Inverse Finance FiRM Audit

BROKEN WITH ❤ BY NOMOI

We reviewed the https://github.com/InverseFinance/FiRM repository at commit c1274c8.

The review started on *Monday, April 17, 2023*.

This report was updated on *Thursday, May 11, 2023*.

Introduction

We have conducted an audit of the FiRM protocol, developed by Inverse Finance, with the objective of providing an independent assessment of the project's smart contracts' security, code quality, and overall functionality. The FiRM protocol is a decentralized finance (DeFi) protocol that enables users to access fixed-rate loans, using a variety of assets as collateral.

The protocol is built around the concept of Markets, which allow users to deposit collateral and borrow DOLA, a stablecoin pegged to the US dollar. Each Market supports a specific type of collateral, with asset prices primarily sourced from Chainlink price feeds. For every Market, the protocol creates a unique escrow contract for each user, enabling them to participate in on-chain governance and other protocol interactions even when their tokens are deposited as collateral in FiRM.

Overall, the codebase is well-documented and easy to comprehend. However, there are several instances where additional safety measures could be implemented to enhance the code's resilience. Many best practices are not employed, such as adhering to the checks-effects-interactions pattern, using reentrancy guards, or utilizing "safe" token function calls. As a result, a significant amount of responsibility is delegated to the governance process, relying on the governance review of proposals to prevent the addition of new markets and escrows that could introduce avoidable vulnerabilities.

It's important to mention that a significant portion of this code has previously undergone a Code4rena audit contest, resulting in a few overlapping findings. We have only included select overlapping findings in our report after reviewing the team's responses during the contest and determining that certain issues still warrant attention and resolution.

The main recommendations provided in this report aim to enhance the protocol's security by preventing issues that arise from unexpected reentrancies and adopting a more secure approach when handling external protocol integrations.

Findings

1. `ConvexCurveEscrow` enables a potential reentrancy

IMPACT HIGH LIKELIHOOD MEDIUM

The `ConvexCurveEscrow.pay` function calls `CvxCrvStakingWrapper.withdraw`, which leaves the system open to a reentrancy that allows a user to fully withdraw their collateral even if they have an open debt position. This happens due to the external calls performed by the staking wrapper:

1. Attacker deposits some collateral.
2. Attacker borrows some DOLA.
3. Attacker calls the `Market.withdraw` function to withdraw part of her collateral.
4. The `Market` will use the `ConvexCurveEscrow.balance` function to compute the withdrawal limit, and then call the `ConvexCurveEscrow.pay` function to transfer tokens to the user.
5. The `ConvexCurveEscrow` contract will then call the `CvxCrvStakingWrapper.getReward` function.
6. The `CvxCrvStakingWrapper` contract will call: a. The `getReward` function of the `CvxCrvStaking` contract, which will in turn call the `getReward` function of all the `extraRewards` contracts. b. The `onRewardClaim` function of the `rewardHook` (if defined)
7. If any of these calls reach an attacker controlled contract, the attacker can reenter the `Market.withdraw` function, which will use the same withdrawal limit as step 4, as the escrow's balance has not been updated yet.

Even though we didn't find a direct way to exploit this issue with Convex's current configuration, this could change if a new reward contract is added or a reward hook is enabled. We consider this issue to be of medium likelihood as the Inverse Finance team has no control over the reward contracts or the reward hook in Convex's staking wrapper.

Note: Related to the issue "ERC777 reentrancy when withdrawing can be used to withdraw all collateral" reported by Code4rena.

Recommendation

Consider using reentrancy guards for all Market functions that perform external calls. Alternatively, for the specific case of `ConvexCurveEscrow` it is possible to use reentrancy guards for all functions that perform external calls such as `pay` and `onDeposit`.

***Update**: As of commit `2ff3c03`, this issue has been resolved by locally storing and updating the staked balance before any calls that could result in a reentrancy.*

2. `Market.withdraw` reentrancy can be used to withdraw more collateral than allowed

The `Market.withdraw` function is susceptible to potential reentrancy through external calls that might be present in the escrow's pay function. This vulnerability could be exploited to withdraw a user's entire collateral, even if they have an open borrow position, by executing a second withdrawal before the amount used to calculate the withdrawal limit is updated. This issue is a generalized version of the "ConvexCurveEscrow enables a potential reentrancy" finding mentioned earlier.

Note: This issue is related to the "ERC777 reentrancy when withdrawing can be used to withdraw all collateral" reported by Code4rena. We chose to include this overlapping issue in our report because we believe its scope is broader than previously identified: it not only involves token standards with callbacks but also any token or escrow integration that execute external calls or have upgradeable components. Additionally, we disagree with the team's comment justifying this behavior due to only accepting ERC20 compliant tokens, as the ERC20 specification does not prohibit tokens from performing external calls or having upgradeable components.

Recommendation

Consider using reentrancy guards in all functions that perform external calls to contracts that are not controlled by Inverse Finance.

3. Potential reentrancy in `INVEscrow`

A reentrancy issue, similar to the one enabled by `ConvexCurveEscrow` but much less likely, is present in the `INVEscrow` contract.

Before transferring the escrow tokens in the `INVEscrow.pay` function, the `xINV.redeemUnderlying` function is called. This external call eventually reaches the `xINV` 's `comptroller` contract, which has upgradeable components. In theory, some external call to an attacker controlled contract could be unintentionally introduced in the future, leading to vulnerabilities such as reentering the `Market.withdraw` function to withdraw more collateral.

Recommendation

Consider using reentrancy guards for all `Market` functions that perform external calls. Alternatively, for the specific case of `INVEscrow` it is possible to use reentrancy guards for all functions that perform external calls.

***Update**: As of commit `ddefe1d` , this issue has been resolved by by locally storing and updating the staked balance before any calls that could result in a reentrancy.*

4. Stale chainlink answers are accepted

The `Oracle.getFeedPrice` function only checks that the returned price is greater than zero. However, there are no checks to prevent using stale prices, which could happen due to issues with Chainlink, or if the price reaches the minimum price configured for that specific feed.

This was also reported by Code4rena and marked as fixed, but only the greater than zero check was implemented.

Recommendation

Consider always checking if the price returned by the Chainlink feed is recent enough.

***Update**: As of commit `e28dae4`, this issue has been resolved by introducing a check in the `BorrowController` which disables borrowing if the feed has't updated the price within the configured threshold.*

5. `onlyINVEscrow` restrictions are too loose

IMPACT MEDIUM LIKELIHOOD LOW

The `DbrDistributor.onlyINVEscrow` modifier is intended to grant access exclusively to escrows of specific markets, such as the `INVEscrow`. In order to achieve this, it checks if the `msg.sender` is a valid escrow by verifying that the associated market is registered in the `DBR` contract.

However, the current implementation fails to effectively limit access to specific markets, and could result in escrows from unintended markets calling these functions. The impact of this will completely depend on the escrow's implementation.

Recommendation

Consider keeping a local registry of the allowed markets in the `DbrDistributor` contract, managed by its `operator`. This will prevent unintended calls from escrows of other markets.

***Update**: As of commit `ac71fff`, this issue has been resolved by hardcoding the `INV` address and checking that it is equal to the market's collateral address.*

6. `BorrowController.onRepay` is not called on liquidation

IMPACT LOW LIKELIHOOD LOW

The `Market.repay` function calls the `DBR.repay` function and `BorrowController.repay` (if defined). On the other hand, the `Market.liquidate` function only calls `DBR.repay`. This could lead to reaching the the `BorrowController`'s daily limit even when loans have been repaid through liquidation.

Someone could also attempt to abuse this and DOS the borrowing functionality:

1. Monitor and wait for a transaction that increases the price of the collateral from the perspective of the oracle being used by the market.
2. Before said transaction, deposit and borrow as much DOLA as possible, ideally close to the daily borrow limit.
3. After the transaction, liquidate the position.

Note: this behavior is unlikely in practice because it would not be free for the attacker and the amount of collateral needed for the deposit might be huge depending on the daily borrow limit.

Recommendation

Consider calling `BorrowController.repay` from `Market.liquidate` the same way in which it is called in the `repay` function.

***Update**: As of commit `5f1f75a` , this issue has been resolved by calling `onRepay` in the `liquidate` function.*

7. Typos

ENHANCEMENT

The codebase contains several typos. A few examples:

- `raio` on `ConvexCurvePriceFeed` .
- `addres` on `BorrowController` here and here.
- `Fed contact` on `Fed` .

8. Remove TODO comment

ENHANCEMENT

Remove the TODO comment in `INVEscrow` .

9. Naming issues

ENHANCEMENT

Rename `onlyINVEscrow` to `onlyEscrow` , as it allows any sender as long as it is an escrow from a valid market.

Rename `threeCurveTokenBps` to `weight` , since that's the nomenclature used on the convex smart contract.

10. Missing error messages

ENHANCEMENT

Some of the `require` statements in following functions don't contain an error message:

- `DbrDistributor.setRewardRateConstraints`
- `DbrDistributor.setRewardRate`
- `Fed.expansion`
- `GOhmTokenEscrow.delegate`
- `GovTokenEscrow.delegate`
- `INVEscrow.delegate`

Consider including error strings in all `require` statements to improve the UX when interacting with these functions.

11. Missing events

ENHANCEMENT

The functions in the `DbrDistributor` contract don't emit events, which might make it difficult for off-chain services to monitor the contract.

The `Fed.takeProfit` function is missing an event that could be useful to, for example, show the profits on a dune dashboard.

Most of the contracts are missing events for important administrative functions that could be useful to increase awareness of critical changes.

12. Multiple storage reads

OPTIMIZATION

There are a few places in the codebase where a storage variable or mapping entry is read multiple times. For example:

- The `Oracle.getNormalizedPrice` function reads `feeds[token]` multiple times.
- The `Market.getEscrow` function reads `escrows[user]` multiple times.
- The `Market.borrowInternal` function reads `borrowController` multiple times.
- The `Market.repay` function reads `borrowController` multiple times.
- The `Market.forceReplenish` function reads `dbr` multiple times.

Recommendation

Consider storing these variable in memory to reduce the number of storage reads.

13. Variable is always 0

OPTIMIZATION

When calculating the `unsafeLiquidationIncentive` in the `Market`'s constructor, the `liquidationFeeBps` storage variable is used. However, this variable is not initialized previously so its value is 0.

Recommendation

Consider removing `liquidationFeeBps` from the calculation of `unsafeLiquidationIncentive` in the `Market`'s constructor.