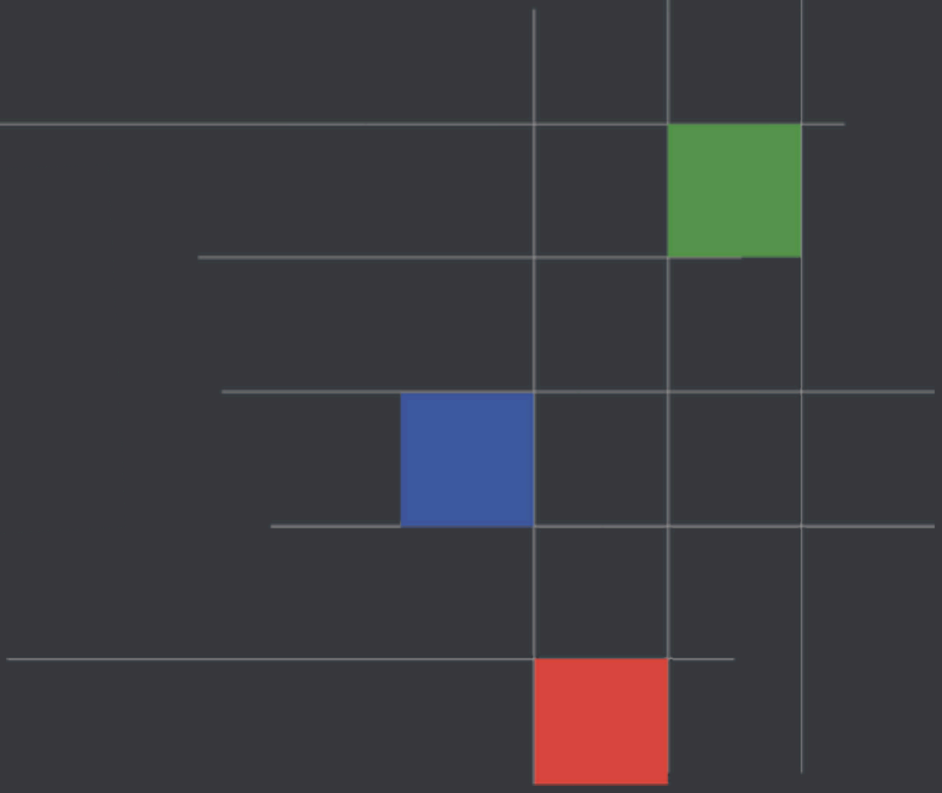
Askolend

Security Assessment

Apr 4th, 2021

For:

Askolend



Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Askolend	Askolend
Description	DeFi
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commit	a6a313a367585c8b83d246488e286afc553d521b 6464339501f781a76a94d496fbe38ddf83005550 d7f78f7b9b3304c703136c4b3f427c1ee543249a 90913acaa3807e80220c9ca954f196a3ea29518f 1385fe96b030cb29b136a44b6d792c98cbbe6dda 3c0aa8a85a966896ac7535ffc133e0348a332d8b 24e9354080212665639948 5cf0182f9f0f65b6e3

Audit Summary

Delivery Date	Apr. 4th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Mar. 19th, 2021 - Apr. 4th, 2021

Vulnerability Summary

Total Issues	29
Total Critical	3
Total Major	3
Total Medium	0
Total Minor	1
Total Informational	22
Total Discussion	0

Executive Summary

This report has been prepared for **Askolend** smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

There are a few external contracts invoked in current project:

`asset` , `interestRateModel` , `MMI` , `MMF` , `UOF` and `uniswapRouter` in contract **AskoRiskToken**;
`ARTF` , `Oracle` , `MMF` and `assets` in contract **MoneyMarketControl**;
`asset` , `AHR` , `ALR` , `MMF` , `UOF` and `ARTF` in contract **MoneyMarketInstance**;
`uniswap_router_add` , `wETH_add` , `factory` and `uniswapRouter` in **UniswapOracleFactory**;
`token0` , `token1` and `tokenA` in **UniswapOracleInstance**.

We assume these contracts are valid and non-vulnerable actors, and implementing proper logic to collaborate with current project.

We also assume all the imported libraries/contracts in the current project are valid and non-vulnerable actors, and implementing proper logic in current project.

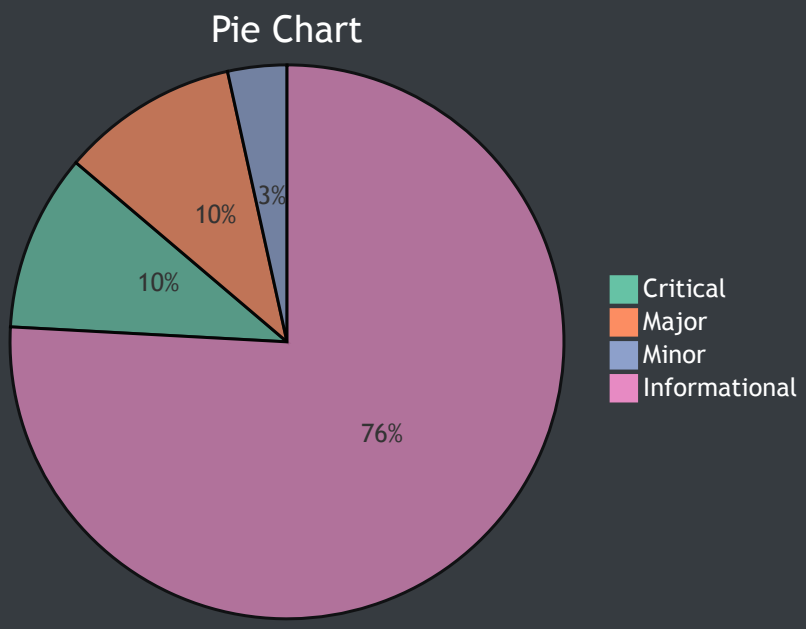
There are a few owner/admin only access functions introducing centralization risk:
`whitelistAsset` , `updateIRM` , `updateRR` , `upgradeOracle` , `upgradeMoneyMarketFactory` , `upgradeARTFactory` , `upgradeMMIOracle` and `changeColateRatio` in **MoneyMarketControl.sol**;
`upgradeMoneyMarketFactory` and `_changeColatRatio` in **MoneyMarketInstance.sol**.

We assume project would update the contract and call aforementioned functions with valid and proper parameters. Meanwhile to improve the trustworthiness of the project, any dynamic runtime updates in the project should be notified to the community. We recommend any plan to invoke aforementioned functions should be also considered to move to the execution queue of Timelock contract.

File in Scope

ID	Contract	SHA-256 Checksum
ART	AskoRiskToken.sol	de4ed303cef9ee2a6f5214477f0043a84aa7d273edb5baec86f2c55f996bed4a
MMC	MoneyMarketControl.sol	a79a1dea9f1146edffe5e945eaa0932c3a82780dae6a650cf424bbbf97591640
MMI	MoneyMarketInstance.sol	322f11d57ad110f310a8ed35ce76c771dc5b60536aaded7268e9253cda3e7a46
UOF	UniswapOracleFactory.sol	d63bdfb9d8eb7c454402f599af88d63dda2ab68a59c275ee8464cc4cf0ca6570
UOI	UniswapOracleInstance.sol	c248b40aee2057184a3c1d234a1f9ea602c2453de01a5de9e48514c5c1e3f097

Findings



ID	Title	Type	Severity	Resolved
ART-01	Unused Variables	Dead Code	Informational	✓
ART-02	Unused Event	Dead Code	Inforamtional	✓
ART-03	Missing Return Value Handling	Logical Issue	Minor	✓
ART-04	Immutable Approve Amount for Uniswap	Logical Issue	Informational	✓
ART-05	WET Code in Function `accrueInterest` I	Coding Style	Informational	✓

ART-06	WET Code in Function `accrueInterest` II	Coding Style	Informational	✓
ART-07	Missing Math Error Handling	Logical Issue	Critical	✓
ART-08	WET Code in Function `borrowBalanceCurrent`	Coding Style	Informational	✓
ART-09	Redundant Struct	Optimization	Informational	✓
ART-10	Missing Checks for Reentrancy	Logical Issue	Major	✓
ART-11	WET Access Control Code	Coding Style	Informational	✓
ART-12	Function Should Be Declared External	Optimization	Informational	✓
ART-13	Incorrect Math Error Handling	Logical Issue	Critical	✓
ART-14	Improper Use of `Storage`	Optimization	Informational	✓
MMC-01	Redundant Authorization Checking	Optimization	Informational	✓
MMC-02	Overwriting Storage Without Existence Checking	Logical Issue	Inforamtional	⚠
MMC-03	Functions Should Be Declared External	Optimization	Informational	✓
MMI-01	Redundant Authorization Checker	Logical Issue	Informational	✓
MMI-02	Missing zero check for `accountBorrowsAHR`	Logical Issue	Inforamtional	✓
MMI-03	Repay AHR Wrong Value	Logical Issue	Critical	✓
MMI-04	Uncertain Behavior from `safeTransferFrom`	Logical Issue	Informational	⚠
MMI-05	Functions Should Be Declared External	Optimization	Informational	✓
MMI-06	Missing Role Check for `liquidateAccount` Function	Logical Issue	Informational	✓

MMI-07	Mismatch Between Comment and Code	Logical Issue	Informational	✓
UOF-01	Functions Should Be Declared External	Optimization	Informational	✓
UOF-02	Missing Role Check for `linkMMI` Function	Logical Issue	Major	✓
UOF-03	Missing Event for Significant Transaction	Logical Issue	Informational	✓
UOI-01	Should Apply SafeMath	Mathematical Operations	Informational	✓
UOI-02	Should Set wETH Address as Constant	Logical Issue	Major	

ART-01: Unused Variables

Type	Severity	Location
Dead Code	Informational	AskoRiskToken.sol: L33, L47

Description:

Constant variables `one` and `nonCompliant` declared in the aforementioned lines are never used within the contract and can be safely omitted.

Recommendation:

We recommend omitting the variables that are never used in the aforementioned lines.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

ART-02: Unused Events

Type	Severity	Location
Dead Code	Informational	AskoRiskToken.sol: L85, L93

Description:

There is no function or transaction matching event `NonCompliantTimerStart` or `NonCompliantTimerReset` in the aforementioned lines and thus the events that are not emitted can be safely omitted.

Recommendation:

We recommend omitting the events that are not emitted in the aforementioned lines.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

ART-03: Missing Return Value Handling

Type	Severity	Location
Logical Issue	🔵 Minor	AskoRiskToken.sol: L131, L359, L442

Description:

`approve` and `transfer` are not void-returning functions per standard IERC20 interface. Ignoring the return value might cause some unexpected exceptions, especially if the callee function doesn't revert automatically when failing.

Recommendation:

We recommend checking the output of the aforementioned functions before continuing processing.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

ART-04: Immutable Approve Amount for Uniswap

Type	Severity	Location
Logical Issue	Informational	AskoRiskToken.sol: L131

Description:

Statement in aforementioned line:

```
    asset.approve(address(uniswapRouter), 100000000000000000000000000);
```

approves `100000000000000000000000000` asset to Uniswap, which cannot be increased anymore once the contract is deployed. Please make sure this is an intended design.

Alleviation:

The development team heeded our advice and resolved this issue in commit d7f78f7b9b3304c703136c4b3f427c1ee543249a.

ART-05: WET Code in Function `accrueInterest` I

Type	Severity	Location
Coding Style	Informational	AskoRiskToken.sol: L161-173

Description:

Code in the aforementioned lines calculates the current borrow interest rate, which has been implemented at function `borrowRatePerBlock`. To keep the code DRY, we suggest replacing the aforementioned lines with the function `borrowRatePerBlock`.

Recommendation:

We recommend modifying the aforementioned lines to

```
    if (accrualBlockNumberPrior != currentBlockNumber) {
        borrowRateMantissa = borrowRatePerBlock();
        ...
    }
```

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

ART-06: WET Code in Function `accrueInterest` II

Type	Severity	Location
Coding Style	Informational	AskoRiskToken.sol: L221-234

Description:

Event `InterestAccrued` is emitted in both `if` and `else` blocks. To simplify, `else` block can be removed and event can be emitted outside if block.

Recommendation

We recommend modifying the aforementioned lines to

```
        if (accrualBlockNumberPrior != currentBlockNumber) {
            ...
        }
        emit InterestAccrued(
                    accrualBlockNumber,
                    borrowIndex,
                    totalBorrows,
                    totalReserves
                );
```

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

ART-07: Missing Math Error Handling

Type	Severity	Location
Logical Issue	Critical	AskoRiskToken.sol: L180, L194, L199, L204, L210, L292, L298, L428, L433, L482, L487, L522, L538, L646, L665, L720, L726, L805, L813

Description:

Math operations in aforementioned lines apply `CarefulMath.sol`, which returns an error and a value `0` instead of reverting the transaction when an overflow happens. Therefore, `mathErr` at the aforementioned lines are expected to be handled properly to guarantee the correctness of calculation.

Recommendation:

We recommend checking the value of `mathErr` in the aforementioned lines, reverting upon failure, bofore any further processing.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

ART-08: WET Code in Function `borrowBalanceCurrent`

Type	Severity	Location
Coding Style	Informational	AskoRiskToken.sol: L282-303

Description:

Code in the aforementioned lines calculates the current exchange rate, which has been implemented at function `exchangeRatePrior`. To keep the code DRY, it is suggested to replace the aforementioned lines with the function `exchangeRatePrior`.

Recommendation:

We recommend replacing the aforementioned code with function `exchangeRatePrior`.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

ART-09: Redundant Struct

Type	Severity	Location
Optimization	Informational	AskoRiskToken.sol: L307

Description:

Struct `MintLocalVars` in the aforementioned line is used in function `mint` but only one of the variables `mintTokens` in the struct is used in the function. Per contract comments `struct used by mint to avoid stack too deep errors`, the author is trying to avoid stack error when using more than 16 local variables in one function, however, the number of local variables in `mint` function is far less than 16. It should be safe to use the plain variable instead of a nested struct.

Recommendation:

We recommend deleting the struct `MintLocalVars` and using `mintTokens` directly.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

ART-10: Missing Checks for Reentrancy

Type	Severity	Location
Logical Issue	Major	AskoRiskToken.sol: L369, L388

Description:

Function `burn` and `mintCollat` have state updates or event emits after external calls and thus are vulnerable to reentrancy attack.

Recommendation:

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation:

The development team heeded our advice and resolved this issue in commit in commit d7f78f7b9b3304c703136c4b3f427c1ee543249a

ART-11: WET Access Control Code

Type	Severity	Location
Coding Style	Informational	AskoRiskToken.sol: L371, L390, L574

Description:

The following check

```
        require(
               msg.sender == address(MMF),
               "msg.sender is not the Money Market Control contract"
           );
```

is performed multiple times within contract `AskoRiskToken`. Extracting the logic and creating a modifier to check if `msg.sender == address(MMF)` could contribute to code optimization.

Recommendation:

We recommend implementing a modifier to check if `msg.sender == address(MMF)` and applying the modifier when the access role is restricted to `address(MMF)`.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

ART-12: Function Should Be Declared External

Type	Severity	Location
Optimization	Informational	AskoRiskToken.sol: L503

Description:

Function `getwETHWorthOfART` which is never called internally within the contract should have external visibility.

Recommendation:

We recommend changing the visibility of `getwETHWorthOfART` to `external`.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

ART-13: Incorrect Math Error Handling

Type	Severity	Location
Logical Issue	Critical	AskoRiskToken.sol: L809-811, L817-819

Description:

The following check

```
        if (mathErr != MathError.NO_ERROR) {
               return (0);
           }
```

returns a value `0` rather than reverting the transaction, when an overflow happens in function `borrowBalancePrior`. If variables within the contract update the value based on the result of `borrowBalancePrior`, it may cause errors.

For example, when function `borrow` in line 416 is called, the function `borrowBalanceCurrent` calls function `borrowBalancePrior`, which would return 0 if an overflow happens inside, and thus `vars.accountBorrows` would be set as `0`. As a result, the record of the previous borrow amount will be `0` and the contract would suffer a heavy loss.

Recommendation:

We recommend adding math error checks in the aforementioned lines like

```
    require(mathErr == MathError.NO_ERROR);
```

before any further processing.

Alleviation:

The development team heeded our advice and resolved this issue in commit 6464339501f781a76a94d496fbe38ddf83005550.

ART-14: Improper Use of `Storage`

Type	Severity	Location
Optimization	Inforamtional	AskoRiskToken.sol: L795

Description:

Keyword `storage` is used in

```
    BorrowSnapshot storage borrowSnapshot = accountBorrows[account];
```

but the object `borrowSnapshot` is not mutated within the function `borrowBalancePrior` .

Recommendation:

We recommend modifying `storage` to `memory` .

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

MMC-01: Redundant Authorization Checking

Type	Severity	Location
Optimization	Informational	MoneyMarketControl.sol L226, L249

Description:

`require(isMMI[msg.sender] || isALR[msg.sender], "not a asko contract");` is redundant when the modifier `onlyMMI` already applies to the function.

Recommendation:

We recommend removing the redundant `require` checking in aforementioned lines.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

MMC-02: Overwriting Storage Without Existence Checking

Type	Severity	Location
Logical Issue	Informational	MoneyMarketControl.sol L126, L127, L128

Description:

In the function `whitelistAsset`, mappings `instanceTracker` and `oracleTracker` are overwritten, and array `assets` will add a new element, directly without any existence checking, even though original one may still being functional. Please make sure this is the intended design.

Alleviation:

(**Askolend Team - Response**)
This is the intended functionality of this function as each MMI is mapped to the asset it represents and this is a protected function. If something goes wrong with the whitelisting of an asset this functionality ensures that the asset can be re-whitelisted and that everything will still map correctly.

MMC-03: Functions Should Be Declared External

Type	Severity	Location
Optimization	Informational	MoneyMarketControl.sol

Description:

Functions which are never called internally within the contract should have external visibility. For example, `getAssets` and `checkCollateralizedALR`.

Recommendation:

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

MMI-01: Redundant Authorization Checker

Type	Severity	Location
Optimization	Informational	MoneyMarketInstance.sol L47

Description:

In this contract, modifiers `onlyMMFactory` and `onlyOwner` behave entirely the same, since `MMF` address is the owner address on contract construction. Please consider reusing the modifier `onlyOwner` if the ownership won't be changed later.

Recommendation:

Please consider reusing the modifier `onlyOwner` and removing the modifier `onlyMMFactory` if the ownership won't be changed later.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

MMI-02: Missing zero check for `accountBorrowsAHR`

Type	Severity	Location
Logical Issue	Informational	MoneyMarketInstance.sol L310-L311

Description:

Before repaying AHR, the contract should check if `accountBorrowsAHR` is zero, similar as the check done in line 302 for ALR.

Recommendation:

We recommend updatig L310-L311 as:

```
        if (accountBorrowsAHR != 0){
            payAmountAHR = AHR.repayBorrow(accountBorrowsAHR, msg.sender);
            asset.safeTransferFrom(msg.sender, address(AHR), accountBorrowsAHR);
        }
```

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

MMI-03: Repay AHR Wrong Value

Type	Severity	Location
Logical Issue	Critical	MoneyMarketInstance.sol L360-L361

Description:

When it reaches line 360, it means
`_repayAmount !=0 && accountBorrowsALR == 0`
and the next step should be user repaying to AHR. The amount to be repayed, should be `_repayAmount`, instead of `payAmountAHR` (which is 0) in line 360.

Recommendation:

We recommend replacing `payAmountAHR` in line 360 with `_repayAmount`.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

MMI-04: Uncertain Behavior from `safeTransferFrom`

Type	Severity	Location
Logical Issue	Informational	MoneyMarketInstance.sol L190, L203, L304, L311, L329, L341, L348, L360

Description:

Please make sure `safeTransferFrom` of the injection dependent contract `asset` would work as your expectation (including inflation/deflation, revert-upon-failure logic).

Alleviation:

(**Askolend Team Response**)
`SafeTransferFrom` was used here to handle non standard ERC20 contracts like the tether contract. These contracts aren't designed to work with inflation/deflation logic and we will need special contracts designed to handle these cases when they arise.

MMI-05: Functions Should Be Declared External

Type	Severity	Location
Optimization	Informational	MoneyMarketInstance.sol

Description:

Functions which are never called internally within the contract should have external visibility. For example, `getAssetAdd` and `checkIfALR`.

Recommendation:

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

MMI-06: Missing Role Check for `liquidateAccount` Function

Type	Severity	Location
Logical Issue	Informational	MoneyMarketInstance.sol L400

Description:

Function `liquidateAccount` will transfer “asset” leftover to `msg.sender`, and it can be called by anyone. We want to confirm if it is an intended design.

Recommendation:

We recommend adding the access control to the `liquidateAccount` function to prevent any unexpected loss.

Alleviation:

(**Askolend Team Response**)
`liquidateAccount` function is intended to be a public ally callable function.

MMI-07: Mismatch Between Comment and Code

Type	Severity	Location
Logical Issue	Informational	MoneyMarketInstance.sol L426-L433

Description:

Code in the aforemetioned lines requires `vars.collatValue` smaller than `vars.borrowedValuecollatRatio`:

```
    require(
              vars.collatValue < vars.borrowedValuecollatRatio,
              "Account is compliant cannot liquidate"
          );
```

However the comment is in the opposite way:

```
    /**
        need to check if the amount of collateral is less than borrowedValuecollatRatio of the
    borrowed amount
        if the collateral value is greater than or equal to borrowedValuecollatRatio of the borrowed
    value than we liquidate
    **/
```

Recommendation:

We recommend modifying the comment to

```
    /**
        need to check if the amount of collateral is less than borrowedValuecollatRatio of the
    borrowed amount
        if the collateral value is greater than or equal to borrowedValuecollatRatio of the borrowed
    value then we cannot liquidate
    **/
```

Alleviation:

The development team heeded our advice and resolved this issue in commit 24e9354080212665639948 5cf0182f9f0f65b6e3.

UOF-01: Functions Should Be Declared External

Type	Severity	Location
Optimization	Informational	UniswapOracleFactory.sol

Description:

Functions which are never called internally within the contract should have external visibility. For example, `createNewOracle`, `linkMMI`, `getUnderlyingPriceofAsset`, `viewUnderlyingPriceofAsset`, `getUnderlyingAssetPriceOfwETH` and `viewUnderlyingAssetPriceOfwETH`.

Recommendation:

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation:

The development team heeded our advice and resolved this issue in commit 90913acaa3807e80220c9ca954f196a3ea29518f.

UOF-02: Missing Role Check for `linkMMI` Function

Type	Severity	Location
Logical Issue	Major	UniswapOracleFactory.sol L63

Description:

Function `linkMMI` is designed to link a `MoneyMarketInstance` to its `oracle` in the oracle factory contract. Therefore, it should not be called by anyone since it mutates significant contract state.

Recommendation:

We recommend adding the access control to the `linkMMI` function to prevent any unexpected loss.

Alleviation:

The development team heeded our advice and resolved this issue in commit 24e9354080212665639948 5cf0182f9f0f65b6e3.

UOF-03: Missing Event for Significant Transaction

Type	Severity	Location
Optimization	Informational	UniswapOracleFactory.sol

Description:

Function `linkMMI` performs a significant role related to the state of `UniswapOracleFactory` cotract. Therefore, logging this action is highly recommended.

Recommendation:

We recommend emitting an event in `linkMMI` function to log the mutation of the contract state.

Alleviation:

The development team heeded our advice and resolved this issue in commit 24e9354080212665639948 5cf0182f9f0f65b6e3.

UOI-01: Should Apply SafeMath

Type	Severity	Location
Mathematical Operations	Informational	UniswapOracleInstance.sol L96, L101, L109, L114

Description:

Although per description from Uniswap Oracle module, the cumulative price should be monotonically increasing with time, we still recommend using SafeMath to handle the calculation since the data is coming from a third party resourse.

Recommendation:

We recommend using `sub` method from SafeMath for subtraction calculation.

Alleviation:

The development team heeded our advice and resolved this issue in commit d7f78f7b9b3304c703136c4b3f427c1ee543249a.

UOI-02: Should Set wETH Address as Constant

Type	Severity	Location
Logical Issue	Major	UniswapOracleInstance.sol L43

Description:

Per comments and logic implementation in the contract file, the token pair of the contract is `_tokenA` (asset token) and `_tokenB` (wETH token). Since wETH token address is a known certain address (0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2), `_tokenB` should not be a injection dependent variable. It should be a const defined in the contract. Otherwise the overall contract logic will not be valid if `_tokenB` is initialized with some other token address.

Recommendation:

We recommend removing the input `_tokenB` , but instead creating and using a predefind const storing wETH address for comparison.

Alleviation:

(**Askolend Team Response**)
These contracts are designed to be run on multiple chains including the Ethereum mainchain, the Binance smart chain, matic and xDAI meaning that the base currency used by the oracle will be different depending on which chain they are deployed to. Due to this I have opted to keep the logic the way it is in this regard as it means not having to rework the contracts for each chain.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

 : Issue resolved

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.