CANTINA

# MakerDAO: Spark ALM Controller

## Security Review

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

December 13, 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 Dec 5th to Dec 10th the Cantina team conducted a review of spark-alm-controller on commit hash ad4391c3. The team identified a total of **7** issues in the following risk categories:

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

The Cantina team reviewed MakerDAO's spark-alm-controller changes holistically on commit hash 2bb26808 and determined that all issues were resolved and no new issues were identified.

# 3 Findings

## 3.1 Medium Risk

### 3.1.1 Malicious Delegate Signer can burn funds from the `ALMProxy`

**Severity:** Medium Risk

**Context:** See below

**Description:** Ethena's minting/redeeming flow uses the Ethena backend API to mint/redeem USDE for a user. The process involves two steps:

- Request an RFQ (see the Ethena docs).
- Submit an order using that RFQ, which will return a transaction hash if the transaction is successfully mined on-chain (see the Ethena docs).

*Note: For this to be successful, the user must approve the EthenaMinting contract to spend USDC or USDE*.

The second step requires gas for the transaction to be submitted on-chain, which is subtracted from the received amount. This means that the minimum amount the user requests for the order must be at least equal to the gas value. This is also outlined in Ethena's Terms and Conditions.

**Example:**

- Assumptions: 1 USDC = 1 USDE

If we want to mint USDE with 100 USDC, with the gas price being $10, we will receive 90 USDE as 10 will be kept for submission gas.

This raises an issue in the `MainnetController` where a malicious delegate signer can submit transactions with slightly greater value -- just enough to exceed the gas price and avoid the error returned by Ethena's API when the gas cost is not covered:

```
"error": {
    "error_code": "8",
    "error_name": "Size must be enough to cover the incurred gas cost."
}
```

By abusing this, the malicious signer can effectively "burn" small amounts of USDC (or USDE in the case of redeeming) at no cost up to approved amount. The signer can burn both USDC by using the minting flow and USDE by using the redeem flow.

By spamming this API flow, the malicious signer would be able to burn large amounts of funds at no cost.

As far as we tested, we did not encounter any rate limits from the Ethena API when generating multiple RFQs, nor did we hit any rate limits when submitting an order. However, we could not perform end-to-end testing because our benefactor is not whitelisted.

**Recommendation:** Consider asking Ethena to reject orders that are lower than a specified minimum amount. Furthermore, we consider that Ethena should rate limit the order submission.

**Maker:** Acknowledged. This is already enforced with the 50bps max slippage control which includes gas cost in slippage. Ethena has assured us they will activate it before we launch.

**Cantina Managed:** Acknowledged.

## 3.2 Informational

### 3.2.1 Documentation/comments issues

**Severity:** Informational

**Context:** See each case below

**Description:**

- The `Controller Functionality` in the README needs to be expanded as the functionality has expanded to `AAVE/Ethena/ERC4626` funds management.
- MainnetController.sol#L385: The `2m per block` reference is incorrect, currently the limits for USDC in the EthenaMinting are: 10.2m for minting and 2m for redeeming.

**Recommendation:** Consider fixing these comments.

**Maker:** Fixed in PR 70.

**Cantina Managed:** Verified.

### 3.2.2 Bad Ethena RFQs could lead to funds being intentionally lost by a compromised relayer

**Severity:** Informational

**Context:** Ethena API docs

**Description:** A compromised relayer can set an account they control as the delegated signer for Ethena API orders. Note that orders are submitted by Ethena and the relayer is *not* free to choose a price, it must be a response from a previous RFQ request (see "RFQ and order match" in the Ethena docs).

If the Ethena backend can be forced to return a bad RFQ, the relayer can submit the `mint` (`burn`) order to mint less USDe (redeem less USDC) to intentionally lose funds to Ethena.

**Recommendation:** The Ethena backend API is not public (to our knowledge) and is considered out-of-scope for this engagement. To further protect against compromised relayers, consider reaching a custom agreement of what orders the Ethena backend should accept for orders with the `order.benefactor = almProxy`.

**Maker:** Ethena has an agreement with us to not fill orders with over a 50bps slippage.

**Cantina Managed:** Acknowledged.

### 3.2.3 Compromised relayer can burn tokens by abusing 4626 vault rounding

**Severity:** Informational

**Context:** MainnetController.sol#L266

**Description:** Most EIP-4626 vaults round *against* the user, rounding down the shares received in a `deposit` and rounding down the assets redeemed in a `redeem`. A compromised relayer can use this rounding behavior to burn a tiny amount of the assets (the vault's share price rounded down) by depositing and withdrawing.

**Recommendation:** As each round of deposits and withdrawals only loses a tiny dust amount to the vault, we consider this scenario very unlikely to happen in practice as there is no economic incentive. The relayer would incur large gas costs and would not even directly receive the "burned" assets to (partially) compensate for it.

**Maker:** Acknowledged, will not implement changes as there is no clear economic incentive to perform this attack.

**Cantina Managed:** Acknowledged.

### 3.2.4 `sUSDe` needs to be added to 4626 withdraw rate limits if `sUSDe.cooldownDuration == 0`

**Severity:** Informational

**Context:** MainnetController.sol#L404-L433

**Description:** The `sUSDE` contract can function in two modes:

- Using a `cooldownDuration > 0`: Alm proxy withdrawals need to be unstaked first and are processed through the `cooldownAssetsSUSDe` followed by the `unstakeSUSDe` functions.
- Using no cooldown `cooldownDuration == 0`: `sUSDe` acts as a normal EIP-4626 contract and Alm proxy withdrawals need to be processed by calling `withdrawERC4626`.

**Recommendation:** Ensure a `LIMIT_4626_WITHDRAW` rate limit is also added for `sUSDe`.

**Maker:** Acknowledged. The `susde` address will be added to all 4626 related rate limits. We assume the cooldown will never be zero so no need to add sUSDe withdrawal rate limit. Ethena will inform us in advance if for some reason this happens.

**Cantina Managed:** Acknowledged.

### 3.2.5 Use same SUSDE amount ⇔ shares conversion in `cooldownSharesSUSDe`

**Severity:** Informational

**Context:** MainnetController.sol#L419

**Description:** The `cooldownSharesSUSDe` function uses the `convertToAssets` function to convert the shares to the asset amount to be rate-limited. The actual `susde.cooldownShares` function uses `previewRedeem`.

```
function cooldownShares(uint256 shares) external ensureCooldownOn returns (uint256 assets) {
  if (shares > maxRedeem(msg.sender)) revert ExcessiveRedeemAmount();

  assets = previewRedeem(shares);

  cooldowns[msg.sender].cooldownEnd = uint104(block.timestamp) + cooldownDuration;
  cooldowns[msg.sender].underlyingAmount += uint152(assets);

  _withdraw(msg.sender, address(silo), msg.sender, assets, shares);
}
```

For EIP-4626, these can use different conversions as `previewRedeem` can factor in slippage:

> Note that any unfavorable discrepancy between convertToAssets and previewRedeem SHOULD be considered slippage in share price or some other type of condition, meaning the depositor will lose assets by redeeming. EIP-4626

Note that for sUSDe there is no difference between these functions and using `convertToAssets` instead of `previewRedeem` has no impact.

**Recommendation:** To be more gas efficient and as a best practice, consider using the return value of the `cooldownSharesSUSDe` for the rate limit.

**Maker:** Fixed in PR 69.

**Cantina Managed:** Fixed.

### 3.2.6 `prepareUSDeMint` resets allowances but burns rate limit

**Severity:** Informational

**Context:** MainnetController.sol#L386

**Description:** The `prepareUSDeMint` sets a new allowance overwriting the existing one. The previous allowance is lost but still contributed to reducing the rate limit. The approvals are not cumulative and the rate limit does therefore not directly reflect the USDe that can be minted. However, it correctly works as an upperbound.

**Recommendation:** To maximize USDe minting ensure:

- Use up any existing approval to mint USDe before calling `prepareUSDeMint` again.
- A compromised relayer can can burn through the rate limit without essentially setting an approval. The rate limits might need to be adjusted after freezing.

**Maker:** Acknowledged, we want to be able to set the approval specifically so we can revoke it/reduce it if needed. Avoiding unintended approval amounts will be managed offchain by querying the current allowance.

**Cantina Managed:** Acknowledged.