

AKIRA TECH

PROJECT

Roe

CLIENT

Ezoia Software
Development Pte Ltd

DATE

April 2022

REVIEWERS

Andrei Simion
[@andreiashu](#)

Table of Contents

- [Details](#)
- [Issues Summary](#)
- [Executive summary](#)
- [Scope](#)
- [Recommendations](#)
 - [Increase the number of tests](#)
- [Issues](#)
 - [FlashLoanReceiverBase_Unwind.clearSingleSidedDebt is vulnerable to price manipulation attacks](#)
 - [premium is sent to the wrong address](#)
 - [LPVault.withdrawOnBehalf might revert in some cases](#)
 - [executeOperation should only permit the LendingPool to call it](#)
 - [FlashLoan_Uniswap should not approve for maximum value](#)
 - [Uniswap V2 deadline argument can be passed as block.timestamp only](#)
- [Artifacts](#)
 - [Surya](#)
- [Sūrya's Description Report](#)
 - [Files Description Table](#)
 - [Contracts Description Table](#)
 - [Legend](#)
- [License](#)

Details

- **Client** Ezoia Software Development Pte Ltd
- **Date** April 2022
- **Reviewers** Andrei Simion ([@andreiashu](#))
- **Repository:** [Roe](#)
- **Commit hash** `2d2018cd30831ceaa3a1f8153714ca525e89f4c9`
- **Technologies**
 - Solidity

Issues Summary

SEVERITY	OPEN	CLOSED
Informational	0	0
Minor	2	0
Medium	2	0
Major	2	0

Executive summary

This report represents the results of the engagement with **Ezoia Software Development Pte Ltd** to review **Roe**.

The review was conducted over the course of **2 weeks** from **April 18th to April 29th, 2022**. A total of **10 person-days** were spent reviewing the code.

Scope

The initial review focused on the [Roe](#) repository, identified by the commit hash `2d2018cd30831ceaa3a1f8153714ca525e89f4c9`.

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/aux/FlashLoan_Uniswap.sol`
- `code/contracts/aux/ETHUSDCLP_Oracle.sol`
- `code/contracts/aux/HistoricPriceConsumerV3.sol`
- `code/contracts/aux/FlashLoan_Unwind.sol`
- `code/contracts/aux/WindUniswap.sol`
- `code/contracts/LPVault.sol`

Excludes:

- `code/contracts/openzeppelin-solidity/*`

Recommendations

I identified a few possible general improvements that are not security issues during the review, which will bring value to the developers and the community reviewing and using the product.

Increase the number of tests

A good rule of thumb is to have 100% test coverage. This does not guarantee the lack of security problems, but it means that the desired functionality behaves as intended. The negative tests also bring a lot of value because not allowing some actions to happen is also part of the desired behavior.

Issues

FlashLoanReceiverBase_Unwind.clearSingleSidedDebt is vulnerable to price manipulation attacks

Status **Open** Severity **Major**

Description

clearSingleSidedDebt function calls Uniswap V2's getReserves :

[code/contracts/aux/FlashLoan_Unwind.sol#L123-L129](#)

```
function clearSingleSidedDebt(address borrowedAsset) external {
    //IERC20 varDebt = IERC20( LENDING_POOL.getReserveData(borrowedAsset).variableDebtTokenAddress );

    uint needA;
    uint needB;
    {
        (uint resA, uint resB, ) = IUniswapV2Pair(borrowedAsset).getReserves();
```

The issue with relying on the getReserves function to calculate pricing data is that it's easy to manipulate its output for profits:

consider a malicious actor who sees this transaction before it is confirmed. They could execute a swap which dramatically changes the DAI/WETH price immediately before the naive swap goes through, wait for the naive swap to execute at a bad rate, and then swap to change the price back to what it was before the naive swap. This attack is fairly cheap and low-risk, and can typically be performed for a profit.

source: [Uniswap V2 Documentation on Pricing](#)

Recommendation

`clearSingleSidedDebt` function should either allow the caller to specify the real price of the tokens involved, or should make use of an external, reliable, oracle to fetch the latest price for a given pair of tokens:

To prevent these types of attacks, it's vital to submit swaps that have access to knowledge about the "fair" price their swap should execute at. In other words, swaps need access to an oracle, to be sure that the best execution they can get from Uniswap is close enough to what the oracle considers the "true" price. While this may sound complicated, the oracle can be as simple as an off-chain observation of the current market price of a pair. Because of arbitrage, it's typically the case that the ratio of the intra-block reserves of a pair is close to the "true" market price. So, if a user submits a trade with this knowledge in mind, they can ensure that the losses due to front-running are tightly bounded.

source: [Uniswap V2 Documentation on Pricing](#)

premium is sent to the wrong address

Status Open Severity Major

Description

During a flash loan operation to the Aave Lending Pool, the execution control is passed to the `executeOperation` function:

[code/contracts/aux/FlashLoan_Uniswap.sol#L32-L38](#)

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

In this call the premium is transferred to the `initiator` of the flash loan:

[code/contracts/aux/FlashLoan_Uniswap.sol#L59](#)

```
IERC20(assets[0]).safeTransfer(initiator, premiums[0]);
```

The issue here is that the initiator is the `msg.sender` (caller) of the `flashLoan` function against the `LendingPool1` contract. The call from Aave's `flashLoan` looks like this:

```
require(  
    flashLoan(  
        assets,  
        amounts,  
        premiums,  
        initiator,  
        params,  
        params.length
```

```
vars.receiver.executeOperation(assets, amounts, premiums, msg.sender, params),
Errors.LP_INVALID_FLASH_LOAN_EXECUTOR_RETURN
);
```

The 4th argument to `executeOperation` callback is the `initiator` in our case:

[code/contracts/aux/FlashLoan_Uniswap.sol#L32-L36](#)

```
function executeOperation(
    address[] calldata assets,
    uint256[] calldata amounts,
    uint256[] calldata premiums,
    address initiator,
```

The amount flash loaded and the premium will be taken back from the contract via `transferFrom` by the `LendingPool` in case of a mode-0 flash loan, or in the current case, it will count towards the debt amount for a mode-2 flash loan.

Recommendation

Remove the transfer of premium from line 59:

[code/contracts/aux/FlashLoan_Uniswap.sol#L59](#)

```
IERC20(assets[0]).safeTransfer(initiator, premiums[0]);
```

LPVault.withdrawOnBehalf might revert in some cases

Status Open Severity Medium

Description

The `LPVault.withdrawOnBehalf` function enables an address to withdraw their `COLLAT_LP` tokens from the Vault:

[code/contracts/LPVault.sol#L278](#)

```
function withdrawOnBehalf(address onBehalfOf) checkInvariant public returns (uint256) {
```

Since the same `COLLAT_LP` tokens are deposited into Aave Lending Pool as a way to facilitate lending them, there is a possibility that the `LPVault` does not have enough funds to cover the withdrawal amount. In this case, the difference in funds is extracted from the Lending Pool:

[code/contracts/LPVault.sol#L293-L298](#)

```
uint vaultBalance = COLLAT_LP.balanceOf( address(this) );
if (exitAmt > vaultBalance) { // If insufficient amou
```

```

        aaveV2LendingPool.withdraw( address(COLLAT_LP), exitAmt - vaultBalance, address(this) );
        collatLentOut -= exitAmt - vaultBalance;
    }
    ERC20(address(COLLAT_LP)).safeTransfer(onBehalfOf, exitAmt);

```

The issue, however, is that since lending will be enabled in the Lending Pool against the LP_COLLAT tokens, there might not be `exitAmt - vaultBalance` amount of tokens in the pool. In that case the `withdraw` function will revert:

[code/contracts/LPVault.sol#L295](#)

```

aaveV2LendingPool.withdraw( address(COLLAT_LP), exitAmt - vaultBalance, address(this) );

```

Recommendation

Ensure that the request for the number of tokens withdrawn from Aave's Lending Pool is at most the funds that the `LPVault` contract has in the Lending Pool.

`executeOperation` should only permit the LendingPool to call it

Status Open Severity Medium

Description

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

[code/contracts/aux/FlashLoan_Uniswap.sol#L32-L38](#)

```

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

```

The `executeOperation` function should whitelist the caller to only allow the Lending Pool to call into 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 `FlashLoanReceiverBase` and `FlashLoanReceiverBase_Unwind` contracts.

References

[UniswapLiquiditySwapAdapter.sol](#)

FlashLoan_Uniswap should not approve for maximum value

Status Open Severity Minor

Description

The `flashLeverage` function issues an approval call for the Aave Lending Pool contract and the Lending Pool's AToken:

[code/contracts/aux/FlashLoan_Uniswap.sol#L64-L74](#)

```
function flashLeverage(address asset, uint amount) external {
    address[] memory assets = new address[](1);
    uint[] memory amounts = new uint[](1);
    uint[] memory flashtype = new uint[](1);
    assets[0] = asset;
    amounts[0] = amount;
    flashtype[0] = 2;
    IERC20(asset).approve( address(LENDING_POOL), 2**256-1);
    IERC20(asset).approve( LENDING_POOL.getReserveData(asset).aTokenAddress, 2**256-1);
    LENDING_POOL.flashLoan( address(this), assets, amounts, flashtype, msg.sender, abi.encode(msg.sender),
}
```

Since the approval is the maximum value, I initially thought it would be best to just move these calls to the `constructor` of the contract since that would save gas. But the issue with this approach is that `ERC20.transferFrom` decreases the allowance on every call (`_approve(sender, _msgSender(), currentAllowance - amount);`), so it's not a reliable solution.

Recommendation

Only issue an `approve` for the exact amount of tokens involved in the flash loan. This approach would also surface any bugs present in the flash loan process.

Uniswap V2 `deadline` argument can be passed as `block.timestamp` only

Status **Open** Severity **Minor**

Description

Calls to Uniswap's V2 Router require passing a `deadline` argument that expresses the last timestamp when the call to the respective function should be considered valid:

[code/contracts/LPVault.sol#L345](#)

```
(uint collat0_amt, uint collat1_amt) = ROUTER.removeLiquidity(address(COLLAT_0), address(COLLA
```

[code/contracts/LPVault.sol#L433-L444](#)

```
swapped = ROUTER.swapExactTokensForTokens(getSwapAmt(asset0_balance - added0, r0 +
(uint256 added0_2, uint256 added1_2, uint256 liquidity) = ROUTER.addLiquidity(addr
added0 += added0_2;
added1 += added1_2;
collatReturned += liquidity;
} else if (asset1_balance >= asset1_needed) {
(added0, added1, collatReturned) = ROUTER.addLiquidity(address(COLLAT_0), address(
address[] memory path = new address[](2);
path[0] = address(COLLAT_1);
path[1] = address(COLLAT_0);
swapped = ROUTER.swapExactTokensForTokens(getSwapAmt(asset1_balance - added1, r1 +
(uint256 added0_2, uint256 added1_2, uint256 liquidity) = ROUTER.addLiquidity(addr
```

In [UniswapV2Router02.sol](#), the `deadline` value is passed to the `ensure` modifier:

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

Since the `deadline` value is checked against the current block's timestamp with a `>=` sign, it's safe for an external contract to only pass the `block.timestamp` value when calling such functions.

Passing a `deadline` as a future timestamp is helpful for front ends that cannot guarantee that the transaction will be mined in the immediate next block.

Recommendation

Replace all `deadline` values that specify a value in the future (`block.timestamp + 3600`) with `block.timestamp`.

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.

Sūrya's Description Report

Files Description Table

File Name	SHA-1 Hash
code/contracts/aux/FlashLoan_Uniswap.sol	98736cb682e16a1900b8902bc7ee
code/contracts/aux/ETHUSDCLP_Oracle.sol	d3a54249a5adb857033941520dbc
code/contracts/aux/HistoricPriceConsumerV3.sol	14fab40089d63046bf2701ce629af8
code/contracts/aux/FlashLoan_Unwind.sol	33f30310d8b3241c462c3824617a6
code/contracts/aux/WindUniswap.sol	770cde1e73eefdf1701718c7d4e5d
code/contracts/LPVault.sol	23cb7ccd0af9d23a01f0e14a7a3aa

Contracts Description Table



Contract	Type	Base Contract
L	Function Name	Visibility
FlashLoanReceiverBase	Implementation	IFlashLoanReceiver
L		Publisher
L	executeOperation	External
L	flashLeverage	External
UniswapV2Pair	Interface	
L	totalSupply	External
L	getReserves	External
LPOracle	Implementation	
L	decimals	External
L	sqrt	Internal
L	getAnswer	Publisher
L	latestAnswer	External
AggregatorInterface	Interface	
L	latestAnswer	External
L	latestTimestamp	External
L	latestRound	External
L	getAnswer	External
L	getTimestamp	External
AggregatorV3Interface	Interface	
L	decimals	External
L	description	External
L	version	External
L	getRoundData	External
L	latestRoundData	External
AggregatorV2V3Interface	Interface	AggregatorV2V3Interface AggregatorV2V3Interface

Contract	Type	Block Explorer
AggregatorProxy	Interface	AggregatorProxy
L	phaseId	External
HistoricalPriceConsumerV3	Interface	
L	getPriceAfterTimestamp	External
L	getLatestPriceX1e6	External
HistoricalPriceConsumerV3_1	Implementation	
L	getHistoricalPrice	Internal
L	getLatestPrice	Internal
L	getPriceAfterTimestamp	External
L	findBlockSamePhase	Internal
L	checkAggregatorDecimals	External
L	getLatestPriceX1e6	Public
HistoricalPriceConsumerV3_RATIO	Implementation	
L		Public
L	getQuotePrice	Public
L	getQuoteMantissa	Internal
L	getHistoricalPrice	Public
L	getLatestPrice	Public
L	findPriceAfterTimestamp	Public
L	getPriceAfterTimestamp	Public
L	findBlockSamePhase	Public
L	checkAggregatorDecimals	External
L	getLatestPriceX1e6	Public
HistoricalPriceConsumerV3_FIXEDPRICE	Implementation	
L		Public
L	setPrice	External
L	setOracle	External
L	getLatestPrice	Public

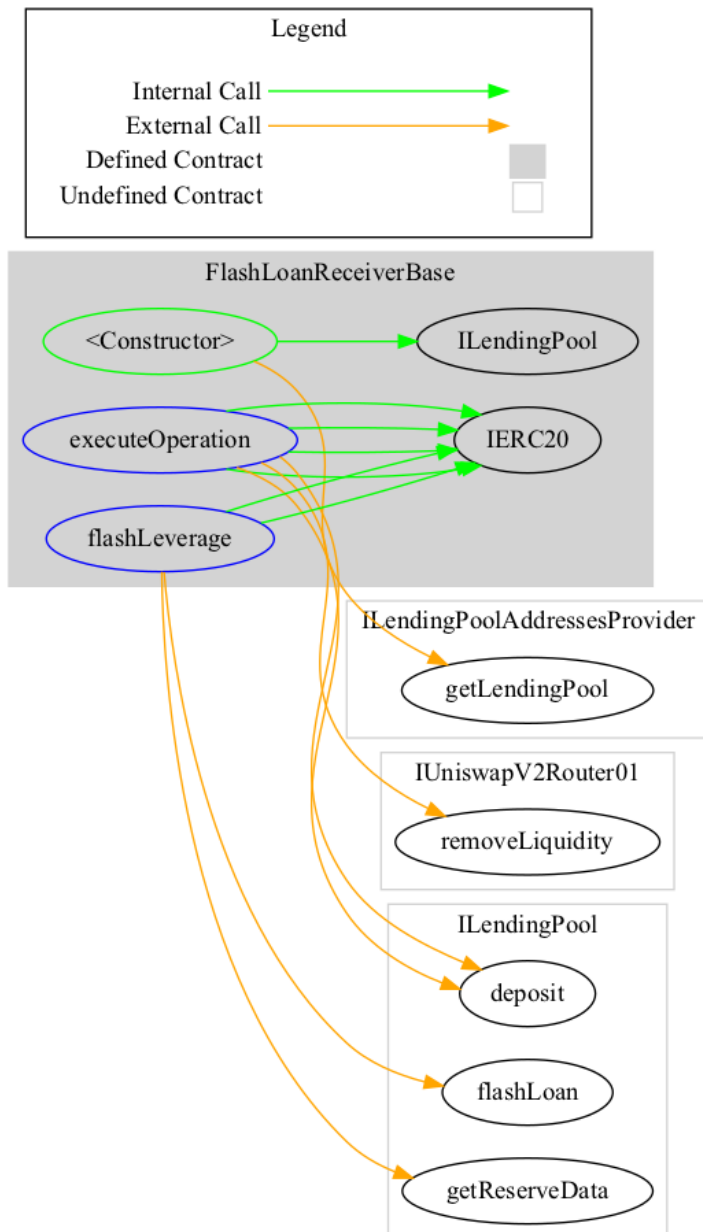
Contract	Type	Base
L	getPriceAfterTimestamp	Publisher
L	getLatestPriceX1e6	Publisher
L	checkAggregatorDecimals	Extension
FlashLoanReceiverBase_Unwind	Implementation	IFlashLoanReceiver
L		Publisher
L	executeOperation	Extension
L	flashDeleverage	Extension
L	clearSingleSidedDebt	Extension
WindUniV2LP	Implementation	Ownable
L		Publisher
L	levUp	Extension
L	levDown	Extension
L	withdraw	Extension
VaultHook	Interface	
L	onInit	Extension
L	onSettle	Extension
LPVault	Implementation	ReentrancyGuard, ERC20
L		Publisher
L	priceWithinRangeX1e6	Integer
L	name	Publisher
L	symbol	Publisher
L	sqrt	Integer
L	epochExpiry	Publisher
L	deposit	Extension
L	depositOnBehalf	Publisher

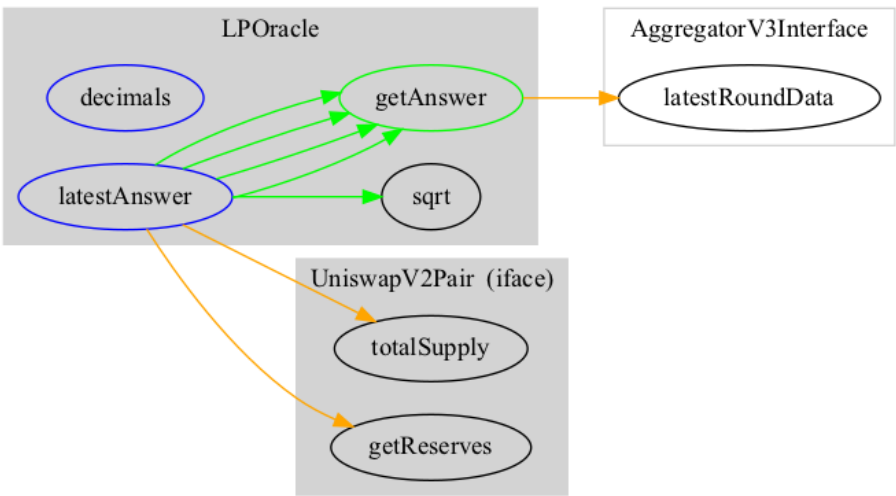
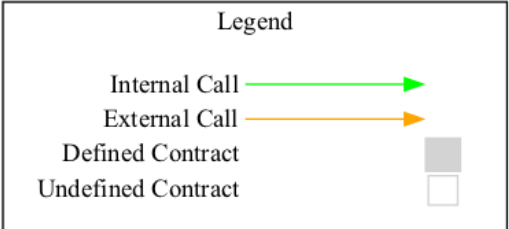
Contract	Type	Block
L	initWithdraw	External
L	initWithdrawOnBehalf	Public
L	directWithdraw	Internal
L	withdraw	External
L	withdrawOnBehalf	Public
L	initNewRound	External
L	getSwapAmt	Internal
L	settle	Public
L	setOwner	External
L	setExpiry	External
L	setMaxCap	External
L	setMaker	External
L	setPriceReader	External
L	emergencyWithdraw	External
L	setAllowInteraction	External
L	setValidator	External
L	setVaultHook	External
L	syncBalance	External
L	setAaveAddressProvider	External
L	depositIntoLendingPool	External
L	withdrawFromLendingPool	External
L	setFeeCollector	External
L	setFeePerYearX1e6	External

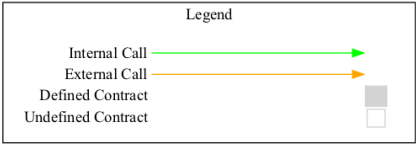
Legend

Symbol	Meaning
	Function can modify state
	Function is payable

Graphs



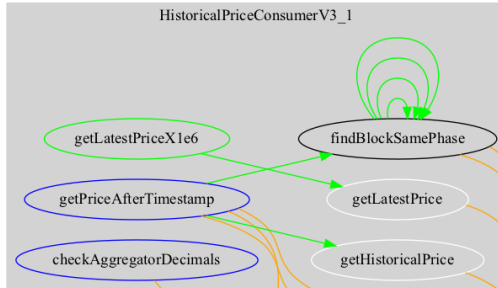




HistoricalPriceConsumerV3_FIXEDPRICE



HistoricalPriceConsumerV3 (iface)



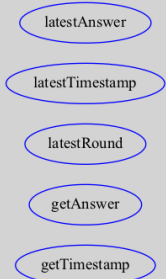
HistoricalPriceConsumerV3_RATIO



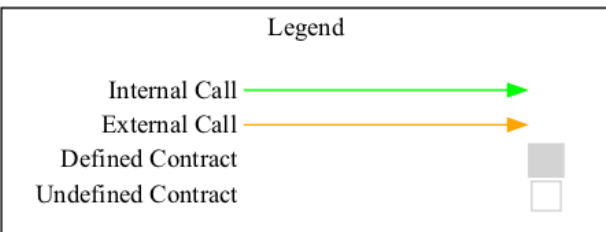
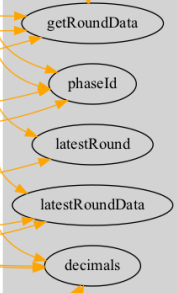
AggregatorV3Interface (iface)



AggregatorInterface (iface)

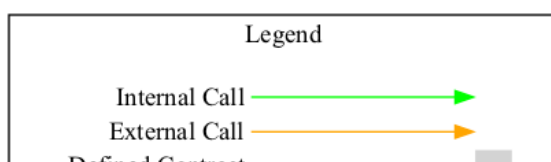
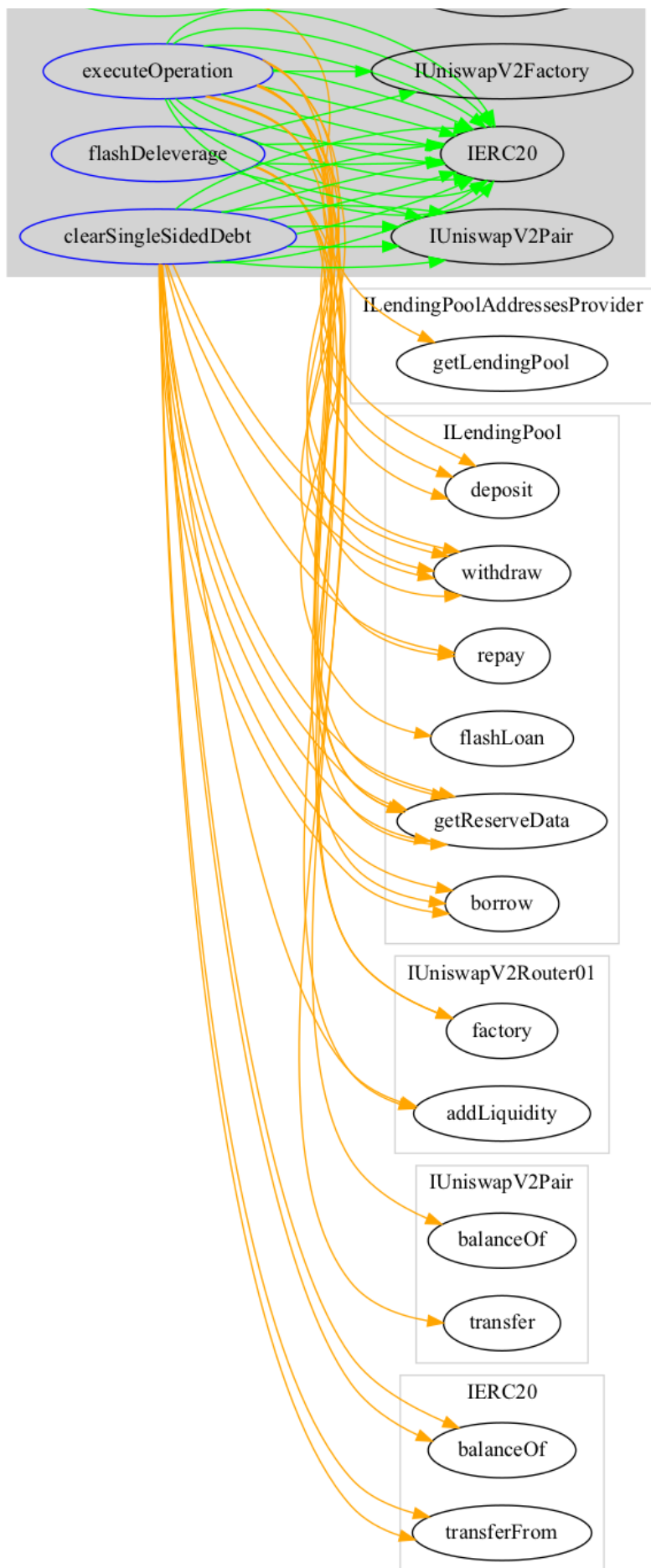


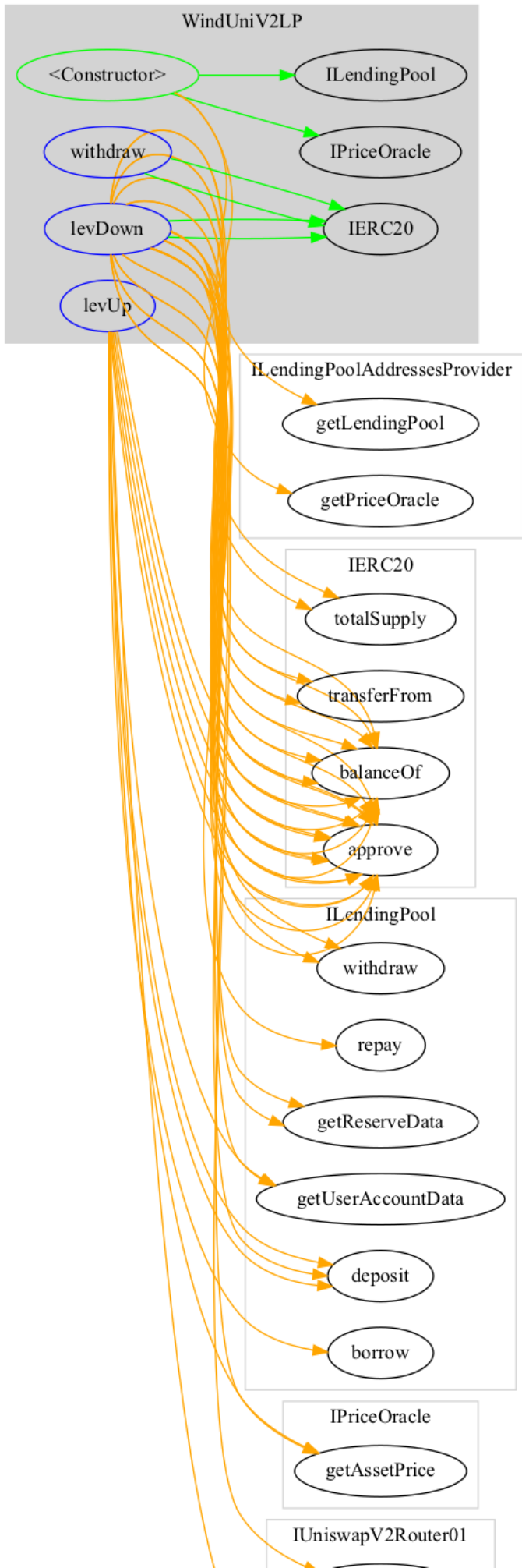
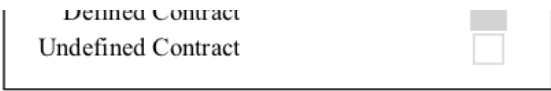
AggregatorProxy (iface)

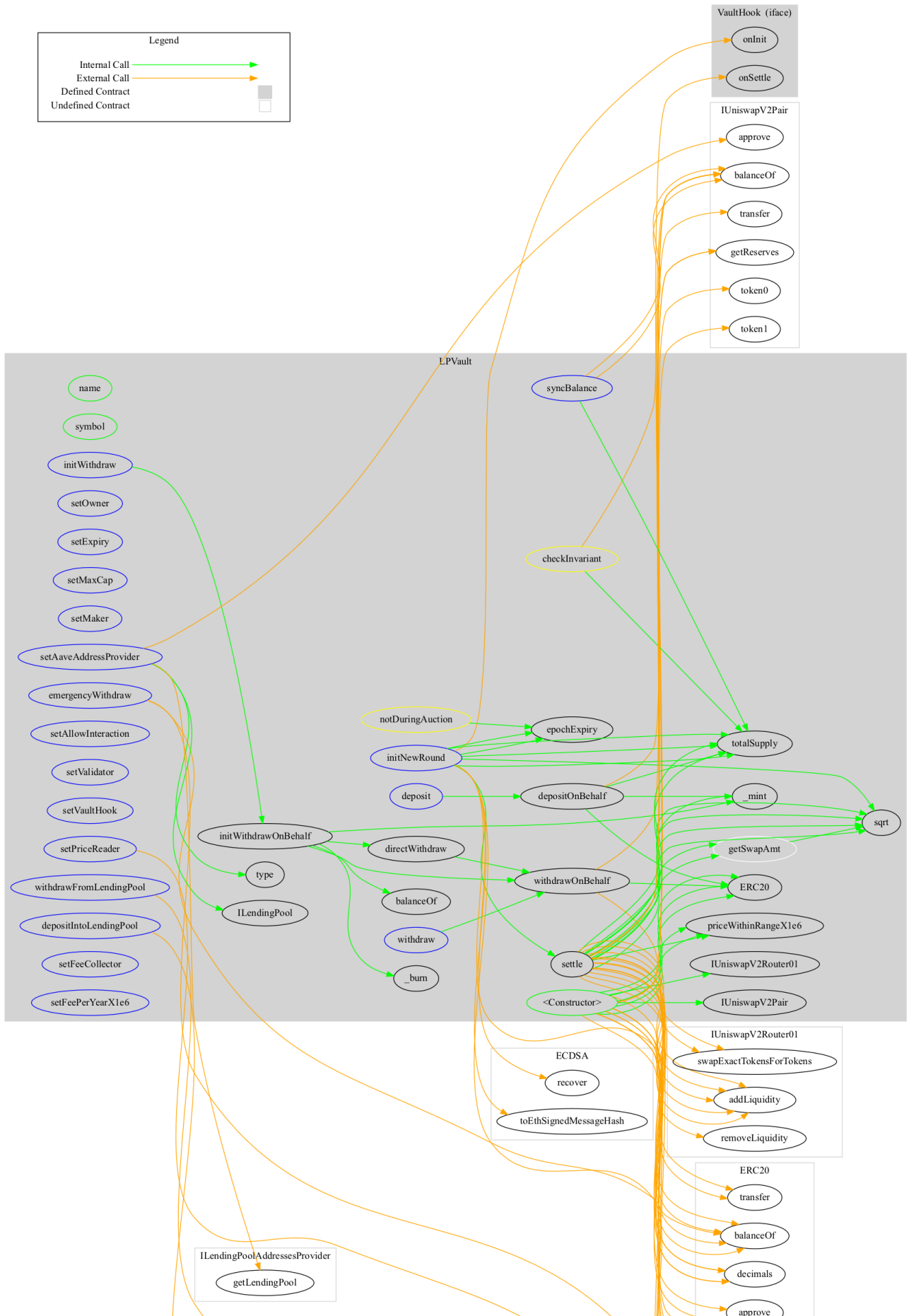
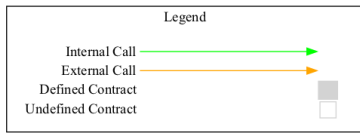
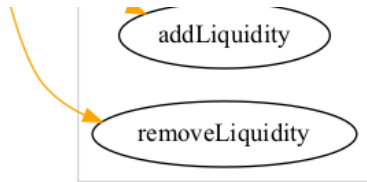


FlashLoanReceiverBase_Unwind



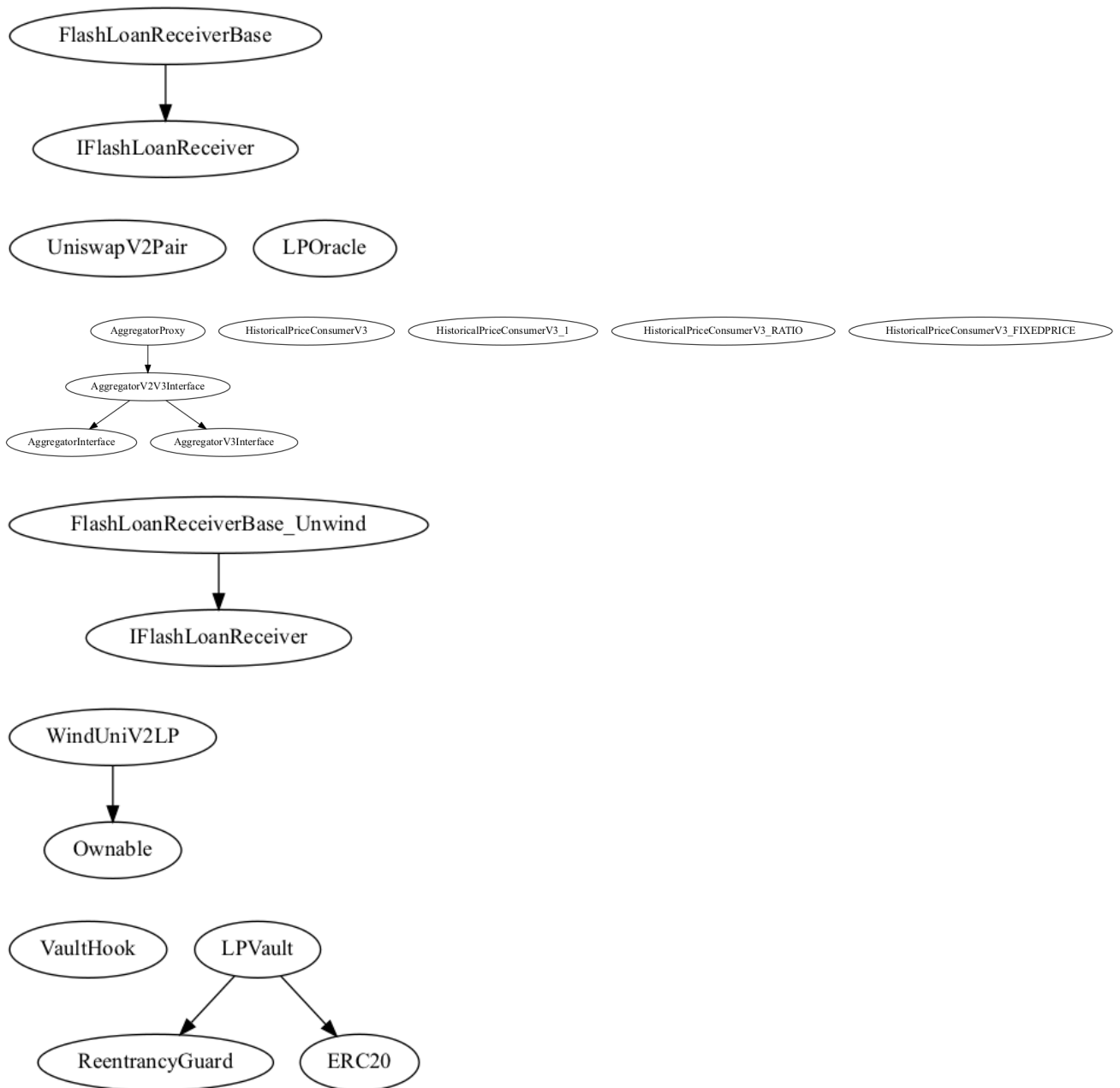








Inheritance



Describe

```
$ npx surya describe code/contracts/aux/FlashLoan_Uniswap.sol code/contracts/aux/ETHUSDCLP_Oracle.sol code
+ FlashLoanReceiverBase (IFlashLoanReceiver)
  - [Pub] <Constructor> #
  - [Ext] executeOperation #
  - [Ext] flashLeverage #

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

+ LPOracle
  - [Ext] decimals
  - [Int] sqrt
  - [Pub] getAnswer
  - [Ext] latestAnswer

+ [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

+ FlashLoanReceiverBase_Unwind (IFlashLoanReceiver)

- [Pub] <Constructor> #
- [Ext] executeOperation #
- [Ext] flashDeleverage #
- [Ext] clearSingleSidedDebt #

+ WindUniV2LP (Ownable)

- [Pub] <Constructor> #
- [Ext] levUp #
- [Ext] levDown #
- [Ext] withdraw #
 - modifiers: onlyOwner

+ [Int] VaultHook

- [Ext] onInit #
- [Ext] onSettle #

+ LPVault (ReentrancyGuard, ERC20)

- [Pub] <Constructor> #
- [Int] priceWithinRangeX1e6
- [Pub] name
- [Pub] symbol
- [Int] sqrt
- [Pub] epochExpiry
- [Ext] deposit #
- [Pub] depositOnBehalf #
 - modifiers: nonReentrant,notDuringAuction,checkInvariant
- [Ext] initWithdraw #
- [Pub] initWithdrawOnBehalf #
 - modifiers: nonReentrant,notDuringAuction,checkInvariant
- [Int] directWithdraw #
- [Ext] withdraw #
- [Pub] withdrawOnBehalf #
 - modifiers: checkInvariant

```
- [Ext] initNewRound #
- [Int] getSwapAmt
- [Pub] settle #
  - modifiers: nonReentrant
- [Ext] setOwner #
- [Ext] setExpiry #
- [Ext] setMaxCap #
- [Ext] setMaker #
- [Ext] setPriceReader #
- [Ext] emergencyWithdraw #
- [Ext] setAllowInteraction #
- [Ext] setValidator #
- [Ext] setVaultHook #
- [Ext] syncBalance #
- [Ext] setAaveAddressProvider #
- [Ext] depositIntoLendingPool #
- [Ext] withdrawFromLendingPool #
- [Ext] setFeeCollector #
- [Ext] setFeePerYearX1e6 #
```

(\$) = payable function

= non-constant function

License

This report falls under the terms described in the included [LICENSE](#).