CANTINA

MakerDAO: Spark ALM Controller

Security Review

Cantina Managed review by:
M4rio.eth, Security Researcher
Jonatas Martins, Associate Security Researcher

September 25, 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 Sep 18th to Sep 19th the Cantina team conducted a review of spark-alm-controller on commit hash 7a0535ee.

The Cantina team reviewed MakerDAO's spark-alm-controller changes holistically on commit hash 2cc82124 and determined that all issues were resolved and no new issues were identified.

The team identified a total of **7** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 1
- Gas Optimizations: 0
- Informational: 6

3 Findings

3.1 Low Risk

3.1.1 More granular limits for CCTP transfers

Severity: Low Risk

Context: MainnetController.sol#L308, ForeignController.sol#L182

Description: The controllers have the capability to transfer USDC from the proxy to various chains using the CCTP protocol, with the destination domains specified as parameters when the transfer function is called.

In our view, it would be more effective to implement granular rate limiting per domain rather than applying a global rate limit across all domains. This approach acknowledges that different domains may carry varying risk-levels, and a lower limit for riskier domains would provide a better security.

Currently, this functionality is not available, leaving open the potential for a relayer to deplete the global limit on just one domain, which could pose unnecessary risks.

Recommendation: Consider making the rate limiting more granulated to improve the security per domain.

MakerDAO: Fixed in commit 5ea7f6d2.

Cantina Managed: Verified.

3.2 Informational

3.2.1 Missing comment about counterintuitive quote amount in swap

Severity: Informational

Context: MainnetController.sol#L243

Description: The `MainnetController` includes a function called `swapUSDSToUSDC`, which allows swapping an amount of `USDS` to `USDC`. This function takes the `USDC` amount, which is denominated in 6 decimals, and then normalizes it to 18 decimals (the denomination of `USDS`). The normalized amount is then used to complete the swap and convert it to `USDC`.

The issue here is the counterintuitive nature of the parameter. Given the function name `swapUSDSToUSDC`, one would logically expect the parameter to be the `USDS` amount (denominated in 18 decimals). However, in this case, the parameter is actually the `USDC` amount. This behavior is also tied to the rate limiter, which uses the USDC amount as a rate-limiting key.

While we understand the design decision behind this approach—specifically to avoid precision loss when converting from 18 decimals to 6 decimals—it may not be immediately apparent to relayers. This could result in confusion and lead them to mistakenly input a `USDS` amount instead of the expected `USDC` amount.

Recommendation: Consider improving the documentation of this behavior, even though the variable is named `usdc` to clearly indicate that a `USDC` amount is expected. It would be beneficial to highlight this aspect further to ensure that users fully understand the function's requirements, helping to prevent potential errors in the future.

MakerDAO: Fixed in commit 5ea7f6d2.

Cantina Managed: Verified.

3.2.2 Unused variables and missing check within `ForeignController`

Severity: Informational

Context: ForeignController.sol#L72-L76

Description: Within the `ForeignController`'s constructor the `psm` variable is initialized supposedly being a PSM3 that contains the following 3 assets: `usds`, `susds`, `usdc`.

We do not see any check that verifies this being true in the constructor, we expect this to be done in the deploy script.

Furthermore, the `usds` and `susds` variables are not used at all in the contract, the functions that deal with these assets actually receive the asset as a parameter.

Recommendation: Consider if the check that the `psm` has the right assets should be performed in the constructor, if not consider removing the unnecessary variables.

MakerDAO: Fixed in PR 26.

Cantina Managed: Verified, fixed by removing the variables `usds` and `susds`.

3.2.3 Improved comments

Severity: Informational

Context: See below

Description: Throughout the code we could notice various comments that could be improved:

- MainnetController.sol#L260: `Approve DAI to PSM from the proxy (assumes the proxy has enough DAI)` could be `Approve DAI to PSM from the proxy` because the `daiUsds.usdsToDai` always returns the `usdsAmount`.
- IRateLimits.sol#L106: `@dev Sets rate limit data for a specific key.` could be `@dev Sets rate limit data for a specific key with lastAmount == maxAmount and lastUpdated == block.timestamp` to specify what happens with the last amount and last updated params.
- IRateLimits.sol: The `IRateLimits` uses only `@dev` NatSpecs while the `IALMProxy` uses `@dev` and `@notice`, consider standardizing the NatSpec format.
- IRateLimits.sol#152: Should specify that this function does not revert if `maxAmount` is reached.
- MainnetController.sol#L284: `// Swap USDC to DAI through the PSM` should be `// Swap USDC to DAI through the PSM 1:1`.

Recommendation: Consider fixing the above comments.

MakerDAO: Fixed in commit 5ea7f6d2.

Cantina Managed: Verified.

3.2.4 The `psm.to18ConversionFactor()` call can be cached

Severity: Informational

Context: MainnetController.sol#L246, MainnetController.sol#L276

Description: The `psm.to18ConversionFactor()` call can be cached as this variable is immutable in the psm.

Recommendation: Consider cache this variable into an immutable variable.

MakerDAO: Fixed in commit 5ea7f6d2.

Cantina Managed: Verified.

3.2.5 The `mintRecipient` is already checked in the `TokenMessenger`

Severity: Informational

Context: MainnetController.sol#L312

Description: In the `MainnetController`, when transferring USDC via CCTP, the `mintRecipient` is checked to not be 0, this check is already performed byt he CCTP contract itself.

Recommendation: Consider removing this redundant check.

MakerDAO: Acknowledged. We are okay to leave this redundant since it is such a high importance check, don't want an external dependency on it.

Cantina Managed: Acknowledged.

3.2.6 The `doCall` forwarding the callvalue is counterintuitive

Severity: Informational

Context: ALMProxy.sol#L32

Description: The `ALMProxy` is a contract that stores assets and allows arbitrary, permissioned calls to it. It has implemented a few functions to allow that:

- `doCall(address target, bytes memory data)`: Performs a `.call` to a target with `msg.value` sent as `callvalue`.
- `doCallWithValue(address target, bytes memory data, uint256 value)`: Performs a `.call` to a target with `value` sent as `callvalue`.
- `doDelegateCall(address target, bytes memory data)`: Performs a `.delegatecall` to a target.

The `doCall` function is ambiguously forwarding the `msg.value` as callvalue as one expects this one to execute a simple call with no value, the `doCallWithValue` should cover all the cases where callvalue must be forwarded.

Recommendation: Consider modifying the `doCall` function by removing the `payable` and make the function just perform a simple `call` without forwarding any `callvalue`.

MakerDAO: Fixed in commit 5ea7f6d2.

Cantina Managed: Verified.