CERTIK

# Preliminary Comments

# OMG Network

Jun 8th, 2021

# Table of Contents

# Summary

This report has been prepared for OMG Network smart contracts to discovering issues and vulnerabilities in the source code of their Smart Contract and 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 minor 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.

# Overview

## Project Summary

| | |
|---|---|
| Project Name | OMG Network |
| Platform | Ethereum |
| Language | Solidity |
| Codebase | https://github.com/omgnetwork/plasma-contracts/tree/master/plasma_framework/contracts/quasar |
| Commit | 08ae3c077420897523f4cb3ddbd5ac4aa99e8605 e0a304c29cf878f54e2bea98bdd99c4b0df0b685 |

## Audit Summary

| | |
|---|---|
| Delivery Date | Jun 08, 2021 |
| Audit Methodology | Static Analysis, Manual Review |
| Key Components | |

## Vulnerability Summary

| | |
|---|---|
| Total Issues | 7 |
| Critical | 0 |
| Major | 0 |
| Medium | 0 |
| Minor | 3 |
| Informational | 4 |
| Discussion | 0 |

## Audit Scope

| ID | file | SHA256 Checksum |
| --- | --- | --- |
| QTC | contracts/quasar/QToken.sol | 1730dd811b74878590a62a337367f80f99f9d8263ebf2398f03f48cdc28c630e |
| QCK | contracts/quasar/Quasar.sol | bdee2278117a799551872050b24c9b47c775b83d77322cabe95ffebb053ab6cc |
| QPC | contracts/quasar/QuasarPool.sol | f97b66fd6232620cf24ee04fbeef1a9848ad4f7c45a6b2159f5d9748782e65d0 |

# Understandings

## Overview

The Quasar contracts implement QToken, Quasar pool, and Quasar ticket system on Ethereum.

The contract `QToken` implements the token that could exchange with other tokens in the Quasar pool.

The contract `QuasarPool` provides a pool that users could deposit and withdraw registered tokens.

The contract `Quasar`, together with the contract `QuasarPool`, constructs a ticket system that users can claim their on-chain assets by creating tickets, claiming tickets, submitting IFE claims, challenging IFE claims, and processing IFE claim. Meanwhile, the exchange rates between asset tokens and QToken will be increased as users pay fees when they claim tickets or processing IFE claims.

## Dependencies

There are a few depending injection contracts or addresses in the current project:

- `token` registered by calling function `QuasarPool.registerQToken()` for contract `QuasarPool`.
- `plasmaFramework`, `paymentExitGame`, `spendingConditionRegistry` for contract `Quasar`.

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

## Priviledged Functions

The contract `QToken` contains the following privileged functions that are restricted by the `onlyFrom(quasarContract)` modifier:

- `QToken.mint()` is used to mint token for accounts.
- `QToken.burn()` is used to burn tokens from accounts.

The contract `QuasarPool` contains the following privileged function that is restricted by the `onlyQuasarMaintainer()` modifier: *`QuasarPool.registerQToken()` is used to register a new token in the Quasar pool.

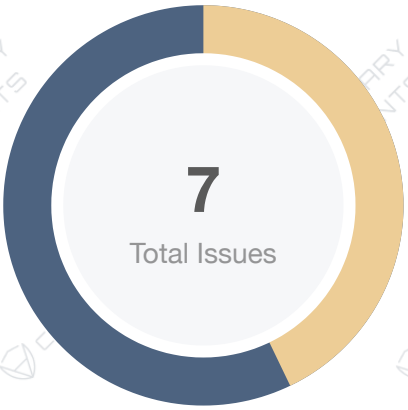
The contract `Quasar` contains the following privileged functions that are restricted by the `onlyQuasarMaintainer()` modifier:

- `Quasar.setSafeBlockMargin()` is used to modify safe block margin for the contract.
- `Quasar.pauseQuasar()` is used to pause the contract.
- `Quasar.resumeQuasar()` is used to resume the contract.

- `Quasar.withdrawUnclaimedBonds()` is used to withdraw unclaimed bonds from the contract.

To improve the trustworthiness of the project, any dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should also consider moving to the execution queue of the `Timelock` contract.

# Findings




| | |
|---|---|
| Critical | 0 (0.00%) |
| Major | 0 (0.00%) |
| Medium | 0 (0.00%) |
| Minor | 3 (42.86%) |
| Informational | 4 (57.14%) |
| Discussion | 0 (0.00%) |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| QCK-01 | Redundant Member `bondValue` in Struct `Ticket` | Gas Optimization, Coding Style | Informational | Resolved |
| QCK-02 | Function Should be Declared External | Gas Optimization | Informational | Resolved |
| **QCK-03** | Centralization Risks | **Centralization / Privilege** | **Minor** | **Resolved** |
| QCK-04 | Logic Related to IFE Claims and Owed Amount | Logical Issue | Minor | Resolved |
| QPC-01 | Function Should be Declared External | Gas Optimization | Informational | Resolved |
| QPC-02 | Lack of Checks for Reentrancy | Logical Issue | Minor | Resolved |
| **QTC-01** | Deployment Risks | **Centralization / Privilege** | **Informational** | **Resolved** |

# QCK-01 | Redundant Member `bondValue` in Struct `Ticket`

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization, Coding Style | Informational | contracts/quasar/Quasar.sol: 57, 174 | Resolved |

## Description

The state `bondValue` of the contract is immutable after contract initialization. According to the code implementation in L174 and L205, the field `bondValue` within any `Ticket` STRUCT instance would be initialized as the same value as the state `bondValue` of this CONTRACT. Afterwards, the state `bondValue` within any struct instance will not be mutated after struct initialization. Therefore, the member `bondValue` in struct `Ticket` is unnecessary. It can be replaced with the state `bondValue` of the contract whenever used.

## Recommendation

It is highly recommended to remove the member `bondValue` from the struct `Ticket` and use state variable `bondValue` of the contract to replace `ticket.bondValue`.

## Alleviation

The OMG Network team heeded our advice and resolved this issue in the commit e0a304c29cf878f54e2bea98bdd99c4b0df0b685.

## QCK-02 | Function Should be Declared External

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | contracts/quasar/Quasar.sol: 119, 128, 143, 150, 157, 173, 222, 247, 279, 320 | Resolved |

### Description

The functions which are never called internally within the contract should have external visibility. For example:

- `Quasar.setSafeBlockMargin()`
- `Quasar.flushExpiredTicket()`
- `Quasar.pauseQuasar()`
- `Quasar.resumeQuasar()`
- `Quasar.withdrawUnclaimedBonds()`
- `Quasar.obtainTicket()`
- `Quasar.claim()`
- `Quasar.ifeClaim()`
- `Quasar.challengeIfeClaim()`
- `Quasar.processIfeClaim()`

### Recommendation

It is highly recommended to change the visibility of the aforementioned functions from `public` to `external` for gas optimization.

### Alleviation

The `Quasar` contract's bytecode size is very close to the EIP-170 limit. Using an `external` function with `calldata` parameters increases the bytecode size. The OMG Network team changed `public` to `external` where it is possible in the commit e0a304c29cf878f54e2bea98bdd99c4b0df0b685.

## QCK-03 | Centralization Risks

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Minor | contracts/quasar/Quasar.sol: 119, 143, 150, 157 | Resolved |

### Description

The role `quasarMaintainer` has authority to:

- modify safe block margin by calling `Quasar.setSafeBlockMargin()`;
- pause the contract by calling `Quasar.pauseQuasar()`;
- resume the contract by calling `Quasar.resumeQuasar()`;
- withdraw unclaimed bonds by calling `Quasar.withdrawUnclaimedBonds()`.

### Recommendation

We advise the client to handle the `quasarMaintainer` account carefully to avoid any potential hack. We also advise the client to consider the following solutions:

1. Apply an associated `Timelock` contract to implement above functions, with reasonable latency for community awareness on privileged operations;
2. Apply Multisig with community-voted 3rd-party independent co-signers;
3. Apply DAO or Governance module to increase transparency and community involvement.

### Alleviation

The OMG Network team implemented the library `TimelockedValue` to update the safe block margin with latency in the commit `e0a304c29cf878f54e2bea98bdd99c4b0df0b685`.

[**OMG Network Team**]: While we agree that the quasarMaintainer account should be carefully managed, the effects of it being compromised are minimal and most of the maintainer methods would not affect the users.

- `pauseQuasar()` only prevents new withdrawals from being started. Existing withdrawals have had their funds reserved and can continue as normal without fear of losing funds.
- `withdrawUnclaimedBonds()` can only withdraw funds that are destined for the `quasarMaintainer` anyway and so should not be considered a Centralization risk as no user funds are in danger
- `setSafeBlockNumber()` does protect the liquidity pool in the event that the plasma chain goes byzantine and the Plasma operator continues publishing blocks. We have added a timelock to this

method to warn liquidity providers when `safeBlockNum` is changed and allow them time to withdraw their funds if they don't agree with it.

# QCK-04 | Logic Related to IFE Claims and Owed Amount

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | contracts/quasar/Quasar.sol: 1 | Resolved |

## Description

According to the code implementation, if a bad IFE claim is not challenged within the eight-day limitation, it would finally get processed. In this case, the attacker could withdraw the tokens that do not belong to him from the contract. This might lead to the contract not having enough balance to pay other users' claims later.

We noticed that in the contract `QuasarPool`, users are allowed to send a certain amount of tokens (amount not exceeding `tokenData[token].owedAmount`), to the contract account by calling the function `QuasarPool.repayOwedToken()`. We hope to confirm with the team about the using scenarios of the function: if the function `QuasarPool.repayOwedToken()` and the variable `tokenData[token].owedAmount` are designed to handle the situation when a bad IFE claim is processed.

## Alleviation

[**OMG Network Team**]: IFE claims are intended as a way of making sure that the user does not lose funds in the event that the user initiated a withdrawal and sent funds to the `quasarOwner`, but the Plasma operator does not include the transaction in a block. This mirrors the Plasma MoreVP protocol. One of the security assumptions of Plasma is that users are able to monitor invalid transactions or IFEs and challenge them. This holds true for Quasar users as well - they can either check for invalid IFEs once every 8 days, or trust someone else to do that for them.

However, in the unlikely event that an invalid IFE does get processed, then yes, the `quasarOwner` can make up the funds by calling `repayOwedToken()`. Note that this means that users must trust the `quasarOwner` to do the right thing at cost to themself. The assumption is that users would prefer to monitor and challenge invalid IFEs instead.

# QPC-01 | Function Should be Declared External

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Gas Optimization | Informational | contracts/quasar/QuasarPool.sol: 40, 50, 79, 110, 120 | Resolved |

## Description

The functions which are never called internally within the contract should have external visibility. For example:

- `QuasarPool.addEthCapacity()`
- `QuasarPool.addTokenCapacity()`
- `QuasarPool.withdrawFunds()`
- `QuasarPool.registerQToken()`
- `QuasarPool.repayOwedToken()`

## Recommendation

It is highly recommended to change the visibility of the aforementioned functions from `public` to `external` for gas optimization.

## Alleviation

The OMG Network team heeded our advice and resolved this issue in the commit e0a304c29cf878f54e2bea98bdd99c4b0df0b685.

# QPC-02 | Lack of Checks for Reentrancy

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Logical Issue | Minor | contracts/quasar/QuasarPool.sol: 50, 79, 120 | Resolved |

## Description

Functions that contain state updates or event emits after external calls are vulnerable to potential reentrancy attacks. For example,

- `QuasarPool.addTokenCapacity()`
- `QuasarPool.withdrawFunds()`
- `QuasarPool.repayOwedToken()`

## Recommendation

It is highly recommended to apply OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent any potential reentrancy attack.

## Alleviation

The OMG Network team heeded our advice and resolved this issue in the commit e0a304c29cf878f54e2bea98bdd99c4b0df0b685.

# QTC-01 | Deployment Risks

| Category | Severity | Location | Status |
| --- | --- | --- | --- |
| Centralization / Privilege | Informational | contracts/quasar/QToken.sol: 22~23, 31 | Resolved |

## Description

According to the contract implementation, the owner account `quasarContract` is capable to mint an unlimited amount of tokens by calling the function `mint`. On the other hand, `quasarContract` is capable to burn all the amount of tokens of an account without any restriction. The concern is if `quasarContract` is not set up properly, or it accidentally calls the aforementioned functions, it might cause some unexpected loss, thus introducing centralization risks.

## Recommendation

We advise the team to review the flow and confirm if it is an intended design. If the owner `quasarContract` is designed to be the contract `QuasarPool`, please ensure `quasarContract` is set up properly, and `QToken` is always bundled with the contract `QuasarPool` to work together, since the contract `QToken` is vulnerable alone.

## Alleviation

[**OMG Network Team**]: That's correct, the owner of the `QToken` contract (stored as `quasarContract`) should be set as the address of the `QuasarPool` contract. If this has been initialized incorrectly, then the `QuasarPool` for that ERC20 token won't work and it should not be used.

# Appendix

## Finding Categories

### Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

### 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.

### 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.