

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
7	Informational	12

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 ArrangerConduit according to [Scope](#) to support you in forming an opinion on their security risks.

MakerDAO implements a contract that is used to give access to funds of Maker SubDAOs to external actors for the purpose of investment into real-world assets.

The most critical subjects covered in our audit are functional correctness and access control. Security regarding all the aforementioned subjects is high.

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.

-Severity Findings	0
-Severity Findings	0
-Severity Findings	0
-Severity Findings	2
•	1
•	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 given repositories based on the documentation files.

Specifically, the following contracts were audited:

In the `dss-conduits` repository:

- `ArrangerConduit.sol`

In the `upgradeable-proxy` repository:

- `UpgradeableProxy.sol`
- `UpgradeableProxied.sol`

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

dss-conduits

V	Date	Commit Hash	Note
1	27 July 2023	8f60eed064ed05706585f48e21da8e691978d341	Initial Version
2	02 September 2023	d9093aea822ce5d9fea3083b580dfa5bff9a9cd5	Second Version
3	17 September 2023	71cc2f0a8bb5f11a478f8741fc3c593893409e18	Third Version
4	09 October 2023	b8b3dac34d307ce1e9f976b4c17457fefa6e2ed4	Fourth Version

upgradeable-proxy

V	Date	Commit Hash	Note
1	27 July 2023	72286d211cb0f7e2ee79bfd216bcbced87cc3c7a	Initial Version
2	27 October 2023	885de5b47aee6a2b492099d163b3e394eb53f852	Second Version

For the solidity smart contracts, the compiler version `0.8.16` 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 () 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.

MakerDAO offers a conduit contract that, in conjunction with the `dss-allocator` system, is used to distribute funds of different SubDAOs in the Maker ecosystem to so-called arrangers which are third-party fund managers that can use these funds to invest into real-world assets like US Bonds.

Funds are located in multiple `AllocatorBuffer` contracts, each belonging to a SubDAO. `NewStable` can be allocated to these buffers from a corresponding `AllocatorVault` but the buffers can also contain other ERC-20 tokens belonging to the respective SubDAO. Approval for each asset can be set from the buffer to the conduit contract. After that, funds can be deposited by the operators of the conduit contract from the buffer by using the function `deposit()`.

The conduit contract contains exactly one arranger address that is allowed to handle the deposited funds of all SubDAOs. This arranger can call `drawFunds()` to transfer the assets of the contract to certain addresses called "brokers". Each broker has to be whitelisted. This can be, for example, a Coinbase account that is used to convert the funds to Fiat currency which can then be used to buy real-world assets.

Operators can request to reclaim assets by calling the function `requestFunds()` with a given amount for a given asset and a given SubDAO. The arranger then returns the funds by sending them to the contract and calling `returnFunds()`. It should be noted that there are no constraints on the amount that is returned in the end. Whether more or less than originally requested is returned, the fund request is always marked as completed. Pending requests can furthermore be canceled by the operators with a call to `cancelFundRequest()`.

Returned funds can finally be withdrawn back to the buffer with the function `withdraw()`. All other previously deposited funds can not be withdrawn until the arranger actively returns them to a given return request.

2.2.1 UpgradeableProxy

The `ArrangerConduit` is an implementation contract for `UpgradeableProxy` which is a simple proxy contract that delegates all function calls to its implementation. It contains the logic for `wards` which are used as access control for certain functions. Since the implementation address and the `wards` are stored in the first two storage slots of the proxy, implementing contracts have to extend `UpgradeableProxied` contract which reserves these slots so that delegated calls to such an implementation do not access the wrong slots.

2.2.2 Roles & Trust Model

Operators are only able to work on an `ilk` (i.e., SubDAO identifier) they are assigned to. The access control is fine-granular and gives each operator access on a function-level only by using the `AllocatorRoles` contract of the allocator system which is part of Maker Core.

The contract also contains `wards` that are allowed to change the arranger, as well as the contract addresses of the allocator system contracts, and are able to whitelist broker addresses. These addresses are assumed to be under the control of an `AllocatorDAO`.

The arranger is a trusted party that has full access (depending on the respective broker addresses) of the deposited funds.

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			
Medium			
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:

- : Architectural shortcomings and design inefficiencies

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

-Severity Findings	0
-Severity Findings	0
-Severity Findings	0
-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.

-Severity Findings	0
-Severity Findings	0
-Severity Findings	0
-Severity Findings	2

- [ERC-20 Missing Return Value](#)
- [Floating Pragma](#)

6.1 ERC-20 Missing Return Value

CS-MDAC-001

`ArrangerConduit` handles ERC-20 transfers in the following way:

```
require(
    ERC20Like(asset).transfer(destination, amount),
    "ArrangerConduit/transfer-failed"
);
```

This assumes that all ERC-20 contracts that can be called return a boolean value in their `transfer()` and `transferFrom()` functions. This is however not the case. Popular tokens like `USDT` are not returning any value in the mentioned functions. If it were to happen that the arranger sends such tokens to the contract, the tokens would be locked and require an update of the contract.

Specification changed:

Transfers are now performed without checking the return values of ERC20 tokens at all. MakerDAO assures that only tokens that revert on failure are used as assets in the contract.

6.2 Floating Pragma

CS-MDAC-002

The `ArrangerConduit` contract is not set to a fixed solidity version - neither in the contract nor in the Foundry configuration. This can lead to unintended side-effects when the contract is compiled with different compiler versions.

Code corrected:



The compiler version 0.8.16 has been added to the Foundry configuration.



7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Incorrect `maxDeposit()` Return Value

CS-MDAC-003

`ArrangerConduit.maxDeposit()` always returns `type(uint256).max`. This value is, however, only correct if the contract does not hold any tokens of the given asset at the time of the call. Furthermore, some tokens have a lower maximum (e.g., `type(uint96).max` in the COMP token).