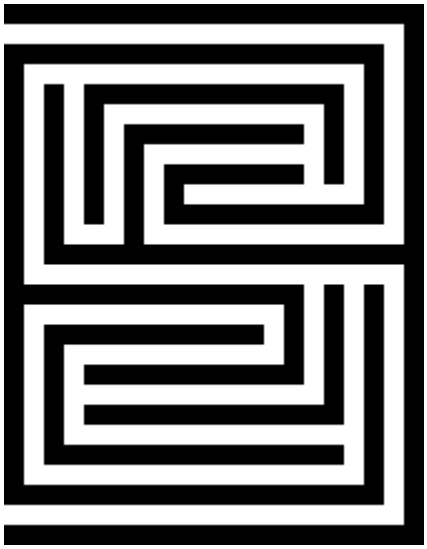

Mai Finance Security Review Public Report

PROJECT: Mai Finance Security Review

March 2021

Prepared For:

Qi Dao Protocol

Mai Finance

Prepared By:

Jonathan Haas | Bramah Systems, LLC.

jonathan@bramah.systems

Table of Contents

Executive Summary **3**
- Scope of Engagement 3
- Timeline 3
- Engagement Goals 3
- Contract Specification 3
- Overall Assessment 4
- Timeliness of Content 5

General Recommendations **6**
- Solidity version misses more recent security updates 6
- Centralized control should be moved to multi-signature wallet 6
- Uint and uint256 used interchangeably could lead to confusion 6

Specific Recommendations **7**
- Division by zero can result through modified EthPriceSource 7
- Setting owner should check for null address 7
- Tautology that always evaluate to true should be removed 7
- Treasury is used without being initialized 7

Toolset Warnings **9**
- Overview 9
- Compilation Warnings 9
- Test Coverage 9
- Static Analysis Coverage 9

Directory Structure **10**

Mai Finance Protocol Review

Executive Summary

Scope of Engagement

Bramah Systems, LLC was engaged in April of 2021 to perform a comprehensive security review of the Mai Finance smart contracts (specific contracts denoted within the appendix). Our review was conducted over a period of two business days by both members of the Bramah Systems, LLC. executive staff.

A secondary audit was performed at the request of the Qi Dao Protocol team, as of commit hash dd9dfc5d4c6db047ebaa41c70df35aa88814aa5c.

Bramah Systems completed the assessment using manual, static and dynamic analysis techniques.

Timeline

Review Commencement: April 12, 2021

Report Delivery: April 14th, 2021

Engagement Goals

The primary scope of the engagement was to evaluate and establish the overall security of the Mai Finance protocol, with a specific focus on trading actions. In specific, the engagement sought to answer the following questions:

- Is it possible for an attacker to steal or freeze tokens?
- Does the Solidity code match the specification as provided?
- Is there a way to interfere with the contract mechanisms?
- Are the arithmetic calculations trustworthy?

Contract Specification

Specification was provided in the form of code comments. The contracts were provided via

GitHub (commit hash **ebc17bc7c27fdea50a6cdfffc59a509067978b38**)

Overall Assessment

Bramah Systems was engaged to evaluate and identify any potential security concerns within the codebase of the Mai Finance protocol. During the course of our engagement, Bramah Systems found multiple instances wherein the team deviated materially from established best practices and procedures of secure software development within DLT. The team has since addressed these issues with a resolution or risk acceptance.

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held, commercially reasonable understanding of Bramah Systems, LLC.'s knowledge of security patterns as they relate to the Mai Finance Protocol, with the understanding that distributed ledger technologies ("DLT") remain under frequent and continual development, and resultantly carry with them unknown technical risks and flaws. The scope of the review provided herein is limited solely to items denoted within "Scope of Engagement" and contained within "Directory Structure". The report does NOT cover, review, or opine upon security considerations unique to the Solidity compiler, tools used in the development of the protocol, or distributed ledger technologies themselves, or to any other matters not specifically covered in this report.

The contents of this report must NOT be construed as investment advice or advice of any other kind. This report does NOT have any bearing upon the potential economics of the Mai Finance protocol or any other relevant product, service or asset of Mai Finance or otherwise. This report is not and should not be relied upon by Mai Finance or any reader of this report as any form of financial, tax, legal, regulatory, or other advice.

To the full extent permissible by applicable law, Bramah Systems, LLC. disclaims all warranties, express or implied. The information in this report is provided "as is" without warranty, representation, or guarantee of any kind, including the accuracy of the information provided. Bramah Systems, LLC. makes no warranties, representations, or guarantees about the Mai Finance Protocol. Use of this report and/or any of the information provided herein is at the users sole risk, and Bramah Systems, LLC. hereby disclaims, and each user of this report hereby waives, releases, and holds Bramah Systems, LLC. harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

Timeliness of Content

All content within this report is presented only as of the date published or indicated, to the commercially reasonable knowledge of Bramah Systems, LLC. as of such date, and may be superseded by subsequent events or for other reasons. The content contained within this report is subject to change without notice. Bramah Systems, LLC. does not guarantee or warrant the accuracy or timeliness of any of the content contained within this report, whether accessed through digital means or otherwise.

Bramah Systems, LLC. is not responsible for setting individual browser cache settings nor can it ensure any parties beyond those individuals directly listed within this report are receiving the most recent content as reasonably understood by Bramah Systems, LLC. as of the date this report is provided to such individuals.

General Recommendations

Best Practices & Solidity Development Guidelines

Solidity version misses more recent security updates

The Solidity version predominantly used, **0.5.2**, is multiple versions out of date and lacks security updates that have been introduced in later Solidity versions.

Update: The newly updated contracts also possess this issue, with a variety of early Solidity versions (including as early as 0.5.5)

Resolution: The team does not intend to update.

Centralized control should be moved to multi-signature wallet

A singular admin (the owner) address controls the price oracle feed (via function **changeEthPriceSource**). This address should be a multi-signature wallet to avoid centralized price control.

Resolution: The team intends to move to a multi-signature wallet for production deployment.

Uint and uint256 used interchangeably could lead to confusion

While both values ultimately alias to the same thing, usage of both variants within the same function (such as **getEthPriceSource**) could lead to confusion.

Resolution: The team has resolved this issue in this commit.

Sensitive variable changes should emit an event

Sensitive functions that change aspects that can materially impact flow of funds to users or the protocol (such as **setFeeAddress**) should emit an event.

Resolution: The team has not resolved this issue.

Owner controlled “fund” function can overflow

The **fund** function within **StakingRewards.sol** can overflow while setting the endBlock. **SafeMath** or a **require** statement should be used to mitigate these concerns.

Resolution: The team has stated they have introduced a modifier to the function including **onlyOwner**. They plan to introduce protection when integrating it with DAO governance

Adding duplicate LP token results in improper rewards

As described in the code below:

// Add a new lp to the pool. Can only be called by the owner.

// DO NOT add the same LP token more than once. Rewards will be messed up if you do.

If a token is inadvertently added more than once, it will reset the rewards variables associated with the token, and as a result, pay out improperly to end users. A control should be added to prevent this from occurring (such as validating against a list of prior submissions)

Resolution: The team has stated “Once decentralized governance is added we will make sure to protect the farm from adding duplicate rewards”

massUpdatePools can experience resource exhaustion

The **massUpdatePools** function within **StakingRewards.sol** can result in resource exhaustion as a result of the gas required in order to properly execute the **updatePool** functionality.

Resolution: The team has provided that “By deploying multiple farms as needed. The gas limit shouldn't be reached when there are <5 farms.”

Interfaces should go in the interfaces folder

Contracts that are interfaces, denoted by the letter **I** at the beginning of their name (and through internal code references to being an interface) should be placed in the “interfaces” folder for clarity. It’s not clear why this division exists, but it should be followed uniformly.

Resolution: The team has stated that “While it is best practice to maintain uniformity, the division does not present a security concern for the contract”

Specific Recommendations

Unique to the Mai Finance Protocol

Division by zero can result through modified EthPriceSource

The following statement can result in a division by zero in the event that the Matic price source has been modified to a value which does not return valid values.

```
uint256 _closingFee  =
(amount.mul(closingFee).mul(getTokenPriceSource())).div(getEthPriceSource().mul(10000));
```

Resolution: The team plans to introduce a middleware contract later to manage this risk, as Chainlink already aggregates data from multiple sources

Setting owner should check for null address

The **nominateNewOwner** function (contracts/Owned.sol#15) lacks a zero-check on **nominatedOwner = _owner** (contracts/Owned.sol#16)

Resolution: The team is not planning to change OpenZeppelin's contract. The team has stated they will note prior making ownership updates to make sure this isn't nulled.

Tautology that always evaluate to true should be removed

The **destroyVault** function (contracts/Stablecoin.sol#124-137) contains a tautology (**vaultCollateral[vaultID] >= 0**) and should be removed (contracts/Stablecoin.sol#127).

Resolution: The team has resolved this issue in this commit.

Treasury is used without being initialized

The Stablecoin **treasury** value (contracts/Stablecoin.sol#22) is used but never initialized.

Resolution: The team intends for the treasury value to be initialized once its contract is created.

Toolset Warnings

Unique to the Mai Finance Protocol

Overview

In addition to our manual review, our process involves utilizing static analysis and formal methods in order to perform additional verification of the presence of security vulnerabilities (or lack thereof). An additional part of this review phase consists of reviewing any automated unit testing frameworks that exist.

The following sections detail warnings generated by the automated tools and confirmation of false positives where applicable.

Compilation Warnings

No compilation warnings were encountered during the course of our audit.

Test Coverage

The contracts possess a number of functional unit tests encompassing various stages of the application lifecycle.

Static Analysis Coverage

The contract repository underwent heavy scrutiny with multiple static analysis agents, including:

- Securify
- MAIAN
- Mythril
- Oyente
- Slither

Directory Structure

At time of review, the directory structure of the Mai Finance smart contracts repository appeared as it does below. Our review, at request of Mai Finance, covers the Solidity code (*.sol) as of commit hash **ebc17bc7c27fdea50a6cdfffc59a509067978b38**

.

```
├── LICENSE
├── README.md
├── contracts
│   ├── Owned.sol
│   ├── PriceSource.sol
│   ├── QiStablecoin.sol
│   └── Stablecoin.sol
├── hardhat.config.ts
├── package-lock.json
├── package.json
├── scripts
│   ├── call.ts
│   └── deploy-miMATIC.ts
└── tsconfig.json
```

2 directories, 12 files