

Code Assessment of the OP Farms Smart Contracts

September 12, 2024

Produced for



by



Contents

1	Executive Summary	3
2	Assessment Overview	5
3	Limitations and use of report	7
4	Terminology	8
5	Findings	9
6	Notes	10

1 Executive Summary

Dear all,

Thank you for trusting us to help MakerDAO with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of OP Farms according to [Scope](#) to support you in forming an opinion on their security risks.

MakerDAO implements a mechanism to distribute rewards originating from a source on Ethereum L1 to a farm contract on OP Stack L2s.

The most critical subjects covered in our audit are functional correctness, asset solvency and cross-chain messaging. Security regarding all the aforementioned subjects is high.

The general subjects covered are code complexity and specification.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical -Severity Findings	0
High -Severity Findings	0
Medium -Severity Findings	0
Low -Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the OP Farms repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	10 Sep 2024	6ce7a1ee7dc11690cbd8309cb2e58106eb354b8f	Initial Version

For the solidity smart contracts, the compiler version 0.8.21 was chosen.

The contracts below were in scope:

```
./src/L1FarmProxy.sol
./src/L2FarmProxy.sol

./deploy/FarmProxyDeploy.sol
./deploy/FarmProxyInit.sol
./deploy/L2FarmProxySpell.sol
```

2.1.1 Excluded from scope

Generally, all files not mentioned above are out of scope. The configuration is out of scope. The OP token bridge and the OP stack's cross chain messaging mechanism is out of scope. The OP token bridge is part of another review.

2.2 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the [Assessment Overview](#).

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

MakerDAO implements a mechanism to distribute rewards originating from a source on Ethereum L1 to a farm contract on OP Stack L2s.

2.2.1 L1FarmProxy

Proxy on L1 Ethereum to receive and forward the reward tokens. Anyone can permissionlessly trigger the transfer of reward tokens to L2 using the public `notifyRewardAmount` function, provided the amount to be transferred exceeds the minimum threshold. The token transfer to L2 makes use of the OP token bridge.

A source transfers the reward to be forwarded to this contract, which is assumed to be a `VestedRewardsDistribution` that receives funds from a `DSSVest` instance that releases tokens over time.

The contract features `recover` to allow retrieving funds by privileged addresses (`wards`) that can be set with the standard `rely / deny` functionality. Further, the authorized addresses can set the `minGasLimit` and `rewardThreshold` with `file`.

The `rewardsToken`, `remoteToken`, `l2Proxy` as well as the `l1Bridge` are set as immutables in the constructor.

2.2.2 *L2FarmProxy*

Receiver of the bridged tokens on L2. Implements `forwardReward`, allowing anyone to push the reward tokens to the farm contract. The farm contract and its corresponding `rewardsToken` are stored as immutables. This function can be executed permissionlessly if the token amount transferred exceeds a set threshold. Additionally, the contract features a `recover` function that works similarly to the one of the L1 farm proxy.

Further, `file` allows for setting the `rewardThreshold` by authorized addresses that can be set with the standard `rely / deny` functionality.

2.2.3 *Deployment Scripts*

FarmProxyDeploy: A library that implements functions to deploy the contracts `L1Proxy` (on L1), `L2Proxy` (on L2) and `L2ProxySpell` (on L2). For contracts with authorized roles (`wards`), specifically the proxies, the ownership is transferred from the deployer to the specified owner address.

FarmProxyInit: Based on the parameters passed by governance, the `init` function performs sanity checks and sets up a `DSSVest` instance. Note that this includes creating a vesting schedule and setting up the `VestedRewardsDistribution` instance. Namely, that requires restricting claiming for the vested tokens. Further, it configures and initializes the `L1Proxy` with `minGasLimit` and `rewardsThreshold`, as well as the `L2Proxy` using a cross-chain message that triggers `L2FarmProxySpell.init()`. Finally, it updates the chainlog with the new L1 addresses.

L2FarmProxySpell: Performs sanity checks on the state and immutables of the `L2Proxy` based on the arguments passed from the cross-chain message call from L1 by the `FarmProxyInit` spell. It sets the `rewardsThreshold` and activates the rewards distribution.

2.2.4 *Trust Model & Roles*

The OP token bridge is fully trusted.

Authorized addresses (`wards`): Fully trusted. Can change parameters of the contracts and add/remove authorized addresses. The configuration is assumed to be correct; incorrect parameters are known to cause problems. For example on L2, the `rewardThreshold` must be large enough to avoid issues when recalculating the reward rate. Operational issues may arise when `minGasLimit` is set incorrectly. All parameters are expected to be set appropriately at all times.

Users/Keepers: untrusted. Assumed to trigger the actions as necessary.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe our findings. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical -Severity Findings	0
High -Severity Findings	0
Medium -Severity Findings	0
Low -Severity Findings	0

6 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

6.1 Deployment Verification

Note Version 1

Since deployment of the contracts is not performed by the governance directly, special care has to be taken that all contracts have been deployed correctly. While some variables can be checked upon initialization through the `PauseProxy`, some things have to be checked beforehand.

We therefore assume that the initcode, bytecode, traces and storage (e.g. mappings) are checked for unintended entries, calls or similar. This is especially crucial for any value stored in a mapping array or similar (e.g. could break access control).