AKIRA TECH

PROJECT

Roe

CLIENT

Roe Finance

DATE

November 2022

REVIEWERS

Andrei Simion

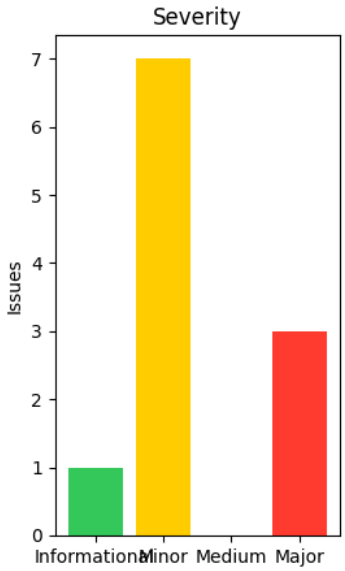
@andreiashu

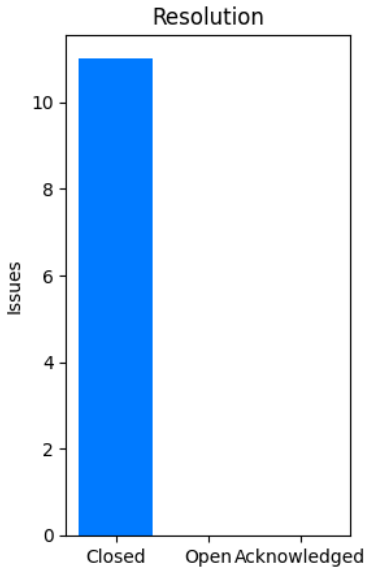
Table of Contents

Details

- **Client** Roe Finance
- **Date** November 2022
- **Reviewer** Andrei Simion (@andreiashu)
- **Repository**: Roe
- **Commit hash** `6230efc13e9b761fb068544b6f8bd41d8f86401c`
- **Technologies**
  - Solidity
  - Python

Issues Summary





SEVERITY	OPEN	CLOSED
Informational	0	1
Minor	0	7
Medium	0	0
Major	0	3

Executive summary

This report represents the results of the engagement with **Roe Finance** to review **Roe**.

The review was conducted over the course of **3 weeks** from **7th of November to 25th of November, 2022**. A total of **15 person-days** were spent reviewing the code.

Please note that this report also covers the Roe Markets project (commit hash `2eb1c1fb574940d81ad4d1c2b3de1629c068fda8`): a fork of Aave V2 Lending Markets that is

used as part of Roe Finance ecosystem.

Scope

The initial review focused on the Roe repository, identified by the commit hash `6230efc13e9b761fb068544b6f8bd41d8f86401c`.

A new code version was pulled in from commit `3f30726839119cc3d5f45396cc327c933f72a7ad`, `f59259a852e9e2ae1f4a4a68d7d67ce29cace756` and `8363cc3fc493f067be6c77c09f91ec0ca269babb` which contained fixes to the issues that I have raised during the review.

I focused on manually reviewing the codebase, searching for security issues such as, but not limited to, re-entrancy problems, transaction ordering, block timestamp dependency, exception handling, call stack depth limitation, integer overflow/underflow, self-destructible contracts, unsecured balance, use of origin, costly gas patterns, architectural problems, code readability.

Includes:

- code/contracts/TokenisableRange.sol
- code/contracts/LonggPositionManager.sol
- code/contracts/OptionsPositionManager.sol
- code/contracts/RangeManager.sol
- code/contracts/helper/LPOracle.sol
- code/contracts/helper/ZapBox.sol
- code/contracts/helper/HistoricPriceConsumerV3.sol
- code/contracts/helper/WindUniswap.sol

Excludes:

- code/contracts/openzeppelin-solidity
- code/contracts/TickMath.sol
- code/contracts/lib/FullMath.sol
- code/contracts/lib/LiquidityAmounts.sol
- code/contracts/lib/FixedPoint96.sol

Issues

[`LonggPositionManager`] swap operations can be price manipulated

Status Fixed Severity Major

Description

There are several instances of the `LonggPositionManager` code calling Uniswap's swap operations that are vulnerable to sandwich attacks:

Where the `amountOutMin` parameter is passed in as `1 wei`:

- https://github.com/akiratechhq/review-roe-finance-roe-2022-11/blob/65c43647b85d6354f42ec3b7155a104c2062619b/code/contracts/LonggPositionManager.sol#L165-L171
- https://github.com/akiratechhq/review-roe-finance-roe-2022-11/blob/65c43647b85d6354f42ec3b7155a104c2062619b/code/contracts/LonggPositionManager.sol#L526

Here the same `amountOutMin` parameter is set to `0`:

code/contracts/LonggPositionManager.sol#L578-L584

```
      ammRouter.swapExactTokensForTokens(
        amount,
        0,
        path,
        address(this),
        block.timestamp + 60
      );
```

In the following case, the `swapTokensForExactTokens` argument `amountInMax` is passed in as `swapAmount * 2`:

code/contracts/LonggPositionManager.sol#L625-L629

```
    swap(ammRouter,
      recvAmount,
      swapAmount * 2,
      path
    );
```

code/contracts/LonggPositionManager.sol#L642-L647

```
    ammRouter.swapTokensForExactTokens(
      recvAmount,
      maxAmount,
      path,
      address(this),
      block.timestamp
```

This is also vulnerable since the value computed (`swapAmount * 2`) is relative to a value that is part of a sandwichable operation, and therefore, the resulting issue is identical to the examples above.

Recommendation

Add invariants that rely on pricing data coming out of bounds (eg. from the UI) or use a Chainlink oracle if one is available for the token pair.

[`OptionsPositionManager`] and [`LonggPositionManager`] should validate the caller in `executeOperation` function

Status Fixed Severity Major

Description

`OptionsPositionManager.executeOperation` is the callback function called by Aave's LendingPool during a `flashLoan` operation:

code/contracts/OptionsPositionManager.sol#L78-L84

```
function executeOperation(
  address[] calldata assets,
  uint256[] calldata amounts,
  uint256[] calldata premiums,
  address initiator,
  bytes calldata params
) override external returns (bool) {
```

The `executeOperation` function should only allow the Lending Pool to call it. There are some examples of this in Aave's repository - `UniswapLiquiditySwapAdapter` contains code to restrict `msg.sender` only to the Lending Pool:

```
require(msg.sender == address(LENDING_POOL), 'CALLER_MUST_BE_LENDING_POOL');
```

Recommendation

Only allow the Lending Pool to call `executeOperation` . This applies to both `OptionsPositionManager` and `LonggPositionManager` contracts.

References

UniswapLiquiditySwapAdapter.sol

[`HistoricPriceConsumerV3`] might be using stale pricing data from Chainlink

Status Fixed Severity Major

Description

There are several places in the `HistoricPriceConsumerV3` contract where Chainlink's `latestRoundData` is being used:

code/contracts/helper/HistoricPriceConsumerV3.sol#L106

```
        ) = priceFeed.latestRoundData();
```

code/contracts/helper/HistoricPriceConsumerV3.sol#L186-L192

```
      (
            ,
            int price,
            ,
            uint timeStamp,
        ) = ratioQuote.latestRoundData();
        require(timeStamp != 0, "RATIO_ORACLE_NOT_READY");
```

code/contracts/helper/HistoricPriceConsumerV3.sol#L214-L220

```
        (
            ,
            int price,
            ,
            uint timeStamp,
        ) = priceFeed.latestRoundData();
        require(timeStamp != 0, "PRICEFEED_TIMESTAMP_NOT_READY");
```

The issue, however, is that the code handling the data feed returned by `latestRoundData` does not perform validation apart from a timestamp check:

code/contracts/helper/HistoricPriceConsumerV3.sol#L219-L220

```
        ) = priceFeed.latestRoundData();
        require(timeStamp != 0, "PRICEFEED_TIMESTAMP_NOT_READY");
```

This is emphasized in Chainlink's code documentation on `latestRoundData` (in v0.6 and v0.7):

> @notice get data about the latest round. Consumers are encouraged to check that they're receiving fresh data by inspecting the updatedAt and answeredInRound

> return values. Note that different underlying implementations of AggregatorV3Interface have slightly different semantics for some of the return values. Consumers should determine what implementations they expect to receive data from and validate that they can properly handle return data from all of them.

Recommendation

Ensure that the price feed data is valid and not stale. An example of validation performed on data returned by `latestRoundData`:

```
        (
          uint80 roundId,
          int256 answer,
          uint256 startedAt,
          uint256 updatedAt,
          uint80 answeredInRound
        ) = priceFeed.latestRoundData();

        require(answer > 0, "INVALID_PRICE");
        require(timestamp != 0, "ROUND_NOT_COMPLETE");
        // stalePriceDelay is a threshold value: price feed data that is
        // older than this value should be considered stale
        require(block.timestamp <= updatedAt + stalePriceDelay, "STALE_PRICE");
        require(answeredInRound >= roundId, "STALE_PRICE");
```

Update: I discussed the issue with the Roe team, and the decision was made to have only the `price > 0` check here. The reason for this is that, for the liquidation system to function, stale pricing data is the lesser of the two evils vs reverting at this stage.

`RangeManager.checkNewRange` gas savings and simpler logic

Status Fixed Severity Minor

Description

`checkNewRange` function is used to ensure that no two ranges overlap each other:

code/contracts/RangeManager.sol#L55-L63

```
/// @notice Checks validity and non overlap of the price ranges
function checkNewRange(uint128 start, uint128 end) internal view {
  require(start < end, "Range invalid");
  for (uint i = 0; i < stepList.length; i++) {
    if ((start < stepList[i].start)  && (end > stepList[i].start)) revert("Range overlap");
    if ((start < stepList[i].end)    && (end > stepList[i].end))   revert("Range overlap");
    if ((start >= stepList[i].start) && (end <= stepList[i].end))  revert("Range overlap");
  }
```

```
  }
```

Recommendation

Below I suggest a more straightforward implementation of the code which will also save a bit of gas, especially for cases where more ranges are managed.

```
function checkNewRange(uint128 start, uint128 end) internal view {
    require(start < end, "Range invalid");
    uint256 len = stepList.length;
    for (uint i = 0; i < len; i++) {
      if (start >= stepList[i].end || end <= stepList[i].start) {
        continue;
      }
      revert("Range overlap");
    }
  }
```

Below is a gas costs comparison for a case where only 3 ranges are managed:

```
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity 0.8.17;

contract StepsOrig {
    struct Step {
        uint128 start;
        uint128 end;
    }

    Step [] public stepList;


    constructor() {
        stepList.push(Step(uint128(0), uint128(2)));
        stepList.push(Step(uint128(5), uint128(6)));
        stepList.push(Step(uint128(8), uint128(10)));
    }

    // start = 10, end = 12
    // gas cost 35713
    function checkNewRange(uint128 start, uint128 end) public view {
        require(start < end, "Range invalid");
        for (uint i = 0; i < stepList.length; i++) {
            if ((start < stepList[i].start)  && (end > stepList[i].start)) revert("Range overlap");
            if ((start < stepList[i].end)    && (end > stepList[i].end))   revert("Range overlap");
            if ((start >= stepList[i].start) && (end <= stepList[i].end))  revert("Range overlap");
        }
    }
}

// more straightforward approach
```

```
contract StepsNew {
    struct Step {
        uint128 start;
        uint128 end;
    }

    Step [] public stepList;

    constructor() {
        stepList.push(Step(uint128(0), uint128(2)));
        stepList.push(Step(uint128(5), uint128(6)));
        stepList.push(Step(uint128(8), uint128(10)));
    }

    // start = 10, end = 12
    // gas cost 31910
    function checkNewRange(uint128 start, uint128 end) public view {
        require(start < end, "Range invalid");
        uint256 len = stepList.length;
        for (uint i = 0; i < len; i++) {
            if (start >= stepList[i].end || end <= stepList[i].start) {
                continue;
            }
            revert("Range overlap");
        }
    }
}
```

Gas saving opportunities

Status Fixed Severity Minor

Description

There are instances whereby gas savings can be achieved.

RangeManager.sol

`ASSET_0` and `ASSET_1` do not change once set in the constructor. They can be set as `immutable` to save gas:

code/contracts/RangeManager.sol#L30-L31

```
  ERC20 public ASSET_0;
  ERC20 public ASSET_1;
```

`TREASURY` state variable is not used and can therefore be removed:

code/contracts/RangeManager.sol#L27

```
address public TREASURY = 0x50101017adf9D2d06C395471Bc3D6348589c3b97;
```

[TokenisableRange] deposit function should validate pricing data feed

Status Fixed Severity Minor

Description

The `feeLiquidity` variable is calculated based on the oracle data feed from Chainlink's `latestRoundData`:

code/contracts/TokenisableRange.sol#L211-L219

```
// Stack too deep, so localising some variables for feeLiquidity calculations
{
    (,int256 TOKEN0_PRICE,,,) = CL_TOKEN0.latestRoundData();
    (,int256 TOKEN1_PRICE,,,) = CL_TOKEN1.latestRoundData();
    // Calculate the equivalent liquidity amount of the non-yet compounded fees
    // Assume linearity for liquidity in same tick range; calculate feeLiquidity equivalent and consid
    feeLiquidity = newLiquidity * ( (fee0 * uint256(TOKEN0_PRICE) / 10 ** TOKEN0.decimals) + (fee1 * u
                                  / ( (n0   * uint256(TOKEN0_PRICE) / 10 ** TOKEN0.decimals) + (n1   * u
}
```

The issue, however, is that the return values `TOKEN0_PRICE`, and `TOKEN1_PRICE` are not checked to ensure that the returned value is > 0.

This means that in the event of an invalid price returned by the feed, the amount of LP tokens minted will be higher since `feeLiquidity` will represent a smaller value:

code/contracts/TokenisableRange.sol#L221-L223

```
lpAmt = totalSupply() * newLiquidity / (liquidity + feeLiquidity);

_mint(msg.sender, lpAmt);
```

Recommendation

Check that both `TOKEN0_PRICE` and `TOKEN1_PRICE` are greater than 0.

[TokenisableRange] obsolete code check in deposit function

Status Fixed Severity Minor

Description

Given that both `added0` and `added1` are defined as `uint256`, the following check is unnecessary:

code/contracts/TokenisableRange.sol#L204

```
    require( (added0 >= 0) && (added1 >= 0), "Deposit Failed No Asset Added");
```

Recommendation

Remove the above mentioned code.

Uniswap's `deadline` argument can be passed as `block.timestamp` instead of a future value

Status Fixed Severity Minor

Description

The `deadline` argument that is defined in Uniswap contracts is used in order to protect transactions coming from off-chain sources from operating in an out-of-date state. For example, a user performing a swap operation on the Uniswap UI would benefit from having their request expire after 60 seconds - if the transaction was not included in any block during that period.

The `deadline` argument is validated in the `ensure` modifier:

contracts/UniswapV2Router02.sol#L18-L21

```
    modifier ensure(uint deadline) {
        require(deadline >= block.timestamp, 'UniswapV2Router: EXPIRED');
        _;
    }
```

Therefore, for operations initialized by other contracts, this argument can be simply be passed as `block.timestamp`. There are 7 instances in several contracts whereby this argument is passed as a future value:

- `WindUniswap.sol` (2 instances)
- `ZapBox.sol` (1 instance)
- `LonggPositionManager.sol` (3 instances)
- `OptionsPositionManager.sol` (1 instance)

code/contracts/LonggPositionManager.sol#L147

```
      block.timestamp + 3600
```

Recommendation

Replace instances of `block.timestamp + #` with `block.timestamp` for any Uniswap operations that require a `deadline` argument.

[`LonggPositionManager`] some state variables can be set to `immutable` or `constant` to save on gas costs

Status Fixed Severity Minor

Description

The `TREASURY` state variable is only set once in the constructor of the contract:

code/contracts/LonggPositionManager.sol#L32

```
address public TREASURY;
```

code/contracts/LonggPositionManager.sol#L62-L64

```
constructor(address treasury_) {
  TREASURY = treasury_;
}
```

To save on gas costs, it can be declared as `immutable`.

Both the `HF_THRESHOLD` and `HF_MAX` state variables can be declared as `constant` since they do not change upon deployment nor once the contract has been deployed:

code/contracts/LonggPositionManager.sol#L33-L34

```
uint public HF_THRESHOLD = 102e16; // Health Factor < 1.02
uint public HF_MAX = 105e16; // Debt repayment cannot excessively reduce debt
```

Recommendation

Declare `TREASURY` as immutable and `HF_THRESHOLD` and `HF_MAX` as constant.

[`OptionsPositionManager`] should validate constructor arguments

Status Fixed Severity Minor

Description

`treasury_` should be non nil and `v2router` can be checked to ensure it's a V2 router by calling a specific function on the contract.

code/contracts/OptionsPositionManager.sol#L66-L69

```
constructor(address treasury_, address v2router) {
  TREASURY = treasury_;
  SWAP_ROUTER_V2 = IUniswapV2Router01(v2router );
}
```

Use `Historical` instead of `Historic` for the `HistoricPriceConsumerV3.sol` filename

Status Fixed Severity Informational

Description

The filename uses `Historic`, but all the contracts that reside in it use `Historical` for the contract name.

Recommendation

Use `Historical` instead of `Historic`.

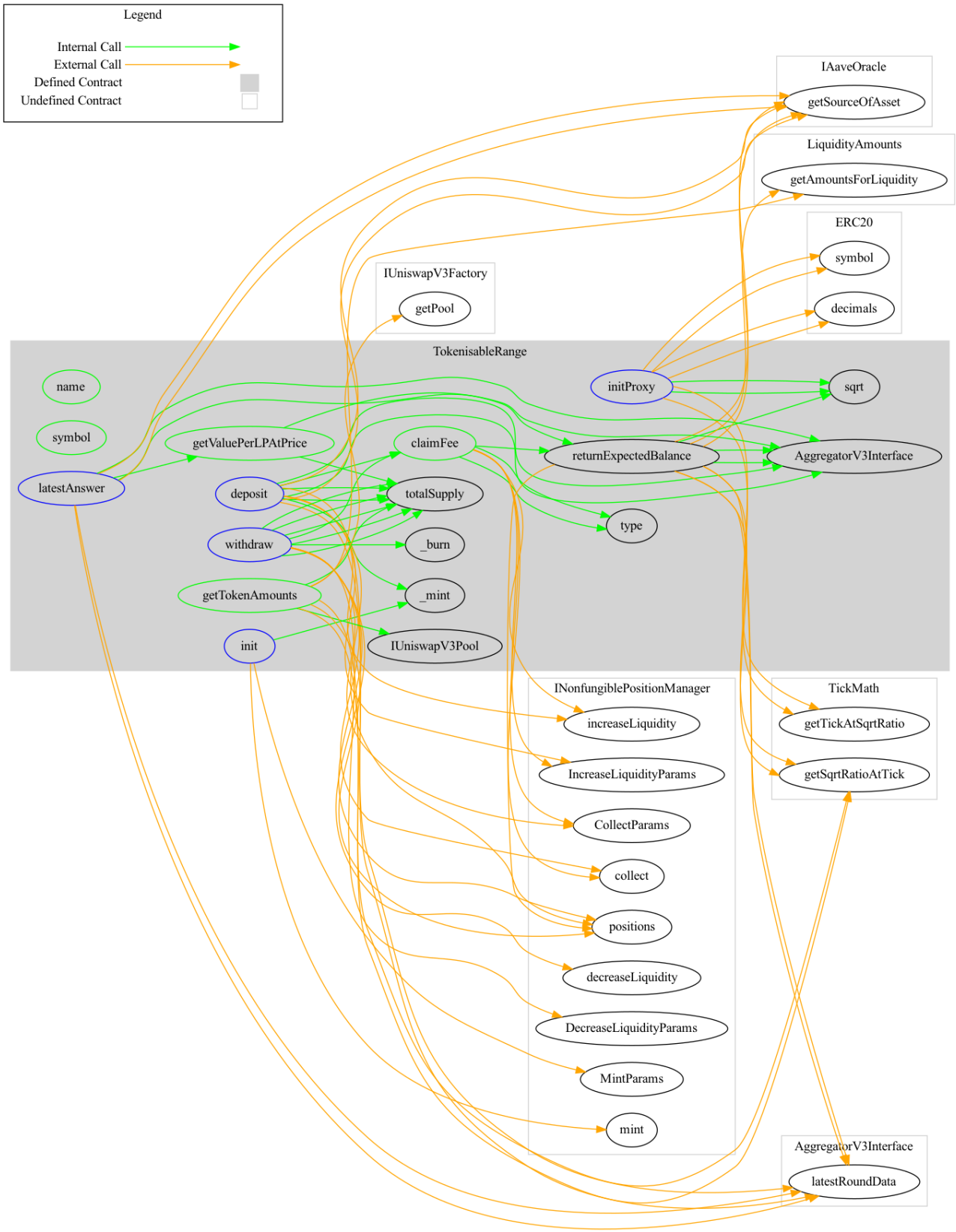
Artifacts

Surya

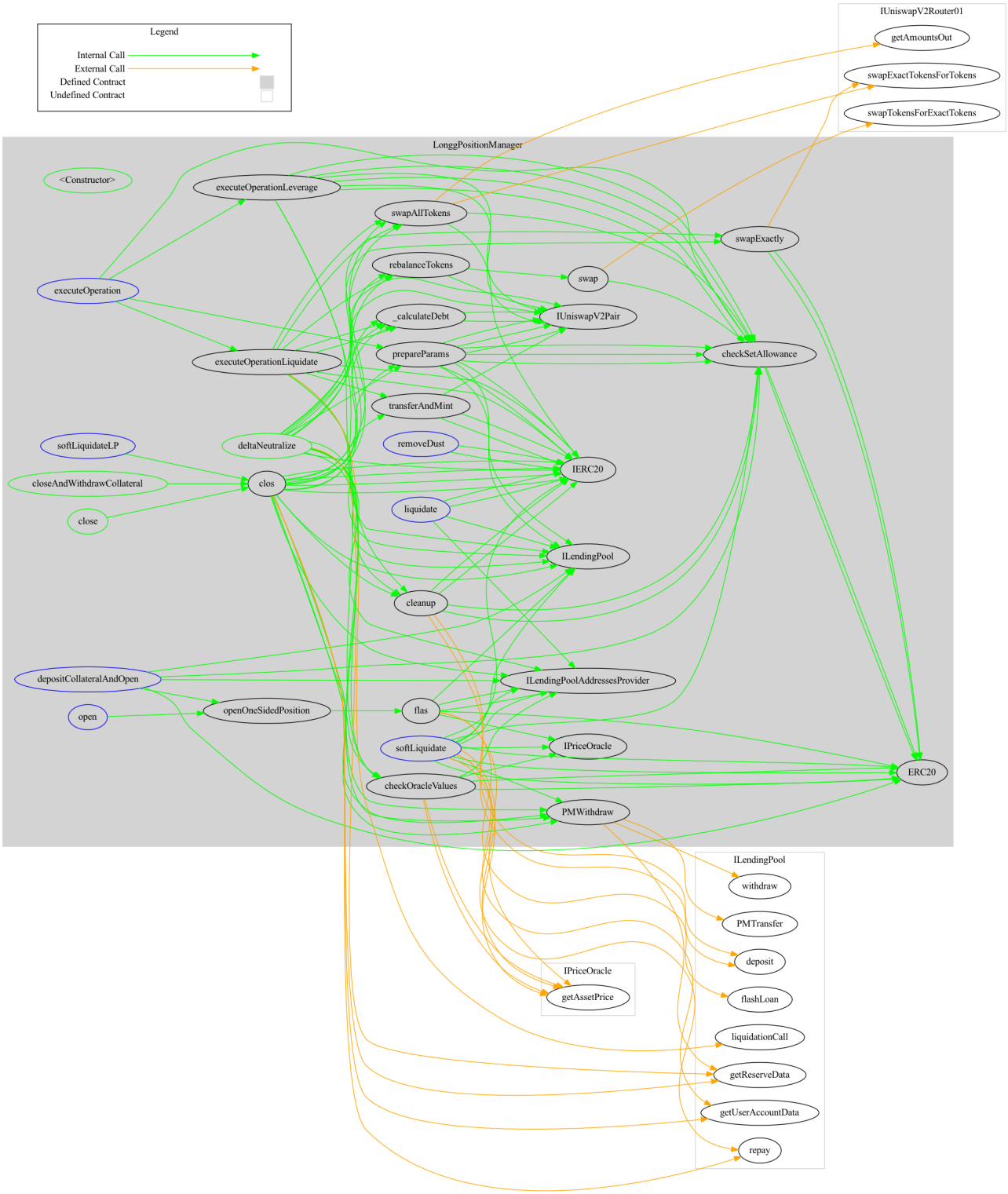
Sūrya is a utility tool for smart contract systems. It provides a number of visual outputs and information about the structure of smart contracts. It also supports querying the function call graph in multiple ways to aid in the manual inspection and control flow analysis of contracts.

Graphs

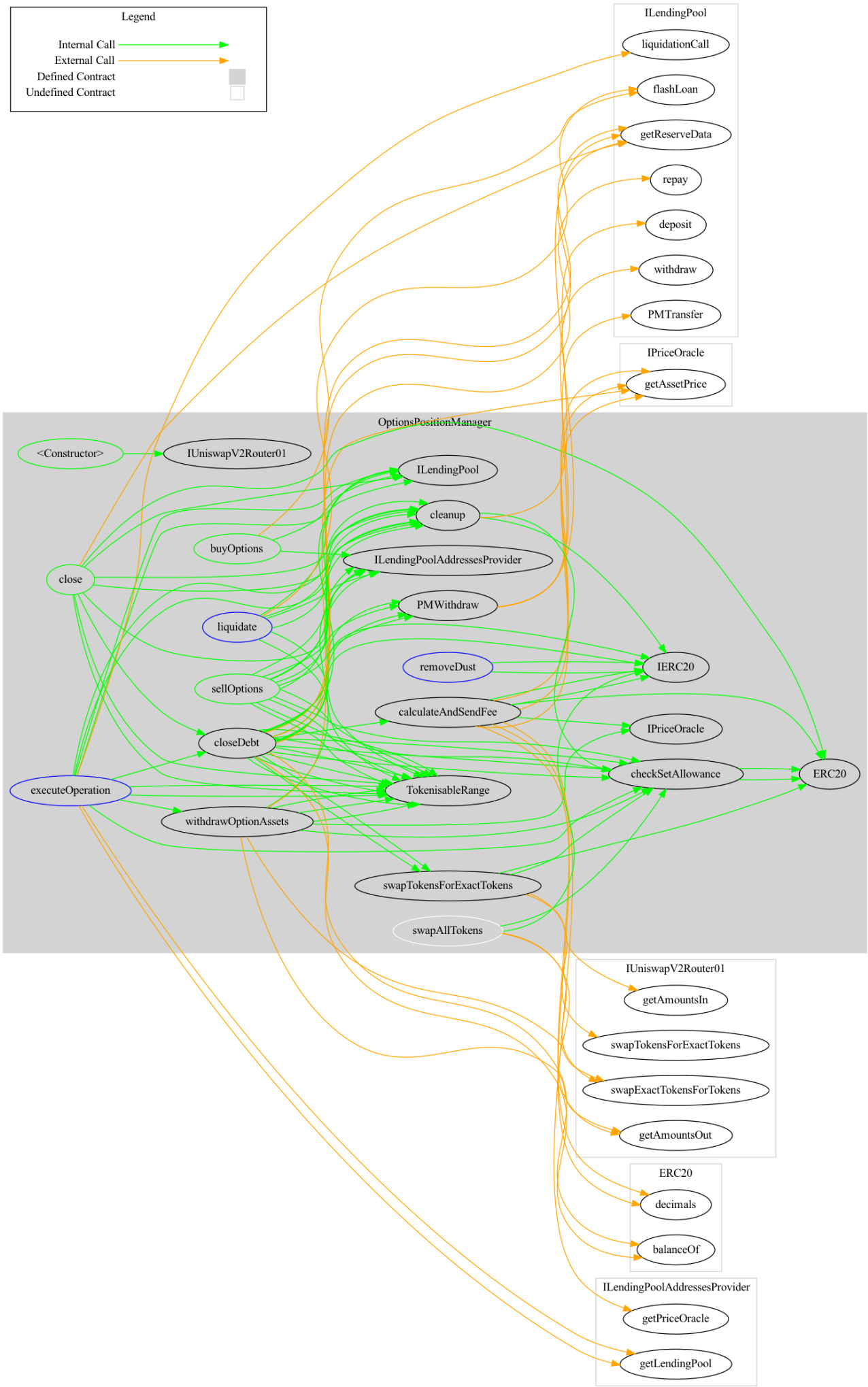
TokenisableRange Graph



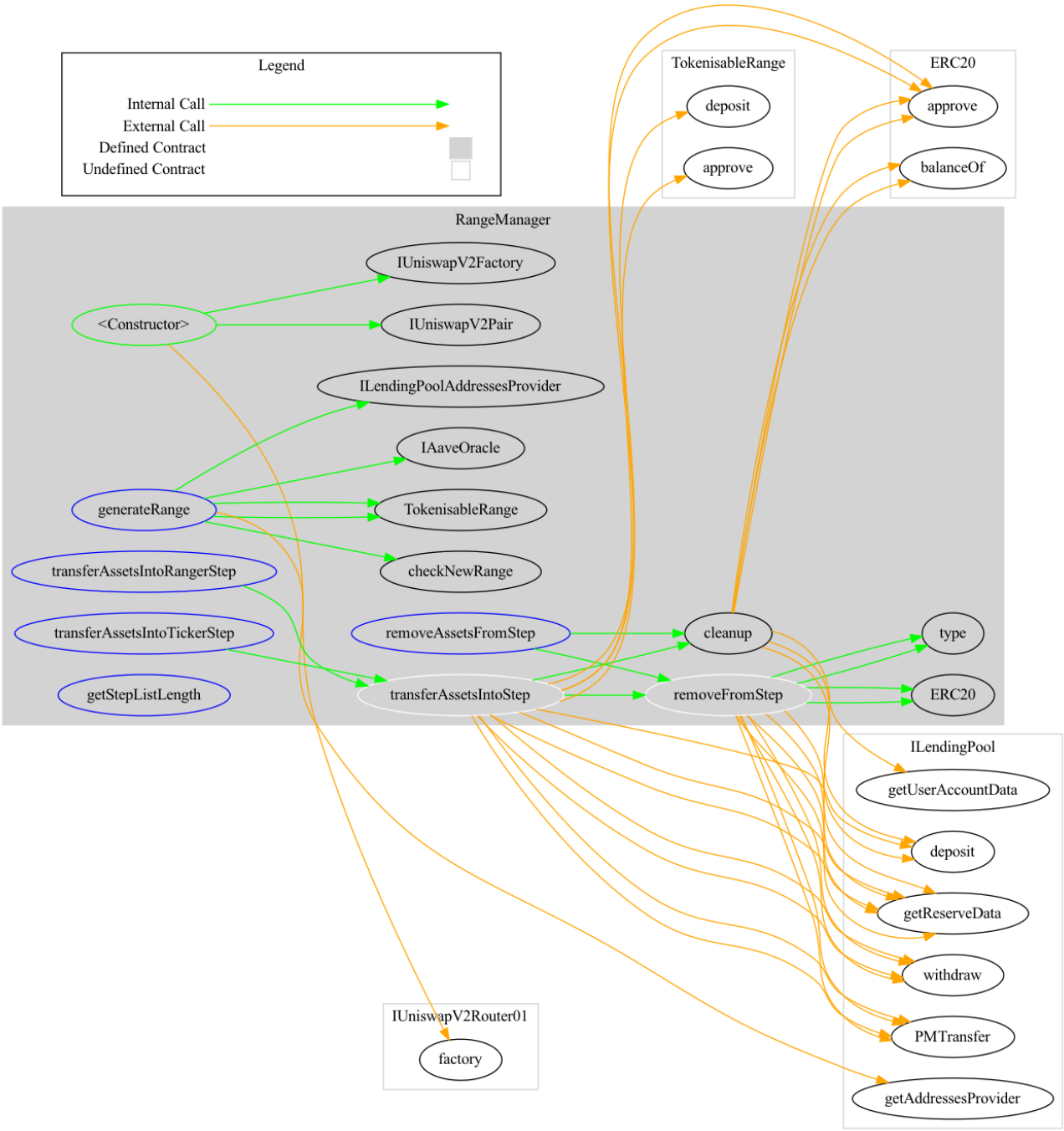
LonggPositionManager Graph



OptionsPositionManager Graph

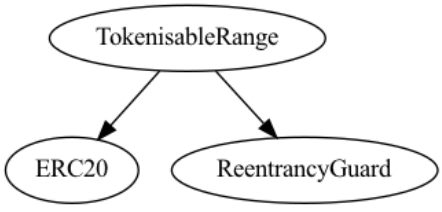

Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ILendingPool
liquidationCall
flashLoan
getReserveData
repay
deposit
withdraw
PMTransfer
IPriceOracle
getAssetPrice
OptionsPositionManager
<Constructor>
IUniswapV2Router01
ILendingPool
cleanup
buyOptions
ILendingPoolAddressesProvider
close
PMWithdraw
liquidate
removeDust
IERC20
sellOptions
calculateAndSendFee
IPriceOracle
closeDebt
checkSetAllowance
ERC20
executeOperation
TokenisableRange
withdrawOptionAssets
swapTokensForExactTokens
swapAllTokens
IUniswapV2Router01
getAmountsIn
swapTokensForExactTokens
swapExactTokensForTokens
getAmountsOut
ERC20
decimals
balanceOf
ILendingPoolAddressesProvider
getPriceOracle
getLendingPool

RangeManager Graph

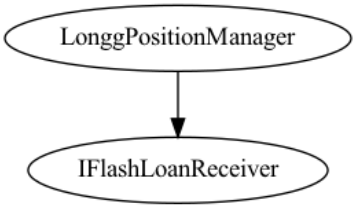


Inheritance

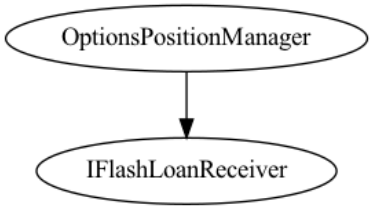
TokenisableRange



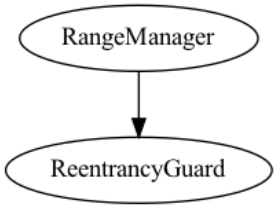
LonggPositionManager



OptionsPositionManager



RangeManager



Describe

```
$ npx surya describe code/contracts/TokenisableRange.sol code/contracts/LonggPositionManager.sol code/cont


 +  TokenisableRange (ERC20, ReentrancyGuard)
    - [Int] sqrt
    - [Ext] initProxy #
    - [Pub] name
    - [Pub] symbol
    - [Ext] init #
    - [Pub] claimFee #
    - [Ext] deposit #
       - modifiers: nonReentrant
    - [Ext] withdraw #
       - modifiers: nonReentrant
    - [Pub] returnExpectedBalance
    - [Pub] getValuePerLPAtPrice
    - [Ext] latestAnswer
    - [Pub] getTokenAmounts

 +  LonggPositionManager (IFlashLoanReceiver)
    - [Pub] <Constructor> #
    - [Ext] executeOperation #
    - [Int] prepareParams #
    - [Prv] executeOperationLeverage #
    - [Ext] depositCollateralAndOpen #
    - [Ext] open #
```

```
    - [Pub] openOneSidedPosition #
    - [Int] flas #
    - [Pub] closeAndWithdrawCollateral #
    - [Pub] close #
    - [Int] clos #
    - [Ext] liquidate #
    - [Prv] executeOperationLiquidate #
    - [Ext] softLiquidateLP #
    - [Ext] softLiquidate #
    - [Pub] deltaNeutralize #
    - [Prv] swapExactly #
    - [Int] checkOracleValues
    - [Int] PMWithdraw #
    - [Int] swapAllTokens #
    - [Int] rebalanceTokens #
    - [Int] swap #
    - [Int] _calculateDebt
    - [Int] transferAndMint #
    - [Ext] removeDust #
    - [Prv] cleanup #
    - [Int] checkSetAllowance #

+   OptionsPositionManager (IFlashLoanReceiver)
    - [Pub] <Constructor> #
    - [Ext] executeOperation #
    - [Prv] withdrawOptionAssets #
    - [Pub] buyOptions #
    - [Ext] liquidate #
    - [Pub] close #
    - [Int] closeDebt #
    - [Int] calculateAndSendFee #
    - [Pub] sellOptions #
    - [Int] PMWithdraw #
    - [Int] swapAllTokens #
    - [Int] swapTokensForExactTokens #
    - [Ext] removeDust #
    - [Prv] cleanup #
    - [Int] checkSetAllowance #

+   RangeManager (ReentrancyGuard)
    - [Pub] <Constructor> #
    - [Int] checkNewRange
    - [Ext] generateRange #
    - [Int] removeFromStep #
    - [Ext] removeAssetsFromStep #
       - modifiers: nonReentrant
    - [Int] transferAssetsIntoStep #
    - [Ext] transferAssetsIntoRangerStep #
       - modifiers: nonReentrant
    - [Ext] transferAssetsIntoTickerStep #
       - modifiers: nonReentrant
```

```
    - [Int] cleanup #
    - [Ext] getStepListLength

+ [Lib] TickMath
    - [Pub] getSqrtRatioAtTick
    - [Pub] getTickAtSqrtRatio

+ [Lib] LiquidityAmounts
    - [Prv] toUint128
    - [Int] getLiquidityForAmount0
    - [Int] getLiquidityForAmount1
    - [Int] getLiquidityForAmounts
    - [Int] getAmount0ForLiquidity
    - [Int] getAmount1ForLiquidity
    - [Int] getAmountsForLiquidity

+ [Lib] FullMath
    - [Int] mulDiv
    - [Int] mulDivRoundingUp

+ [Lib] FixedPoint96

+ [Int] UniswapV2Pair
    - [Ext] totalSupply
    - [Ext] getReserves
    - [Ext] token0
    - [Ext] token1

+ [Int] IERC20
    - [Ext] decimals

+  LPOracle
    - [Pub] <Constructor> #
    - [Ext] decimals
    - [Int] sqrt
    - [Int] getAnswer
    - [Ext] latestAnswer

+ [Int] ERC2612
    - [Ext] permit #

+  ZapBox
    - [Pub] <Constructor> #
    - [Pub] zapIn #
    - [Pub] zapInETH ($)
    - [Pub] zapInSingleAsset #
    - [Pub] zapInSingleAssetETH ($)
    - [Pub] zapOut #
    - [Pub] zapOutWithPermit #
    - [Ext] <Receive Ether> ($)
    - [Prv] cleanup #
```

```
    - [Prv] checkSetApprove #
    - [Int] getSwapAmt
    - [Int] sqrt

+ [Int] AggregatorInterface
    - [Ext] latestAnswer
    - [Ext] latestTimestamp
    - [Ext] latestRound
    - [Ext] getAnswer
    - [Ext] getTimestamp

+ [Int] AggregatorV3Interface
    - [Ext] decimals
    - [Ext] description
    - [Ext] version
    - [Ext] getRoundData
    - [Ext] latestRoundData

+ [Int] AggregatorV2V3Interface (AggregatorInterface, AggregatorV3Interface)

+ [Int] AggregatorProxy (AggregatorV2V3Interface)
    - [Ext] phaseId

+ [Int] HistoricalPriceConsumerV3
    - [Ext] getPriceAfterTimestamp
    - [Ext] getLatestPriceX1e6

+  HistoricalPriceConsumerV3_1
    - [Int] getHistoricalPrice
    - [Int] getLatestPrice
    - [Ext] getPriceAfterTimestamp
    - [Int] findBlockSamePhase
    - [Ext] checkAggregatorDecimals
    - [Pub] getLatestPriceX1e6

+  HistoricalPriceConsumerV3_RATIO
    - [Pub] <Constructor> #
    - [Pub] getQuotePrice
    - [Int] getQuoteMantissa
    - [Pub] getHistoricalPrice
    - [Pub] getLatestPrice
    - [Pub] findPriceAfterTimestamp
    - [Pub] getPriceAfterTimestamp
    - [Pub] findBlockSamePhase
    - [Ext] checkAggregatorDecimals
    - [Pub] getLatestPriceX1e6

+  HistoricalPriceConsumerV3_FIXEDPRICE
    - [Pub] <Constructor> #
    - [Ext] setPrice #
    - [Ext] setOracle #
```

```
    - [Pub] getLatestPrice
    - [Pub] getPriceAfterTimestamp
    - [Pub] getLatestPriceX1e6
    - [Ext] checkAggregatorDecimals

 +  WindUniV2LP (Ownable)
    - [Pub] <Constructor> #
    - [Ext] levUp #
    - [Ext] levDown #
    - [Ext] withdraw #
       - modifiers: onlyOwner


 ($) = payable function
 # = non-constant function
```

Coverage

Tests

```
❯ brownie test
Brownie v1.19.0 - Python development framework for Ethereum

New compatible solc version available: 0.8.11
Compiling contracts...
  Solc version: 0.8.11
  Optimizer: Enabled  Runs: 200
  EVM Version: Istanbul
Generating build data...
 - LonggPositionManager
 - OptionsPositionManager
 - RangeManager
 - TickMath
 - TokenisableRange
 - AggregatorInterface
 - AggregatorProxy
 - AggregatorV2V3Interface
 - AggregatorV3Interface
 - HistoricalPriceConsumerV3
 - HistoricalPriceConsumerV3_1
 - HistoricalPriceConsumerV3_FIXEDPRICE
 - HistoricalPriceConsumerV3_RATIO
 - IERC20
 - LPOracle
 - UniswapV2Pair
 - OracleConvert
 - WindUniV2LP
```

- ERC2612
- ZapBox
- FixedPoint96
- FullMath
- LiquidityAmounts
- AccessControl
- AccessControlEnumerable
- IAccessControl
- IAccessControlEnumerable
- Ownable
- PaymentSplitter
- VestingWallet
- Governor
- IGovernor
- TimelockController
- GovernorCompatibilityBravo
- IGovernorCompatibilityBravo
- GovernorCountingSimple
- GovernorPreventLateQuorum
- GovernorProposalThreshold
- GovernorSettings
- GovernorTimelockCompound
- ICompoundTimelock
- GovernorTimelockControl
- GovernorVotes
- GovernorVotesComp
- GovernorVotesQuorumFraction
- IGovernorTimelock
- IVotes

- Votes
- IERC1271
- IERC1363
- IERC1363Receiver
- IERC1363Spender
- IERC2981
- IERC3156FlashBorrower
- IERC3156FlashLender
- IERC2612
- ERC2771Context
- MinimalForwarder
- AccessControlEnumerableMock
- AccessControlMock
- AddressImpl
- ArraysImpl
- BadBeaconNoImpl
- BadBeaconNotContract
- Base64Mock
- BitMapMock
- CallReceiverMock
- CheckpointsImpl
- ClashingImplementation

- ClonesMock
- ConditionalEscrowMock
- ContextMock
- ContextMockCaller
- CountersImpl
- Create2Impl
- DummyImplementation
- DummyImplementationV2
- Impl
- ECDSAMock
- EIP712External
- ERC1155BurnableMock
- ERC1155Mock
- ERC1155PausableMock
- ERC1155ReceiverMock
- ERC1155SupplyMock
- ERC1271WalletMock
- ERC165InterfacesSupported
- SupportsInterfaceWithLookupMock
- ERC165MissingData
- ERC165NotSupported
- ERC165CheckerMock
- ERC165Mock
- ERC165StorageMock
- ERC1820ImplementerMock
- ERC20BurnableMock
- ERC20CappedMock
- ERC20DecimalsMock
- ERC20FlashMintMock

- ERC20Mock
- ERC20PausableMock
- ERC20PermitMock
- ERC20SnapshotMock
- ERC20VotesCompMock
- ERC20VotesMock
- ERC20WrapperMock
- ERC2771ContextMock
- ERC3156FlashBorrowerMock
- ERC721BurnableMock
- ERC721EnumerableMock
- ERC721Mock
- ERC721PausableMock
- ERC721ReceiverMock
- ERC721RoyaltyMock
- ERC721URIStorageMock
- ERC721VotesMock
- ERC777Mock
- ERC777SenderRecipientMock
- EnumerableMapMock
- EnumerableAddressSetMock
- EnumerableBytes32SetMock

- EnumerableUintSetMock
- EtherReceiverMock
- GovernorCompMock
- GovernorCompatibilityBravoMock
- GovernorMock
- GovernorPreventLateQuorumMock
- GovernorTimelockCompoundMock
- GovernorTimelockControlMock
- GovernorVoteMocks
- ConstructorInitializableMock
- InitializableMock
- MathMock
- MerkleProofWrapper
- MulticallTest
- MulticallTokenMock
- SampleChild
- SampleFather
- SampleGramps
- SampleHuman
- SampleMother
- OwnableMock
- PausableMock
- PullPaymentMock
- ReentrancyAttack
- ReentrancyMock
- Implementation1
- Implementation2
- Implementation3
- Implementation4

- SafeCastMock
- ERC20NoReturnMock
- ERC20ReturnFalseMock
- ERC20ReturnTrueMock
- SafeERC20Wrapper
- SafeMathMock
- SignatureCheckerMock
- SignedSafeMathMock
- MigratableMockV1
- MigratableMockV2
- MigratableMockV3
- StorageSlotMock
- StringsMock
- TimersBlockNumberImpl
- TimersTimestampImpl
- UUPSUpgradeableBrokenMock
- UUPSUpgradeableMock
- UUPSUpgradeableUnsafeMock
- VotesMock
- CompTimelock
- MyGovernor1
- MyGovernor2

- MyGovernor
- Clones
- ERC1967Proxy
- ERC1967Upgrade
- Proxy
- BeaconProxy
- IBeacon
- UpgradeableBeacon
- ProxyAdmin
- TransparentUpgradeableProxy
- Initializable
- UUPSUpgradeable
- Pausable
- PullPayment
- ReentrancyGuard
- ERC1155
- IERC1155
- IERC1155Receiver
- ERC1155Burnable
- ERC1155Pausable
- ERC1155Supply
- IERC1155MetadataURI
- ERC1155PresetMinterPauser
- ERC1155Holder
- ERC1155Receiver
- ERC20
- IERC20
- ERC20Burnable
- ERC20Capped

- ERC20FlashMint
- ERC20Pausable
- ERC20Snapshot
- ERC20Votes
- ERC20VotesComp
- ERC20Wrapper
- IERC20Metadata
- ERC20Permit
- IERC20Permit
- ERC20PresetFixedSupply
- ERC20PresetMinterPauser
- SafeERC20
- TokenTimelock
- ERC721
- IERC721
- IERC721Receiver
- ERC721Burnable
- ERC721Enumerable
- ERC721Pausable
- ERC721Royalty
- ERC721URIStorage
- IERC721Enumerable

- IERC721Metadata
- ERC721Votes
- ERC721PresetMinterPauserAutoId
- ERC721Holder
- ERC777
- IERC777
- IERC777Recipient
- IERC777Sender
- ERC777PresetFixedSupply
- ERC2981
- Address
- Arrays
- Base64
- Checkpoints
- Context
- Counters
- Create2
- Multicall
- StorageSlot
- Strings
- Timers
- ECDSA
- MerkleProof
- SignatureChecker
- EIP712
- ConditionalEscrow
- Escrow
- RefundEscrow
- ERC165

- ERC165Checker
- ERC165Storage
- ERC1820Implementer
- IERC165
- IERC1820Implementer
- IERC1820Registry
- Math
- SafeCast
- SafeMath
- SignedSafeMath
- BitMaps
- EnumerableMap
- EnumerableSet
- DataTypes
- ILendingPool
- IAaveOracle
- IERC721Permit
- IFlashLoanReceiver
- ILendingPoolAddressesProvider
- INonfungiblePositionManager
- IPeripheryImmutableState
- IPeripheryPayments

```
 - IPoolInitializer
 - IPriceOracle
 - ISwapRouter
 - IUniswapV2Factory
 - IUniswapV2Pair
 - IUniswapV2Router01
 - IUniswapV3Factory
 - IUniswapV3Pool
 - IUniswapV3SwapCallback
 - PoolAddress

Generating interface ABIs...
==================================== test session starts ====================================
platform darwin -- Python 3.10.4, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /akiratech/review-roe-finance-roe-2022-11/code
plugins: eth-brownie-1.19.0, forked-1.4.0, xdist-1.34.0, hypothesis-6.27.3, web3-5.29.1
collected 31 items
Attached to local RPC client listening at '127.0.0.1:8545'...

tests/test_pm.py ........                                                          [ 25%]
tests/test_pm_ranger.py ..                                                 [ 32%]
tests/test_ranger.py ........                                              [ 58%]
tests/test_ranger_WBTCUSDC.py .......                          [ 80%]
tests/test_zap.py ......                                       [100%]

-- Docs: https://docs.pytest.org/en/stable/warnings.html
========= 31 passed, 1 warning in 908.04s (0:15:08) ==========
```

License

This report falls under the terms described in the included LICENSE.