certora

Security Assessment

Compound V3 - Multiplier Plugin

November 2025

Prepared for Compound

Table of contents

Project Summary

Project Scope

Project Name	Repository	Commit Hashes	Platform
Compound Multiplier	Compound Multiplier	Initial: 75b6224 Final: 6bfa3c2	EVM

Project Overview

This document describes the manual code review of **Compound Multiplier Plugin**

The work was a 9 days effort undertaken between **11/2025** and **12/2025**

The following contract list is included in our scope:

- plugins/flashloan/*
- plugins/swap/* (OneInchV6Plugin excluded)
- CometFoundation.sol

During the manual audit, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.

Audit Goals

The audit scope concentrated on a new functionality plugin, designed for V3 markets, which facilitates the process of creating and managing leveraged exposure (referred as multiplication, also known as looping) as well as swapping collateral on existing loans by utilizing flash loans. The goals of the plugin are:

- **Introduce multiply, cover and exchange**
  - The functions essentially create leveraged exposure (multiply), reduce/manage said exposure (cover) and swap collateral token on an existing loan (exchange) while maintaining good health.
- **Provide a wide choice of swap/flash loan plugins**
  - A variety of swap and flash loan providers is introduced to give maximum flexibility of users paired with most optimal swap paths.

Coverage

- **Plugin validity is verified correctly**
  - Plugins are arbitrarily input and the main contract performs a `delegatecall` towards them which is dangerous, however proper authorization is performed.
- **Multiply, cover and exchange sanity checks**
  - All 3 methods alter positions in Comets which change health ratios too. This is taken into consideration with the introduction of `maxHealthFactorDrop`, validating that users can't (unwillingly) end up in worse health than what they started with

Remarks

The Compound Multiplier repo's contracts follow best security practices. The main contract, `CometFoundation.sol`, is technically almost stateless, does not hold any user funds or track balances - it just enables a one-click ability to take out a flash loan, adjust collateral to a comet and perform swaps. This greatly limits the attack surface. The in-code implementation matches the intended functionality. Some remarks:

- `CometFoundation` approval is not revoked automatically - currently poses no risk since all `Comet` interactions are performed on behalf of `msg.sender` so it's not possible for a third party to abuse the given authority, just a heads-up.
- During the kickoff, the researchers were informed that the support of weird tokens is expected and issues related to non-compliant ERC20s are welcomed. Currently the contract does not interact well with such tokens.
- Some plugins have certain functionalities and quirks which are currently unsupported. Their core functionality works correctly, however their full potential is not captured. All mentioned in the report.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	2	2	2
Low	3	3	3
Informational	4	4	2
Total	9	9	7

Severity Matrix

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

Detailed Findings

ID	Title	Severity	Status
M-01	Unspent srcToken is not returned to the user after swap	Medium	Fixed
M-02	Native ETH cannot be rescued	Medium	Fixed
L-01	Max health factor drop is not applied to the multiply function	Low	Fixed
L-02	Multiply function does not support fee-on-transfer tokens	Low	Fixed
L-03	Calls to smartSwapTo and smartSwapByOrderId revert due to an incorrect interface definition	Low	Fixed
I-01	Use of .transfer() may cause transaction to revert when sending ETH to the treasury	Informational	Fixed
I-02	Limited LiFi plugin functionality	Informational	Acknowledged
I-03	Immutable plugin mapping	Informational	Acknowledged
I-04	Collateralization factor is not checked during leverage calculations	Informational	Fixed

Medium Severity Issues

M-01 Unspent srcToken is not returned to the user after swap		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: LiFiPlugin.sol#L48-L55	Status: Fixed	

Description: When swapping using the `LiFiPlugin`, the function `GenericSwapFacetV3.swapTokensMultipleV3ERC20ToERC20` is called. In the case of positive slippage, any unspent `srcToken` is returned to the receiver, which in this case is the `CometFoundation` contract.

```
    function swapTokensMultipleV3ERC20ToNative(
        ...
   ) external {
        _depositMultipleERC20Tokens(_swapData);
@>          _executeSwaps(_swapData, _transactionId, _receiver);
        ...
   }
function _executeSwaps(
        LibSwap.SwapData[] calldata _swapData,
        bytes32 _transactionId,
        address _receiver
   ) private {
     ...
       if (sendingAssetId != receivingAssetId)
@>            _returnPositiveSlippageERC20(sendingAsset, _receiver);
   }
```

However, the `LiFiPlugin.swap` function does not forward the unspent `srcToken` to the user. As a result, any leftover funds remain in the `CometFoundation` contract and can only be recovered by the treasury, causing user funds to be lost.

Recommendations: In the `LiFiPlugin.swap` function, after the swap execution, transfer any remaining `srcToken` back to the user.

Customer response: Fixed in 4097622 and 60b300a. Unspent token is now returned to the user after swap.

Fix review: Fix confirmed.

M-02 Native ETH cannot be rescued		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: CometFoundation.sol#L111	Status: Fixed	

Description: The `CometFoundation.rescue` function is designed to recover funds stuck in the contract, including native ETH. However, attempting to rescue native ETH will always revert because it calls `balanceOf` on the zero address, preventing the treasury from retrieving these funds

```
function rescue(IERC20 token) external {
  _dust(treasury, token, IComet(address(0)), token.balanceOf(address(this)));
}
```

Recommendations: Use `address(this).balance` instead of `token.balanceOf(address(this))` when rescuing native ETH.

Customer response: Fixed in bf0290e

Fix review: Fix confirmed.

Low Severity Issues

L-01 Max health factor drop is not applied to the multiply function

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: CometFoundation.sol#L111 CometFoundation.sol#L438	Status: Fixed	

Description: The `CometFoundation.multiply` function applies leverage but has no slippage protection against a health factor drop. As a result, when using a flash loan provider that applies fees, the resulting fee can reduce the user's health factor beyond the expected by the leveraged position.

Recommendations: Add a `maxHealthFactorDrop` parameter to `multiply` function and revert if the resulting health factor after flash-loan fees falls below the allowed threshold.

Customer response: Fixed in a81e367 and 9f01ae1. `maxHealthFactorDrop` is added to the `multiply` function and used to limit the maximum possible leverage the user can execute.

Fix review: Fix confirmed.

L-02 Multiply function does not support fee-on-transfer tokens

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: CometFoundation.sol#L329	Status: Fixed	

Description: The `multiply` function in `CometFoundation` uses `safeTransferFrom` to pull the specified `collateralAmount`. However, when interacting with fee-on-transfer tokens, the actual amount received will be less than the requested amount due to transfer fees.

```
function _multiply() internal {
   ...
   else {
@>      collateral.safeTransferFrom(msg.sender, address(this), collateralAmount);
   }
   ...
}
```

The inflated `collateralAmount` causes the function `_supplyWithdraw` to revert, since `comet.supplyTo` attempts to supply more collateral than the `CometFoundation` contract actually holds.

Recommendations: Instead of assuming the full amount is received, set `collateralAmount` to the actual balance difference before and after the transfer.

Customer response: Fixed in cde3ad9

Fix review: Fix confirmed.

L-03 Calls to smartSwapTo and smartSwapByOrderId revert due to an incorrect interface definition

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: IOKX.sol#L78-L99	Status: Fixed	

Description: The OKX interface definitions for `smartSwapTo` and `smartSwapByOrderId` are missing the final parameter `extraData` that exists in the actual OKX Router contract. This mismatch causes different function selectors to be computed. When `OKXPlugin.swap` receives `swapData` with the correct router selectors, it fails to match them against the stored constants and reverts with `InvalidSelector`.

Recommendations: Update the interface and the decoding logic to include the missing parameter `extraData`.

Customer response: Fixed in 6bfa3c2

Fix review: Fix confirmed.

Informational Issues

I-01: Use of .transfer() may cause transaction to revert when sending ETH to the treasury

Description: The `CometFoundation._dust` function uses `.transfer()` to send native ETH to the treasury. Since `.transfer()` compiles to a low-level CALL with a fixed 2,300 gas stipend, it will revert if the treasury is a smart contract whose fallback function requires more than 2,300 gas to execute.

Recommendations: Use `.call` instead of `.transfer` to send native ETH to the treasury.

Customer response: Fixed in 60b300a

Fix Review: Fix confirmed.

I-02: Limited LiFi plugin functionality

Description: The LiFi plugin supports multiple token deposits within a single swap:

```
  function swapTokensMultipleV3ERC20ToNative(
      ...
      LibSwap.SwapData[] calldata _swapData
  ) external {
@>>   _depositMultipleERC20Tokens(_swapData);
      _executeSwaps(_swapData, _transactionId, _receiver);
      ...
  }
```

This functionality is unsupported in `CometFoundation` currently since the contract gives approval only of a single input token.

Recommendations: In the `LifiPlugin.swap` function, add approval for the router to spend the sending tokens when `requiresDeposit` is true.

Customer response: Acknowledged

I-03: Immutable plugin mapping

Description: Currently all plugins are initialized in the constructor and the contract lacks functions to adjust them in case of malfunction of either of them. This means if even one is sunset/address changes/malfunctions/etc. or if a new plugin is to be added, the contract must be re-deployed

Recommendations: Optional, as the current design eliminates the need for an authority to modify the contract.

Customer response: Acknowledged

I-04: Collateralization factor is not checked during leverage calculations

Description: During `multiply`, leverage calculations are performed without taking into consideration the borrowing power of the supplied funds. For instance, a user requesting 10x leverage on a 1k deposit would supply 10k collateral and take a 9k loan which is possible only for markets where CF is below 10k/9k = 1.(11), otherwise it would revert due to not meeting health calculations.

Recommendations: Enforce the invariant `(userDeposit + loanAmount) * collateralizationFactor >= loanAmoun`t.

Customer response: Fixed in a81e367

Fix Review: Fix confirmed. Max leverage calculation is now based on the borrow collateral factor.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.