PUBLIC

Code Assessment

of the DSS Allocator

Smart Contracts

September 09, 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 DSS Allocator according to Scope to support you in forming an opinion on their security risks.

MakerDAO implements a token allocation system for AllocatorDAOs which consists of a core, a funnel and an automation layer.

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

The general subjects covered are specification and integration with 3rd party systems. All the aforementioned subjects are covered well.

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	0
Low-Severity Findings	2
• Code Corrected	2

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 DSS Allocator 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	17 Aug 2023	5af7242af3825009b7463c6842cbde5e075dcc16	Initial Version
2	23 Aug 2023	40005875d2eb3f6dd67b6267e6091d75e0eed565	Second Version
3	30 Aug 2023	a5469884813528b414ec1c2b985f52dd192455a1	Third Version
4	13 Oct 2023	7692abec1b85d9533b9f54392e7f081159e8d343	Fourth Version
5	13 Nov 2023	7268b572fc82d59d0bc2c552a8e8612e42c41e67	SwapperCalleePsm
6	12 Dec 2023	584dab103749ab6be3005b27fb2468440ab62e68	VaultMinter
7	28 August 2024	226584d3b179d98025497815adb4ea585ea0102d	Renaming

For the solidity smart contracts, the compiler version `0.8.16` was chosen.

The following files were in scope:

```
src/AllocatorVault.sol
src/AllocatorOracle.sol
src/IAllocatorConduit.sol
src/AllocatorRoles.sol
src/AllocatorBuffer.sol
src/funnels/callees/SwapperCalleeUniV3.sol
src/funnels/Swapper.sol
src/funnels/DepositorUniV3.sol
src/funnels/automation/StableDepositorUniV3.sol
src/funnels/automation/StableSwapper.sol
src/funnels/automation/ConduitMover.sol
src/funnels/uniV3/LiquidityAmounts.sol
src/funnels/uniV3/TickMath.sol
src/funnels/uniV3/FullMath.sol
src/AllocatorRegistry.sol
```

In (Version 5), the following file has been added:

```
src/funnels/callees/SwapperCalleePsm.sol
```

In (Version 6), the following file has been added:

```
src/funnels/automation/VaultMinter.sol
```

2.1.1 Excluded from scope

All files not mentioned above are out of scope. Additionally, Uniswap V3 is out-of-scope. All interacted tokens with transfers / approvals are expected to be standard ERC-20 tokens that revert on failure. Further, the selection of parameters is out of scope. Governance is trusted to set parameters correctly to achieve the desired properties. See Notes for examples of assumptions that should be considered.

Note that shutdown related functionality is explicitly excluded from this contract since, according to scoping, it is assumed to be unreachable.

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.

MakerDAO offers a token allocation system for AllocatorDAOs which consists of a core, a funnel and an automation layer.

2.2.1 Core Layer

Each AllocatorDAO will have a specific ilk for the minted NST to be allocated. For each such ilk, an `AllocatorVault` will be the only owner of the urn holding an ink balance for the particular ilk. The ink balance is created with the function `init()` by an authorized role (from the gem balance created by the governance through a call to `Vat.slip()`). Using the ink balance, the `AllocatorVault` draws debt and mints NSTs through `draw()`. With `wipe` NSTs are burned and debt is repaid. `wipe()` intentionally calls `Jug.drip()` before repaying debt (it is also expected that the AllocatorDAOs will not try to bypass this to have less debt). Note that with function `file()` the `Jug` can be replaced by an authorized role.

The funds generated are moved to the `AllocatorBuffer` which can give token approvals to arbitrary addresses (e.g. funnels or conduits) through the authorized function `approve()`. Note that the buffer may also hold tokens other than NSTs as a result of a funnel execution.

The `AllocatorRegistry` will keep track of mapping ilks to the aforementioned buffers. A new entry can be generated by the authorized function `file()`. However, note that the registry is not used to enforce any properties. Further, note that each gem for the ilk will have a constant price feed of 1M NST, see `AllocatorOracle`.

Each core contract implements the common local access control with `rely()` / `deny()` (except for the `AllocatorOracle`) that grants the rights to call the privileged functions. The `AllocatorVault` additionally can have custom roles defined in the `AllocatorRoles` contract. The `AllocatorRoles` contract allows defining more fine-grained access control which can be done by a role admin that can be defined with `setIlkAdmin()` for a given ilk. That role admin can then proceed to assign roles based on the target contract and function selectors for the ilk he manages (i.e. allowance to call a given function on a given target contract). New roles are created with `setRoleAction()` while roles are assigned with `setUserRole()`.

While the `AllocatorVault` and the `AllocatorBuffer` are infrastructure individual to the AllocatorDAO, the remaining contracts of the core are shared infrastructure.

2.2.2 Funnel (& Conduit) Layer

Note that conduits are not in scope of this audit.

Funnels & Conduits allow allocating the funds held by the `AllocatorBuffer` if it has given them the required approvals. Given the buffer-specific nature of funnels, they are to be deployed per buffer. While there is no whitelisting of funnels, MakerDAO implements two funnels:

1. `Swapper`: Authorized addresses can swap funds held by the buffer through a call to function `swapCallback()` on an arbitrary swap callee. However, a token rate limit per era must be respected which must be set through `setLimits()` by authorized addresses. Even though the swapper can be arbitrary, MakerDAO implements `SwapperCalleeUniV3` whose `swapCallback()` routes trades through the Uniswap V3 router.
2. `DepositorUniV3`: Authorized addresses can manage Uniswap V3 positions with `deposit()`, `withdraw()` and `collect()` to add and remove liquidity as well as to collect fees. Note that pool-based limits are in place. Namely, a token rate limit per token per era for both deposits and withdrawals must be respected.

Note that `setLimits()` resets the end to zero, allowing for immediate execution with the new parameters.

Each contract implements the common local access control with `rely()` / `deny()` and pairs it with the usage of the `AllocatorRoles` control.

2.2.3 Automation Layer

The automation layer enables the use of funnels and conduits in an automated fashion. More specifically, they wrap funnels/conduits and set further restrictions on the interaction with the underlying contracts. Then, keepers can call the available functions to trigger an allocation execution.

Three such contracts are introduced:

1. `StableSwapper`: Wraps the `Swapper`. With `setConfig()` the number, frequency, source token amount and slippage protection of swaps from source to destination token can be limited. `swap()` wraps `Swapper.swap()`.
2. `StableDepositorUniV3`: Wraps the `DepositorUniV3`. With `setConfig()` the number, frequency, desired token amounts and slippage protection for deposits and withdrawals can be limited. `deposit()`, `withdraw()` and `collect()` wrap the corresponding functions accordingly.
3. `ConduitMover`: Wraps conduits that match the interface. Withdraws from a conduit and deposits to another conduit. Note that in case the buffer is specified as the from or to address, the deposit/withdrawal is not called respectively. With `setConfig()` the number of movements, amounts and frequency from a source to a destination are limited.

Note that `setConfig()` resets the end to zero, allowing for immediate execution with the new parameters.

Each contract implements the common local access control with `rely()` / `deny()` paired with whitelisting functionality `kiss()` / `diss()`.

2.2.4 Changes in Version 2

In (Version 2), the `init()` function has been removed. The functionality will be executed by governance through a governance spell.

2.2.5 Changes in Version 4

In (Version 4), the Conduit Mover has been updated to check the return value when withdrawing from a conduit (`conduit.withdraw()`). The return value should be identical to the intended withdrawn amount (`cfg.lot`), otherwise, it will revert.

The UniswapV3 Swapper Callee has been updated to check the first token address in the encoded `path` matches the input `src`. This forbids the potential lingering approval on the callee (i.e. a lingering positive approval of USDT may block the consecutive swaps).

2.2.6 Changes in Version 5

In (Version 5), SwapperCalleePsm has been added.

SwapperCalleePsm facilitates the swapping between Dai and Gem tokens via the PSM (Peg Stability Module). The SwapperCalleePsm is bound to a Gem as immutable. It approves `max(uint256)` allowance of both Dai and Gem to PSM for the future swaps, and set `msg.sender` as the admin (a ward) in the constructor. It implements the common local access control with `rely()` / `deny()` and the wards have the privilege to swap on PSM with no fees via `swapCallback`.

2.2.7 Changes in Version 6

In (Version 6), the price returned by the `AllocatorOracle` has been changed from `1M (WAD)` to `1 (WAD)`.

The contract `VaultMinter` has been added. The `VaultMinter` sits on top and must be a `ward` of an `AllocatorVault`. It also implements the `draw` and `wipe` functions, but the amount, delay between calls, and number of time they can be called are limited by a set of rules defined in the `MinterConfig` struct. The caller of those functions must be a `bud`, and only a `ward` can update the rules (`minterConfig`). When `minterConfig` is updated, `zzz` is reset to `0` and a `draw/wipe` action can be triggered immediatley after the update. Note that the `VaultMinter` can be either in one of the `draw` or `wipe` modes at a time.

2.2.8 Changes in Version 7

In (Version 7), NST has been renamed to USDS.

2.2.9 Roles & Trust Model

The Allocation System assumes that the ESM threshold is set large enough prior to its deployment, so Emergency Shutdown can never be called.

The `PauseProxy` as well as the `AllocatorDAO Proxy` are expected to be the ultimate admins of the Allocation System. They perform actions through spells with a governance delay.

1. The `PauseProxy` is expected to be the ward of the `AllocatorRoles` and `AllocatorRegistry` singleton contract.
2. The `AllocatorDAO Proxy` is expected to be the ward of the individual `AllocatorVault` and the `AllocatorBuffer` contract. It is also expected to be an `ilkAuth` of its own `ilk` in `AllocatorRoles`, to add operators to its funnel network. Furthermore, it should be in charge of setting the rate-limiting parameters for its operators.

The operators (e.g. the automation contracts) can be whitelisted through the `AllocatorRoles` contract by the respective Proxy. They can perform predefined actions on the `AllocatorVault`, the funnels and the conduits. It is assumed the operators are inspected and set carefully by the governance.

The keepers are optional actors that can be setup to trigger the automation contracts in case repetitive actions are required, such as swap, deposit, withdraw, collect fees, and move funds between conduits. Their ability to arbitrage the system is always limited by the config the governance set.

We expect the governance to carefully inspect the proposal upon deployment of the contracts as well as any privileged actions. In case there are any other wards than the aforementioned proxies, they are assumed to be trusted and never act against the interest of the system and users. Otherwise, the funds could be stolen from the buffer. The keepers are not trusted and the configs are expected to be set correctly to limit their arbitrage ability.

In (Version 5), it is assumed that `SwapperCalleePsm` has privilege to swap with no fee on PSM (this requires `bud` role on `DssLitePsm`). In addition, we expect the `Swapper` being a `ward` of `SwapperCalleePsm`. It is further assumed that all the parties that have wards role on

`SwapperCalleePsm` are intended and trusted by the governance, becuase they can directly use `SwapperCalleePsm.swapCallback()` to swap with no fee.

In Version 6, it is assumed the `VaultMinter` has a `ward` role on the connected `AllocatorVault`.

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:

- 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	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	2

- Impossibility to Create One-Side Token1 Liquidity Code Corrected
- Incorrect Uniswap V3 Path Interpretation Code Corrected

Informational Findings	1

- Gas Inefficiencies Code Corrected

6.1 Impossibility to Create One-Side Token1 Liquidity

Correctness Low Version 1 Code Corrected

CS-MKALLOC-003

The `DepositorUniV3` funnel has a `uniswapV3MintCallback()` function for properly integrating with Uniswap V3 and move the funds. However, it only moves funds if the owed amount in token0 is greater than 0. Hence, if the current tick is outside of the position's tick range so that it leads to one-sided liquidity in token1, no funds will be transferrable. Ultimately, one-sided token1 liquidity cannot be added. Thus, deposits could be temporarily DOSed.

Code corrected:

`amt1Owed` is now used for transfers of token1.

6.2 Incorrect Uniswap V3 Path Interpretation

Correctness Low Version 1 Code Corrected

CS-MKALLOC-004

The swapper callback contract for UniswapV3 interprets the last tokens as follows:

```
lastToken := div(mload(sub(add(add(path, 0x20), mload(path)), 0x14)), 0x1000000000000000000000000)
```

Namely, it loads the last 20 bytes as the last token. However, the path may have some additional unused data so that the last token does not have any effect on the execution. Consider the following example:

1. The path is encoded as `[srcToken fee randomToken dstToken]`.
2. The swapper will interpret `dstToken` as the last token.

3. However, in UniswapV3, `randomToken` will be received.
4. In case no slippage requirements for the amount out are present, `randomToken` will be received successfully and will be stuck in the swapper contract.

Ultimately, the path is wrongly interpreted which could, given some configurations, lead to tokens lost unnecessarily due to bad input values.

Code corrected:

The check towards the correctness of path encoding has been removed, as it provides a false sense of security. Ultimately, the swap is protected by the minimum output token amount requirement.

MakerDAO states:

```
These checks were only meant to provide more explicit revert reasons for a subset
of (common) path misconfigurations and were not meant to catch all possible incorrect
path arrays. Ultimately the ""Swapper/too-few-dst-received"" check is the only one
that matters. But since that seems to cause confusion, we just removed the checks.
```

6.3 Gas Inefficiencies

Informational **Version 1** **Code Corrected**

CS-MKALLOC-005

Below is a non-exhaustive list of gas inefficiencies:

1. In `AllocatorVault.wipe()`, the call `vat.frob()` takes `address(this)` as an argument for the gem balance manipulation. However, due to the gem balance not being interacted with, using `address(0)` may improve gas consumption minimally.
2. In the withdrawal and deposit functions of the `UniV3Depositor`, an unnecessary `MSTORE` operation is performed when caching `era` into memory. Using only the `SLOAD` could be sufficient.

Code corrected:

Code has been corrected to optimize the gas efficiency.

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 Lack of Sanity Checks

Informational Version 1 Acknowledged

CS-MKALLOC-002

The code often lacks sanity checks for setting certain variables. The following is a non-exhaustive list:

1. On deployment, the conduit mover does not validate whether the ilk and the buffer match against the registry.
2. Similarly, that is the case for the allocator vault.

MakerDAO states:

```
The sanity checks are done as part of the init functions (to be called in the
relevant spell).
```

7.2 Potential Rounding Errors of SwapCallback

Informational Version 5 Specification Partially Changed

CS-MKALLOC-001

In case the caller swaps Dai for Gem tokens through the `swapCallback`, the input `amt` would be in Dai decimals. It is converted to Gem decimals to comply to the interface of `psm.buyGemNoFee`. However, this may introduce rounding errors if `amt` is not a multiple of `to18ConversionFactor`. The remainder of the Dai will be trapped and accumulated in the Callee contract. Consider the following example:

1. `cfg.lot` in the `StableSwapper` automation is `10**18+to18ConversionFactor-1`
2. A keeper calls `` `StableSwapper.swap() `` which calls `Swapper.swap()` leading to a call to the `SwapperCalleePsm's swapCallback()` with the `lot` as the amount and Dai as the source token.
3. The PSM will only pull `10**18` Dai from the callee.
4. The swap succeeds. Due to some slippage tolerance, the swap is accepted by `Swapper`.

In addition, once there is excess Dai, it is possible to swap another token instead of Dai to Gem through the PSM. Consider the following example:

1. Assume there are 100 Dai trapped in the callee contract for a gem unequal to USDT.
2. `swapCallback` is invoked with `src==USDT`, `amt==100*10**18`.
3. It goes into the `else` branch and falsely assumes the input token is Dai.
4. Eventually, it will swap the trapped 100 Dai to 100 Gem. And the input 100 USDT will be trapped in this contract.

In practice, it is expected to be invoked with an `amt` being a multiple of `to18ConversionFactor`. (e.g. the `cfg.log` of `funnels/automation/StableSwapper.sol` for this token pair). Otherwise the rounding error may appear and accumulate. Further, note that the rounding errors are dependent on the `to18ConversionFactor` and maybe different tokens (e.g. if the token has 6 decimals, it would require around 1M such iterations to reach a value of 1 USD).

A note has been added to clarify the potential accumulated dust in case `amt` is not a multiple of `to18ConversionFactor`. And MakerDAO states:

```
As both the callee's swapCallback() and the swapper's swap() are authenticated
functions, dust can only accumulate in the callee if a trusted operator uses an
incorrect `amt`. Operators are expected to always use `amt` that are multiple
of `to18ConversionFactor` when `src != gem`.
```

8 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.

8.1 1T NST Minting

Note Version 1

The documentation specifies that a maximum of 1T NST should be placed and that at most 1T NST should be mintable. However, that may not be the case if the spotter has `mat` and `par` set to unsuitable values. Technically, `Vat.rate` could be decreasing (depending on the jug). Hence, with a decreasing rate, more than 1T NST could be minted. Additionally, governance is expected to provide the allocator vault with a gem balance through `Vat.slip()`. Calling this multiple times would allow to re-initialize the allocator vault multiple times to create more ink than intended (and, hence, allowing for more debt than expected).

Ultimately, governance should be careful when choosing properties.

8.2 Deposit and Withdraw Share the Same Capacity

Note Version 1

The governance can set a `PairLimit` in `DepositorUniV3`, which limits the maximum amount of a pair of tokens that can be added or removed from the pool per era. Instead of setting two capacity parameters for adding liquidity and removing liquidity respectively, both actions share the same capacity.

8.3 Potentially Outdated Debt Estimation

Note Version 1

In contract `AllocatorVault`, `debt()` returns an estimation of the debt that is on the Vault's urn. This estimation could be outdated if the vat's rate has not been updated by the `jug.drip()` in the same block.

The getter `debt()` has been removed (along with `line()` and `slot()`). MakerDAO states that they are not strictly needed and can be implemented in another contract as well.

8.4 Shutdown Not Considered

Note Version 1

The shutdown was not in scope and users should be aware that consequences of a potential shutdown have not been investigated as part of this audit.

8.5 Topology May Break the Intended Rate Limit

Note **Version 1**

The keepers' ability to move funds between conduits/buffer and swapping tokens is limited by the triplets (`from`, `to`, `gem`) and (`src`, `dst`, `amt`) respectively. However, the actual funds flow between `from` and `to` (`src` and `dst`) could exceed the config dependent on the topology of the settings.

Assume there is a config that limits moving NST between conduits `CA` and `CB` to `100` per hop: `(CA, CB, 100)`. If there are another two configs `(CA, CX, 40)` and `(CX, CB, 60)` exist, then keepers can move at most `100 + 40 = 140` DAI from `CA` to `CB` per hop.

The same situation applies to Swapper. Therefore, the topology of the configs should be carefully inspected.

MakerDAO states:

```
The rate limit for each swap/move pair is an authed configuration of the allocator
proxy. It is therefore assumed to know what it is doing and is allowed to set any
configuration regardless of paths or duplication.
```