PROTOFIRE

SMART CONTRACT AUDIT REPORT

ROE Markets

Version: 2.0

Protofire
January, 2023

Table Of Contents

The ROE Markets audit report

The audit report was prepared for **ROE Markets** and includes following Github repositories and files (not all repository contracts):

1) https://github.com/RoeFinance/RoeMarkets audited commit 12b65d8b5568f09c47e54279a251300553b6aaee:

   /contracts/protocol/lendingpool/LendingPool.sol.0x13

2) https://github.com/ezoia-com/roe audited commit f7a8f873b21d24de5aa3cf1ea1b3a53c040f429a:

   /contracts/helper/HistoricalPriceConsumerV3.sol
   /contracts/helper/LPOracle.sol
   /contracts/helper/OracleConvert.sol
   /contracts/helper/ZapBox.sol
   /contracts/helper/LonggPositionManager.sol
   /contracts/helper/OptionsPositionManager.sol
   /contracts/helper/RangeManager.sol
   /contracts/helper/TokenisableRange.sol

Report Update - 1

The ROE Markets team has updated its code in accordance with the issues identified in the first review.
The updated code was checked at the following commits:

1) https://github.com/RoeFinance/RoeMarkets audited commit 4bb8a4b61ba78c5f3394226c96b3719fd6b3daec

   (LendingPool.sol.0x14)

3) https://github.com/ezoia-com/roe audited commit 0ff8877503340eeb64ad46b7e5ce84ff10a48be9:

   Added new file: RoeRouter.sol

Procedure

The audit includes the following procedures:

- code analysis by the Slither static analyzer, followed by manual analysis and selection of problems
- manual code analysis, including logic check
- checking the business logic for compliance with the documentation (or white paper)

In addition, during the audit, we also provide recommendations on optimizing smart contracts and using best practices.

Summary of audit findings

Severity	Count
HIGH	4
MEDIUM	9
LOW	26
INFORMATIONAL	8
TOTAL	**47**

Severity classification

High severity issues

High severity issues can lead to a direct or indirect loss of funds by both users and owners, a serious violation of the logic of the contract, including through the intervention of third parties. Such issues require immediate correction.

Medium severity issues

Medium severity issues may cause contract functionality to fail or behave incorrectly. Also, medium severity issues can cause financial damage. Such issues require increased attention.

Low severity issues

Low severity issues do not have a major security impact. It is recommended that such issues be taken into account.

Informational severity issues

These issues provide general guidelines for writing code as well as best practices.

Smart contracts HistoricalPriceConsumer [RM-01]

Contracts allow to receive and process historical prices from price oracles.

Update

After the update, this contract is no longer used.

ID: **RM01-01**	Severity: **Medium**	Status: **Not relevant**

Price-feed validation

1) Contracts use data from Chainlink price feeds. In this case, the validation of the received data is not carried out in any way.
   We recommend that you implement validation of the data received when calling priceFeed.latestRoundData().

2) The HistoricalPriceConsumerV3_FIXEDPRICE contract allows users to manually set prices. At the same time, the setPrice() function allows setting the value to zero.
   We recommend:
   a. add validation of input values;
   b. use this contract with caution, as it is not a full-fledged oracle and can only be used for testing purposes.

Update

After the update, this contract is no longer used.

ID: **RM01-02**	Severity: **Low**	Status: **Not relevant**

Gas saving

1) The functions getLatestPriceX1e6() (L151, L270, L311), getPriceAfterTimestamp() (L232, L304), getLatestPrice() (L300), could be declared with 'external' visibility type to save gas.

2) The variable `ratioQuote` (L179) can be declared as 'immutable' to save gas.

Update

After the update, this contract is no longer used.

ID: **RM01-03**	Severity: **Low**	Status: **Not relevant**

Code quality

a. Variable visibility

The variable 'ratioQuote' (L179) has default visibility. We recommend explicitly setting the visibility for variables.

b. Different reason messages for require statements

We recommend using different reason messages for require statements in the findBlockSamePhase() function (L257-258). This will make it easier to debug when testing and using the contract.

c. Events

Consider emitting events in the setOracle(), setPrice() functions.

d. Typos in code

We recommend fixing following typos in the code:

- L83, L171, 'ROUNDED';
- L149, L267 'normalizes'.

e. NatSpec documentation

We recommend writing NatSpec documentation for every function. This will help in identifying bugs during development, as well as making it easier for users to use the contract.

f. Non-fixed pragma

We recommend using the latest and fixed pragma version for each contract to prevent deployment with non-recommended compiler versions. You can use the '0.8.17' version for this contract.

Update

After the update, this contract is no longer used.

Smart contract LPOracle [RM-02]

The contract allows calculating the price of the LP token (uniswapV2 pair token).

The price is calculated based on the reserves of a pair of tokens and their separate value obtained from the oracle.

ID: **RM02-01**	Severity: **Low**	Status: **Fixed**

Gas saving

The variables 'decimalsA', 'decimalsB' (L22-23) can be declared as 'immutable' to save gas.

ID: **RM02-02**	Severity: **Low**	Status: **Fixed**

Parameter Validation

1) We recommend adding non-zero validation for the constructor() parameters to prevent incorrect initialization.

2) Since the formula on L95 uses decimals values, we recommend adding validation for the decimals values in the contract constructor:

```
require(decimalsA <= 18 && decimalsB <= 18, "Incorrect tokens");
```

ID: **RM02-03**	Severity: **Info**	Status: **Fixed**

Code quality

a. NatSpec documentation

We recommend writing NatSpec documentation for every function. This will help in identifying bugs during development, as well as making it easier for users to use the contract.

b. Non-fixed pragma

We recommend using the latest and fixed pragma version for each contract to prevent deployment with non-recommended compiler versions. You can use the '0.8.17' version for this contract.

Smart contract OracleConvert [RM-03]

The contract serves to calculate price using the middle asset, if there is no direct price feed.

Contract should have correct initialization. The 'clToken0' token denominator should be the same as the 'clToken1' numerator.

ID: **RM03-01**	Severity: **Low**	Status: **Fixed**

Parameter Validation

1) We recommend adding non-zero validation for the constructor() parameters to prevent incorrect initialization.

2) Since the formula on L47 uses decimals values, we recommend adding validation for the decimals values in the constructor:

```
require(CL_TOKENA.decimals() + CL_TOKENB.decimals() >= 18, "Decimals error message");
```

ID: **RM03-02**	Severity: **Info**	Status: **Fixed**

Code quality

a. NatSpec documentation

We recommend writing NatSpec documentation for every function. This will help in identifying bugs during development, as well as making it easier for users to use the contract.

b. Non-fixed pragma

We recommend using the latest and fixed pragma version for each contract to prevent deployment with non-recommended compiler versions. You can use the '0.8.17' version for this contract.

Smart contract ZapBox [RM-04]

An Uniswap v2 add/remove liquidity wrapper that also deposits/withdraw tokens from the lending pool.

ID: **RM04-01**	Severity: **Medium**	Status: **Fixed**

Dangerous Arguments

Consider using constant variables for **LendingPool** and **Router** instead of receiving as an argument, to prevent malicious contracts interacting directly with this one.

In zapInSingleAssetETH() function, if an attacker codes a malicious router, it will get max allowance.

ID: **RM04-02**	Severity: **Low**	Status: **Partially fixed**

Gas Saving

1) The following functions can be declared as external:
   - zapIn()
   - zapInETH()
   - zapInSingleAsset()
   - zapInSingleAssetETH()
   - zapOutWithPermit()

2) A constant variable can be used to define `2**256-1` instead of calculating the expression every time is needed.

Update
In the updated code we recommend declaring the `'ROEROUTER'` variable as immutable.

ID: **RM04-03**	Severity: **Info**	Status: **Fixed**

Code quality

a. NatSpec documentation

We recommend writing NatSpec documentation for every function. This will help in identifying bugs during development, as well as making it easier for users to use the contract.

b. nonReentrant modifier

As a security improvement, we recommend using the nonReentrant modifier to functions managing assets. But more important to the ones managing ETH directly.

c. Emitting event

No events are emitted except for the ERC20 transfers (zapIn/zapOut). It could be a good improvement adding events to main functions especially when storage is changed.

d. Non-fixed pragma

We recommend using the latest and fixed pragma version for each contract to prevent deployment with non-recommended compiler versions. You can use the '0.8.17' version for this contract.

Smart contract TokenisableRange [RM-05]

A tokenized range instance holds information about a range, i.e. lower/upper ticks, liquidity. TokenisableRange is designed to be a logic proxy, so the initialization code is held in the init function.

ID: **RM05-01**	Severity: **Medium**	Status: **Fixed**

Arguments manipulation (L219)

A malicious user can increase `n1` or `n2` parameters (and manipulate with one parameter) to decrease the `'feeLiquidity'` value in the deposit() function. It allows an increase of `lpAmt` value.

We recommend using `'added0'` and `'added1'` instead of `'n0'` and `'n1'`.

ID: **RM05-02**	Severity: **Low**	Status: **Fixed**

Gas Saving

1) The function getTokenAmounts() can be declared as 'external' to save gas.

2) L80-L94: We recommend using local variables for `upperTick` and `lowerTick` values before final assigning (to prevent multiple reading/writing to storage variables).

3) No need to update `fee0` and `fee1` variables of the claimFee() function in case when `newFee0==0` and `newFee1==0`. Thus, we recommend moving L154 before fee updating (to L149).

ID: **RM05-03**	Severity: **Low**	Status: **Fixed**

Validations

1) There is no validation for the incoming `_oracle` argument in the initProxy() function.

2) L289 Consider adding a validation for the case when `totalSupply==0`, to prevent failure on L292.

3) Consider adding an access validation to **init()** function.

ID: **RM05-04**	Severity: **Info**	Status: **Partially fixed**

Code quality

a. NatSpec documentation

We recommend writing NatSpec documentation for every function. This will help in identifying bugs during development, as well as making it easier for users to use the contract.

b. Outdated UniswapV3 library

As a security improvement, consider using the latest version of UniswapV3 library: lib/LiquidityAmounts.sol.

c. Commented code

We recommend removing commented code on L15. Commented code can signal that the development is not complete.

d. Hardcoded addresses

L115-L116, L193-L194: Consider using a constant variables, or an admin updatable variables for such addresses

e. Emitting event

No events are emitted except for the public functions. It could be a good improvement adding events to main functions especially when storage is changed.

f. Non-fixed pragma

We recommend using the latest and fixed pragma version for each contract to prevent deployment with non-recommended compiler versions. You can use the '0.8.17' version for this contract.

Update

Points c, d, e, f have been fixed in the updated code.
Point a: we recommend adding a description for the return statements for the returnExpectedBalance(), getValuePerLPAtPrice(), getTokenAmounts() functions.

Smart contract RangeManager [RM-06]

RangeManager is a TokenisableRange holder. A ranger instance is deployed for each Uniswap v3 pool supported, and holds TR, preventing overlap.

ID: **RM06-01**	Severity: **Medium**	Status: **Fixed**

Arguments manipulation (L72)

We recommend adding access restrictions for the generateRange() function, to prevent creating incorrect or unwanted ranges by other (malicious) users. Because if an invalid (or unwanted) ‘range’ is created by a malicious user, such “range” cannot be corrected.

ID: **RM06-02**	Severity: **Low**	Status: **Partially fixed**

Validations

1) Consider adding argument validation to incoming arguments in the constructor L47:
   `lendingPool != address(0)`.

   Also, to prevent incorrect calculations in the OptionsPositionManager.withdrawOptionAssets() it could be better to do ordering for tokens:
   `ASSET_0 = _asset0 < _asset1 ? _asset0 : _asset1`.

2) Consider adding validation to the `beacon` parameter of the generateRange() function on L72.

3) L124, L151, L157: Consider adding a validation for the step parameter since it's an array index.

Update
Points 1 and 2 have been fixed in the updated code.

ID: **RM06-03**	Severity: **Low**	Status: **Not fixed**

Gas Saving

The following variables can be declared as immutable:

- LENDING_POOL
- UNI_LP (if being used)
- ASSET_0
- ASSET_1

ID: **RM06-04**	Severity: **Info**	Status: **Fixed**

Code quality

a. NatSpec documentation

We recommend writing NatSpec documentation for every function. This will help in identifying bugs during development, as well as making it easier for users to use the contract.

b. Unused variables

L26,L35 variables `UNI_LP` and `V3_FACTORY` are never used in the contract source code.

c. Commented code

Remove commented code L85 to L92.

d. Unused Event

The event **ReducedPosition** is not being used in the contract source code.

e. Emitting events

No event is emitted in this contract. We recommend adding one on generateRange(). It could be a good improvement adding events to main functions especially when storage is changed.

f. Non-fixed pragma

We recommend using the latest and fixed pragma version for each contract to prevent deployment with non-recommended compiler versions. You can use the '0.8.17' version for this contract.

ID: **RM06-05**	Severity: **Info**	Status: **Found**

Leftovers after adding liquidity (L98)

Using the initRange() function the owner can initialize TokenizableRange contract.

The TokenizableRange.init() function returns remaining assets to `msg.sender` after adding liquidity (L152-153 TokenizableRange.sol). In this case, these leftovers will not be returned to the owner of the contract RangeManager, because such functionality is not implemented in the initRange() function.

Smart contract LonggPositionManager [RM-07]

The contract allows users to trade on margin (with leverage). Positions in such trading are opened using a flashloan.

The contract also allows liquidation or performing partial (soft) liquidation of positions with a weak health-factor.

LP-token is used as a trading asset (pair contract compatible with Uniswap dex).

ID: **RM07-01**	Severity: ~~High~~ **Low**	Status: **Fixed**

Withdrawing user funds with external (malicious) Router (L451)

At the beginning of the attack, the attacker must rebalance tokens A and B by swapping these tokens. (After the attack, it can be swapped back).

The attacker can call the softLiquidateLP() or liquidate() functions with following arguments:

`user` - address of the user who has an open position

`repayAmount` - at least 1 token

`ammRouter` - malicious router address

When closing (liquidating) a position, the rebalanceTokens() function will be called. This will allow an attacker to make a "fake exchange" with a malicious router contract and withdraw one of the user's assets.

Such an attack is possible, among other things, for the following reasons:

1) the 'debt' parameter is not checked in any way in the transferAndMint() function

2) the user's healthFactor may not exceed 1.05 by the end of the transaction.

Also, the healthFactor check on L364, L485 allows to complete the execution of functions with a health factor below 1.00, which will later lead to liquidation.

Recommendations:

1) Avoid the possibility of users using a custom router in all functions.

2) Add a check for the "debt" parameter of the transferAndMint() function.

3) make sure the health factor check (L364, 485) is working as planned.

4) Add more tests for the healthfactor (including integration tests). Because in the current tests, there is only one case with a healthFactor check and it is disabled! (/roe/tests/test_pm.py L183-184).

Issue update

After discussion with the developer, the severity of the issue was decreased. The risk of withdrawing funds is negligible. Nevertheless, the audit team insists on the implementation of recommendations.

ID: **RM07-02**	Severity: **High**	Status: **Fixed**

Fee manipulation (L463)

Any user (especially `TREASURY` address owner) can use the softLiquidate() function with a malicious provider address to provide an incorrect PriceOracle address. Such malicious PriceOracle can return a huge value to manipulate the `'feeAmount'` variable on L477. This can lead to withdrawing feeAmount from user collateral.

We recommend using a constant variable for the provider address.

ID: **RM07-03**	Severity: **High**	Status: **Fixed**

Blocking when closing/liquidating a position (L555-L556)

The function checkOracleValues() is used to check prices of the different assets. It uses following formula:

```
valueA = amountA * oracle.getAssetPrice(assetA) / 10**ERC20(assetA).decimals();
```

When using this formula, there may be situations when the 'amount' (for example, with partial close or liquidation) and the price of tokens are small, and decimals==18. That is, the product of amount and price will be equal to '1e18 - 1'. In this case, the formula will return zero. At the same time, for another asset, the product of quantity and price will be '1e18 + 1'. This will cause the function to stop (revert) at line 557 (or 558) and the user will not be able to close or liquidate the position.

Recommendation: It is not necessary to divide the product of 'amount' and 'price' by 'decimals'. It is enough to leave only multiplication, the meaning of this will not change. Or, you can divide by a significantly smaller number (eg 1e8 instead of decimals).

ID: **RM07-04**	Severity: **Low**	Status: **Fixed**

Zero result of the getTargetAmountFromOracle() (L185-L190)

The function getTargetAmountFromOracle() will return zero in cases when the assetA.decimals() is greater than the assetB.decimals(). (For example assetA is AAVE with 18 decimals, and assetB is USDT with 6 decimals.)

This issue will affect the executeOperationLeverage() function on L167, and allows 100% slippage. Also the function executeOperationLeverage() must be well tested, because there can be many situations where it is difficult to achieve 1% slippage.

ID: **RM07-05**	Severity: **Medium**	Status: **Fixed**

Function call restriction (L73)

The executeOperation() function can be executed by anyone. Thus, the attacker can execute this function in the same way as the removeDust() function. To do this, the user must specify the following parameters:

```
mode=0,
assetSold=tokenToWithdraw,
router=maliciousRouter,
amount=AmountToWithdraw
```

After calling the executeOperation() function, the nested function executeOperationLeverage() will be called. Further, when calling the checkSetAllowance() function (L163), the malicious router will receive allowance to spend tokens and will be able to transfer them to any account using a call to the fake swapExactTokensForTokens() function on L167.

Recommendation:

Add a restriction on the ability to call the executeOperation() function by any user and cover it with tests.

ID: **RM07-06**	Severity: **Low**	Status: **Fixed**

Gas savings

1) The `TREASURY` (L32) variable can be declared as 'immutable' to save gas.
2) The `HF_MAX` (L34) variable can be declared as 'constant'.
3) We recommend using constant variables instead of evaluating expressions:
   - L113-115, L135-136: `10**60`;
   - L726: `2**256-1`.
4) The `HF_THRESHOLD` (L33) variable is never used in the contract code. Consider removing it.
5) Remove unused imports on L12, 14, 17.
6) Consider removing redundant parameter `address(0x0)` on L229.
7) The functions closeAndWithdrawCollateral(), close(), deltaNeutralize() can be declared as 'external' to save gas
8) No need to check `uint` value for >= 0. Remove redundant check on L227 for the `amountSoldInPercent` parameter.

ID: **RM07-07**	Severity: **Low**	Status: **Fixed**

Closing position with another collateral token

It is possible that a user has `tokenC` as collateral, and at the same time opens a position by buying LP tokenAB (where `token0=tokenA`, `token1=tokenB`). If the price of an open position does not change (or worsens), then the user will not be able to close such a position. This will be due to the fact that interest coverage for the position cannot be paid with the `tokenC`.

According to the developer's comments, it follows that when using the front-end of the project, users will not be able to open positions with collateral in the form of a token that is not an integral part of the LP token.

But since the risks for users still remain (e.g. in case of failure of the front-end), we strongly recommend that NatSpec be supplemented with documentation explaining how users should open and close positions in such cases. In addition, such a case should be described in the documentation for the project.

ID: **RM07-08**	Severity: **Low**	Status: **Fixed**

Third-party application risk

There may be situations where the front-end of an application can be hacked (or, for example, not work). In such cases, an attacker can deliberately forge the address of the `ammRouter` and (or) `provider` (LendingPool) so that users cannot close their positions on their own. This could be to give the attacker the opportunity to earn on the liquidations of positions.

Thus we recommend defining constant variables for `Router`, `LendingPool` and `AddressProvider`.

Note. Third-party applications of the current project are not included in the audit scope.

ID: **RM07-09**	Severity: **Low**	Status: **Fixed**

Validation of oracle answer

Consider adding validation for the oracle price answer on L189 (getTargetAmountFromOracle() function).

At least non-zero check:

```
oracle.getAssetPrice(assetA) > 0;
oracle.getAssetPrice(assetB) > 0.
```

ID: **RM07-10**	Severity: **Low**	Status: **Not Fixed**

Incentives for soft-liquidators

There is no clear incentive for any users to call softLiquidation(), as the fees go to the Treasury contract.

Recommendations:

1) Consider transferring fee to caller (as incentive).
2) Provide documentation describing incentives for the soft-liquidators.

Update

Development team decided to do soft-liquidation by themself. It's up to them as this is a recommendation issue.

ID: **RM07-11**	Severity: **Low**	Status: **Fixed**

Code quality

a. Commented lines of code

We recommend removing commented code on L16, L654. Commented code can signal that the development is not complete.

b. Unused variables

The variables `ADDRESSES_PROVIDER` and `LENDING_POOL` (L21-22) are never used in the contract code. Consider using these values.

c. Code style

Remove 'spacebar' for the '`param. amtA`' variable on L94.

d. Function modifier

Add '`view`' modifier for the getTargetAmountFromOracle() function.

e. Typos in code

We recommend fixing following typos in the code:

- L376: 'to **g**et back' **?;**
- L647: 'ro**ut**er'.

f. NatSpec documentation

We recommend writing NatSpec documentation for every function, **input parameters and returning values**. This will help in identifying bugs during development, as well as making it easier for users to use the contract.

g. Non-fixed pragma

We recommend using the latest and fixed pragma version for each contract to prevent deployment with non-recommended compiler versions. You can use the '0.8.17' version for this contract.

Smart contract OptionsPositionManager [RM-08]

The contract allows users to trade on margin (with leverage). Positions in such trading are opened using a flashloan.

The contract also allows liquidation or performing partial liquidation of positions with a weak health-factor.

TokenisableRange tokens are used as a trading asset.

ID: **RM08-01**	Severity: ~~High~~ **Info**	Status: **Not relevant**

Buy Options problem (L90-102)

Using the buyOptions() function, the user can take multiple flash loans for different assets. When the executeOperation() function is called back, the withdrawOptionAssets() function will be called for all borrowed assets. This will allow getting the underlying assets for the 'OptionsPositionManager' contract.

But then only the assets of the first flash loan will be correctly processed and credited to the user's deposit in the Lending Pool.

Thus, all assets of the second and subsequent flashloans will remain on the 'OptionsPositionManager' contract. (This situation may well occur when the user has a large collateral or when flashloans are small).

The remaining assets can be easily withdrawn:

1) for the Treasury contract using the removeDust() functions

2) through an attack on the open for everyone function executeOperation(). The attacker will call the function with parameters: mode=0, provider=maliciousProvider. Such a malicious provider would allow the withdrawOptionAssets() function to be executed on behalf of the attacker, and further calls to cleanup() on lines 100 and 102 would deposit the assets to the address of the attacker.

Recommendations:

1) The function executeOperation() (mode=0) does not look correct. Perhaps the for loop should be extended to line 104. Please, recheck the intended logic.

2) We strongly recommend making it possible to call the executeOperation() function only for the LendingPool contract.

3) We also recommend using the global variable for the LendingPoolAddressesProvider contract in the withdrawOptionAssets(), buyOptions(), liquidate(), close(), closeDebt(), sellOptions() functions, instead of using it as function parameter.

Issue update

According to the developer team comment one LendingPool can only handle one pair of assets. All the options in one lending pool will be of the type of same assets (for example tokenA-TokenB) with various strikes. The 2 assets of the first flashloan asset are the same as for the following assets, that's why we only check those 2.

Thus, the problem was not confirmed and is not relevant for the current project.

However, the audit team suggests following the recommendations in points (2) and (3).

Also, consider using mapping for allowed options assets and validating their value when operating with option assets.

The severity of the issue has decreased after the update.

Issue update 2

All necessary recommendations were implemented by the development team.

ID: **RM08-02**	Severity: **High**	Status: **Fixed**

Malicious provider (L323, L365-368)

Since the 'provider' address can be specified when calling the closeDebt() function, this can be manipulated to increase the fee in the calculateAndSendFee() function. Because a malicious provider address can provide correct addresses for the LendingPool and malicious addresses for PriceOracles on L370-373.

We recommend defining the provider's address as a global contract variable and using it in all functions (where necessary).

ID: **RM08-03**	Severity: **Medium**	Status: **Fixed**

Function call restriction (L80)

The executeOperation() can be called by anyone.

We strongly recommend adding restrictions for every user (or contract) to call this function. Consider adding a modifier that only allows the LendingPool contract to call this function.

ID: **RM08-04**	Severity: ~~Med~~ **Low**	Status: **Not relevant**

Several options liquidation (L215, 241-248)

The liquidation of several options looks half-baked. The liquidate() function assumes the possibility of liquidating multiple options, since the function accepts an 'options' array. But at the same time, the reward for the liquidator will be received only for the first option.

We recommend that lines 241-248 be added to the for loop to enable rewards for all liquidated options.

Also, note that the liquidation functionality with multiple assets is not tested at all.

Issue update

According to the developer team comment one LendingPool can only handle one pair of assets. All the options in one lending pool will be of the type of same assets (for example tokenA-TokenB) with various strikes. The 2 assets of the first flashloan asset are the same as for the following assets, that's why we only check those 2.

Thus, the problem was not confirmed and is not relevant for the current project.

However, the audit team recommends following the testing recommendations.

The severity of the issue has decreased after the update.

ID: **RM08-05**	Severity: ~~Medium~~ **Low**	Status: **Not Fixed**

Importance of the liquidation (L354-355)

There may be a situation where liquidating a bad position is much more important than countering a sandwich attack. In this regard, the require conditions on L354-355 look quite complex and should be well tested with various values to prevent blocking on liquidations.

```
require(expected0 * token1Amount <= token0Amount * expected1 * 101 / 100;
require(expected0 * token1Amount >= token0Amount * expected1 * 99 / 100;
```

Otherwise, the boundaries must be expanded.

It should be noted that only one case of liquidation was covered by the tests, which, in our opinion, is extremely small for the trade-lending project.

Also, it should be noted that such a strict price check should be implemented in the functions for opening positions to prevent sandwich attacks.

Issue update

According to the developer team the PositionManagers rely on the oracle value, but if necessary a position can be liquidated by a direct call to the lending pool.

The severity of the issue was reduced. At the same time, we believe that the problem remains relevant when the user closes his position. Thus, we recommend adding more tests and considering expanding boundaries.

The bottom line for this issue is that we feel the requirement for the slippage is too tight. And in a context where the user needs to close a position due to a price change or something, we feel it is better to widen those requirements and not get a revert instead of tightening them up and losing the window the user has chosen to close its position. Either way, we feel it’s very important to test these conditions widely to avoid blocking a user.

ID: **RM08-06**	Severity: **Medium**	Status: **Fixed**

Amounts check (L468-469)

The '`amountsIn[0] > 0`' check is done twice on lines 468, 469. Check intended logic.

Remove the check in one place to save gas.

Also, consider adding following check instead, to prevent incorrect swapping:

```
amountsIn[0] <= ERC20(path[0]).balanceOf(address(this)).
```

ID: **RM08-07**	Severity: **Medium**	Status: **Fixed**

Traces of incomplete code

1) The contract contains 'TODO' comment in the code (L159), which may indicate that the functionality is incomplete.

   One way to prevent sandwich attack is to use slippage tolerance (%) for prices from oracles.

2) The test file /tests/test_pm_ranger.py contains 'TODO' comment on L276.

   Also, the mentioned test file contains a lot of commented lines of assertions.

   We strongly recommend:

   1) fixing test
   2) uncomment all assertions in tests
   3) writing more tests

ID: **RM08-08**	Severity: **Low**	Status: **Fixed**

Gas savings

1) The `TREASURY` (L35) and `SWAP_ROUTER_V2` (L36) variables can be declared as 'immutable' to save gas.
2) Remove unused imports on L13, 14, 15.
3) No need to create new local variables inside the for-loop on L91-93 for the `'asset'`, `'amount'` and `'target'` values to call withdrawOptionAssets() function (use `assets[k]`, `amounts[k]`, `sourceSwap[k]` instead).
4) The local variable `'amounts_'` is never used in the buyOptions(), liquidate() functions. Also, no need to create the same variable `'assets'` for the `'options'` parameter (L190-191, on L229-230).
5) The function swapAllTokens() (L435) is never used in contract code. Remove unused function.
6) We recommend using constant variables instead of evaluating expression on L502: `2**256-1`. You can use `type(uint256).max`.
7) The functions buyOptions(), close(), sellOptions() can be declared as 'external' to save gas.

ID: **RM08-09**	Severity: **Low**	Status: **Partially fixed**

Code quality

a. Commented lines of code

We recommend removing commented code on L17, L34, L104. Commented code can signal that the development is not complete.

b. Unused variables

The variables `ADDRESSES_PROVIDER` and `LENDING_POOL` (L22-23) are never used in the contract code. Consider using these values.

c. Variable visibility

The variable `'SWAP_ROUTER_V2'` (L35) has default visibility. We recommend explicitly setting the visibility for variables.

d. Variable naming

The variable name `'target'` on L93 is misleading since it serves as 'source' of the swap. Rename it. Allso, we recommend correcting reason message on L153 in the same way.

e. Different reason messages for require statements

We recommend using different reason messages for require statements in the closeDebt() function on L354-355, because the current message is already used in the contract LonggPositionManager.

f. Typos in code

We recommend fixing following typos in the code:

- L113: ‘transfer**r**ed’;
- L175: ‘serie’ ?;
- L176: ‘lpop’ ?;
- L386: ‘repres**e**nting’

g. NatSpec documentation

We recommend writing NatSpec documentation for every function, **input parameters and returning values**. This will help in identifying bugs during development, as well as making it easier for users to use the contract.

h. Non-fixed pragma

We recommend using the latest and fixed pragma version for each contract to prevent deployment with non-recommended compiler versions. You can use the ‘0.8.17’ version for this contract.

Issue update

In the updated code points a, b, c, d, e, and f were fixed.

The NatSpec documentation still lacks return values for the closeDebt(), withdrawOptionAssets() functions.

ID: **RM08-10**	Severity: **Low**	Status: **Fixed**

Order of mathematical operations (L374)

The calculations in the calculateAndSendFee() function on L374 allow division before multiplication. This may result in incorrect rounding.

Make sure the function works as intended for all valid values.

ID: **RM08-11**	Severity: **High**	Status: **Fixed**

Withdrawing collateral (L393)

Using the sellOptions() function, the user (attacker) can withdraw all or part of his collateral.

The attack can be made in the following way.

1. The attacker opens a position using the LonggPositionManager contract (for example, with the LP TokenBC). Due to this, the attacker will "receive" collateral in the form of a-TokenB and a-TokenC.

Note: Another way to open a position is to buy an option using OptionsPM.

2. The attacker calls the sellOptions() function with the argument `optionAddress=MaliciousContract`.

3. Such a Malicious contract will return the correct addresses of tokenB and tokenC on L401, 402.

4. Call to PMWithdraw() will withdraw funds to the OptionsPositionManager contract (L404-405).

5. On L407-408, the Malicious contract will receive allowance for transferring tokens.

6. On L409, tokens B and C will be withdrawn to a controlled address.

A similar problem (with the `'debtAsset'` parameter) is typical for the close() (and closeDebt()) function.

Recommendation: It is necessary to prevent the possibility of using an incorrect optionAddress argument. You may need to use a whitelist for tokens.

Smart contract LendingPool [RM-09]

A fork of the original LendingPool contract from the Aave V2 project.

Functionality has been added to the contract to interact with contracts LonggPositionManager and OptionsPositionManager.

ID: **RM09-01**	Severity: **Medium**	Status: **Fixed**

Updating ‘pm’ mapping (L438, 456)

The function PMSet() allows the admin to add a new address, which can call the PMTransfer() function. Since the PMTransfer() function allows to perform aToken transfers without Health-Factor validation, such a function should be well protected from attacks.

A situation may occur when the administrator of the LendingPool contract may behave dishonestly (for example, as a result of the loss of a private key or internal disagreements of the project owners). In this case, a malicious address can be added by the admin using the PMAssign() or PMSet() functions.

This will allow the malicious address to transfer the user's aTokens to its address when the user’s Health-Factor is below `softLiquidationThreshold` value.

In this regard, we recommend:

1) use a multisig wallet for the LandingPool administrator;

2) consider using only two addresses (Option and LongPM) to call the PMAssign() and PMSet() functions, and/or add new addresses to the mapping `‘pm’` only with a delay of at least 24 hours (using a Time-Lock contract so that users can make sure in the reliability of the new address).

At the same time, the functionality of disabling addresses can be simplified.

Issue update

The LendingPool admin will be deployed behind a 48h Timelock.

Users working with the project should keep track of changes to the ‘pm’ mapping.

Also, for the convenience of users, we recommend adding a list of current PM contracts and a view function for them. This will not lead to significant gas costs, but it will simplify verification for users.

ID: **RM09-02**	Severity: **Low**	Status: **Fixed**

Free flashloans (L103)

With the `_flashLoanPremiumTotal` variable set to 0, the contract allows everyone to make a free flashloan. Thus, the project will not be able to earn money on the use of flashloans by third-party users.

Make sure that this was intended, or consider branching by introducing a free flashloan for the OptionsPositionManager and LonggPositionManager contracts only (and `premium>0` for other users).

ID: **RM09-03**	Severity: **Info**	Status: **Fixed**

Emitting events

Consider adding an event to the setSoftLiquidationThreshold() function.

This is good practice and will make it easier to keep track changes off-chain.

Smart contract RoeRouter [RM-10]

Contract RoeRouter holds a list of whitelisted ROE lending pools with token addresses and routers.

ID: **RM10-01**	Severity: **Low**	Status: **Found**

Gas savings

a. The functions getPoolsLength(), deprecatePool(), addPool(), setTreasury() can be declared as 'external' to save gas.

b. The deprecatePool() function can check if the pool is already deprecated to avoid writing to storage if it is.

c. The deprecatePool() function can check if the pool is already deprecated to avoid writing to storage if it is.

Code quality

NatSpec documentation

We recommend writing NatSpec documentation for every function. Natspect return statement missing on getPoolsLength and addPool

Smart contract PositionManager [RM-11]

The PositionManager contract is used as the parent contract for contracts LonggPositionManager and OptionsPositionManager.

ID: **RM11-01**	Severity: **Low**	Status: **Found**

Gas savings

a. The variable `ROEROUTER` can be declared as immutable.

b. The variables `ADDRESSES_PROVIDER` and `LENDING_POOL` are never used and can be removed.

General issues and recommendations

A. Testing

1) We recommend adding instructions for running the tests, as well as all the associated configuration files for them (proper yaml or json files). The instructions should describe the required dependencies (e.g. InfuraID, etc.).

2) Tests should be made in such a way that when forking the network at any time they do not fail (for example, you can use a fork from a specific block).

3) We strongly recommend adding integration tests that fully simulate trading situations with various trading pairs. In such tests, cases with sharp price changes should be considered.

4) It is imperative to achieve as close as possible to 100% test coverage in a project.

5) During the audit, the tests were updated several times. In this case, there was not a single 100% correct result.

B. Generic Code Readability improvements

1) Review Natspec comments. In many places it lacks the '`@return`' value. Or the function definition is not present at all.

2) Consider one methodology to write functions along the repo. If putting the return statement at the function end, put it in all functions. If there's no return statement and it is defined in the function signature, try to follow that rule for all functions. This is mostly a readability improvement, to keep the same writing pattern.

3) In many ERC20 transfers and approvals, the returned value is not being checked. This goes against the EIP-20 which states:
`Callers MUST handle false from returns (bool success). Callers MUST NOT assume that false is never returned.` (link, link)
It's up to the development team to follow or not this point.

C. Project Management

1) Consider using alert service (eg. Forta Network) to track all significant trades and events for your project. Using such applications, you can notice suspicious activity on the part of the attackers and freeze the project for detailed investigations. This can help prevent significant losses (eg. Aave loss case).

2) We recommend that you carefully and cautiously approach the addition of tradable tokens (assets) in the project. Added tokens must be verified, as well as have sufficient liquidity in the market (to reduce the risk of price manipulation). In addition, such assets should have reliable and resistant to manipulation price oracles.
Also, it is not recommended to use tokens:

a) with fee on transfers
b) rebase tokens
c) tokens (and LP-pairs) with low liquidity in the market or low distribution among users
d) unverified tokens

3) Please note there's an infura ID in the readme file and a private key in many files of the test cases.

List of changes for draft versions

Versions: rc0.1 -> 1.0:

- **Added new high issue RM08-11;**
- RM07-01 issue's severity was decreased;
- RM08-01 issue is not relevant, but need to perform recommendations;
- RM08-04 issue is not relevant, but need to perform recommendations;
- RM09-01 issue status was updated, but consider performing recommendations;
- 'General issues and recommendations' added points C.1), D.2), D.3), D.4).

Versions: 1.0 -> 2.0 new issues:

- Added new issue RM06-05;
- Added new issue RM10-01;
- Added new issue RM11-01.

Conclusion

During the audit, a large number of problems were identified.

Such a result may also indicate poor testing in the project. Such a lend-trading project should be very well covered with tests. We strongly recommend writing more tests with several trading pairs, as well as testing for cases of extreme sharp changes in the prices of trading assets.

Please note that auditing is not a complete replacement for unit tests and integration tests.

Conclusion update

All of the high and medium issues have been resolved on the code update. There are some minor issues remaining which we don’t think represent possible danger of tampering with the contract suite.

The audit team was unable to successfully run the tests nor the coverage (without errors) from the repository.

Disclaimer

Note that smart contract audit provided by Protofire is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, Protofire recommends proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Smart contract audit provided by Protofire shall not be used as investment or financial advice.