PUBLIC

Code Assessment

of the Multiply FMM extension Smart Contracts

October 20, 2021

Produced for

by

Contents

1 Executive Summary 3
2 Assessment Overview 4
3 Limitations and use of report 8
4 Terminology 9
5 Findings 10
6 Resolved Findings 11

1 Executive Summary

Dear Sir or Madam,

First and foremost we would like to thank Oazo Apps Limited for giving us the opportunity to assess the current state of their Multiply FMM extension system. This document outlines the findings, limitations, and methodology of our assessment.

We hope that this assessment provides more insight into the current implementation and provides valuable findings. We are happy to receive questions and feedback to improve our service and are highly committed to further support your project.

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	2
• Code Corrected	2
Low-Severity Findings	1
• Risk Accepted	1

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 `MultiplyProxyActions.sol`, `Exchange.sol` and `ExchangeData.sol` inside the Multiply FMM extension repository based on the documentation files. This assessment targeted only the changes between the last reviewed commit of previous engagement (`10cfe4f37e9c5b9c5456e967b967fa623e03f632`) and the commits listed in the table below.

The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	14 October 2021	ac939ddb362f43b4244a84806ad9cf547346f02c	Initial Commit
2	19 September 2021	e277ac1471a95138aaa93b39cf2c16c36c769740	Final Commit

For the solidity smart contracts, the compiler version `0.7.6` was chosen.

2.2 System Overview

This system overview describes the changes implemented in the initially received version (Version 1) of the contracts as defined in the Assessment Overview. For completeness, the second subsection contains the system overview of the referenced initial report.

2.2.1 Changes in the updated version

The flashloan provider was replaced to be the Maker Flash Mint Module. Now, instead of the AaveLendingPoolProvider's address, the address of the flashloan provider is directly passed. However, any address implementing EIP-3165 can be passed as an argument.

Furthermore, the flashloan provider remains fully trusted.

2.2.2 System Overview of previous report

Oazo provides a frontend for interacting with the Maker system which allows users to easily open a vault, deposit collateral and generate `DAI` backed by the locked collateral. Each user first deploys a `DSProxy` contract which is used to interact with the functionality provided. The proxy allows the user to execute code of the Oazo smart contracts aggregating functionality to perform certain actions wrapped in one transaction.

Like leverage trading that creates a larger position from a smaller investment amount, it is possible to use borrowed `DAI` from locked collateral to buy more collateral and use this collateral to borrow more `DAI`. By doing this repeatedly, long chains of exposure to the collateral can be generated. The new `MultiplyProxyActions` introduces the support for leverage actions while reducing the number of total transactions to one and the total number of transfers to the vault to one deposit by leveraging flash loans. Oazo's new `MultiplyProxyActions` contract contains functionality allowing users to easily increase and decrease the multiply factor and, thus, simplifies actions for creating, withdrawing and modifying leveraged positions. The core process can be described in three steps:

1. All values for the desired exposure / multiply factor are precomputed by the front end.

2. The respective function on the stateless smart contract is called and the parameters passed contain all necessary information.
3. The required large amount of collateral can be deposited all at once by leveraging flash loans. Therefore, `DAI` is generated only once. However, much more `DAI` will be available now but will be also needed to pay back the flash loan.

Effectively, gas costs are significantly reduced since there is no repetition of the traditional leverage actions.

The flash loan is always taken in `DAI` and swapped into collateral if needed. When reducing the multiply factor some collateral can be withdrawn from the vault. To repay the debt incurred by the flash loan, collateral is swapped into `DAI`. An exchange connector is deployed to support swapping `DAI` to collateral, and vice versa.

Following functions are provided by the smart contract:

Increasing Exposure

`openMultiplyVault` Opens a new vault for the user before calling `increaseMultipleDepositCollateral`.

`increaseMultipleDepositCollateral` Allows a user to increase the multiply factor on his vault while depositing collateral.

`increaseMultipleDepositDai` Allows a user to increase the multiply factor on his vault while depositing `DAI`.

`increaseMultiple` Allows a user to increase the multiply factor on his vault without depositing additional funds.

Decreasing Exposure

`decreaseMultiple`

Allows a user to decrease the multiply factor on his vault without withdrawing funds.

`decreaseMultipleWithdrawCollateral`

Allows a user to decrease the multiply factor on his vault while withdrawing collateral.

`decreaseMultipleWithdrawDai`

Allows a user to decrease the multiply factor on his vault while receiving the withdrawn collateral swapped into `DAI`.

`closeVaultExitCollateral`

Allows a user to close his vault (return all his debt) and withdraw the collateral.

`closeVaultExitDai`

Allows a user to close his vault (return all his debt) and receive the withdrawn collateral swapped into `DAI`.

Execution:

Changing the multiply factor on a user's vault of a certain `ilk` (collateral type) is done as follows:

Through his `DSProxy`, the user executes the appropriate function of the `MultiplyProxyAction` contract. Note that this executes the code of the `MultiplyProxyAction` in the context of the user's `DSProxy` contract since it is being executed as `DELEGATECALL`. All data required for the actions on the vault and the exchange are precomputed off chain by the front-end and passed as function parameters.

For the Vault action, all information related to the collateralized debt position is passed:

- `address gemJoin`: The Token Adapter for this collateral in the Maker system
- `address payable fundsReceiver`: The address withdrawn funds are to be transferred to

- `uint256 cdpId`: The unique id of the collateralized debt position in the cdp manager. Note that vaults not opened through the CdpManager are not supported. If the vault opening action is called, then the ID is zero.
- `bytes32 ilk`: The identifier of the vault type for the collateral. When the execution enters the code of MultiplyProxyActions, this is set to zero. It is set in the smart contract.
- `uint256 requiredDebt`: Amount of `DAI` required / to be flashloaned
- `uint256 borrowCollateral`: Amount of collateral needed to perform the action on the vault (will be swapped from the flashloaned `DAI`)
- `withdrawCollateral`: Amount of collateral to be withdrawn from the vault
- `withdrawDai`: Amount of `DAI` to be withdrawn from the vault
- `depositDai`: Amount of `DAI` to be deposited to the vault
- `depositCollateral`: Amount of collateral to be deposited into the vault.

These values must be initialized as required, not all values are required for all actions.

For the token swaps, more parameters are required:

- `address fromTokenAddress`: Address of the source token, either `DAI` or the collateral
- `address toTokenAddress`: Address of the tokens the source token is to be exchanged into, either `DAI` or the collateral
- `uint256 fromTokenAmount`: Amount of the source token, in respective tokenwei
- `uint256 toTokenAmount`: Amount of the target token, in respective tokenwei
- `uint256 minToTokenAmount`: Minimum amount of the target token to be received after the exchange
- `address exchangeAddress`: Address of the exchange to be used
- `bytes _exchangeCalldata`: Calldata for the call to the exchange

Addresses to be interacted with are passed as an address registry. This struct contains following data:

- `address jug`: Contract of the Maker system for the stability fee
- `address manager`: `CDPManager` contract address
- `address multiplyProxyActions`: The `MultiplyProxyActions` contract address
- `address aaveLendingPoolProvider`: The AaveLendingPoolProvider is used to query the address of Aave's flash loan contract
- `address feeRecepient`: (unused) the fee recepient
- `address exchange`: The address of the exchange contract

If necessary, setup actions such as opening a new vault or calculating the debt to pay when closing a value is done. If required by the chosen action, funds are transferred from the `DSProxy` to the `MultiProxyAction` account. Next, allowance to operate on the vault is given to the `CDPManager` and the call for the flash loan is crafted and being called. The funds are made available and the callback to function `executeOperation()` in the `MultiplyProxyActions` contract is made. Note that at this stage, the execution is no longer in the context of the `DSProxy` which executed a `DELEGATECALL`, but is in the context of `MultiplyProxyActions` since the callback from the flash loan provider executes the code of `MultiplyProxyActions` as a normal call.

Depending on the chosen action, the multiply factor will be either increased or decreased. In case of an increase, more debt in the vault is created and `DAI` is generated and is used to pay back the flash loan debt while the remaining generated `DAI` is either swapped to collateral and transferred back or directly transferred back to the `fundsReceiver`. In case of a decrease action, depending on the action, the collateral received is either partially swapped so that the flash loan can be paid back and the remaining

collateral can be returned to the `fundsReceiver`, or completely swapped to pay off the flash loan debt and to return `DAI` to the `fundsReceiver`. Note that during token swaps, a fee is paid to a fee recipient.

Once the callback has finished and the flash loan is paid back, the context switches back to the `DSProxy`. If the flash loan has not been returned, the transaction will revert. Finally, the allowance given to the `CDPManager` is removed.

2.2.2.1 Roles & Trust Model

User: Untrusted, any user may interact with the `MultiplyProxyActions` contract to perform action on his own vault. Each user is responsible for the correctness of the input parameters passed to the function. The user may use the official frontend and trust it to aggregate all values correctly. That also includes trusting the third-party APIs used by the Oazo front-end (e.g., 1inch API generating swap request data).

CdpManager: Trusted, the `CDPManager` contract of Maker. During the action allowance is given to the `CDPManager` to manipulate the vault of the user.

Oazo: Owner of the smart contracts. Operates the Frontend used by most users to interact with the smart contract. The Frontend aggregates all values for the operation of the smart contract.

Exchange (system contract): Fully trusted. Called by the `MultiplyProxyActions` contract to facilitate the token swap via the external third-party exchange. Collects the fee for the system. Ensures that the minimum expected token amount has been returned by the external exchange.

Exchange (third party): `1inch` swap aggregator is to be used. Untrusted, with the exception that we assume it does not reenter into `MultiplyProxyAction`. The system exchange contract provides allowance for the source token and ensures that after the call a minimum amount of the tokens has been exchanged.

FlashLoan Provider: Fully trusted. Called from the proxy context after preparing the action, calls back into `MultiplyProxyActions.executeOperation()` passing the arguments to execute the operation. Fully trusted to do so honestly and correctly. Notably this includes that it does not reenter any other public function of the `MultiplyProxyActions` contract. In this version of the smart contract Aave v2 is used.

2.2.2.2 Changes in Version 2

- It's now possible to skip the flashloan in case no extra funds are required to execute the multiply action
- The top level function called when entering MultiplyProxyActions is now tracked and emitted in an Event

2.2.2.3 Changes in Version 3

- In the Exchange contract access control has been removed from the swap functions. This is necessary for the skipFL functionality to work.

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 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 associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. 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 the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than 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 severities. 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 any open findings. Findings that have been resolved, have been moved to the Resolved Findings section. All of the findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

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	1

- Unhandled Alternative Flashloan Providers Risk Accepted

5.1 Unhandled Alternative Flashloan Providers

Design Low Version 1 Risk Accepted

Currently the MultiplyProxyActions relies on the assumption that the lender will call `onFlashLoan()` and revert if the flashloan fails. However, EIP-3156 does not require `flashLoan()` to revert if it is unsuccessful. In any scenario where the flash loan fails but does not revert, the funds sent to the MultiplyProxyActions could remain in the contract (since it is possible that `onFlashLoan()` is not called) and can be accessed by anyone via direct call to the `onFlashLoan()`. In addition, only relying on the `bool` return value of `flashLoan()` is insufficient since returning `true` is also allowed in case of failures.

The DssFlashmit contract reverts if `onFlashLoan()` fails. However, using another flashloan EIP-3156 lender contract can potentially lock the funds of the users.

Risk accepted:

Oazo Apps Limited plans on using FMM solely.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

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	2

- Incorrect Balance Check for Vault Code Corrected
- Incorrect Conversion to 18 Decimals Code Corrected

Low-Severity Findings	0

6.1 Incorrect Balance Check for Vault

Correctness Medium Version 1 Code Corrected

Checking whether the MultiplyProxyActions contract holds enough funds has been modified to:

```
require(
  cdpData.requiredDebt.add(cdpData.depositDai) >= IERC20(DAI).balanceOf(address(this)),
  "requested and received amounts mismatch"
);
```

The check should ensure that the MultiplyProxyActions contract holds enough Dai for the operation on the vault. Thus, if less DAI than needed is available the code should revert while a surplus of DAI could be tolerated. However, the change proceeds with the execution if the balance is lower than the amount needed while it reverts if there is a surplus of DAI.

Code corrected:

The condition has been changed to <=.

6.2 Incorrect Conversion to 18 Decimals

Correctness Medium Version 1 Code Corrected

An additional call to `convertTo18()` has been added compared to the codebase from the previous report.

In `_closeWithdrawCollateralSkipFL()`, the following code is executed:

```
cdpData.withdrawCollateral = convertTo18(cdpData.gemJoin, cdpData.withdrawCollateral);

wipeAndFreeGem(
  addressRegistry.manager,
  cdpData.gemJoin,
```

```
  cdpData.cdpId,
  cdpData.requiredDebt,
  cdpData.withdrawCollateral
);
```

`wipeAndFreeGem()` contains the following code where `collateralDraw` is equal to `cdpData.withdrawCollateral`:

```
uint256 wadC = convertTo18(cdpData.gemJoin, collateralDraw);
```

Thus, `wadC` will be `withdrawCollateral*10**(18-gem.decimals())*10**(18-gem.decimals())`. Ultimately, that results in an incorrect amount.

Code corrected:

The conversion to 18 decimals has been removed in `_closeWithdrawCollateralSkipFL()`.