CANTINA

Maker Dao - DSS Allocator

Security Review

Cantina Managed review by:
Christoph Michel, Lead Security Researcher
M4rio.eth, Security Researcher
Shung, Associate Security Researcher

November 24, 2023

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

The Maker Protocol, also known as the Multi-Collateral Dai (MCD) system, allows users to generate Dai (a decentralized, unbiased, collateral-backed cryptocurrency soft-pegged to the US Dollar) by leveraging collateral assets approved by the Maker Governance, which is the community organized and operated process of managing the various aspects of the Maker Protocol.

From Sep 26th to Oct 9th the Cantina team conducted a review of dss-allocator on commit hash dc192486. The team identified a total of **11** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 4
- Gas Optimizations: 2
- Informational: 5

3 Findings

3.1 Low Risk

3.1.1 `SwapperCalleeUniV3` is incompatible with USDT

Severity: Low Risk

Context: SwapperCalleeUniV3.sol#L46

Description: The `SwapperCalleeUniV3` can be used by the `Swapper` as a Uniswap-V3 swap fulfilment callee for arbitrary tokens. The preliminary technical specification of the Allocation System mentions USDT among other tokens.

1. The `ApproveLike` interface defines an ERC20-compliant `approve` function that returns a `boolean`. USDT does not return a boolean and approve call `ApproveLike(src).approve(uniV3Router, amt)` would revert.
2. A second issue with USDT is that its `approve` function reverts when trying to set a non-zero value while the existing approval is already non-zero.

   ```
   require(!((_value != 0) && (allowed[msg.sender][_spender] != 0)));
   ```

   This can be exploited by an attacker that leaves the `callee` in a state with non-zero USDT approvals after the `swap` call: Anyone can call the `callee`; it is possible to use `src=USDT` but a different `path` value of `WETH -> MKR`. The approval to the Uniswap V3 router will be set, but the WETH pay-by-ETH special case in its `pay` function can be used to avoid the `WETH.transferFrom` call. The swap will succeed and the USDT approval will still be set on the callee. Future USDT swap calls will revert when trying to approve the UniswapV3 router a second time.

Recommendation: To address both issues, consider the following:

1. Use an interface that does not expect a return value for ERC20 functions, like `GemLike` in `AllocatorVault`.
2. Lingering approvals on the `SwapperCalleeUniV3` should not lead to additional attack vectors that aren't already possible by a keeper (as keepers are whitelisted and we can assume the `path` does not contain malicious tokens). Therefore, consider simply solving the issue for USDT by checking that `src == path[0]`, i.e., that the `src` token matches the first token in the `path`. This will ensure that the approved amount will indeed be transferred for USDT, and therefore reset the approval to `0`. Alternatively, use a function like `forceApprove` instead of `approve`.

Maker: `approve()` interface fixed and test added in commit e454ce4d. In commit e56ac158, reintroduced some of the checks removed in commit 1eebe56b.

Cantina: Fixed, path decoding matches Uniswap's.

3.1.2 Two separate auth systems can result in mistakes

Severity: Low Risk

Context: AllocatorVault.sol#L75-L79, Swapper.sol#L61-L64, DepositorUniV3.sol#L108-L111

Description: `AllocatorVault`, `Swapper`, and `DepositorUniV3` use two separate auth systems, one internal and one through `AllocatorRoles`. The `auth` modifier in these contracts passes if the address is authorized in either of these auth systems. For example, the `auth` modifier looks like below in `AllocatorVault`.

```
modifier auth() {
    require(roles.canCall(ilk, msg.sender, address(this), msg.sig) ||
            wards[msg.sender] == 1, "AllocatorVault/not-authorized");
    _;
}
```

This design will result in `deny` function to not have any effect if an address is authorized in `AllocatorRoles`.

```
function deny(address usr) external auth {
    wards[usr] = 0;
    emit Deny(usr);
}
```

This can be an issue if admin revokes the authorization of an address through the contract, but the address is also authorized in `AllocatorRoles`. Or, if the address is only authorized through `AllocatorRoles` but admin erroneously calls `deny` function of the relevant contract. Both of these scenarios can result in lingering authorizations.

Recommendation: Consider any of the following.

- Run a simulated test any time you update an authorization to ensure the intended effect is achieved.
- Add a check in `deny` functions to ensure the address is not also authorized through `AllocatorRoles`. This would alert the admin to first remove the authorization in `AllocatorRoles`.

Maker: Yes it is a caveat we will need to consider, but we need to keep both. In the practice wards will only be used by the subdao pause proxy, so should be only that address, which will be the admin of the roles contract for that `ilk`.

Cantina: Acknowledged.

3.1.3 The `AutoLine` can be called immediately after initialization

Severity: Low Risk

Context: deploy/AllocatorInit.sol

Description: Within the scoped commit for the audit, in the `AllocatorInit.sol` setup, the `Line` and `line` for a specific `ilk` was set to a debt ceiling which could be very high.

- deploy/AllocatorInit.sol#L198-L200

```
require(cfg.debtCeiling < WAD, "AllocatorInit/incorrect-ilk-line-precision");
 dss.vat.file(ilk, "line", cfg.debtCeiling * RAD);
 dss.vat.file("Line", dss.vat.Line() + cfg.debtCeiling * RAD);
```

After discussing with the client the decision was that they will be using the `AutoLine` approach, which let's you increase an `ilk` debt ceiling gradually accordingly with a `gap` (amount) and `ttl` (cooldown time). A new commit was provided with the `AutoLine` configuration.

Within this commit, the initial debt ceiling was set to the `gap` then the `AutoLine` is configured for the specific `ilk`. The `AutoLine` uses a public `exec` function which checks if the `ttl` was passed and that the `maxLine` was not reached, if these conditions were met then the `Line` and `line` will be increased with the `gap`.

Because the initialization increases the gap directly within the `vat`:

```
dss.vat.file(ilk, "line", cfg.gap);
dss.vat.file("Line", dss.vat.Line() + cfg.gap);
```

Then setting the `AutoLine`:

```
AutoLineLike(dss.chainlog.getAddress("MCD_IAM_AUTO_LINE")).setIlk(ilk, cfg.maxLine, cfg.gap, cfg.ttl);
```

Anyone can call the `exec` immediately and increase the debt ceiling (line) one more time due to the fact that the `AutoLine` considers that when a new `ilk` is set the debt ceiling should be increased afterwards using the `exec` function so the `ttl` to enter in effect.

Recommendation: Consider setting the `AutoLine` first then call the `exec` to setup the initial debt ceiling for the `ilk`.

Maker: So far this has been the way to set the line initially for all the `ilk`s that have been created since auto line exists. But it is definitely a good point to think about.

Cantina: Acknowledged.

3.1.4 The `withdraw` function can withdraw less than `lot`

Severity: Low Risk

Context: ConduitMover.sol#L102-L103

Description: The`ConduitMover` can be used to move funds between their `AllocatorBuffer` and the conduits in an automated manner. The operations of moving funds are done within some boundaries set by the facilitator, these boundaries are defined within the config of the `ConduitMover`.

```
struct MoveConfig {
    uint64   num; // The remaining number of times that a `from` to `to` gem move can be performed by keepers
    uint32   hop; // Cooldown period it has to wait between `from` to `to` gem moves
    uint32   zzz; // Timestamp of the last `from` to `to` gem move
    uint128  lot; // The amount to move every hop for a `from` to `to` gem move
}
```

If a whitelisted keeper performs an action of moving funds from one conduit to another by using the `move` function, he needs to define the `from` and `to` parameters. If these parameters are not the `buffer`, which means it does not need to move funds from/to the buffer, then a `withdraw` then `deposit` is performed on `from` and `to` with the configured `lot`.

Currently, one of the conduits that will be used is the ArrangerConduit which, when a `withdraw` is performed, can return less amount than requested, depending on how many funds are ready.

```
amount = maxAmount > withdrawableFunds_ ? withdrawableFunds_ : maxAmount;
```

This can break the following invariant within the `ConduitMover`: `withdraw amount == deposit amount == lot`. We've seen throughout the automation contracts that the config parameter's boundaries are not broken and should be respected by the keepers.

Recommendation: Consider adding a check that the actual `withdraw` amount is equal to the `lot`.

Maker: Fixed in commit 5dba577a.

Cantina: Fixed.

3.2 Gas Optimization

3.2.1 Unnecessary custom bitwise negation function

Severity: Gas Optimization

Context: AllocatorRoles.sol#L67-L69

Description: `AllocatorRoles._bitNot` function behaves identical to `NOT` opcode that is made available in Solidity with `~` sign. Therefore, this function is unnecessary and costlier.

```
function _bitNot(bytes32 input) internal pure returns (bytes32 output) {
    output = (input ^ bytes32(type(uint256).max));
}
```

Recommendation: Consider removing the function `_bitNot` and replacing all instances of `_bitNot(mask)` in `AllocatorRoles` contract with `~mask`.

Maker: Fixed in commit 184e4191.

Cantina: Fixed.

3.2.2 Left shift operation is cheaper than base two exponentiation

Severity: Gas Optimization

Context: AllocatorRoles.sol#L58, AllocatorRoles.sol#L62, AllocatorRoles.sol#L91, AllocatorRoles.sol#L101

Description: In AllocatorRoles, there are 256 possible roles, which are defined by their IDs from 0 to 255. The conversion from the role IDs to their respective bit position is done using a base two exponentiation. This is equivalent to a bit shift, which is cheaper and also better conveys the idea of mapping roles to bit positions.

Recommendation: All instances of `2 ** uint256(role)` in `AllocatorRoles` can be replaced with `uint256(1) << role` for reduced execution cost.

Maker: Fixed in commit 73cf7f33.

Cantina: Fixed.

3.3 Informational

3.3.1 There are unused functions in GemLike interface

Severity: Informational

Context: AllocatorBuffer.sol#L19-L24

Description: In `AllocatorBuffer.sol::GemLike` interface, all except one function is unused. Although there is no issues with this, it is not in line with the minimal code style of the rest of the codebase.

Recommendation: Consider removing all functions except `approve` from the relevant GemLike interface.

Maker: Fixed in commit 122b6a11.

Cantina: Fixed.

3.3.2 The `fee` parameter is missing from event emitting within the `DepositorUniV3.sol`

Severity: Informational

Context: DepositorUniV3.sol#L254, DepositorUniV3.sol#L298, DepositorUniV3.sol#L327

Description: Within the `deposit`, `withdraw`, and `collect` functions, an event is emitted with various params. The `fee` param is missing which can help the off-chain infra to get the right pool that the deposit/withdraw was done with. The logic to get a Uniswap v3 pool is the following:

```
_getPool(p.gem0, p.gem1, p.fee)
```

Recommendation: Consider adding the `fee` parameter to the events within the deposit, withdraw, and collect.

MakerDAO: Fixed in commit 5dea8ea.

Cantina: Fixed.

3.3.3 Wrong loss caps in Security Model documentation

Severity: Informational

Context: Documentation

Description: The documentation mentions the following constraints on losses in the "Security Model" section:

> A keeper can not incur a loss of more than the funnel operator can, and any loss it can incur is also constrained by `req` or `req0` and `req1` for a specific configuration.

While we believe the code matches the desired behavior, this section is unclear in its meaning.

Recommendation: The `req*` variables are the min outputs of swaps / allocations, the loss should be defined on the *input* amounts, like was done for the funnel operators with `cap`. To match the loss definition

of the funnel operator, consider changing this section to "is also constrained by `lot` or `amt0` and `amt1` for a specific configuration (invocation)".

Maker: Clarified the keeper's assumptions in commit 4928ce01.

Cantina: Fixed.

3.3.4 Various typos within the codebase

Severity: Informational

Context: See bellow

Description: Throughout the codebase we found various typos:

- StableSwapper.sol#L55: permissionned → permissioned.
- StableSwapper.sol#L103: form → from.
- README.md#L93: AllocatoBuffer → AllocatorBuffer.
- README.md#L133: opeator → operator.
- SwapperCalleeUniV3.sol#L23: broken `SwapRouter` link in. It should be `https://github.com/Uniswap/v3-periphery/blob/b06959dd01f5999aa93e1dc530fe573c7bb295f6/contracts/SwapRouter.sol`.

Recommendation: Consider fixing the mentioned typos for better readability.

Maker: Fixed in commits 03ca8e33 and e454ce4d.

Cantina Fixed.

3.3.5 Custom division function `_divup` misbehaves when `0` is divided by `0`

Severity: Informational

Context: AllocatorVault.sol#L102-L106

Description: `AllocatorVault._divup` returns `0` when `0` is divided by `0`. The expected behaviour aligned with Solidity division operator should be a revert.

```
function _divup(uint256 x, uint256 y) internal pure returns (uint256 z) {
    unchecked {
        z = x != 0 ? ((x - 1) / y) + 1 : 0;
    }
}
```

As seen above, `x == 0` will return `0`, even if `y` is also a `0`. However, this function is only called as `dart = _divup(wad * RAY, rate)` and `rate` is never `0`. So there is no direct impact.

Recommendation: Consider fixing or documenting this feature so there is no bug introduced if this function is used elsewhere.

Maker: Added comment here. We are acknowledging the issue but leaving it as it is for this repo.

Cantina: Acknowledged.

4 Additional Comments

The Cantina team reviewed MakerDao's dss-allocator changes holistically on commit hash 7692abec1b85d9533b9f54392e7f081159e8d343 and determined that all issues were resolved and no new issues were identified.