Maker DAO: Lockstate Engine

Security Review

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

June 26, 2024

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 May 20th to May 30th the Cantina team conducted a review of lockstate on commit hash 735e1e85.

The specific scope of the review included the following components:

- All files within lockstate/src
- The deployment scripts at lockstate/deploy

The Cantina team reviewed MakerDao's Lockstate Engine changes holistically on commit hash de66d6fc3b7478f0b5c88fbcfe79bbe899be65e4 and determined that all issues were resolved and no new issues were identified.

The team identified a total of **14** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 1
- Gas Optimizations: 3
- Informational: 10

3 Findings

3.1 Low Risk

3.1.1 The debt ceiling can be increased immediately after initialization

Severity: Low Risk

Context: LockstakeInit.sol#L170-L177

Description: Within the `LockstakeInit` script, the debt ceiling has to be set to the `gap`: `dss.vat.file(cfg.ilk, "line", cfg.gap);` then the `AutoLine` is configured for the `ilk`.

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

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`.

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

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

Maker We decided to accept this trade-off and keep the consistency with the other init scripts.

Cantina Managed: Acknowledged.

3.2 Gas Optimization

3.2.1 Inefficient reentrancy lock

Severity: Gas Optimization

Context: LockstakeClipper.sol#L156-L162

Description: The `lock` modifier of `LockstakeClipper` uses `0` and `1` to represent reentrancy lock state. This is inefficient due to zero-to-nonzero storage changes requiring 17100 extra gas. This extra amount would be refunded when `lock` is set to its original value, but it still increases the required gas limit of the transaction.

Recommendation: Consider using `1` and `2` to represent reentrancy lock state (see Solmate's `ReentrancyGuard.sol`).

Maker: Considering the final gas usage is fine, we will keep as it is.

Cantina Managed: Acknowledged.

3.2.2 Unnecessary checks in `LockstakeClipper.take`

Severity: Gas Optimization

Context: LockstakeClipper.sol#L400

Description: Within the `take` function an external call is made to the `who.clipperCall`. This external call is made only if the `data` is not empty and `who` is not the three authorized contracts in the clipper: `vat`, `dog` and `engine`.

These three contracts do not implement the `clipperCall` function which renders this check unnecessary. The contracts' implementation can be checked here:

- `vat.sol`.
- `dog.sol`.
- engine.

Recommendation: Consider removing the checks regarding `who` not being the three authorized contracts.

Maker: Acknowledged. This was done at the time just to be extra careful, thinking that there could also be a selector collision.

Even though this could be ensured off-chain we opted not to remove this check and align to the other clippers.

Cantina Managed: Acknowledged.

3.2.3 `kiss` optimization in `LockstakeInit`

Severity: Gas Optimization

Context: LockstakeInit.sol#L184-L187

Description: In this initialization script, we have 4 `kiss` actions for four addresses, one kiss per address. The `PipLike` also has a function that can accept multiple addresses into one single `kiss`: `kiss(address[] a)`.

Recommendation: Consider using the `kiss` function that accepts multiple addresses as a parameter:

```
address[4] memory kisses = [address(dss.spotter), address(clipper), clipperMom, address(dss.end)];
PipLike(pip).kiss(kisses);
```

Maker: Leaving as it is as gas efficiency is not really relevant for the init scripts.

Cantina Managed: Acknowledged.

3.3 Informational

3.3.1 Lack of input sanitization in `LockstakeClipper.file()`

Severity: Informational

Context: LockstakeClipper.sol#L170-L187

Description: `file()` functions of `LockstakeClipper` do not sanitize input values and accept invalid inputs. Below snippet shows that `chip` and `tip` values can truncate, and `stopped` can be set beyond accepted range.

```
else if (what == "chip")       chip = uint64(data);   // Percentage of tab to incentivize (max: 2^64 - 1 =>
↪  18.xxx WAD = 18xx%)
else if (what == "tip")         tip = uint192(data);  // Flat fee to incentivize keepers (max: 2^192 - 1 =>
↪  6.277T RAD)
else if (what == "stopped") stopped = data;           // Set breaker (0, 1, 2, or 3)
```

Recommendation: Consider explicitly checking if an input is within the accepted range, and reverting if the input is invalid.

Maker: `chip` was already checked in the scripts, so we added a similar boundary check for `tip` which prevents the casting overflow, see commit de66d6fc.

Cantina Managed: Acknowledged. Checks for `tip` and `chip` are implemented in the deployment scripts. The issue still exists if `file` will be called outside of the deployment scripts in the future.

3.3.2 Small `bark`s can delay the `selectVoteDelegate`

Severity: Informational

Context: LockstakeEngine.sol#L260

Description: The liquidation system in Maker DAO is managed by the `dog` contract.

If a vault is unhealthy, one can call `dog.bark` to start a Dutch auction to sell its collateral for `DAI`. In most cases, the full amount of collateral is sold except when the target amount of `DAI` to be raised in the resulting auction (debt of Vault + liquidation penalty) causes either `Dirt` to exceed `Hole` or `ilk.dirt` to exceed `ilk.hole` by an economically significant amount, we call this a partial liquidation.

In partial liquidation, one can buy only part of it with a minimum amount of `dust`, defined for each `ilk` in the `vat` system leaving room for another auction to be triggered to recover more of the debt. This denotes the fact that one vault can have multiple Auctions started to recover the debt.

In the `LockstakeEngine`, when an Auction is started, a count is incremented:

```
// --- liquidation callback functions ---

function onKick(address urn, uint256 wad) external auth {
    // Urn confiscation happens in Dog contract where ilk vat.gem is sent to the LockstakeClipper
    (uint256 ink,) = vat.urns(ilk, urn);
    uint256 inkBeforeKick = ink + wad;
    _selectVoteDelegate(urn, inkBeforeKick, urnVoteDelegates[urn], address(0));
    _selectFarm(urn, inkBeforeKick, urnFarms[urn], address(0), 0);
    lsmkr.burn(urn, wad);
    urnAuctions[urn]++; // @audit-info <== Count incremented
    emit OnKick(urn, wad);
}
```

And when an Auction is closed, the same count is decremented.

During the Liquidation processes, as long as Auctions are opened a urn can not be used to select votes or select farms due to this check:

```
require(urnAuctions[urn] == 0, "LockstakeEngine/urn-in-auction");
```

A malicious actor can delay selecting votes of whales that are in the liquidation process if the system is in the following state:

- The `DAI` raised would cause a partial liquidation.
- `dust` is low enough so the malicious actor could open multiple auctions.

The delay can be caused by spamming multiple small auctions (each auction value being at least the 'dust' value) to fill the entire block or multiple blocks. This will result in time delays in various governance processes.

We have classified this issue as informational because the likelihood of it occurring is very low.

Recommendation: There is not much that can be done to combat this without refactoring the existing contracts, one countermeasure would be to have the `dust` value high enough to discourage this behavior.

Maker: `dust` is assumed to be appropriately set.

Cantina Managed: Acknowledged.

3.3.3 Additional deployment config checks

Severity: Informational

Context: LockstakeInit.sol#L161

Description: The deployment script performs checks on its configuration variables.

Recommendation: Consider also checking that `cfg.gap <= cfg.maxLine` for the `autoLine`, ensuring that the initial line of `gap` is less than the max defined line.

Maker: Fixed in commit 98fd5420.

Cantina Managed: Verified.

3.3.4 `Multicall` revert errors might be incomprehensible

Severity: Informational

Context: Multicall.sol#L14-L21

Description: When a `Multicall` subcall reverts, the revert data is assumed to be from a revert error encoding a string:

```
if (!success) {
    // Next 5 lines from https://ethereum.stackexchange.com/a/83577
    if (result.length < 68) revert();
    assembly {
        result := add(result, 0x04)
    }
    revert(abi.decode(result, (string)));
}
```

If the revert data is less than 68 or the string decoding fails, the call reverts with an opaque reason. This can naturally happen if the subcall reverts with a `Panic(uint256)` or a custom error selector.

Recommendation: Consider bubbling up the revert data in the `Multicall` instead of assuming a type and trying to decode it.

```
if (!success) {
    if (result.length == 0) revert("multicall failed");
    assembly ("memory-safe") {
        revert(add(32, result), mload(result))
    }
}
```

Maker: Fixed in commit d3878d40.

Cantina Managed: Verified.

3.3.5 `LockstakeUrn` incompatible with farms with non-standard reward tokens

Severity: Informational

Context: LockstakeUrn.sol#L78, StakingRewards.sol

Description: The `farm` contracts `StakingRewards` use `SafeTransfer` to be compatible with non-standard ERC20 reward tokens.

Recommendation: Even though the `LockstakeUrn` is currently only planned to use farms that use either NST or a subDAO token as a reward, consider performing a `SafeTransfer` call in `LockstakeUrn.getReward` to be compatible with all farms in case the scope expands in the future.

Maker: Acknowledged. The only tokens being used are NST and SubDAO ones.

On the other hand, `getReward` will just work for any reasonable token, as the interface doesn't expect a return value, meaning that even those known tokens that do not return anything will work. The only case that is problematic is if the token doesn't revert on failure. Those tokens are not supported but they should be extreme rare cases.

Cantina Managed: The issue has been acknowledged by the client.

3.3.6 The `getUrn` function does not guarantee the `urn` existance

Severity: Informational

Context: LockstakeEngine.sol#L221-L229

Description: The `getUrn` function calculates the urn address using `create2`. This will also work for future urn addresses that are not actually created.

Recommendation: Consider adding a comment in the `getUrn` function. Furthermore, consider creating a second function that calculates an `urn` address and calls `urn.vat()` to check if the urn actually exists.

Maker: We decided to add a comment to the `getUrn` specifying this behavior, see commit f240c591.

Cantina Managed: Verified.

3.3.7 IOUs locked in vote delegate instead of sent to user

Severity: Informational

Context: VoteDelegate.sol#L89, Old VoteDelegate.sol#L86

Description: Unlike the existing vote delegate contracts, the new contract does not transfer the IOU tokens received from the `chief` to the delegating user. It was stated that the IOUs are obsolete and not planned to be used.

Recommendation: Consider documenting this difference in IOU behavior, either in the Lockstake Engine vote delegate changes section 3a) or in the vote delegate repository.

Maker: Fixed, a comment has been added in the readme (see commit 5132023e):

```
In order to simplify the logic, the IOU tokens generated by DSChief are kept in the new VoteDelegate contract.
```

Cantina Managed: Verified.

3.3.8 Stakers need to reselect their vote delegate and farms after liquidation

Severity: Informational

Context: LockstakeEngine.sol#L425-L426

Description: When a liquidation auction is `kick`ed off, the urn unselects the current vote delegate and farm.

Recommendation: If the liquidation was a partial auction or an auction with leftover collateral, the user needs to remember to select the vote delegate and the farm again to be able to vote and earn rewards. Users should monitor their positions.

Maker: Fixed by adding a comment in the readme to document this behavior (see commit d2e3c596):

```
Users need to manually delegate and stake again if there are leftovers after liquidation finishes.
```

Cantina Managed: Verified.

3.3.9 Urn operators can create and remove urn operators

Severity: Informational

Context: LockstakeEngine.sol#L247

Description: The `LockstakeEngine.hope` function allows assigning new operators for the urn. The function is itself authorized by the `urnAuth` modifier, allowing operators to create new operators or remove other operators. If a single urn operator is compromised, they can revoke all other operators and create new compromised operators (potentially faster than the urn owner can remove them).

Recommendation: Consider changing `hope` and `nope`'s authorization to the urn owner.

Maker: Acknowledged. We prefer to leave the extra flexibility of this function and accept a bit of extra risk, which in our opinion shouldn't be much as a malicious contract will directly empty the urn.

Cantina Managed: The client has acknowledged the issue.

3.3.10 Named parameters in mapping types

Severity: Informational

Context: LockstakeEngine.sol#L69-L76

Description: The code uses mapping types without named parameters and uses comments to document the parameters:

```
mapping(address => uint256)                     public wards;            // usr => 1 == access
mapping(address => FarmStatus)                  public farms;            // farm => FarmStatus
mapping(address => uint256)                     public usrAmts;          // usr => urns amount
mapping(address => address)                     public urnOwners;        // urn => owner
mapping(address => mapping(address => uint256)) public urnCan;           // urn => usr => allowed (1 = yes, 0
↪  = no)
mapping(address => address)                     public urnVoteDelegates; // urn => current associated
↪  voteDelegate
mapping(address => address)                     public urnFarms;         // urn => current selected farm
mapping(address => uint256)                     public urnAuctions;      // urn => amount of ongoing
↪  liquidations
```

Recommendation: Consider using named parameters in mapping types. They could replace most of the current comments for the mappings, making the code more readable.

```
mapping(address farm => FarmStatus farmStatus) public farms;
mapping(address usr => uint256 urnAmount) public usrAmts;
mapping(address urn => address owner) public urnOwners;
// ...
```

Maker: Fixed in commit d93b7f78.

Cantina Managed: Verified.