Maker Dao - Proxy & Helpers

Security Review

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

November 24, 2023

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 Medium Risk . . . 4
  - 3.1.1 Proxy does not support `payable` functions in its implementation . . . 4
- 3.2 Low Risk . . . 4
  - 3.2.1 Uncallable functions on implementation contract . . . 4
  - 3.2.2 The `implementation` must be trusted . . . 4
- 3.3 Informational . . . 5
  - 3.3.1 NatSpec inaccuracies for upgradeable proxy interfaces . . . 5
  - 3.3.2 Prefer `encodeCall` to `encodeWithSelector` . . . 5

4 Additional Comments **6**

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.

On Nov 3rd the Cantina team conducted a review of erc20-helpers and upgradeable-proxy on commit hashes 8fe5ef3e and 885de5b4 respectively. The team identified a total of **5** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 1
- Low Risk: 2
- Gas Optimizations: 0
- Informational: 2

3 Findings

3.1 Medium Risk

3.1.1 Proxy does not support `payable` functions in its implementation

Severity: Medium Risk

Context: UpgradeableProxy.sol#L37

Description: The `UpgradeableProxy.fallback` function that forwards the calls to the implementation contract is not defined as `payable`. If the implementation contract defines any functions that require ETH to be sent, they will be inaccessible as the `UpgradeableProxy.fallback` will revert.

Recommendation: Consider adding the `payable` keyword to the `UpgradeableProxy`'s `fallback` function.

```
- fallback() external {
+ fallback() external payable {
```

Maker: Acknowledged. After internal discussion, it was decided that we won't use payable functionality, so this won't be added.

Cantina: Acknowledged.

3.2 Low Risk

3.2.1 Uncallable functions on implementation contract

Severity: Low Risk

Context: UpgradeableProxy.sol#L37

Description: The `UpgradeableProxy` only forwards calls from its `fallback` function to the implementation contract. However, it defines the following three public functions besides its fallback:

- `function deny(address usr) external;`
- `function rely(address usr) external;`
- `function setImplementation(address implementation_) external;`

If the implementation contract defines a function with the same signature as one of these, it will be unable to be called through the proxy.

Recommendation: If these restrictions are acceptable, check that the implementation does not define one of these functions itself. Alternatively, the functions could be moved to the implementation contract (or `UpgradeableProxied`) as virtual functions.

Maker: Acknowledged, will keep current functionality and ensure no collisions through adequate testing.

Cantina: Acknowledged.

3.2.2 The `implementation` must be trusted

Severity: Low Risk

Context: UpgradeableProxy.sol#L8

Description: The `implementation` variable is used to `delegatecall` the calls received by the proxy to the implementation. This pattern is a pretty common pattern used in upgradeable contracts, where a proxy contract (usually called a storage) uses an implementation (usually called logic) to perform certain actions.

Because the `implementation` variable can be changed at any point in time, via the `auth` call `setImplementation`, we recommend to carefully review every new implementation before calling this function. A malicious implementation can perform dangerous actions like:

- Modifying `slot0` to a new value, basically replacing the `implementation` from a call that is not `auth`ed.
- Calling `selfdestruct` rendering the proxy unusable.
- Modifying `wards` slots, relying/denying targets without `auth`ed consent.

Recommendation: We recommend for every new implementation version, a security review to be performed, making them as trustable as possible to be used within the proxy.

Maker: Acknowledged. This is standard practice we will make sure it's done always.

Cantina: Acknowledged.

3.3 Informational

3.3.1 NatSpec inaccuracies for upgradeable proxy interfaces

Severity: Informational

Context: IUpgradeableProxy.sol#L11, IUpgradeableProxy.sol#L17, IUpgradeableProxied.sol#L7, IUpgradeableProxy.sol#L39

Description: The NatSpec documents the following:

- IUpgradeableProxy.sol#L11, IUpgradeableProxy.sol#L17: "removed/added from the Conduit." The upgradeable proxy repo does not specifically talk about conduits anywhere else and can act as a more general proxy for other contracts besides conduits. Consider changing it to "removed/added from the proxy".
- IUpgradeableProxied.sol#L7, IUpgradeableProxy.sol#L39: "Returns a 0 or 1 depending on if the user has been added as an admin." It does not state what value of 0 and 1 indicates admin status. While the default boolean interpretation is correct here, consider properly specifying it in the documentation.

Recommendation: Consider applying the mentioned recommendations.

Maker: Fixed in PR 4.

Cantina: Fixed.

3.3.2 Prefer `encodeCall` to `encodeWithSelector`

Severity: Informational

Context: SafeERC20.sol#L17, SafeERC20.sol#28, SafeERC20.sol#39, SafeERC20.sol#44

Description: The contract performs low-level calls to an `IERC20` contract. It is using `abi.encodeWithSelector` to encode the parameters.

Recommendation: Consider using `abi.encodeCall` as it type-checks that the parameters are indeed the correct types defined as the arguments for the `IERC20` functions. Wrong or missing argument types can be caught already during compile time, whereas `encodeWithSelector` does not give this guarantee.

```
// example for IERC20.transfer
- abi.encodeWithSelector(IERC20.transfer.selector, to, amount)
+ abi.encodeCall(IERC20.transfer, (to, amount))
```

Maker: Fixed in PR 4.

Cantina: Fixed.

4 Additional Comments

The Cantina team reviewed MakerDao's erc20-helpers and upgradeable-proxy changes holistically on commit hashes 9cf80a0d21692a4442c38bf2cb76c866ee3b7d85 and 48ef24898e5dc21ff17e86c56189d188fbd421fd respectively, and determined that all issues were resolved and no new issues were identified.