PUBLIC

Code Assessment

of the RWA Toolkit Smart Contracts

July 06, 2023

Produced for

by

Contents

1 Executive Summary 3
2 Assessment Overview 5
3 Limitations and use of report 7
4 Terminology 8
5 Findings 9
6 Resolved Findings 10

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 RwaMultiSwapOutputConduit contract according to Scope to support you in forming an opinion on their security risks.

RwaMultiSwapOutputConduit allows priviledged users to convert DAI held by the smart contract into other stablecoins and transfer them to off-chain funds, using one of the Peg Stability Modules (PSM). Configurations need resetting after each use for security.

The most critical subjects covered in our audit are access control, functional correctness and the intergrations into the existing DSS system. After the intermediate report all uncovered issues have been resolved.

In summary, we find that the codebase provides a good 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	1
• Code Corrected	1
Low-Severity Findings	1
• Code Corrected	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 file inside the RWA Toolkit repository based on the documentation files.

1. src/conduits/RwaMultiSwapOutputConduit.sol

This review focused on this file only, all other smart contract of the repository have not been reviewed.

The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	3 July 2023	1e84bfce78573ec4e3772100b4b0c774d2c65454	Initial Version
2	5 July 2023	3ba40e14c660caac8f2dd966d29b4e1457a46ebc	After Intermediate Report

For the solidity smart contracts, the compiler version `0.6.12` was chosen.

2.1.1 Excluded from scope

Any other file not explicitly mentioned in the scope section. In particular tests, scripts, external dependencies, and configuration files are not part of the audit scope.

2.2 System Overview

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

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Contract *RwaMultiSwapOutputConduit* implements an output conduit for the real world assets (RWA) functionality alllowing to deposit funds to addresses belonging to trusted counterparties operating off-chain fund allocation. DAI is deposited in the contract, is then converted to an output stablecoin through one of MakerDAO Peg Stability Modules (PSM), and is then transferred to the recieving address. A set of user roles and whitelists govern who can set the recieving address and PSM, configure the whitelists, assign user roles, and operate the main functionality.

2.2.1 Main functionality: Push

A user with the `may` role enabled, or any user when address `0` has the `may` role, can operate the main functionality of the contract through the `push` method. The method, overloaded as `push()` and `push(uint256 wad)`, swaps an amount of DAI for the target stablecoin through the configured PSM, and transfers it to the configured recipient. The version of `push()` without arguments swaps the whole DAI contract balance, while the version with the `wad` argument allows the caller to specify the amount to swap.

The contract holds balance in DAI, and the PSM allows swapping it to another stablecoin through its `buyGem(address usr, uint256 gemAmt)` method, by specifying the output amount `gemAmt`. The PSM Stablecoin and DAI are assumed to have the same value, but fees can apply on swaps. When DAI is swapped for stablecoin, the PSM `tout` (toll out) fee applies. Because of the fee, the input amount might not be the same as the output amount. The `push` method computes the ouptut amount for a given `wad` of input DAI through the `expectedGemAmt(uint256 wad)` public method. Since in the PSM the input amount for a given output amount is `input = (1 + fee) * output`, the `expectedGemAmt` method computes the required output that needs to be specified from a given input as the inverse formula `output = input/(1 + fee)`.

`buyGem` is then called with the calculated output amount, and with the content of storage variable `to` as the recipient of the stablecoin.

Finally the `to` variable, containing the configured recipient, and the `psm` variable, with the address of the PSM in use, are cleared, so that reconfiguration of the contract is required before reuse of the `push` method.

A user with the `may` role can also chose to call the `quit(wad)` function, transferring `wad` DAI (or the whole balance if `wad` is left unspecified) to address in the `quitTo` storage variable.

2.2.2 Roles, Whitelists and Trust Model

The contract enumerates several roles, each with specific capabilities.

1. `ward`**, granted with** `rely()` **and revoked with** `deny`**,**

   is the admin of the contract. The deployer is first assigned the `ward` role. It can access methods guarded by the `auth` modifier. Therefore, it grants and revokes all the roles, it can set the `quitTo` address through the `file` method, and it can `yank()`, that is transfer away, any balance of any token present in the contract. It can add and remove potential recipients to the `bud` recipient whitelist, respecitvely with methods `kiss` and `diss`, and add or remove PSMs to the `pal` whitelist, with methods `clap` and `slap`. This role is fully trusted to act correctly and honestly at all times.

2. `can`**, granted with** `hope()`**, revoked with** `nope()`**,**

   allows the user who is granted this role to `pick()` and `hook()`. `pick()` allows the `can` user to pick a recipient, to be set as the `to` address in storage, who will receive the output stablecoin on the next call of `push()`. `hook()` lets the `can` user to select a PSM that is set as the PSM in use by the contract. The `to` address chosen with `pick()` must be in the `bud` whitelist, and the psm chosen with `hook()` mus be in the `pal` whitelist.

3. `may`**, granted with** `mate` **and revoked with** `hate`**,**

   allows access to functions guarded by the `onlyMate` modifier. These are the `push()` and `push(wad)` functions, and the `quit()` and `quit(wad)` functions. If address `0` is granted the `may` role, `onlyMate` functions are unpermissioned.

4. **The** `bud` **whitelist**

   includes addresses that can be set as the swap recipient with `pick()` by `can` users, setting the `to` storage variable. The `ward` includes addresses to the whitelist with `kiss()`, or it can remove them from the whitelist with `diss()`. If they are the current recipient when *dissed*, the recipient is reset to the zero address.

5. **The** `pal` **whitelist**

   comprehends the addresses that can be selected as valid PSMs by the `can` users with `hook()`. A `ward` can add addresses to the list with `clap()` and remove them with `slap()`. If the current PSM is *slapped*, the `psm` storage variable is reset to the zero address. When a PSM is added to the whitelist with `clap()`, it is given infinite approval for DAI from the RwaMultiSwapOutputConduit contract. When a PSM is removed from the whitelist with `slap()`, the approval is set to zero.

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:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies

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	1

- Unpermissioned Access to onlyMate Methods Code Corrected

Low-Severity Findings	1

- Use of Unsafe Math Subject to Overflows Code Corrected

6.1 Unpermissioned Access to onlyMate Methods

Security Medium Version 1 Code Corrected

CS-MRT-001

If the `may` role is assigned to address zero, methods guarded by the `onlyMate` modifier become unpermissioned. The purpose of this is unclear, as conflicting functionality can be accessed through `onlyMate` methods. A user could call `push()` to transfer the whole balance to the configured recipient, or `push(1)` to transfer a very low amount to the recipient and disable further `push()` access, since the `to` and `psm` variables are reset to zero after `push()` is called. Likewise, a user could call `quit()`, which transfers the DAI balance to the configured `quitTo` address.

Since mutually exclusive functionality is accessible through the `onlyMate` modifier, leaving it unpermissioned opens the door to race conditions and unpredictable behavior. Only trusted parties should be granted access to `onlyMate` guarded methods.

Similarly, but to a lesser extent, `pick()` and `hook()` are unpermissioned when address zero is granted the `can` role.

Code corrected:

MakerDAO realized output conduits should never be permissionless. The abilitiy to make the `may` and `can` roles unpermissioned has been removed.

6.2 Use of Unsafe Math Subject to Overflows

Design Low Version 1 Code Corrected

CS-MRT-002

Since the contract uses version `0.6.12` of solidity, unchecked arithmetics are used by default. Methods `expectedGemAmt()` and `requiredDaiWad()` are subject to possible artihmetic overflows. if their `wad` or `amt` parameters are set large enough.

Since no accounting state is held by the contract, but operations are performed on the current `DAI` balance, the overflows cannot be exploited, even by a malicious `may` user. However, external contracts

and off-chain users relying on the correctness of `expectedGemAmt()` and `requiredDaiWad()` might be negatively affected.

Code corrected:

SafeMath like methods were introduced to ensure the calculations in `expectedGemAmt()` and `requiredDaiWad()` cannot overflow.