1inch MoneyMarket Audit

December 2021

By CoinFabrik

Introduction

CoinFabrik was asked to audit the contracts for the 1inch's MoneyMarket project. A set of contracts allows an owner to deploy money markets for different tokens. A money market for a token allows users to lend and borrow these tokens, respectively receiving or being charged an interest.

First we will provide a summary of our discoveries and then we will show the details of our findings.

Scope

The contracts audited are from the https://github.com/1inch/money-market-protocol git repository. The audit is based on the commit `316f0769a4f85fcc678bfd0b02da1ff47870d160`.

The audited contracts are:

- BaseMarket.sol
- Debtoken.sol
- DebtokenDeployer.sol
- Formula.sol
- Lentoken.sol
- LentokenDeployer.sol
- MoneyMarket.sol
- MoneyMarketToken.sol
- libs/MovingValue.sol

Analyses

The following analyses were performed:

- Misuse of the different call methods
- Integer overflow errors
- Division by zero errors
- Outdated version of Solidity compiler
- Front running attacks

- Reentrancy attacks
- Misuse of block timestamps
- Softlock denial of service attacks
- Functions with excessive gas cost
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Failure to use a withdrawal pattern
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures

Findings and Fixes

ID	Title	Severity	Status
CR-01	Unrestricted Borrowing Through User Health Toggling	Critical	Not fixed
ME-01	Reentrancies in Lentoken	Medium	Not fixed
MI-01	Duplicated Code May Cause Inconsistent Liquidations	Minor	Not fixed
MI-02	The 18 Decimals Assumption May Fail	Minor	Not fixed
MI-03	Unchecked Token Remove Operations	Minor	Not fixed

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.

- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.
- **Enhancement:** These kinds of findings do not represent a security risk. They are best practices that we suggest to implement.

This classification is summarized in the following table:

SEVERITY	EXPLOITABLE	ROADBLOCK	TO BE FIXED
Critical	Yes	Yes	Immediately
Medium	In the near future	Yes	As soon as possible
Minor	Unlikely	No	Eventually
Enhancement	No	No	Eventually

Issues Found by Severity

Critical Severity Issues

CR-01 Unrestricted Borrowing Through User Health Toggling

The function `BaseMarket.isHealthy()`, which is used to validate that a user has a healthy balance, can be controlled by an arbitrary user.

This function is used in `MoneyMarket.borrow()` and `MoneyMarket.liquidate()` to validate that the message sender is healthy before allowing him to borrow, or respectively liquidate an arbitrary account. It is also used (via the modifier `MoneyMarketTomen.stayHealthy()`) to validate the message sender of `Lentoken.withdrawTo()` will stay healthy after withdrawing tokens (and burning lenTokens) to an arbitrary account. It is similarly used by `LenToken.borrow()`, `LenToken.flashBorrow()`, `LenToken.transfer()` and `LenToken.transferFrom()`.

Function `isHealthy()` can return user-controlled values via the contract's external function `MoneyMarket.toggleFlash()`.

```
function toggleFlash(address account, bool entry) external override {
    require(_lentokenExists(ILentoken(msg.sender)) ||
_debtokenExists(IDebtoken(msg.sender)), "1MM: Market do not exist");
    entry ? _flashCounter[account]++ : _flashCounter[account]--;
}
```

`flashCounter[account]>0`. Since the following function can be called by the user to increase `_flashCounter[]`

It follows that he can convert to health. So an arbitrary user can modify the value of `isHealthy(account)` for any account (and consequently that of `stayHealthy(account))`

Recommendation

Make the function `toggleFlash()` private.

Solution

Unresolved.

Medium Severity Issues

ME-01 Reentrancies in Lentoken

The functions `_borrowTo()`, `repayFor()`, `depositFor()` and `flashBorrow()` in the Lentoken contractare susceptible to reentrancy when calling `token.safeTransfer()` or `token.transfer()`. Notice that if the attacker controls the token contract, he can implement reentrancy attacks in the `transfer()` or `safeTransfer()` function respectively calling to`_borrowTo()`, `repayFor()`, `depositFor()`, `flashBorrow()` recursively. In the case of the first three functions, the attacker can use the reentrancy to skip the event emission, e.g., in `_borrowTo()` he would call `mint()`, `safeTransfer()` twice and emit only one `Borrowed()` event).

In the case of _flashBorrow() there's one point of reentrancy in `_borrow()` and a second point of reentrancy in `provideCollateralOrReturnBorrowed()`. This second point of reentrancy is more dangerous, since it could allow the attacker to call `toggleFlash()` once (to increase `_flashCounter[account]` without decreasing it, hence setting the account to healthy and then launching the attacks described in CR-01).

Recommendation

Use reentrancy guard or place reentrancy checks.

Minor Severity Issues

MI-01 Duplicated Code May Cause Inconsistent Liquidations

Functions `MoneyMarket.isHealty()` and `MoneyMarket().healthStatus()` perform the same calculation of the account health status, returning a boolean that is false if the user debts are bigger than a percentage of the collateral. This calculation could be refactored so code is not duplicated.

A more dangerous problem is that both functions can return different results, Ex.: `isHealty()` will always return true if the `_flashcounter[]` array is not zero, indicating the account is healthy, while `healthStatus()` will ignore `_flashcounter[]` and can return zero, indicating the account is unhealthy, thus returning inconsistent results.

Recommendation

Eliminate one of the functions, or refactor the code so duplicate calculations are performed in a different shared function.

MI-02 The 18 Decimals Assumption May Fail

Throughout the contracts, it is assumed that `IERC20(token).decimals() = 18`. While this is true now, it may change in the future, breaking assumptions.

Recommendation

The number of decimals (and constants such as `_ONE = 1e18`) should be defined during construction when `IERC20` is imported.

MI-03 Unchecked Token Remove Operations

The `MoneyMarket.removeDebtoken()` and `MoneyMarket.removeLentoken()` functions should check that the balance is zero before removing the tokens from the account. Currently, this is checked by the callers, but not the functions themselves.

Recommendation

This check can be refactored and moved into those functions so those operations are always checked.

Enhancements

EN-01 Division Before Multiplication in Lentoken._interest()

In the function `_interest(debLiq, liq, _computedDebtInterest)` of Lentoken.sol the following assignment is made.

```
x = x / liq * (_ONE - reserveRatio) / _ONE;
```

Note that the precision error would be reduced if division took place last as there is no chance of overflow.

EN-02 Uninitialized variable and no return statement in _totalDebt()

The internal function `MoneyMarket._totalDebt()` follows.

```
function _totalDebt(address account) internal view override returns(uint256 totalDebt) {
        address[] memory debts = _userDebts[account].items.get();
        for (uint i = 0; i < debts.length; i++) {
            IDebtoken debtoken = IDebtoken(debts[i]);
            totalDebt += debtoken.underlyingBalanceOf(account) *
_getPrice(debtoken.token());
        }
    }
```

Note that the variable `totalDebt` is not initialized and there is no return statement. While the code still works, we recommend doing both for code readability and ease of maintenance.

EN-03 safeMath Library No Longer Required

The library safeMath is included in several contracts (e.g., DebToken) but it is no longer necessary while using solidity 0.8.0 or above. We recommend that you remove the inclusion to save gas. We further note that none of its functions are used.

EN-04 Miscellaneous

- Check that `_reserveRatio M<=1` in LenMarket's constructor.
- In the contract `MoneyMarketToken` casting `moneyMarket` in the modifier `onlyMoneyMarket()` is unnecessary since `moneyMarket` already is an address by construction:

```
constructor(IMoneyMarket moneyMarket_, IERC20 token_) {
        moneyMarket = address(moneyMarket_);
        token = token_;
 }
```

- The value `MAX_PERCENT - MIN_PERCENT` is computed for each call of `Formula.getAPR()`. Replace this by a constant to save gas.
- The value `365e18 days` in the contract MovingValue should be set as a constant. Also, either adding documentation or replacing it with `365 days * 1e18` would improve code readability.

Other Considerations

Missing Docstrings and References

The reviewed contracts and functions lack documentation. Documentation makes it easier to read the code and helps reviewers to understand its intention. Consider documenting functions using the Ethereum Natural Specification Format (NatSpec).

Centralization

The contracts `BaseMarket.sol` and `MoneyMarket.sol` allow the contract owner to change the formula, reserve ratio and price oracle. Users of the MoneyMarket

contract need to trust the owner of the contracts because of this fact.

Conclusion

We found the contracts to be simple and straightforward. The code is not documented. We found a critical and a medium severity issue that require immediate fixing. We also found a few issues of lesser severity and proposed some enhancements that should be looked into.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the 1inch Money Market project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.