin.sol: 11, 26, 31	Resolved

Description

Addresses should be checked before assignment to make sure they are not zero addresses. This suggestion is not limited to these codes but also applies to other similar codes.

Recommendation

Consider adding a check like below:
constructor():

```
require(_premint != address(0), "_premint address cannot be 0");
require(_investor != address(0), "_investor address cannot be 0");
```

setpool():

```
require(_pool != address(0), "_pool address cannot be 0");
```

mint():

```
require(_to != address(0), "_to address cannot be 0");
```

Alleviation

The team heeded our advice and changed related codes. Code change was applied in commit 49f9b6d4175d55dd25171cc59dd0ca1728f4222c.

MMC-02 | Proper Usage of `public` And `external` Type

Category	Severity	Location	Status
Gas Optimization	Informational	Meshcoin.sol: 31	Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

Consider using the `external` attribute for functions never called from the contract.

Alleviation

The team heeded our advice and changed related codes. Code change was applied in commit 49f9b6d4175d55dd25171cc59dd0ca1728f4222c.

MMC-03 | Integer Overflow Risk

Category	Severity	Location	Status
Mathematical Operations	Informational	Meshcoin.sol: 18~20	Resolved

Description

Using `*` in the method directly to calculate the value of the variable may overflow. `SafeMath` provides a method to verify overflow, and it is safer to use the method provided.

Recommendation

Using the functions in `SafeMath` library for mathematical operations. For example:

```
using SafeMath for uint256;
constructor (
            uint256 _totalSupply,
            address _premint,
            address _investor
        ) public ERC20('Meshcoin', 'MSC') {
        capmax =  _totalSupply;
        _mint(_premint, _totalSupply.mul(20).div(100).div(10000));
        _mint(_investor, _totalSupply.mul(3).div(100));
        _mint(address(0x0000000000000000000000000000000000000001),
_totalSupply.mul(70).div(100));
    }
```

Alleviation

The team heeded our advice and changed related codes. Code change was applied in commit 49f9b6d4175d55dd25171cc59dd0ca1728f4222c.

MPM-01 | Missing Zero Address Validation

Category	Severity	Location	Status
Logical Issue	Informational	MeshcoinPools.sol: 88, 89, 319, 325	Resolved

Description

Addresses should be checked before assignment to make sure they are not zero addresses. This suggestion is not limited to these codes but also applies to other similar codes.

Recommendation

Consider adding a check like below:
constructor():

```
require(_devaddr != address(0), "_devaddr address cannot be 0");
require(_opeaddr != address(0), "_opeaddr address cannot be 0");
```

dev():

```
require(_devaddr != address(0), "_devaddr address cannot be 0");
```

ope():

```
require(_opeaddr != address(0), "_opeaddr address cannot be 0");
```

Alleviation

The team heeded our advice and changed related codes. Code change was applied in commit 49f9b6d4175d55dd25171cc59dd0ca1728f4222c.

MPM-02 | Check Effect Interaction Pattern Violated

Category	Severity	Location	Status
Logical Issue	Informational	MeshcoinPools.sol: 255~260	Resolved

Description

The order of external call/transfer and storage manipulation must follow check effect interaction pattern.

Recommendation

We advice client to check if storage manipulation is before the external call/transfer operation by considering following modification:

```
            user.rewardDebt = totalRewards(pool, user);
            if(_amount > 0) {
                user.amount = user.amount.add(_amount);
                pool.totalAmount = pool.totalAmount.add(_amount);
                pool.lpToken.safeTransferFrom(address(msg.sender), address(this),
_amount);
            }
```

Alleviation

The team heeded our advice and changed related codes. Code change was applied in commit 49f9b6d4175d55dd25171cc59dd0ca1728f4222c.

MPM-03 | Missing Emit Event

Category	Severity	Location	Status
Coding Style	Informational	MeshcoinPools.sol: 106, 121, 126, 132, 139, 143	Declined

Description

Some functions should be able to emit event as notifications to customers because they change the status of sensitive variables.This suggestion is not limited to these codes, but also applies to other similar codes.

Recommendation

Consider adding an emit after changing the status of variables.

Alleviation

No alleviation.

MPM-04 | Proper Usage of `public` And `external` Type

Category	Severity	Location	Status
Gas Optimization	Informational	MeshcoinPools.sol: 106	Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

Consider using the `external` attribute for functions never called from the contract.

Alleviation

The team heeded our advice and changed related codes. Code change was applied in commit 49f9b6d4175d55dd25171cc59dd0ca1728f4222c.

MPM-05 | add() Function Not Restricted

Category	Severity	Location	Status
Volatile Code	Major	MeshcoinPools.sol: 106	Resolved

Description

The comment in line L105, mentioned // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.

The total amount of reward eggReward in function `updatePool()` will be incorrectly calculated if the same LP token is added into the pool more than once in function `add()`.

However, the code is not reflected in the comment behaviors as there isn't any valid restriction on preventing this issue.

The current implementation is relying on the trust of the owner to avoid repeatedly adding the same LP token to the pool, as the function will only be called by the owner.

Recommendation

Detect whether the given pool for addition is a duplicate of an existing pool. The pool addition is only successful when there is no duplicate. Using a mapping of `addresses` -> `booleans`, which can restricted the same address being added twice.

Alleviation

The team heeded our advice and changed related codes. Code change was applied in commit 49f9b6d4175d55dd25171cc59dd0ca1728f4222c.

MPM-06 | Recommended Explicit Pool Validity Checks

Category	Severity	Location	Status
Logical Issue	Informational	MeshcoinPools.sol: 189, 214, 247, 265, 282, 296	Resolved

Description

There's no sanity check to validate if a pool is existing.

Recommendation

We advise the client to adopt following modifier `validatePoolByPid` to functions `claimAll()`, `deposit()`, `withdraw()`, `emergencyWithdraw()`, `pendingRewards()` and `updatePool()`.

```
1 modifier validatePoolByPid(uint256 _pid) {
2     require (_pid < poolInfo.length , "Pool does not exist") ;
3     _;
4 }
```

Alleviation

The team heeded our advice and changed related codes. Code change was applied in commit 49f9b6d4175d55dd25171cc59dd0ca1728f4222c.

MPM-07 | Discussion For `MeshCoin` Contract

Category	Severity	Location	Status
Logical Issue	Informational	MeshcoinPools.sol: 20	Resolved

Description

When deploying the contract, 70% of the tokens will be allocated to `address(0x0000000000000000000000000000000000000001)`, which means that this part of the tokens cannot be retrieved. Can you provide a detail introduction about it please?

Alleviation

The team had removed related codes. Code change was applied in commit 49f9b6d4175d55dd25171cc59dd0ca1728f4222c.

MPM-08 | Lack of Conditional Judgement

Category	Severity	Location	Status
Logical Issue	Informational	MeshcoinPools.sol: 153	Acknowledged

Description

According to the logic of the function, there is only one interval to be used to calculate block reward. What if `_from` to `_to` cross several reduction intervals? Each interval has a different `rewardPerBlock`. Have you considered this situation?

Alleviation

Customer team response:
During the period of real reward mining activity, the update of rewards will be very frequent. The calculation of rewards across multiple reward intervals is not considered temporarily. Now, only consider two reward intervals. If there are more than two intervals, reward calculation will based on the nearest interval.

MPM-09 | Lack Of Operations On `rewardDept` And `rewardRemain`

Category	Severity	Location	Status
Logical Issue	Minor	MeshcoinPools.sol: 301	Resolved

Description

When users call the `emergencyWithdraw` function, `user.rewardDebt` and `user.rewardRemain` should be set to 0.

Recommendation

Consider setting `user.rewardDebt` and `user.rewardRemain` to 0.

Alleviation

The team heeded our advice and changed related codes. Code change was applied in commit 49f9b6d4175d55dd25171cc59dd0ca1728f4222c.

MPM-10 | Incorrect Calculation For `value`

Category	Severity	Location	Status
Logical Issue	Minor	MeshcoinPools.sol: 173	Declined

Description

When the value of `rewardDistributionFactor` is `1e9`, the calculation is meaningless. If its value was changed after the contract is deployed. The returned value of `getBlocksReward()` will lose or add some precision. For example, if `rewardDistributionFactor` was set to `1e18`, `value` will contains `1e9` and this precision should be removed in the subsequent formula.

Recommendation

Consider removing the calculation.

Alleviation

Customer team response:
`rewardDistributionFactor` is used as a profit adjustment parameter under special circumstances.

MPM-11 | Check Effect Interaction Pattern Violated

Category	Severity	Location	Status
Logical Issue	Informational	MeshcoinPools.sol: 288~290	Resolved

Description

The order of external call/transfer and storage manipulation must follow check effect interaction pattern.

Recommendation

We advice client to check if storage manipulation is before the external call/transfer operation by considering following modification:

```
            user.rewardRemain = 0;
            user.rewardDebt = totalRewards(pool, user);
            safeMscTransfer(msg.sender, value);
```

Alleviation

The team heeded our advice and changed related codes. Code change was applied in commit 49f9b6d4175d55dd25171cc59dd0ca1728f4222c.

MPM-12 | Discussion For The `getBlockReward` Function

Category	Severity	Location	Status
Logical Issue	Informational	MeshcoinPools.sol: 177~200	Acknowledged

Description

According to the contract design, block rewards gradually decreases, and eventually stabilize. The calculation of the reward in the function is based on the current block using a fixed proportion calculation. We think that it should be calculated separately for the attenuation block, and finally sum all block rewards. For example, Assuming that the reward for each block is 100, the reward is reduced from the second block, and the final reward is:

```
100 + (100 * 0.8) + 100 * (0.8 ** 2 ) + 100 * (0.8 ** 3)
```

but not :

```
100 * 4 * (0.8 **3)
```

Alleviation

Customer team response:
During the period of real reward mining activity, the update of rewards will be very frequent. The calculation of rewards across multiple reward intervals is not considered temporarily. Now, only consider two reward intervals. If there are more than two intervals, reward calculation will based on the nearest interval.

MPM-13 | Check Effect Interaction Pattern Violated

Category	Severity	Location	Status
Logical Issue	Informational	MeshcoinPools.sol: 272~277	Resolved

Description

The order of external call/transfer and storage manipulation must follow check effect interaction pattern.

Recommendation

We advice client to check if storage manipulation is before the external call/transfer operation by considering following modification:

```
            user.rewardDebt = totalRewards(pool, user);
            if(_amount > 0) {
                user.amount = user.amount.sub(_amount);
                pool.totalAmount = pool.totalAmount.sub(_amount);
                pool.lpToken.safeTransfer(address(msg.sender), _amount);
        }
```

Alleviation

The team heeded our advice and changed related codes. Code change was applied in commit 49f9b6d4175d55dd25171cc59dd0ca1728f4222c.

MPM-14 | Meaningless Operation On `user.rewardDept`

Category	Severity	Location	Status
Logical Issue	Informational	MeshcoinPools.sol: 271	Resolved

Description

`totalReward` function and `user.rewardDept` are not related, the operation is meaningless, we recommend removing this operation.

Recommendation

Consider removing `user.rewardDebt = 0;`.

Alleviation

The team heeded our advice and changed related codes. Code change was applied in commit 49f9b6d4175d55dd25171cc59dd0ca1728f4222c.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings 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 findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

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

Volatile Code

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

Coding Style

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

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 assessment. This report does 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.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.