dss-charter Assessment

In the dates of 2021-09-05 to 2021-09-12 B.Protocol's devs, led by Yaron Velner, were engaged by MakerDAO's Protocol Engineering team to perform a security audit for the implementation of `dss-charter` module.

UPDATE: On the 2021-09-14 the Protocol Engineering team addressed some of the comments.

Table of contents

Scope

The `dss-charter` module implements three (potentially independent) features:

1. *Origination fees*: This feature enables the protocol to collect a (one-time) minting fee over the DAI debt, in addition to the applicable stability fee.
2. *Permissioned vaults*: This allows only permissioned white-listed addresses to accrue DAI debt for collateral types that use this module.
3. *Initial margin*: This feature make it possible for the protocol to define an initial margin collateral ratio for the user, and users below that margin are not allowed to decrease their collateral ratio. The initial margin could be potentially higher than the liquidation ratio (a.k.a Maintenance margin).

We reviewed `CharterManager.sol` from commit hash `6ab5cd69323cf2686d863e337d9e4e635bf79ad1` and `join-managed.sol` from commit hash `74d2b020b3b3c76feaa24e13cbaef016eb10e528`.

UPDATE: On the 2021-09-14 the Protocol Engineering team submitted a new version with few fixes. `CharterManager.sol` new commit hash is `ab207e9d3e1a35c8b2923527153918b126639b92` (which is identical to squashed commit hash `b0e8443be7217c3c44d58d2e694162e8ba8990e8`), and `join-managed.sol` new commit hash is `244430936ccf58b70e6974525bb4f243cd1ae2d4`.

Findings

Overall the code is of high quality and only low severity issues were found. In addition to the issues, we also suggest some easy gas optimizations, and give few notes on the design for the team to consider.

UPDATE: The team addressed the issues and comments.

1. Correctness

1.1 Wrong event logging

The `Rely` and `Deny` events log the `msg.sender` instead of the `usr`.

UPDATE: Issue was fixed.

1.2 ERC20 function return values are ignored

The implementation of `CharterManager.sol` ignores the return value of `transferFrom` and `approve`. We recommend to either check the return values, or implement a safeERC20 logic that would be able to handle also with non compliant ERC20 implementation, such as USDT.

We also note that this is inconsistent with the implementation of `join-managed.sol`, where return values are checked.

UPDATE: The issue was acknowledged by the team.

> *The token will be correctly handled as each adapter will take care of the tokens idiosyncrasies. We prefer not to support only certain tokens or complicate the code.*

1.3 Tokens with 19 or more decimals should be blocked

The `ManagedGemJoin` constructor does not validate that `gem.decimals() <= 18`. As a result, overflows might occur here and here.

UPDATE: Issue was fixed.

1.4 README typos

1. The README file does not explicitly mentions the initial margin feature.
2. The README file explains that the `dss-charter` wraps the `AuthJoin`, however in practice it wraps `ManagedGemJoin`.

UPDATE:

1. Some additional text was added at line 11 of the README file. We recommend to add a description already at line 3.
2. Issue was fixed.

2. Styling

Overall the code is nicely organized and clean. There are few places where the style seems to be inconsistent with the rest of the `dss` code.

2.1 `TokenLike`

Typically this is named `GemLike`.

UPDATE: Issue was fixed.

2.2 `rely`, `deny` and `auth`

These functions are usually implemented in a single line (e.g., here. This is not the case for `CharterManager.sol`.

UPDATE: The issue was acknowledged by the team.

> *Multiple expressions in each, split across lines is reasonable.*

2.3 Scattered event definitions

Typically event definitions are grouped together. In `CharterManager.sol` they are scattered over many places in the code.

UPDATE: Issue was fixed.

2.4 Redundant parameters in `frob` function

The `frob` function has 2 redundant parameters. Indeed, the implementation forces `u` and `v` to be the same, and forces `w` to be equal to `msg.sender`. The current `frob` prototype is already different than the one in the `vat`. Hence, it is not clear how these extra parameters contributes to the styling of the code.

UPDATE: The issue was acknowledged by the team. In addition, following comment 3.4 the prototype was changed and it is now identical to the one that is in the `vat`.

> *We left all the parameters so in the future (if needed) we can extend the functionality to a full frob flexibility without having to change the abi.*

3. Gas optimization

3.1 `UrnProxy`

1. Consider removing the `usr` field. Which is not used anywhere.
2. Consider calling `selfdestruct` after the `vat.hope` is executed.

UPDATE: The issue was acknowledged by the team.

> *proxy->usr retrieval needed in CropManager, sticking with same UrnProxy implementation for both contracts.*

3.2 Duplicated external calls

In the `join` function there are two external calls for `gemJoin.gem()`. Consider storing the result of the first call in a local variable, and avoid making the second call. This can also mitigates malicious `gemJoin` input which returns different result for the first and second call.

UPDATE: Issue was fixed.

3.3 Avoid `validate` logic execution when repaying debt

The `validate` function can check if `dink >= 0 && dart <= 0` before calculating the `tab`, and in particular before making an external call to `vat.urns()`.

UPDATE: Issue was fixed.

3.4 `frob` and `flux` can get `ilk` as input

Currently `frob` and `flux` get `gemJoin` as input, and it is only used to make an external call to `gemJoin.ilk()`. The extra calls can be avoided by having `ilk` as input instead of `gemJoin`. This is already done, e.g., in `quit`.

UPDATE: Issue was fixed.

3.5 `draw` function

Technically it is possible to replace this sequence of a single `frob` and two `move`s with a sequence of two `frob`s. This will save an external contract call, and can also help towards simplifying the code, however a simulation is required to determine if it will actually save gas, and it will have the byproduct of effectively increasing the `dust` level.

UPDATE: The issue was acknowledged by the team.

> *Subjective, we think current implementation is elegant and the suggested optimization will not necessarily save gas.*

3.6 `mload` op

The `mload` operation is not needed, and one can just set `ptr = 0`. The saving is very small though, so it is mostly a matter of taste. New proxy implementations, e.g., Gnosis and OpenZepplin just set it to 0.

UPDATE: The issue was acknowledged by the team.

> *Although free memory ptr is 0 (start of an external call), we will still use the ptr for safety.*

4. Notes

This section contains general comments and thoughts about the implementation and design of the code. Unlike the other sections, the findings here are not necessarily a recommendation for a change.

4.1 The new Vault implementation is not compatible with DssCdpManager

In the new Vault, users will not be able to call `vat` functions directly. As a result `DssCdpManager` can no longer be used. As the Vault is new, it is does not have any affect on existing systems that are integrated with MakerDAO, but moving forward it will require a new implementation from interfaces such as Oasis.app, Instadapp and potentially also DeFiSaver.

UPDATE: The issue was acknowledged by the team.

4.2 Upgradable proxy Pattern

The `CharterManager` is implemented as an upgradable proxy. If your implementation is expected to have multiple upgrades during the product life-time, you might want to consider separating the contract storage from its logic, e.g., have a v0, v1, v2, v3... storage, where v_{n+1} inherits from v_n, and each impl inherits from storage_{v_n}.

On the other hand, MakerDAO code styling typically avoids inheritance patterns, and favors a flat implementation. If the upgradability is only to mitigate unexpected events and bugs, then the current implementation would suffice.

UPDATE: The issue was acknowledged by the team.

> *Makerdao preferences self contained code without inheritance if possible. No upgrades are purposely planned. Added comments to emphasize storage layout.*

4.3 Immutable `spotter` and `vow`

In the general `dss` system, the `spotter` and `vow` values could change over time. In the current implementation this will require to redeploy a new proxy for `CharterManager`. This can be avoided if you allow these values to be mutable and configurable via the `file` function.

UPDATE: The issue was acknowledged by the team.

> *Vow and Spot upgrades should be rare.*

4.4 `CharterManager` may call malicious `GemJoin` contracts

Letting the contract call arbitrary `gemJoin` contract is not a best practice. In the current implementation anyone can force the contract to give infinite allowance of any token to a malicious contract. This on its own does not put funds at risk, however, as the contract is upgradable, and since funds might get stuck in the contract due to user interaction errors, this might be an issue in the future. I would suggest to have a configurable mapping of ilk to gemJoin to mitigate it.

Alternatively, a more light weight mitigation would be to call `gemJoin.gem()` only once at `join` function.

UPDATE: The issue was mitigated by verifying that `gemJoin` is an authorized contract in the dss system (specifically by the `vat` contract). The mitigation is satisfactory, and nicely balances simplicity and gas costs concerns vs hypothetical future security concerns. A more comprehensive solution would be to verify `gemJoin` is listed in the `IlkRegistry` contract, however it would entail higher gas cost.