Smart Contract Audit Report

for

MaxiDao

TRUSTLOOK

Version 0.1

Trustlook Blockchain Labs

Email: bd@trustlook.com

TRUSTLOOK

Project Overview

Project Name	MaxiDao
Contract codebase	N/A
Platform	Ethereum
Language	Solidity
Submission Time	2021.08.27

Report Overview

Report ID	TBL_20210827_00
Version	0.1
Reviewer	Trustlook Blockchain Labs
Starting Time	2021.08.27
Finished Time	2021.08.29

Disclaimer

Trustlook audit reports do not provide any warranties or guarantees on the vulnerability-free nature of the given smart contracts, nor do they provide any indication of legal compliance. The Trustlook audit process is aiming to reduce the high level risks possibly implemented in the smart contracts before the issuance of audit reports. Trustlook audit reports can be used to improve the code quality of smart contracts and are not able to detect any security issues of smart contracts that will occur in the future. Trustlook audit reports should not be considered as financial investment advice.

About Trustlook Blockchain Labs

Trustlook Blockchain Labs is a leading blockchain security team with a goal of security and vulnerability research on current blockchain ecosystems by offering industry-leading smart contracts auditing services. Please contact us for more information at (https://www.trustlook.com/services/smart.html) or Email (bd@trustlook.com)

The Trustlook blockchain laboratory has established a complete system test environment and methods.

Black-box Testing	The tester has no knowledge of the system being attacked. The goal is to simulate an external hacking or cyber warfare attack.
White-box Testing	Based on the level of the source code, test the control flow, data flow, nodes, SDK etc. Try to find out the vulnerabilities and bugs.
Gray-box Testing	Use Trustlook customized script tools to do the security testing of code modules, search for the defects if any due to improper structure or improper usage of applications.

Introduction

By reviewing the implementation of MaxiDao's smart contracts, this audit report has been prepared to discover potential issues and vulnerabilities of their source code. We outline in the report about our approach to evaluate the potential security risks. Advice to further improve the quality of security or performance is also given in the report.

About MaxiDao

MaxiDAO is a Decentralized Autonomous Organization built by and for crypto miners and pools. It started with a decentralized mining pool protocol on the Chia blockchain and built cross-chain bridges of Chia to the DeFi ecosystem.

About Methodology

To evaluate the potential vulnerabilities or issues, we go through a checklist of well-known smart contracts related security issues using automatic verification tools and manual review. To discover potential logic weaknesses or project specific implementations, we thoroughly discussed with the team to understand the business model and reduce the risk of unknown vulnerabilities. For any discovered issue, we might test it on our private network to reproduce the issue to prove our findings.

The checklist of items is shown in the following table:

Category	Type ID	Name	Description
Coding Specification	CS-01	ERC standards	The contract is using ERC standards.
	CS-02	Compiler Version	The compiler version should be specified.
	CS-03	Constructor Mismatch	The constructor syntax is changed with Solidity versions. Need extra attention to make the constructor function right.
	CS-04	Return standard	Following the ERC20 specification, the transfer and approve

			functions should return a bool value, and a return value code needs to be added.
	CS-05	Address(0) validation	It is recommended to add the verification of require(_to!=address(0)) to effectively avoid unnecessary loss caused by user misuse or unknown errors.
	CS-06	Unused Variable	Unused variables should be removed.
	CS-07	Untrusted Libraries	The contract should avoid using untrusted libraries, or the libraries need to be thoroughly audited too.
	CS-08	Event Standard	Define and use Event appropriately
	CS-09	Safe Transfer	Using transfer to send funds instead of send.
	CS-10	Gas consumption	Optimize the code for better gas consumption.
	CS-11	Deprecated uses	Avoid using deprecated functions.
	CS-12	Sanity Checks	Sanity checks when setting key parameters in the system
Coding Security	SE-01	Integer overflows	Integer overflow or underflow issues.
	SE-02	Reentrancy	Avoid using calls to trade in smart contracts to avoid reentrancy vulnerability.
	SE-03	Transaction Ordering Dependence	Avoid transaction ordering dependence vulnerability.
	SE-04	Tx.origin usage	Avoid using tx.origin for authentication.
	SE-05	Fake recharge	The judgment of the balance and the transfer amount needs to use the “require function”.
	SE-06	Replay	If the contract involves the demands for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks.
	SE-07	External call checks	For external contracts, pull instead of push is preferred.
	SE-08	Weak random	The method of generating random numbers on smart contracts requires more considerations.
Additional Security	AS-01	Access control	Well defined access control for functions.
	AS-02	Authentication management	The authentication management is well defined.
	AS-03	Semantic Consistency	Semantics are consistent.
	AS-04	Functionality checks	The functionality is well implemented.

	AS-05	Business logic review	The business model logic is implemented correctly.

The severity level of the issues are described in the following table:

Severity	**Description**
Critical	The issue will result in asset loss or data manipulations.
High	The issue will seriously affect the correctness of the business model.
Medium	The issue is still important to fix but not practical to exploit.
Low	The issue is mostly related to outedate, unused code snippets.
Informational	This issue is mostly related to code style, informational statements and is not mandatory to be fixed.

TRUSTLOOK

Audit Results

Here are the audit results of the smart contracts. The new release of the smart contracts add more features to restrict the privilege of the owner to reduce the risk of private key loss or hacking events.

Scope

Following files have been scanned by our internal audit tool and manually reviewed and tested by our team:

File names	Sha1
Bridge.sol	632facd15d4de2ce2d6834f854961aac176a9914
Controller.sol	b3452e3e821a522fe4bcad9f7a8695f23db7732a
Members.sol	251c62a43a6cd28ad2017e7bedc25a8a48e2a459
WXCH.sol	e494bc1c8791cb8b30086eae4583f919257b3a81

Summary

Issue ID	Severity	Location	Type ID	Status
TBL_SCA_001	Info	Bridge.sol:569	CS-10	open
TBL_SCA_002	Info	Bridge.sol:601	CS-10	open
TBL_SCA_003	Medium	WXCH.sol:653	AS-01	closed

Details

• ID: TBL_SCA-001

• Severity: Informational

• Type: CS-10 (Gas consumption)

• Description:

The validation of "*broker*" to be *address(0)* or not is not necessary, since the later validation "*controller.isBroker(broker)*" will cover this validation. Because there is already validation of broker before adding into the brokers list:

```
function addBroker(address broker) external override onlyOwner returns (bool) {
    require(broker != address(0), "invalid broker address");
    require(brokers.add(broker), "broker add failed");

    emit BrokerAdd(broker);
    return true;
}
```

• Remediation:

• ID: TBL_SCA-002

• Severity: Informational

• Type: CS-10 (Gas consumption)

• Description:

The parameter *depositAddress* is not necessary since it must be the value of "*custodianDepositAddress[msg.sender])*" when there is a validation of the value at line 608.

• Remediation:

• ID: TBL_SCA-003

• Severity: Medium

• Type: AS-01 (Access Control)

• Description:

Function burn() is designed to be called only by smart contract Controller, therefore, the modifier onlyOwner is needed.

• Remediation:

The team has fixed this issue in a new release.