PUBLIC

Code Assessment

of the XChain DSR Oracle Smart Contracts

April 08, 2024

Produced for

MAKER

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help MakerDAO with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of XChain DSR Oracle according to Scope to support you in forming an opinion on their security risks.

MakerDAO implements cross-chain oracles for the DAI Savings Rate where update messages are sent to L2s from Ethereum Mainnet.

The most critical subjects covered in our audit are functional correctness, access control and message passing.

The general subjects covered are code complexity and specification.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the XChain DSR Oracle repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	02 April 2024	01481b10aabb6b8a2c6afacec3cad19a90ecd7b1	Initial Version
2	05 April 2024	ccd09c8cf122200dc66affee7c07f8843cb4c684	After Intermediate Report
3	06 April 2024	463e012c51d50cd24d96fc5738e26f7ca624c94c	Added Base Support

For the solidity smart contracts, the compiler version `0.8.20` was chosen.

The following files were in scope:

```
src/DSRAuthOracle.sol
src/receivers/DSROracleReceiverOptimism.sol
src/receivers/DSROracleReceiverGnosis.sol
src/adapters/DSRBalancerRateProviderAdapter.sol
src/forwarders/DSROracleForwarderGnosis.sol
src/forwarders/DSROracleForwarderBase.sol
src/forwarders/DSROracleForwarderOptimism.sol
src/forwarders/DSROracleForwarder.sol
src/DSRMainnetOracle.sol
src/DSROracleBase.sol
```

2.1.1 Excluded from scope

All other files including tests and helpers. The correctness of the external systems (i.E. bridges) is out of scope. Of the xchain-helpers, only the functions used have been reviewed.

2.2 System Overview

MakerDAO offers a framework to report values related to the DAI Savings Rate (DSR) from Ethereum Mainnet to various chains. The set of contracts consists of forwarders on L1 (Ethereum Mainnet) and receivers on L2. Currently, Gnosis, Optimism and Base are supported L2s. Oracle contracts are provided for users to access these cached values.

2.2.1 Oracles

DSROracleMainnetOracle

This oracle is deployed on mainnet and pulls its data directly from the Pot, the DAI Savings Rate contract. The values cached locally can be updated permissionlessly using the function `refresh()`.

DSRAuthOracle

Authenticated oracle which receives updates from Mainnet through the bridge. Exposes `setPotData()` for the cross-chain message receiver to update the data. The oracle only accepts increasing `rho` values that are in the past, DSR rates greater than one and less than the upper bound `maxDSR` which is defined by governance with `setMaxDSR()`, and increasing `chi` values that are upper bounded by a maximum chi that could have occurred assuming that the DSR never exceeded its maximum.

All Oracle contracts, through inheritance of DSROracleBase, expose the following public functionality to access the data:

- `getPotData()`: Returns the struct `PotData` which consists of: `uint96 dsr`, the DAI Savings Rate in per second value (in the unit of `ray`), `uint120 chi`, the last computed conversion rate (to DAI) in `ray` and `rho` the timestamp of the last update (computation of `chi`) in seconds.
- `getDSR()`: returns the current DSR value stored.
- `getChi()`: returns the current Chi value stored.
- `getRho()`: returns the current Rho value stored.
- `getAPR()`: calculates and returns the Annual Percentage Rate calculated using the stored DSR.
- `getConversionRate()`: returns the conversion rate at the current timestamp.
- `getConversionRate(uint256 timestamp)`: returns the conversion rate at the given timestamp (now or in the future).
- `getConversionRateBinomialApprox()`: returns the binomial approximated conversion rate at the current timestamp.
- `getConversionRateBinomialApprox(uint256 timestamp)`: returns the binomial approximated conversion rate that the given timestamp (now or in the future)
- `getConversionRateLinearApprox()`: returns the linear approximated conversion rate at the current timestamp.
- `getConversionRateLinearApprox(uint256 timestamp)`: returns the linear approximated conversion rate at the given timestamp (now or in the future).

All values returned are based on the cached values which might be outdated. Updates must be triggered when the DSR changes, in between updates of the DSR the current `chi` can be calculated accurately based on the cached data. Calculating values for timestamps in the future may be inaccurate if the DSR changes.

2.2.2 Bridging framework

Canonical bridging is used to pass data. Forwarder and Receiver contracts are used to implement the chain-specific messaging process.

1. Forwarders on Mainnet

Forwards expose `refresh()` allowing any caller to push current data to the respective chain.

DSROracleForwarderGnosis uses `XchainForwarders.sendMessageGnosis()`, DSROracleForwarderOptimism uses `XChainForwarders.sendMessageOptimismMainnet()`, and DSROracleForwarderBase uses `XChainForwarders.sendMessageBase()` for message passing.

All three inherit from the base contract DSROracleForwarder which handles message packing and stores the last seen Pot data. It implements functions `getLastSeenPotData()`, `getLastSeenDSR()`, `getLastSeenChi()` and `getLastSeenRho()` to query the respective data.

2. Xchain-helpers

This library exposes helpers for cross-chain messaging. The following functions are used by the XChain DSR Oracle:

- `sendMessageGnosis()`: Sends the message using the canonical Gnosis Messenger (invokes `l1CrossDomain.requireToPassMessage(target, message, gasLimit)`).
- `sendMessageOptimismMainnet()`: Sends the message using the canonical Optimism Messenger (invokes `l1CrossDomain.sendMessage(target, message, gasLimit)`).
- `sendMessageBase()`: Sends the message using the canonical Messenger (invokes `l1CrossDomain.sendMessage(target, message, gasLimit)`).

The messenger contract addresses are hardcoded in the XChainForwarders.

3. Receivers on L2

Decodes the received messages and forwards the data to the DSRAuthOracle.

DSROracleReceiverGnosis and DSROracleReceiverOptimism (used for Optimism and Base) implement `setPotData()` which is callable only by the respective canonical messenger.

2.2.3 Adapter

Furthermore, DSRBalancerRateProviderAdapter is provided which exposes `getRate()`, a function returning the value of sDAI in terms of DAI.

2.3 Trust Model & Roles

- Pot: DAI Savings Rate contract. Fully trusted, source of Data.
- Bridges: Fully trusted. Transmit the message, they must not alter any data nor censor any message. However, note that there are some limitations put in place to reduce the trust in bridges.
- Governance: Fully trusted. The governance can replace the callers of `setPotData()` which could lead to oracle manipulations.

Receiver contracts (DSROracleReceiverGnosis/DSROracleReceiverOptimism) update DSRAuthOracle based on received messages, which must originate from the designated contract on the L1 chain. At the deployment of these receiver contracts, the messaging contract and L1 origin parameters are initialized, and the correct configuration is essential.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6.1 Inconsistencies

Informational Version 1 Code Corrected

CS-XDSR-002

The codebase is inconsistent in style across files and functions. Below is an incomplete list:

1. The DSRAuthOracle explicitly uses `1e27`. However, in other places, the ray constant RAY is used.
2. The `getConversionRate` function does not perform the time-delta computation `timestamp - rho` in an `unchecked` block while `getConversionRateLinearApprox()` and `getConversionRateBinomialApprox()` do.

Code corrected:

The code has been changed accordingly.

6.2 Redundant Getters

Informational Version 1 Code Corrected

CS-XDSR-001

`_lastSeenPotData` is a `public` variable and thus has an autogenerated public getter. Nevertheless, there is a second getter `getLastSeenPotData()`. Ultimately, both getters return the same since no complex data types are used in `PotData`.

Code corrected:

`_lastSeenPotData` is now `private`.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Delayed Oracle Consequences

Note Version 1

Users of the crosschain DSR oracle should be aware that the oracles could hold outdated values. They should ensure that the oracle is kept alive when changes in the DSR occur. Otherwise, the impact of the scenarios below could be more impactful to their application than expected.

Namely, the following scenarios could occur:

1. The DSR decreased on Mainnet. However, an L2 oracle has not been updated. Consequently, the L2 oracle will overvalue `chi` and thus will overvalue sDAI.
2. The DSR increased on Mainnet. However, an L2 oracle has not been updated. Consequently, the L2 oracle will undervalue `chi` and thus will undervalue sDAI.

7.2 Oracle Functions May Behave Differently

Note Version 1

Users and integrators of the DSR oracles should be aware that DSR functions `getConversionRate()`, `getConversionRateLinearApprox()` and `getConversionRateBinomialApprox()` do not differ only in accuracy but could also differ in terms of reverting behaviour of overflows. For example, there is a set of numbers for which the third function would revert values whereas the second would not.