PUBLIC

Code Assessment

of the D3M

Smart Contracts

November 17, 2021

Produced for

MAKER

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Maker team,

First and foremost we would like to thank MakerDAO for giving us the opportunity to assess the current state of their D3M system. This document outlines the findings, limitations, and methodology of our assessment.

The code reviewed was of a high standard. However, limited documentation for the smart contracts reviewed was provided for the audit. Given the potential implications of this module on the global settlement during the shutdown, proper documentation should be available to all users of Maker.

No significant issue has been uncovered during the audit. After the intermediate report which contained low severity findings all issues have been either corrected or acknowledged. For a complete list of issues please refer to the Findings section.

The communication with your team during the audit was very good and helped to resolve arising questions quickly.

We hope that this assessment provides more insight into the current implementation and provides valuable findings. We are happy to receive questions and feedback to improve our service and are highly committed to further support your project.

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	2
• Code Corrected	1
• Acknowledged	1

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 D3M 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	18 October 2021	cfbaa8cf69d01873f391d48cc611f563303ff179	Initial Version
2	17 November 2021	6ce0efcee4c51828be9c955a67f258bcc27f5a1d	After Intermediate Report

For the solidity smart contracts, the compiler version `0.6.12` was chosen. This version, although being deprecated, has been chosen explicitly to be consistent through all newly developed maker modules.

The files in scope for this review were: `DirectDepositMom.sol` and `DssDirectDepositAaveDai.sol`.

2.1.1 Excluded from scope

All files not listed above.

2.2 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

The Direct Deposit Module (D3M) enables the interaction of the Maker ecosystem with third-party lending pools. `DssDirectDepositAaveDai` is a smart contract of this module that enables the transaction of DAI tokens from Maker to the respective lending pool of Aave. The goal of this smart contract is to ensure that the maximum variable interest rate for borrowing stays below a targeted interest rate decided by the Maker governance. In Aave, the variable interest rate of a pool depends on the utilization of that pool, which is the ratio of the total debt taken over the total liquidity put in the pool. Therefore, the higher the utilization of a pool, the higher becomes the variable interest rate. This strategy motivates liquidity providers to deposit capital in the pool when utilization is high.

The goal of DssDirectDepositAaveDai is to limit the maximum variable interest rate for the DAI pool in Aave by depositing or withdrawing DAI from the pool as needed. To achieve this functionality, the DssDirectDepositAaveDai needs:

1. to be an authorized ward in the `Vat`, and
2. operate on a special `ilk`.

The essential feature of this `ilk` is that it allows the DssDirectDepositAaveDai to generate DAI tokens on the fly without requiring a traditional collateral in another token. The `ilk` should have the `rate` set to 1, and the `spot` price fixed to 1. Note that, the generated DAI over this `ilk` can only be transferred to the

DAI pool in Aave, hence the `ink` that the contract has in `Vat` is backed with the `aDAI` (interest-bearing token in Aave pegged to the value of DAI at 1:1 ratio) the contract holds. This way, the `aDAI` amount owned by DssDirectDepositAaveDai in Aave serves as `ink` in this special `ilk` for the generated DAI. It is important to note that the Aave pool is fully trusted to behave correctly.

Whenever the variable interest rate of the pool is below the targeted threshold, DssDirectDepositAaveDai withdraws (if possible) liquidity from the pool and pays back the DAI debt in `Vat` and destroys the respective gem amount. Finally, all interests earned in the Aave pool by the contract are transferred to the `Vow` contract.

DssDirectDepositAaveDai provides the following functionalities:

- `calculateTargetSupply`: can be called by anyone and computes the total liquidity required given the total debt taken from a pool to reach a targeted interest rate.
- `exec`: can be called by anyone and computes on-chain if the contract should deposit of withdraw DAI from the Aave pool. Depending on the current state of the pool, the function calls `_wind()` (to deposit DAI) or `_unwind()` (to withdraw DAI).
- `reap`: can be called by anyone and collects the interests earned in the Aave pool. The interests are always transferred to the `Vow` contract.
- `collect`: can be called by anyone and calls the `rewardsClaimer` with the predefined `king` address.
- `exit`: accessible only during general shutdown of `Vat`, allows users to exit their DAI.
- `cage`: accessible only by a `ward`, or by anyone if the address of the interest strategy contract in Aave changes. This function performs the shutdown of the module and sets the `tic`.
- `cull`: accessible only in case of module shutdown while the `Vat` is live, calls `Vat.grab()` for the existing `ink` and `art` and the `Vow` address as `w`.
- `uncull`: accessible only if the D3M module is shutdown, then `cull()` is called, and afterwards the `Vat` is shutdown. In this unlikely scenario, `uncull()` reverts the `cull()` updates.
- `quit`: can be called only by authorized `wards` while the `Vat` is live, and transfers all `aDAI` tokens owned by DssDirectDepositAaveDai to another address.
- `file`: can be called only by authorized `wards` and allows to set the values of `bar` (targeted interest rate), `tau`, and `king`.
- The standard authorization functions `rely()` and `deny()`.

DirectDepositMom provides the following functionalities:

- `disable`: can be called only by authorized `wards` to set the `bar` to `0` for the DssDirectDepositAaveDai.
- `setOwner` and `setAuthority`: can be called only by the `owner`.

2.2.1 Trust Model & Roles

Wards: Fully trusted to behave honestly and correctly at all times. They can set the parameters `bar`, `tau`, `king`, can shut down the module by calling `cage()`, or trigger the emergency quit via `quit()`.

External users: Untrusted. Can call the functions of DssDirectDepositAaveDai with arbitrary parameters.

Aave pool is trusted to behave correctly and to preserve its functionality and integrity when the contract is upgraded.

We assume the DssDirectDepositAaveDai operates on a special `ilk`, which should be initialized with the correct parameters to ensure that the `rate` of the `ilk` remains constant at `1`, and the price oracle returns a spot price of `1` always. Moreover, no adapter (join) should allow users to interact with this `ilk`. Finally, the DssDirectDepositAaveDai is limited by the debt ceiling of the `ilk` on the amount of DAI it can deposit in the Aave pool, therefore if this limit is reached, the contract cannot ensure that the target interest rate is maintained.

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 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 associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. 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 the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than 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 severities. 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:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

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	1

- Redundant External Calls Acknowledged

5.1 Redundant External Calls

Design Low Version 1 Acknowledged

The function `calculateTargetSupply()` makes redundant calls to the interest strategy contract to get the `variableRateSlope1` value. It would be more efficient if the return value of the first call is stored in memory, hence avoiding the second call.

Similarly, the function `exec()` makes two external calls to get the total debt (from `stableDebt` and `variableDebt`), and then calls the `calculateTargetSupply()` which makes again the same external function calls.

Acknowledged:

Maker acknowledges:

```
This function is only called by a keeper bot, and likely infrequently enough
to not overly worry about gas savings. Will re-assess if exec() starts getting called a lot.
```

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	1

- ink Used as daiDebt Code Corrected

6.1 ink Used as daiDebt

Correctness Low Version 1 Code Corrected

Function _unwind() contains following comment:

```
// IMPORTANT: this function assumes Vat rate of this ilk will always be == 1 * RAY (no fees).
```

This means the debt in art equals the amount in DAI.

The value for daiDebt is queried as follows:

```
if (mode == Mode.NORMAL) {
    // Normal mode or module just caged (no culled)
    // debt is obtained from CDP ink (or art which is the same)
    (daiDebt,) = vat.urns(ilk, address(this));
```

Note that the first return value is the ink while the second one is the art. Hence the ink and not the art is used as value for the debt in DAI. As documented, this requires that the rate for the collateral is 1. However, additionally it is required that the spot price of the collateral must always be 1. This is not fully documented.

In Function reap the daiDebt is the art:

```
(, uint256 daiDebt) = vat.urns(ilk, address(this));
```

Code corrected:

The value for daiDebt in function _unwind is now based on the art of the CDP.

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 Possible Overflow for `2 ** 255`

Note **Version 1**

The functions `_unwind()`, `exit()`, `cull()` and `quit()` check if a conversion of a `uint256 value` to a negative `int256` overflows: `require(value <= 2 ** 255)`. Theoretically, if `value == 2 ** 255` the overflow will happen twice, but the result matches the expected value in such cases.

7.2 Shutdown Vat and `exit()`

Note **Version 1**

`exit()` does not explicitly check whether the Vat is not `live`. Technically this seems to be unnecessary as during normal operation only this contract can have a non-zero entry for this special `gem`. However given that it's so unlikely that the `exit` function is used and the dire consequences if for any other reason an account has a non-zero balance of this `gem`, it is worth considering adding the explicit check as a precautionary measure.

7.3 Shutdown of Vat

Note **Version 1**

In case of a shutdown of the Vat, the ideal scenario for this module is that all its debt can be unwound in the first phase of the shutdown.

Should this not be possible, e.g. because there was no sufficient liquidity in the Aave pool or no one called `exec` before `MCD_END.debt()` was set, the settlement of the remaining DAI holders gets complicated:

In this phase of the settlement, users receive a share of all `ilks`. These shares are jointly worth one DAI for each DAI redeemed by the user. These shares will include the collateral share of this special ilk. For this ilk, special handling is necessary. However, after some steps, users have a `Vat.gem` balance which they can exit.

For this special `ilk` no join adapter exists and users will have to use the `exit()` function of this contract and receive aDAI.

Two scenarios can happen:

- This aDAI is worthless as there is no liquidity in the aDAI pool and users are unable to redeem their aDAI for DAI. This could happen because free DAI have been submitted in the shutdown process. During global settlement, however, the price of one aDAI was taken as 1 DAI. The overall consequence is that users receive less than one dollar worth of collaterals for 1 DAI "cashed out".
- There is remaining liquidity in the Aave pool and users can redeem their aDAI for DAI. Having received this new DAI, the user can now redeem this DAI, again receiving a share of all collaterals including aDAI again. Note that, as many ilks exist, in practice this will be a very small percentage overall. Hence users may do following workaround:

1. Exit their share of the gem for this special ilk and receive aDAI first.
2. Convert this aDAI to DAI by withdrawing from the pool.
3. Redeeming this DAI. This increases the gem balance of the users for all ilks.
4. Only now exit the other gems. The aDAI received at this step may be forfeited as their value like is negligible and the user was able to redeem his DAI for collateral being worth close to 1 dollar.