CERTIK

Security Assessment

Maidcoin

Aug 3rd, 2021

Table of Contents

Summary

This report has been prepared for Maidcoin to discover issues and vulnerabilities in the source code of the Maidcoin project 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 findings that ranged from critical to informational. 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 that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Majority of the findings are of informational nature with three minor two medium and one major findings. The minor findings comprise lack of validation for constructor or function parameters and non-restriction of public state modifying function. The major finding comprises wrong `require` check. The medium findings comprise signature malleability and lack of input sanitization. All of the findings are remediated as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

Overview

Project Summary

Project Name	Maidcoin
Description	The audited codebase comprises ERC721 contracts of CloneNurse and Maid, ERC20 contracts of MaidCoin and MasterCoin, and ERC1155 contract of NursePart. Users can support a maid by providing LP tokens in Maid contract, then deposit MaidCoin and Maid to enter a Raid in NurseRaid contract and receive NursePart when exiting. The NursePart can be used in CloneNurse contract to deposit power in TheMaster contract in exchange of CloneNurse token and receive rewards. CloneNurse contract also allows supporting of existing CloneNurse tokens through TheMaster contract. TheMaster contract distributes MaidCoin to the users participating in pools.
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/maidcoingit/maidcoin/tree/51cfc4e6778bb46572d8a507337fae3292b243ba/contracts
Commit	51cfc4e6778bb46572d8a507337fae3292b243ba

Audit Summary

Delivery Date	Aug 03, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

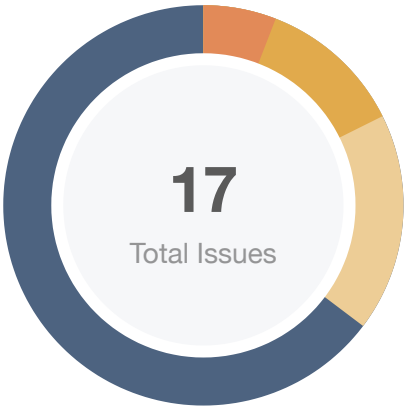
Vulnerability Summary

Vulnerability Level	Total	Pending	Partially Resolved	Resolved	Acknowledged	Declined
Critical	0	0	0	0	0	0
Major	1	0	0	1	0	0
Medium	2	0	0	2	0	0
Minor	3	0	0	3	0	0
Informational	11	0	0	10	0	1
Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
ICN	interfaces/ICloneNurse.sol	381e1b2b0e05a7b59513e17b736921204b0ec8239fd7fbe61f25ed580c655ae8
IER	interfaces/IERC1271.sol	1192e26fc5f75b40512cb8dcaa3e32a611228150df4e909a064902fef78ebf0b
IMT	interfaces/IMaid.sol	8ef09eee9054ab9bf4c20cd5150f059ad10be9eee6174e22310c6caabf42e8d8
IMC	interfaces/IMaidCoin.sol	f2de8c086fc0784281356eabdd0ae9b9c170c9f1387d03aac3ba4552498354af
INP	interfaces/INursePart.sol	e2bea73a5c620cc3ef127cde9277fd21ed0eb1927debf14452df546f4f0077a1
INR	interfaces/INurseRaid.sol	b29a8e41269acabca42230a4e82b3e5444e6bc9f6e85744c39a7115bf2e3a3a3
IRN	interfaces/IRNG.sol	d2082d2058ed06678901396e9881c0785ead24bfe621ebfe036ec1d9a59bdeb2
IRC	interfaces/IRewardCalculator.sol	c7f1da4aacc1e4efce995a0f13e0e87768f20feb66d05351495c3418098de91a
IST	interfaces/ISupportable.sol	c60161888e284ab94d1278b97cfe471ca089aaea32d44e9fdbc0997b26901269
ITM	interfaces/ITheMaster.sol	8881f1ed63a9ec3c816d8593721ada2ccb65d943a8a7e348e9452697167702c8
CNT	CloneNurse.sol	9c58d86c456b92c6c04615f8ac10b8ccb2245c8a1bd8dfe61c0f68961d9496f4
MAI	Maid.sol	eb313438acc93e9e81bcc3dfe37255ce9e5380c21c4c458533e65fb53da07990
MCT	MaidCoin.sol	3521b6b66f7677e07446e6e6bded4c5c3167aa6d07594144149684647c1b320e
MCI	MasterCoin.sol	96431910f850d42dac36fb8750f0c6e3c212bcc028e2667eb0418526b0e5f1c4
NPT	NursePart.sol	46f290c8cf2ffe36b69dddd20dbd2d26d1cd06e0303140215a044fc861a917c6
NRT	NurseRaid.sol	17f4aec4c549f39c8cd5713c7e68696a1be186ad55bdd99a02cbf0c7363c4c51
TMT	TheMaster.sol	52d8c93357fd60a569b5446a590058bf09b7d7d37164988699d88ba5370b3e96

Findings



Critical	0 (0.00%)
Major	1 (5.88%)
Medium	2 (11.76%)
Minor	3 (17.65%)
Informational	11 (64.71%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Unlocked Compiler Version	Language Specific	Informational	Declined
CNT-01	Missing Error Messages	Language Specific	Informational	Resolved
CNT-02	Non-standard contract's order of layout	Language Specific	Informational	Resolved
CNT-03	Function is publicly callable	Volatile Code	Minor	Resolved
CNT-04	Ineffectual `require` check	Volatile Code	Informational	Resolved
MAI-01	Missing Error Messages	Language Specific	Informational	Resolved
MAI-02	Non-standard contract's order of layout	Language Specific	Informational	Resolved
MAI-03	Possibility of Replay Attack in `permit` and `permitAll`	Logical Issue	Minor	Resolved
MAI-04	Signature malleability	Volatile Code	Medium	Resolved
NPT-01	Missing Error Messages	Language Specific	Informational	Resolved
NPT-02	Possibility of Replay Attack in `permit` and `permitAll`	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
NRT-01	Missing Error Messages	Language Specific	Informational	Resolved
NRT-02	Non-standard contract's order of layout	Language Specific	Informational	Resolved
NRT-03	Lack of input sanitization	Data Flow	Medium	Resolved
TMT-01	Inefficient storage read	Gas Optimization	Informational	Resolved
TMT-02	Comparison with literal `true`	Gas Optimization	Informational	Resolved
TMT-03	Wrong conditional in `require` statement	Logical Issue	Major	Resolved

GLOBAL-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	Global	Declined

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.8.5` the contract should contain the following line: `pragma solidity 0.8.5;`.

Alleviation

No alleviations.

CNT-01 | Missing Error Messages

Category	Severity	Location	Status
Language Specific	Informational	CloneNurse.sol: 11	Resolved

Description

The contract is missing error messages for most of the `require` calls.

Recommendation

We recommend adding error messages for `require` calls to increase the legibility of codebase.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

CNT-02 | Non-standard contract's order of layout

Category	Severity	Location	Status
Language Specific	Informational	CloneNurse.sol: 13~15	Resolved

Description

The contract has non-standard order of layout.

Recommendation

We advise to revisit the contract and observe the recommended order of layout from Solidity's official documentation.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

CNT-03 | Function is publicly callable

Category	Severity	Location	Status
Volatile Code	Minor	CloneNurse.sol: 138	Resolved

Description

The function `checkSupportingRoute` on the aforementioned line changes the token id that a supporter supports if the current supporting token has been destroyed. Although, its public state of access restriction does not possess any vulnerability, we still advise to make this function `internal` as it modifies the contract's state unless there is a reason to keep its visibility `public`.

Recommendation

we advise to make the aforementioned function `internal` as it can be called by anyone and modifies contract's state.

Alleviation

The client provided feedback that "If there's a serial destroying of CloneNurse in a future, some users can't claim their yield or withdraw their properties. Because while loop is in checkSupportingRoute function.".

CNT-04 | Ineffectual `require` check

Category	Severity	Location	Status
Volatile Code	Informational	CloneNurse.sol: 169	Resolved

Description

The `require` check on the aforementioned line is ineffectual as the enclosing function is `internal` and the calling function `shareRewards` already contains the aforementioned `require` check.

Recommendation

We advise to remove the ineffectual `require` check from the aforementioned line.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

MAI-01 | Missing Error Messages

Category	Severity	Location	Status
Language Specific	Informational	Maid.sol: 11	Resolved

Description

The contract is missing error messages for most of the `require` calls.

Recommendation

We recommend adding error messages for `require` calls to increase the legibility of codebase.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

MAI-02 | Non-standard contract's order of layout

Category	Severity	Location	Status
Language Specific	Informational	Maid.sol: 13	Resolved

Description

The contract has non-standard order of layout.

Recommendation

We advise to revisit the contract and observe the recommended order of layout from Solidity's official documentation.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

MAI-03 | Possibility of Replay Attack in `permit` and `permitAll`

Category	Severity	Location	Status
Logical Issue	Minor	Maid.sol: 99, 132	Resolved

Description

The `permit` and `permitAll` functions performs the operation of deriving signer address from the signature values of `v`,`r` and `s`. The state variable `DOMAIN_SEPARATOR` that is used to calculate hash has a value of `chainid` that is derived only once in constructor, which does not change after contract deployment. The issue arises in the event of fork when the cross-chain replay attacks can be executed.

The attack scenario can be thought of as if a fork of Ethereum happens and two different networks have id of for example 1 and 9 . The chainid coded in `DOMAIN_SEPARATOR` will be the same on contracts residing in both of the forks. If the chainid 1 is stored in the contract then the permit transaction signed for chainid 1 will be executable on both of the forks.

Recommendation

We advise to construct the `DOMAIN_SEPRATOR` hash inside the `permit()` and `permitAll` functions so the current `chainid` could be fetched and only the transactions signed for current network could succeed.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

MAI-04 | Signature malleability

Category	Severity	Location	Status
Volatile Code	Medium	Maid.sol: 154, 69	Resolved

Description

The signature system is prone to signature malleability as it does not account for the pitfalls of EIP-2(https://eips.ethereum.org/EIPS/eip-2)

Recommendation

To fix this we would recommend adding check from EIP-2, point 2 (https://eips.ethereum.org/EIPS/eip-2), and also check for the v value to ensure off chain library is properly used. Look into ecrecoverFromSig function from SWC-117 (https://swcregistry.io/docs/SWC-117).

OpenZeppelin ECDSA library contract contains proper implementation for recovering address from signature that isn't prone to signature malleability. We suggest importing that and using it in the contract.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

NPT-01 | Missing Error Messages

Category	Severity	Location	Status
Language Specific	Informational	NursePart.sol: 9	Resolved

Description

The contract is missing error messages for most of the `require` calls.

Recommendation

We recommend adding error messages for `require` calls to increase the legibility of codebase.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

NPT-02 | Possibility of Replay Attack in `permit` and `permitAll`

Category	Severity	Location	Status
Logical Issue	Minor	NursePart.sol: 47	Resolved

Description

The `permit` and `permitAll` functions performs the operation of deriving signer address from the signature values of `v`,`r` and `s`. The state variable `DOMAIN_SEPARATOR` that is used to calculate hash has a value of `chainid` that is derived only once in constructor, which does not change after contract deployment. The issue arises in the event of fork when the cross-chain replay attacks can be executed.

The attack scenario can be thought of as if a fork of Ethereum happens and two different networks have id of for example 1 and 9 . The chainid coded in `DOMAIN_SEPARATOR` will be the same on contracts residing in both of the forks. If the chainid 1 is stored in the contract then the permit transaction signed for chainid 1 will be executable on both of the forks.

Recommendation

We advise to construct the `DOMAIN_SEPRATOR` hash inside the `permit()` and `permitAll` functions so the current `chainid` could be fetched and only the transactions signed for current network could succeed.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

NRT-01 | Missing Error Messages

Category	Severity	Location	Status
Language Specific	Informational	NurseRaid.sol: 7	Resolved

Description

The contract is missing error messages for most of the `require` calls.

Recommendation

We recommend adding error messages for `require` calls to increase the legibility of codebase.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

NRT-02 | Non-standard contract's order of layout

Category	Severity	Location	Status
Language Specific	Informational	NurseRaid.sol: 27~34, 45	Resolved

Description

The contract has non-standard order of layout.

Recommendation

We advise to revisit the contract and observe the recommended order of layout from Solidity's official documentation.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

NRT-03 | Lack of input sanitization

Category	Severity	Location	Status
Data Flow	Medium	NurseRaid.sol: 67, 149	Resolved

Description

The `maxRewardCount` parameter should be sanitized to be not greater than `55` as `2` is raised to the power of `maxRewardCount` on `L149` and having value greater than `55` f.e. `2**56` will revert the transaction.

Recommendation

We advise to revisit the code and introduce a validation check for `maxRewardCount`.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

TMT-01 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	Informational	TheMaster.sol: 63~64	Resolved

Description

The aforementioned lines repeatedly perform storage read `pool.lastRewardBlock` which results in significant gas cost.

Recommendation

We advise to utilize a local variable to the address `pool.lastRewardBlock` and then utilize it in the aforementioned lines to save gas cost associated with repeated storage reads.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

TMT-02 | Comparison with literal `true`

Category	Severity	Location	Status
Gas Optimization	Informational	TheMaster.sol: 118	Resolved

Description

The aforementioned line performs comparison with literal `true` which can be substituted with the expression itself to save gas cost.

Recommendation

We advise to substitute the comparison with literal `true` with the expression itself.

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

TMT-03 | Wrong conditional in `require` statement

Category	Severity	Location	Status
Logical Issue	Major	TheMaster.sol: 366	Resolved

Description

The condition in `require` statement on the aforementioned line expects that `supportable` address to be zero for the desupport of a pool. The support/desupport can only be performed for the pools that have non-zero `supportable` addresses.

Recommendation

We advise to rectify the conditional on the aforementioned line.

```
require(address(supportable) != address(0), "TheMaster: use emergencyWithdraw func");
```

Alleviation

Alleviations are applied as of commit hash `4e61aa1605b210a38bcf65aea729ba52417f5552`.

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.

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.

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 in-storage one.

Language Specific

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

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.