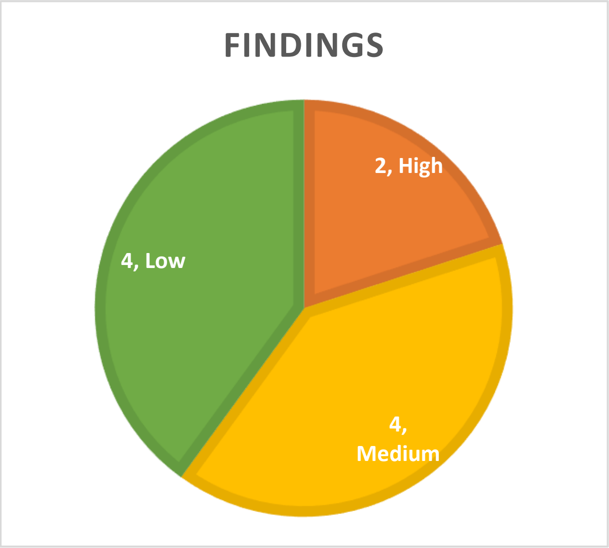
Trust Security

Smart Contract Audit

rysk UniswapV3RangeOrderReactor

23/12/2022

Executive summary

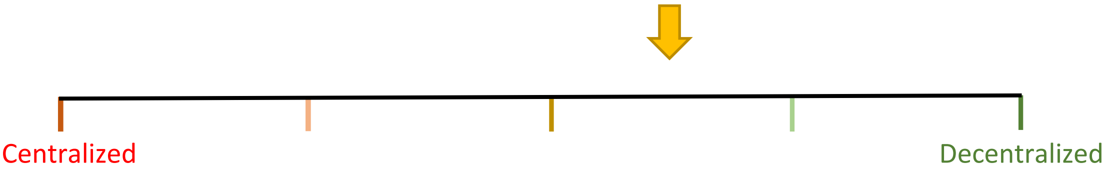


Category	Hedging
Audited file count	1
Lines of Code	433
Auditor	Trust
Time period	21/12-23/12

Findings

Severity	Total	Fixed	Fix issues	Acknowledged	Disputed
High	2	2	1	-	-
Medium	4	4	0	-	-
Low	4	3	1	1	-

Centralization score



Signature

Digitally signed by Trust
Date: 2023-01-09 17:32:01

Document properties

Versioning

Version	Date	Description
0.1	23/12/2022	Client report
0.2	09/01/2023	Mitigation review

Contact

Or Cyngiser, AKA Trust

boss@trustindistrust.com

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate. Following the initial audit, the parties have reengaged for another round where the mitigations were reviewed.

Scope

- contracts/hedging/UniswapV3RangeOrderReactor.sol

Repository details

- **Repository URL:** https://github.com/rysk-finance/dynamic-hedging
- **Commit hash:** 7c329955fd443a56111d45e73f31ef64fa4b0496
- **Mitigation review hash:** 9d81909f4ee1507e6900fd3ae806d313efddca89

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Trust is the leading auditor at competitive auditing service Code4rena, reported several critical issues to Immunefi bug bounty platform and is currently a Code4rena judge.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from many different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	Rating	Comments
Code complexity	**Good**	Project is neatly structured and manages to keep state simple
Documentation	**Excellent**	Project is mostly very well documented.
Best practices	**Excellent**	Project adheres to best practices and industry standards
Centralization risks	**Moderate**	Project has some centralization concerns

Findings

High severity findings

TRST-H-1 createUniswapRangeOrder() charges manager instead of pool

- **Category:** Logic flaw
- **Source:** UniswapV3RangeOrderReactor.sol
- **Status:** Fixed

Description

_createUniswapRangeOrder() can be called either from manager flow, with *createUniswapRangeOrder(),* or pool-induced from *hedgeDelta()*. The issue is that the function assumes the sender is the parentLiquidityPool, for example:

```
if (inversed && balance < amountDesired) {
    // collat = 0
    uint256 transferAmount = amountDesired - balance;
    uint256 parentPoolBalance =
ILiquidityPool(parentLiquidityPool).getBalance(address(token0));
    if (parentPoolBalance < transferAmount) { revert
CustomErrors.WithdrawExceedsLiquidity(); }
    SafeTransferLib.safeTransferFrom(address(token0), msg.sender,
address(this), transferAmount);
}
```

Balance check is done on pool, but money is transferred from sender. It will cause the order to use manager's funds.

```
function createUniswapRangeOrder(
    RangeOrderParams calldata params,
    uint256 amountDesired
) external {
    require(!_inActivePosition(), "RangeOrder: active position");
    _onlyManager();
    bool inversed = collateralAsset == address(token0);
    _createUniswapRangeOrder(params, amountDesired, inversed);
}
```

Recommended mitigation

Ensure safeTransfer from uses parentLiquidityPool as source.

Team response

Fixed

Mitigation Review

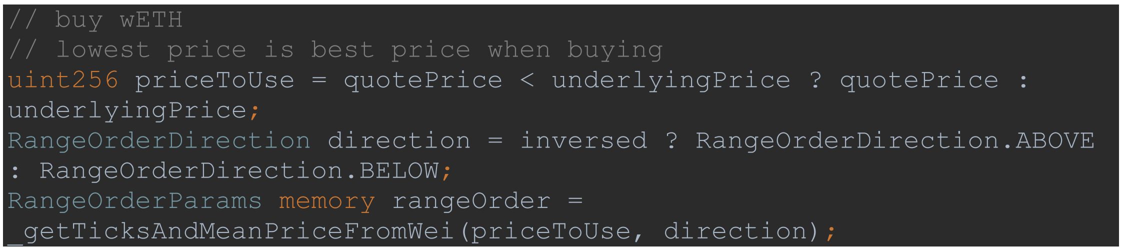
The transfers are now implemented in *_transferFromParentPool()* which ensures **from** is always parentLiquidityPool.

TRST-H-2 hedgeDelta() priceToUse is calculated wrong, which causes bad hedges

- **Category:** Logic flaw
- **Source:** UniswapV3RangeOrderReactor.sol
- **Status:** Partially fixed

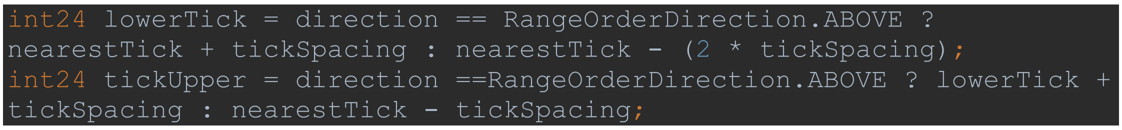
Description

When _delta parameter is negative for *hedgeDelta()*, **priceToUse** will be the minimum between quotePrice and underlyingPrice.



```
// buy wETH
// lowest price is best price when buying
uint256 priceToUse = quotePrice < underlyingPrice ? quotePrice :
underlyingPrice;
RangeOrderDirection direction = inversed ? RangeOrderDirection.ABOVE
: RangeOrderDirection.BELOW;
RangeOrderParams memory rangeOrder =
_getTicksAndMeanPriceFromWei(priceToUse, direction);
```

This works fine when direction is **BELOW**, because the calculated lowerTick and upperTick from _getTicksAndMeanPriceFromWei are guaranteed to be lower than current price.



```
int24 lowerTick = direction == RangeOrderDirection.ABOVE ?
nearestTick + tickSpacing : nearestTick - (2 * tickSpacing);
int24 tickUpper = direction ==RangeOrderDirection.ABOVE ? lowerTick +
tickSpacing : nearestTick - tickSpacing;
```

Therefore, the fulfill condition is not true and we mint from the correct base. However, when direction is **ABOVE,** it is possible that the oracle supplied price (underlyingPrice) is low enough in comparison to pool price, that the fulfill condition is already active. In that case, the contract tries to mint from the wrong asset which will cause the wrong tokens to be sent in. In effect, the contract is not hedging.

A similar situation occurs when _delta parameter is greater than zero.

Recommended mitigation

Verify the calculated priceToUse is on the same side as pool-calculated tick price.

Team response

Fixed

Mitigation Review

The issue has been solved in the **_delta < 0** branch of *hedgeDelta()*, however it still exists in the else clause. Make sure to use the new *getPriceToUse()* utility in both cases.

Medium severity findings

TRST-M-1 multiplication overflow in getPoolPrice() likely

- **Category:** overflow
- **Source:** UniswapV3RangeOrderReactor.sol
- **Status:** Fixed

Description

getPoolPrice() is used in hedgeDelta to get the price directly from Uniswap v3 pool:

```
function getPoolPrice() public view returns (uint256 price, uint256
inversed){
    (uint160 sqrtPriceX96, , , , , , ) = pool.slot0();
    uint256 p = uint256(sqrtPriceX96) * uint256(sqrtPriceX96) * (10
** token0.decimals());
    // token0/token1 in 1e18 format
    price = p / (2 ** 192);
    inversed = 1e36 / price;
}
```

The issue is that calculation of **p** is likely to overflow. sqrtPriceX96 has 96 bits for decimals, *10** token0.decimals()* will have 60 bits when decimals is 18, therefore there is only *(256 – 2 * 96 – 60) / 2 = 2* bits for non-decimal part of sqrtPriceX96.

Recommended mitigation

Consider converting the sqrtPrice to a 60x18 format and performing arithmetic operations using the PRBMathUD60x18 library.

Team response

Fixed

Mitigation Review

Calculations are now performed safely using the standard FullMath library.

TRST-M-2 Hedging won't work if token1.decimals() < token0.decimals()

- **Category:** overflow
- **Source:** UniswapV3RangeOrderReactor.sol
- **Status:** Fixed

Description

_*tickToToken0PriceInverted()* performs some arithmetic calculations. It's called by _*getTicksAndMeanPriceFromWei*(), which is called by *hedgeDelta()*. This line can overflow:

```
uint256 intermediate = inWei.div(10**(token1.decimals() -
token0.decimals()));
```

Also, this line would revert even if the above calculation was done correctly:

```
meanPrice = OptionsCompute.convertFromDecimals(meanPrice,
token0.decimals(), token1.decimals());
```

```
function convertFromDecimals(uint256 value, uint8 decimalsA, uint8
decimalsB)
   internal
   pure
   returns (uint256) {
   if (decimalsA > decimalsB) {
      revert();
   }
…
```

The impact is that when token1.decimals() < token0.decimals(), the contract's main function is unusable.

Recommended mitigation

Refactor the calculation to support different decimals combinations. Additionally, add more comprehensive tests to detect similar issues in the future.

Team response

Fixed

Mitigation Review

The code has been refactored, there is no longer risk of overflow.

TRST-M-3 Overflow danger in _sqrtPriceX96ToUint

- **Category:** overflow
- **Source:** UniswapV3RangeOrderReactor.sol
- **Status:** Fixed

Description

_sqrtPriceX96ToUint will only work when the non-fractional component of sqrtPriceX96 takes up to 32 bits. This represents a price ratio of 18446744073709551616. With different token digits it is not unlikely that this ratio will be crossed which will make hedgeDelta() revert.

```
function _sqrtPriceX96ToUint(uint160 sqrtPriceX96)
    private
    pure
    returns (uint256)
{
    uint256 numerator1 = uint256(sqrtPriceX96) *
uint256(sqrtPriceX96);
    return FullMath.mulDiv(numerator1, 1, 1 << 192);
}
```

Recommended mitigation

Perform the multiplication after converting the numbers to 60x18 variables.

Team response

Fixed

Mitigation Review

New utility function *sqrtPriceX96ToUint* correctly uses SafeMath, and also multiplies in a different order depending on price size to ensure no overflows occur:

```
if (sqrtPrice > Q96) {
    uint256 sqrtP = FullMath.mulDiv(sqrtPrice, 10 ** token0Decimals, 
Q96);
    return FullMath.mulDiv(sqrtP, sqrtP, 10 ** token0Decimals);
} else {
    uint256 numerator1 = FullMath.mulDiv(sqrtPrice, sqrtPrice, 1);
    uint256 numerator2 = 10 ** token0Decimals;
    return FullMath.mulDiv(numerator1, numerator2, 1 << 192);
}
```

TRST-M-4 hedgeDelta(0) doesn't behave properly

- **Category:** Logic flaw
- **Source:** UniswapV3RangeOrderReactor.sol
- **Status:** Fixed

Description

hedgeDelta() is called again by the pool when the exposure to underlying asset needs to change. If it was previously non-zero and the pool wishes to reset the delta to zero, *hedgeDelta(0)* would be called. Unfortunately, it will never execute.

Flow will enter the sell wETH branch and call _*createUniswapRangeOrder*() with 0 delta. Eventually it will try minting a UniswapV3 position with 0 liquidity, which reverts at the Uniswap level.

As a result, the previous exposure remains as *_yankRangeOrderLiquidity()* is not called.

Recommended mitigation

Add branching logic for hedgeDelta. If delta is 0, do nothing.

Team response

Fixed

Mitigation Review

hedgeDelta() now correctly implements an early-exit in case **_delta** is 0.

Low severity findings

TRST-L-1 createUniswapRangeOrder() does not validate direction for hedge

- **Category:** safety checks
- **Source:** UniswapV3RangeOrderReactor.sol

- **Status:** Acknowledged

Description

_createUniswapRangeOrder() is an internal function that receives parameters for hedge action, including lower/upper tick and direction. It can be called from *hedgeDelta()*, in that case parameters are ensured to be correct by the in-contract creation. However, when called from *createUniswapRangeOrder(),* manager is responsible for passing these params. They can easily get wrong the RangeOrderDirection parameter, which will make the hedge only fulfillable from the wrong side. It is also not checked that lower tick < upper tick, but UniswapV3 logic ensures that property.

Recommended mitigation

Insert validity checks for *createUniswapRangeOrder()* parameters.

Team response

Manager may need to place an order that is outside the scope of a normal order according to hedgeDelta this includes orders that maybe in range or the on the opposite side of what the delta would dictate. Manager can also withdraw range liquidity at any time using exitActiveRangeOrder

Mitigation Review

As long as described behavior is intended and documented, it is not an issue.

TRST-L-2 Insufficient dust checks

- **Category:** Logical flaw
- **Source:** UniswapV3RangeOrderReactor.sol
- **Status:** Fixed

Description

In *hedgeDelta()*, there is a dust check in the case of sell wETH order:

```
// sell wETH
uint256 wethBalance = inversed ? amount1Current : amount0Current;
if (wethBalance < minAmount) return 0;
```

However, the actual used amount is _delta.

```
uint256 deltaToUse = _delta > int256(wethBalance) ? wethBalance :
uint256(_delta);
_createUniswapRangeOrder(rangeOrder, deltaToUse, inversed);
```

The check should be applied on deltaToUse rather than wethBalance because it will be the minimum of wethBalance and _delta.

Additionally, there is no corresponding check for minting with collateral in case **_delta** is negative.

Recommended mitigation

Correct current dust checks and add them also in the if clause.

Team response

This feature is more useful on ethereum mainnet than L2 will consider if it makes sense to implement dust check on collateral size as well

Mitigation Review

The dust check is now applied on **deltaToUse**. It is up to the project if they wish to perform a dust check when **_delta** is negative.

TRST-L-3 Lack of logging in important functions

- **Category:** Missing events
- **Source:** UniswapV3RangeOrderReactor.sol
- **Status:** Fixed + new issue

Description

For the sake of transparency, it is recommended to emit events during maintenance transfer of funds into and out of contracts. Make sure to add events for *withdraw(), recoverETH()* and *recoverERC20().*

Recommended mitigation

Add the events listed above.

Team response

Fixed

Mitigation Review

The issue was fixed with additional logging. However, the fix introduced an issue. In the event that logs withdraw, if withdrawal amount is greater than balance than the log will be incorrect.

```
if (_amount <= balance) {
    SafeTransferLib.safeTransfer(ERC20(collateralAsset), msg.sender,
_amount);
    emit Withdraw(_amount);
    // return in collateral format
    return _amount;
} else {
    SafeTransferLib.safeTransfer(ERC20(collateralAsset), msg.sender,
balance);
    emit Withdraw(_amount);
    // return in collateral format
    return balance;
}
```

Correct behavior would be to log **balance**.

TRST-L-4 _getUnderlyingBalances() does unnecessary computation when not in active position

- **Category:** Excessive gas usage
- **Source:** UniswapV3RangeOrderReactor.sol
- **Status:** Fixed

Description

If the RangeOrderReactor contract is not currently active, it should simply return the current token balances. However, it does a lot of expensive logic to calculate position value.

```
(
    uint128 liquidity,
    uint256 feeGrowthInside0Last,
    uint256 feeGrowthInside1Last,
    uint128 tokensOwed0,
    uint128 tokensOwed1
) = pool.positions(_getPositionID());
// compute current holdings from liquidity
(amount0Current, amount1Current) =
LiquidityAmounts.getAmountsForLiquidity(
    sqrtRatioX96,
    currentPosition.activeLowerTick.getSqrtRatioAtTick(),
    currentPosition.activeUpperTick.getSqrtRatioAtTick(),
    liquidity
);
// compute current fees earned
uint256 fee0 =
    _computeFeesEarned(true, feeGrowthInside0Last, tick, liquidity) +
        uint256(tokensOwed0);
uint256 fee1 =
    _computeFeesEarned(false, feeGrowthInside1Last, tick, liquidity)
+
        uint256(tokensOwed1);
```

Recommended mitigation

Perform early exit in case position is not active.

Team response

Fixed

Mitigation Review

Issue was addressed with correct early exit.

Additional recommendations

More comprehensive testing

The current test suite does not stress the contract in many important ways. It needs to create a variety of pools, with different tokens, token decimals and inversion. Consider fuzz testing the fulfillment and hedgeDelta() functions.

Safety checks

The contract is somewhat lacking in safety checks. fulfillActiveRangeOrder does not verify the contract is in active position. Addresses should not be zero. The oracle calculated price should be close to pool-generated price. Additional checks will increase the robustness of the contract when moving forward.

Hedging assumptions

Hedging is only activated when crossing the ticks into active territory. If price stays on the same side, the LMT order won't execute. This should be clearly stated as a limitation of the reactor.

Centralization risks

Governor has unlimited access to contract's funds

Governor is able to call recoverETH(), recoverERC20() and exitActiveRangeOrder(). It introduces significant risks in the event of a private key compromise or a rug pull. The recommendation is to delegate complete access to the parent pool and that Governor is only able to get delayed access to the funds.

Changes to onlyAuthorizedFulfill take effect immediately

Owner can lock access to fulfillActiveRangeOrder() without prior warning. Such an ability may catch users off guard, so it is best to implement a delay.

Manager is able to create arbitrary orders

Manager is able to call createUniswapRangeOrder() with controlled RangeOrderParams, meaning it can be used for a completely different use case than hedging strategy. It is recommended to allow only very specific parameters to be controlled by manager, such as tick width.