DefiEdge Audit Report

Sep 6, 2022

Table of Contents

Summary

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

The auditing process pays special attention to the following considerations:

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

Overview

Project Summary

Project Name	**DefiEdge**
Codebase	**https://github.com/unbound-finance/defiedge-core**
Commit	**356410ded8a68f44b7583d77355c8879a2e08d93**
Language	**Solidity**

Audit Summary

Delivery Date	**Sep 6, 2022**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**20**

WP-C1: Attacker can manipulate the price of the primary tick and mint more shares to steal funds from the `Strategy`

Critical

Issue Description

When the Strategy has more than 1 tick, the first tick will be the primary tick and all the new `mint()` will add liquidity to that tick.

`mint()` calls `mintLiquidity()` on the primary tick, which will pull funds from the caller's wallet according to the current proportion of the tick and the desired `_amount0` and `_amount1` .

`mintLiquidity()` returns the actual amounts used (`amount0` , `amount1`) for the liquidity added to the primary tick:

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/base/UniswapV3LiquidityManager.sol#L47-L69

```
function mintLiquidity(
    int24 _tickLower,
    int24 _tickUpper,
    uint256 _amount0,
    uint256 _amount1,
    address _payer
) internal returns (uint256 amount0, uint256 amount1) {
    uint128 liquidity = LiquidityHelper.getLiquidityForAmounts(
        pool,
        _tickLower,
        _tickUpper,
        _amount0,
        _amount1
    );
    // add liquidity to Uniswap pool
    (amount0, amount1) = pool.mint(
        address(this),
        _tickLower,
        _tickUpper,
        liquidity,
        abi.encode(MintCallbackData({payer: _payer, pool: pool}))
    );
}
```

The returned `amount0` , `amount1` will then be used for `issueShare()` .

The amount of shares to be minted is calculated based on token with the bigger amount in face value.

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/libraries/ShareHelper.sol#L26-L74

```
function calculateShares(
    FeedRegistryInterface _registry,
    IUniswapV3Pool _pool,
    IStrategyManager _manager,
    bool[2] memory _useTwap,
    uint256 _amount0,
    uint256 _amount1,
    uint256 _totalAmount0,
    uint256 _totalAmount1,
    uint256 _totalShares
) public view returns (uint256 share) {

    require(_amount0 > 0 && _amount1 > 0, 'INSUFFICIENT_AMOUNT');

    _amount0 = OracleLibrary.normalise(_pool.token0(), _amount0);
    _amount1 = OracleLibrary.normalise(_pool.token1(), _amount1);
    _totalAmount0 = OracleLibrary.normalise(_pool.token0(), _totalAmount0);
    _totalAmount1 = OracleLibrary.normalise(_pool.token1(), _totalAmount1);

    // price in USD
    uint256 token0Price = OracleLibrary.getPriceInUSD(
        _pool,
        _registry,
        _pool.token0(),
        _useTwap,
        _manager
    );

    uint256 token1Price = OracleLibrary.getPriceInUSD(
        _pool,
        _registry,
        _pool.token1(),
        _useTwap,
        _manager
    );

    if (_totalShares > 0) {

        if(_amount0 < _amount1){
            share = FullMath.mulDiv(_amount1, _totalShares, _totalAmount1);
        } else {
            share = FullMath.mulDiv(_amount0, _totalShares, _totalAmount0);
        }

    } else {
        share = ((token0Price.mul(_amount0)).add(token1Price.mul(_amount1)))
            .div(DIVISOR);
    }
}
```

However, since the Strategy can have more than 1 tick, and each tick may have a different proportion ratio between `token0` and `token1` .

The primary tick's proportion ratio can be different from the avg proportion ratio of the whole Strategy.

In which case, the shares minted can be fewer or larger than expected.

An attacker can exploit this by manipulating the price of the primary tick with flashloan, getting shares at a lower cost, and burning the shares to steal funds from the Strategy.

PoC

Given:

- The Strategy has 2 positions:
  - `tick0` = {amount0: `0.001 DAI` , amount1: `1 ETH` }
  - `tick1` = {amount0: `10000 DAI` , amount1: `0.001 ETH` }
- Current `_totalShares` : 1e18

The attacker can:

1. `mint()` with amount0 = `0.001 DAI` , amount1 = `1 ETH` ;
2. `share` minted = `amount1 * _totalShares / _totalAmount1` ⊠ `1e18` ;
3. `burn()` all shares and received `1 ETH` and `5000 DAI` .

The attacker's profit is `5000 DAI` .

The precondition of this attack is that the primary tick's proportion ratio is different from the overall ratio.

While regular market movements can naturally create such conditions, it's not very practical to wait until such condition is matured by nature.

The desired conditions can be created by manipulating the price of the Uniswap v3 pool with flashloan.

When the cost of the price manipulation is lower than the profit, this attack can also be valid.

Recommendation

Consider calculating the required amounts (`amount0` and `amount1`) proportional to the overall ratio. The unused funds from `addLiquidity()` should be left in the balance.

Status

✓ **Fixed**

WP-H2: Newly collected fees should be considered in `burn()`

High

Issue Description

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/DefiEdgeStrategy.sol#L173-L242

```
173  function burn(
174      uint256 _shares,
175      uint256 _amount0Min,
176      uint256 _amount1Min
177  ) external returns (uint256 collect0, uint256 collect1) {
178
179      require(manager.isUserWhiteListed(msg.sender), "UA");
180
181      // check if the user has sufficient shares
182      require(balanceOf(msg.sender) >= _shares && _shares != 0, "INS");
183
184      uint256 amount0;
185      uint256 amount1;
186
187      // give from unused amounts
188      collect0 = IERC20(token0).balanceOf(address(this));
189      collect1 = IERC20(token1).balanceOf(address(this));
190
191      uint256 _totalSupply = totalSupply();
192
193      if (collect0 > 0) {
194          collect0 = FullMath.mulDiv(collect0, _shares, _totalSupply);
195      }
196
197      if (collect1 > 0) {
198          collect1 = FullMath.mulDiv(collect1, _shares, _totalSupply);
199      }
200
201      // burn liquidity based on shares from existing ticks
202      for (uint256 i = 0; i < ticks.length; i++) {
203          Tick storage tick = ticks[i];
204
205          uint256 fee0;
```

```
            uint256 fee1;
            // burn liquidity and collect fees
            (amount0, amount1, fee0, fee1) = burnLiquidity(
                tick.tickLower,
                tick.tickUpper,
                _shares,
                0
            );

            // add to total amounts
            collect0 = collect0.add(amount0);
            collect1 = collect1.add(amount1);

            tick.amount0 = tick.amount0 >= amount0
                ? tick.amount0.sub(amount0)
                : 0;
            tick.amount1 = tick.amount1 >= amount1
                ? tick.amount1.sub(amount1)
                : 0;
        }

        // check slippage
        require(_amount0Min <= amount0 && _amount1Min <= amount1, "S");

        // burn shares
        _burn(msg.sender, _shares);

        // transfer tokens
        if (collect0 > 0) {
            TransferHelper.safeTransfer(address(token0), msg.sender, collect0);
        }
        if (collect1 > 0) {
            TransferHelper.safeTransfer(address(token1), msg.sender, collect1);
        }

        emit Burn(msg.sender, _shares, collect0, collect1);
    }
```

`burnLiquidity()` returns `amount0` = `tokensBurned0` and `amount1` = `tokensBurned1` , and the amounts of fees collected (`fee0` and `fee1`), which belong to all the share holders, including the caller.

However, in the current implementation, the fees collected alongside with the `burnLiquidity()` call are not considered as part of the liabilities to the caller.

As a result, the caller will lose part of the newly collected fees that belongs to them.

PoC

Alice is the only user of the Strategy.

1. Alice `mint()` with `100e18` token0 and `100e18` token1, received `100e18` share;
2. A few days later, a certain amount of fees are accumulated: `tokensOwed0` in all position = `10e18` , `tokensOwed1` in all position = `10e18` ;
3. Alice `burn()` all the shares (`100e18`), received `100e18` token0 and `100e18` token1.

Expected results: Alice to receive `110e18` token0 and `110e18` token1, all the principal plus the fees earned.

Recommendation

Change to:

```
uint256 _totalSupply = totalSupply();

if (total0 > collect0) {
    collect0 = collect0.add(FullMath.mulDiv(total0 - collect0, _shares, _totalSupply));
}

if (total1 > collect1) {
    collect1 = collect1.add(FullMath.mulDiv(total1 - collect1, _shares, _totalSupply));
}
```

Status

✓ Fixed

WP-M3: Wrong implementation of slippage control in `burn()`

Medium

Issue Description

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/DefiEdgeStrategy.sol#L173-L229

```
function burn(
    uint256 _shares,
    uint256 _amount0Min,
    uint256 _amount1Min
) external returns (uint256 collect0, uint256 collect1) {

    require(manager.isUserWhiteListed(msg.sender), "UA");

    // check if the user has sufficient shares
    require(balanceOf(msg.sender) >= _shares && _shares != 0, "INS");

    uint256 amount0;
    uint256 amount1;

    // give from unused amounts
    collect0 = IERC20(token0).balanceOf(address(this));
    collect1 = IERC20(token1).balanceOf(address(this));

    uint256 _totalSupply = totalSupply();

    if (collect0 > 0) {
        collect0 = FullMath.mulDiv(collect0, _shares, _totalSupply);
    }

    if (collect1 > 0) {
        collect1 = FullMath.mulDiv(collect1, _shares, _totalSupply);
    }

    // burn liquidity based on shares from existing ticks
    for (uint256 i = 0; i < ticks.length; i++) {
        Tick storage tick = ticks[i];

        uint256 fee0;
206            uint256 fee1;
207            // burn liquidity and collect fees
208            (amount0, amount1, fee0, fee1) = burnLiquidity(
209                tick.tickLower,
210                tick.tickUpper,
211                _shares,
212                0
213            );
214
215            // add to total amounts
216            collect0 = collect0.add(amount0);
217            collect1 = collect1.add(amount1);
218
219            tick.amount0 = tick.amount0 >= amount0
220                ? tick.amount0.sub(amount0)
221                : 0;
222            tick.amount1 = tick.amount1 >= amount1
223                ? tick.amount1.sub(amount1)
224                : 0;
225        }
226
227        // check slippage
228        require(_amount0Min <= amount0 && _amount1Min <= amount1, "S");
```

`_amount0Min` and `_amount1Min` of `burn()` are used for slippage control, the `burn()` transcation should revert if the `amount0` and `amount1` to be received are larger than the mimimum amounts.

However, in the current implementation, the `amount0` and `amount1` at L228 are the last values in the last iteration rather than the total amounts to be received.

The slippage control should check `collect0 >= _amount0Min` and `collect1 >= _amount1Min` instead.

Recommendation

Change to:

```
    // check slippage
    require(_amount0Min <= collect0 && _amount1Min <= collect1, "S");
```

Status

✓ Fixed

WP-M4: Unsafe `ERC20.approve()`

Medium

Issue Description

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/base/UniswapV3LiquidityManager.sol#L273-L288

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/personal/UniswapV3PrivateLiquidityManager.sol#L317-L332

```
function swap(bytes calldata data) external onlyOperator {
    (IERC20 srcToken, IERC20 dstToken, uint256 amount) = OneInchHelper
        .decodeData(IERC20(token0), IERC20(token1), data);

    require(
        (srcToken == token0 && dstToken == token1) ||
            (srcToken == token1 && dstToken == token0),
        "IA"
    );

    srcToken.approve(address(oneInchRouter), amount);

    // Interact with 1inch through contract call with data
    (bool success, bytes memory returnData) = address(oneInchRouter).call{
        value: 0
    }(data);
```

`UniswapV3LiquidityManager#swap()` calls `IERC20.approve()` before the swap call to `oneInchRouter` ,

However, there are many Weird ERC20 Tokens that won't work correctly using the standard `IERC20` interface.

For example, the `USDT` token on mainnet does not return a bool on the `approve()` method.

As a result, when calling `IERC20(USDT).approve()` , the transaction will revert with an error: ”function returned an unexpected amount of data”, as the expected return data is a bool, but it actually does not return any data.

This means that `UniswapV3LiquidityManager#swap()` doesn’t work well with one of the most popular tokens.

Recommendation

Consider using `SafeERC20.safeIncreaseAllowance()` .

Status

✓ Fixed

WP-H5: `getAUMWithFees()` PerformanceFees should be excluded from uncollected/pending trading fees on the Uni v3 pool in the AUM

High

Issue Description

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/DefiEdgeStrategy.sol#L74-L165

```
function mint(
    uint256 _amount0,
    uint256 _amount1,
    uint256 _amount0Min,
    uint256 _amount1Min,
    uint256 _minShare
)
    external
    onlyValidStrategy
    returns (
        uint256 amount0,
        uint256 amount1,
        uint256 share
    )
{
    require(manager.isUserWhiteListed(msg.sender), "UA");

    // get total amounts with fees
    (uint256 totalAmount0, uint256 totalAmount1, , ) = this
        .getAUMWithFees(false);
```

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/base/UniswapV3LiquidityManager.sol#L409-L482

```
435 if (currentLiquidity > 0) {
436     // calculate current positions in the pool from currentLiquidity
437     (uint256 position0, uint256 position1) = LiquidityHelper
438         .getAmountsForLiquidity(
439             pool,
440             tick.tickLower,
441             tick.tickUpper,
442             currentLiquidity
443         );
444 
445     // update fees earned in Uniswap pool
446     // Uniswap recalculates the fees and updates the variables when amount is
    passed as 0
447     pool.burn(tick.tickLower, tick.tickUpper, 0);
448 
449     // fees are credited as tokensOwed in Uniswap when burn is called with 0
450     //
    https://github.com/Uniswap/v3-core/blob/main/contracts/interfaces/pool/IUniswapV3PoolActions.
451     (, , , uint256 tokensOwed0, uint256 tokensOwed1) = pool
452         .positions(
453             PositionKey.compute(
454                 address(this),
455                 tick.tickLower,
456                 tick.tickUpper
457             )
458         );
459 
460     totalFee0 = totalFee0.add(tokensOwed0);
461     totalFee1 = totalFee1.add(tokensOwed1);
462 
463     amount0 = amount0.add(position0);
464     amount1 = amount1.add(position1);
465 }
```

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/base/StrategyBase.sol#L121-L123

```
121     function totalSupply() public view override returns (uint256) {
122         return _totalSupply.add(accManagementFee);
123     }
```

In the current implmentation, when the user calls `mint()` , the contract will calculate the total value of the strategy, including the pending fees with `getAUMWithFees()` , and then issue shares based on the AUM (`totalAmount0` and `totalAmount1`).

However, a portion of the pending/uncollected lp fees from the Uni v3 pool is yet to be settled (transferred out) as the performance fees.

That part of the fees should not be included in `totalAmount0` and `totalAmount1` , otherwise, the AUM will be higher than the actual value, and the price pre share is overvalued.

PoC

Given:

- `managementFee` = 0%
- token0 is `DAI`
- token1 is `LUSD`
- `protocolFee` = 0%
- `manager.performanceFee` = 0%
- `protocolPerformanceFee` = 20%

1. Alice called `mint()` with `10e18` DAI and `10e18` LUSD, received `2e17` shares
2. A few days later, the position incurred some lp fees:

- `tokensOwed0` = `1e18`
- `tokensOwed1` = `1e18`

1. Bob called `mint()` with `11e18` DAI and `11e18` LUSD, received `2e17` shares

- before the `mint` : `totalSupply` = `2e17`
- `totalAmount0` = `11e18`
- `totalAmount1` = `11e18`

1. Bob called `burn()` for `2e17` shares, received `10.9e18` DAI and `10.9e18` LUSD back (We consider the [WP-H2] is fixed here);

- protocolFeeToken0 = `2e17`
- protocolFeeToken1 = `2e17`
- amount0 = `10.5e18`

- amount1 = `10.5e18`
- fee0 = `8e17`
- fee1 = `8e17`

The expected result is to receive `11e18` DAI and `11e18` LUSD.

The actual result is `0.1e18` fewer than expected. That's because the `burn()` triggered the settlement of the performanceFees and the total amount of fees that can be redeemed based on gets lower.

Recommendation

Consider excluding the PerformanceFees from pending/uncollected lp fees in `getAUMWithFees()` .

Status

✓ Fixed

WP-M6: `DefiEdgeStrategy.sol#mint()` Improper slippage control

Medium

Issue Description

When the user `mint()` new shares, there is a parameter called `_minShare` , which represent `Minimum amount of shares to be received to the user` .

However, in the current implementation, the `_minShare` is checked against the pre-fee shares amount.

As a result:

When `managementFee` > 0, a portion of the shares is taken as `managerShare` .

https://github.com/unbound-finance/defiedge-core/blob/d8d3ead886f6d92ae6245e4e3c178d1e4f78ee01/contracts/DefiEdgeStrategy.sol#L125-L145

```
        // issue share based on the liquidity added
        share = issueShare(
            amount0,
            amount1,
            totalAmount0,
            totalAmount1,
            msg.sender
        );

        // prevent front running of strategy fee
        require(share >= _minShare, "SC");

        // price slippage check
        require(amount0 >= _amount0Min && amount1 >= _amount1Min, "S");

        uint256 _shareLimit = manager.limit();
        // share limit
        if (_shareLimit != 0) {
            require(totalSupply() <= _shareLimit, "L");
        }
        emit Mint(msg.sender, share, amount0, amount1);
```

If the `managementFee` gets updated just before the user's `mint()` transaction, the deviation between `_minShare` and the actual amount of shares received by the user can be quite large.

In other words, the user may not receive the expected `>= _minShare` amount of shares, which means the slippage control has failed for this scenario.

The value of `share` in the `Mint` event is also inaccurate.

https://github.com/unbound-finance/defiedge-core/blob/d8d3ead886f6d92ae6245e4e3c178d1e4f78ee01/contracts/base/StrategyBase.sol#L87-L119

```
function issueShare(
    uint256 _amount0,
    uint256 _amount1,
    uint256 _totalAmount0,
    uint256 _totalAmount1,
    address _user
) internal returns (uint256 share) {
    // calculate number of shares
    share = ShareHelper.calculateShares(
        chainlinkRegistry,
        pool,
        manager,
        useTwap,
        _amount0,
        _amount1,
        _totalAmount0,
        _totalAmount1,
        totalSupply()
    );

    require(share > 0, "IS");

    uint256 managerShare;
    uint256 managementFee = manager.managementFee();
    // strategy owner fees
    if (managementFee > 0) {
        managerShare = share.mul(managementFee).div(FEE_PRECISION);
        accManagementFee = accManagementFee.add(managerShare);
    }

    // issue shares
    _mint(_user, share.sub(managerShare));
}
```

Recommendation

Change to:

```
    // strategy owner fees
    if (managementFee > 0) {
        managerShare = share.mul(managementFee).div(FEE_PRECISION);
        accManagementFee = accManagementFee.add(managerShare);
        share = share.sub(managerShare);
    }

    // issue shares
    _mint(_user, share);
}
```

Status

✓ **Fixed**

WP-H7: `UniswapV3TwapLiquidityManager.getAUMWithFees()` The PerformanceFees can be escaped by calling `getAUMWithFees(true)`

High

Issue Description

https://github.com/unbound-finance/defiedge-core/blob/f3f544a287a6495d5bc6c01236ef73601b580f77/contracts/twap/base/UniswapV3TwapLiquidityManager.sol#L409-L482

```
467  // collect fees
468  if (_claimFee) {
469      (uint256 collect0, uint256 collect1) = pool.collect(
470          address(this),
471          tick.tickLower,
472          tick.tickUpper,
473          type(uint128).max,
474          type(uint128).max
475      );
476      emit FeesClaim(address(this), collect0, collect1);
477  }
```

When `getAUMWithFees` is called with `_claimFee: true` , at L469-476, the pending/uncollected LP fees will be collected into the balance without any performanceFees.

In comparison with the correct implementation, which will transfer the performance fees out, see L138 of `burnLiquidity()` :

https://github.com/unbound-finance/defiedge-core/blob/f3f544a287a6495d5bc6c01236ef73601b580f77/contracts/twap/base/UniswapV3TwapLiquidityManager.sol#L77-L139

```
118  // collect fees
119  (collect0, collect1) = pool.collect(
120      address(this),
    _tickLower,
    _tickUpper,
    type(uint128).max,
    type(uint128).max
);

fee0 = collect0 > tokensBurned0
    ? uint256(collect0).sub(tokensBurned0)
    : 0;
fee1 = collect1 > tokensBurned1
    ? uint256(collect1).sub(tokensBurned1)
    : 0;

collect0 = tokensBurned0;
collect1 = tokensBurned1;

// transfer performance fees
_transferPerformanceFees(fee0, fee1);
```

Recommendation

Consider adding `_transferPerformanceFees()` to `getAUMWithFees()` .

Status

✓ Fixed

WP-M8: sqrtRatioX96^2 can overflow and cause the malfunction of `OracleLibrary.consult()`

Medium

Issue Description

https://github.com/unbound-finance/defiedge-core/blob/d8d3ead886f6d92ae6245e4e3c178d1e4f78ee01/contracts/libraries/OracleLibrary.sol#L320-L353

```
function consult(address _pool, uint32 _period)
    internal
    view
    returns (uint256 price)
{
    int24 tick = getTick(_pool, _period);

    uint160 sqrtRatioX96 = TickMath.getSqrtRatioAtTick(tick);

    uint256 ratioX192 = uint256(sqrtRatioX96).mul(sqrtRatioX96);

    // return price from TWAP in 1e18
    price = FullMath.mulDiv(ratioX192, BASE, 1 << 192);

    uint256 token0Decimals =
IERC20Minimal(IUniswapV3Pool(_pool).token0()).decimals();
    uint256 token1Decimals =
IERC20Minimal(IUniswapV3Pool(_pool).token1()).decimals();

    bool decimalCheck = token0Decimals > token1Decimals;

    uint256 decimalsDelta = decimalCheck
        ? token0Decimals - token1Decimals
        : token1Decimals - token0Decimals;

    // normalise the price to 18 decimals
    if (token0Decimals == token1Decimals) {
        return price;
    }

    if (decimalCheck) {
        price = price.mul(CommonMath.safePower(10, decimalsDelta));
    } else {
        price = price.div(CommonMath.safePower(10, decimalsDelta));
    }
}
```

As the market price of `pool.token0` , `pool.token1` changes, the $ratio = \frac{token1Amount}{token0Amount}$ will change accordingly.

When $ratio = \frac{token1Amount}{token0Amount}$ is large enough, L329 `uint256(sqrtRatioX96).mul(sqrtRatioX96)` will revert with an error: ”SafeMath: multiplication overflow”.

As a result, as the $ratio = \frac{token1Amount}{token0Amount}$ changes, `OracleLibrary.consult()` will be malfunctioning from time to time, causing the whole system to be unstable.

PoC

Uniswap V3: WBTC-SHIB
https://etherscan.io/address/0x1153c8f2b05fdde2db507c8d16e49d4c7405c907#readContract

- token0: WBTC 0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599
- token1: SHIB 0x95aD61b0a150d79219dCF64E1E6Cc01f0B64C4cE

Current value (Aug 16)

- slot0.sqrtPriceX96: 331157352196766793165814541259643006294
- slot0.tick: 443092

$$ratioX192 = sqrtRatioX96^2 = 331157352196766793165814541259643006294^2$$

= 1.0966519191397344 $\times 10^{77}$

$$ratioX192 = ratio \times 2^{192} = 1.0001^{tick} \times 2^{192} = 1.0001^{443092} \times 2^{192}$$

= 1.0965744612088456 $\times 10^{77}$

This is near, but not yet exceeds $type(uint256).max = 2^{256} - 1 = 1.157920892373162 \times 10^{77}$

Therefore, the `OracleLibrary.consult()` function can work normally.

June 1

- BTC: 31925.73 USD
- SHIB: 0.00001179 USD

1.0 BTC -> 31925.73 USD -> $\frac{31925.73}{0.00001179} = 2707865139.949109$ SHIB

$$ratioX192 = ratio \times 2^{192} = \frac{token1Amount}{token0Amount} \times 2^{192}$$

$$= \frac{2707865139.949109 \times 10^{SHIB.decimals}}{1.0 \times 10^{WBTC.decimals}} \times 2^{192}$$

$$= \frac{2707865139.949109 \times 10^{18}}{1.0 \times 10^{8}} \times 2^{192}$$

= 1.6997544969167648 $\times 10^{77}$

This exceeds $type(uint256).max = 2^{256} - 1 = 1.157920892373162 \times 10^{77}$

`OracleLibrary.consult()` will revert due to overflow.

Recommendation

Consider using `ratioX128` when `sqrtRatioX96 > type(uint128).max` , just like `Uniswap/v3-periphery/contracts/libraries/OracleLibrary.sol` :

https://github.com/Uniswap/v3-periphery/blob/5bcdd9f67f9394f3159dad80d0dd01d37ca08c66/contracts/libraries/OracleLibrary.sol#L43-L69

```
    /// @notice Given a tick and a token amount, calculates the amount of token received in exchange
    /// @param tick Tick value used to calculate the quote
    /// @param baseAmount Amount of token to be converted
    /// @param baseToken Address of an ERC20 token contract used as the baseAmount denomination
    /// @param quoteToken Address of an ERC20 token contract used as the quoteAmount denomination
    /// @return quoteAmount Amount of quoteToken received for baseAmount of baseToken
    function getQuoteAtTick(
        int24 tick,
        uint128 baseAmount,
        address baseToken,
        address quoteToken
    ) internal pure returns (uint256 quoteAmount) {
        uint160 sqrtRatioX96 = TickMath.getSqrtRatioAtTick(tick);

        // Calculate quoteAmount with better precision if it doesn't overflow when multiplied by itself
        if (sqrtRatioX96 <= type(uint128).max) {
            uint256 ratioX192 = uint256(sqrtRatioX96) * sqrtRatioX96;
            quoteAmount = baseToken < quoteToken
                ? FullMath.mulDiv(ratioX192, baseAmount, 1 << 192)
                : FullMath.mulDiv(1 << 192, baseAmount, ratioX192);
        } else {
            uint256 ratioX128 = FullMath.mulDiv(sqrtRatioX96, sqrtRatioX96, 1 << 64);
            quoteAmount = baseToken < quoteToken
                ? FullMath.mulDiv(ratioX128, baseAmount, 1 << 128)
                : FullMath.mulDiv(1 << 128, baseAmount, ratioX128);
        }
    }
```

Status

✓ Fixed

WP-M9: Lack of price freshness check in `OracleLibrary.sol#getChainlinkPrice()` may cause a stale price to be used

Medium

Issue Description

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/libraries/OracleLibrary.sol#L88-L107

```
function getChainlinkPrice(
    FeedRegistryInterface _registry,
    address _base,
    address _quote
) internal view returns (uint256 price) {
    (, int256 _price, , , ) = _registry.latestRoundData(_base, _quote);

    // normalise the price to 18 decimals
    uint256 _decimals = _registry.decimals(_base, _quote);

    if (_decimals < 18) {
        uint256 missingDecimals = uint256(18).sub(_decimals);
        price = uint256(_price).mul(10**(missingDecimals));
    } else if (_decimals > 18) {
        uint256 extraDecimals = _decimals.sub(uint256(18));
        price = uint256(_price).div(10**(extraDecimals));
    }

    return price;
}
```

In the current implmentation of `OracleLibrary.sol#getChainlinkPrice()` , there is no freshness check being done. This could lead to stale prices being used.

If the market price of the token drops very quickly (”flash crashes”), and Chainlink’s feed does not get updated in time, the smart contract will continue to believe the token is worth more than the market value.

A stale price can cause the malfunction of multiple features across the system:

1. `isSwapExceedDeviation()` and `allowSwap()` is using the price to check if the price deviation of the swap excees the `allowedSwapDeviation` ; A stale price will malfunction the swap;
2. ShareHelpercalculateShares() is using the price to calculate the amount of shares to be minted; A stale price means the amount can be lower or higher than expected;
3. The rebase can only be done when `hasDeviation()` ; A stale price will prevent the strategy from doing rebases.

Chainlink also advise us to check for the `updatedAt` before using the price:

> Your application should track the latestTimestamp variable or use the updatedAt value from the latestRoundData() function to make sure that the latest answer is recent enough for your application to use it. If your application detects that the reported answer is not updated within the heartbeat or within time limits that you determine are acceptable for your application, pause operation or switch to an alternate operation mode while identifying the cause of the delay.

And they have this heartbeat concept:

> Chainlink Price Feeds do not provide streaming data. Rather, the aggregator updates its latestAnswer when the value deviates beyond a specified threshold or when the heartbeat idle time has passed. You can find the heartbeat and deviation values for each data feed at data.chain.link or in the Contract Addresses lists.

The `Heartbeat` on Polygon is usually `5m` or `24s` .

Source: https://docs.chain.link/docs/matic-addresses/

Recommendation

Consider adding the missing freshness check for stale price:

```
(uint80 roundID, int256 answer, , uint256 updatedAt, uint80 answeredInRound) =
_registry.latestRoundData(_base, _quote);
uint validPeriod = _registry.validPeriod(_base, _quote);
if (block.timestamp - updatedAt > validPeriod)) {
    return 0;
}
if (answer <= 0) {
    return 0;
}
...
```

The `validPeriod` can be based on the `Heartbeat` of the feed.

Status

✓ Fixed

WP-M10: `DefiEdgePrivateManager.rebalance()` Using `this.swap(_swapData)` (an external call) will change the `msg.sender` in the context to `address(this)` , affecting the `onlyOperator` modifier on `swap()`

Medium

Issue Description

In `DefiEdgePrivateManager.rebalance()` , when there is a `_swapData` provided, `this.swap(_swapData);` will be called as a **external call**:

See: https://docs.soliditylang.org/en/v0.7.6/control-structures.html#function-calls

> The expressions this.g(8); and c.g(2); (where c is a contract instance) are also valid function calls, but this time, the function will be called "externally", via a message call and not directly via jumps. Please note that function calls on this cannot be used in the constructor, as the actual contract has not been created yet.

This should usually work just fine, besides some gas overhead. However, there is an access control modifier (`onlyOperator`) that will check the address of the caller.

And by using an external call instead of an internal call, the `msg.sender` has been changed from the original caller to `address(this)` .

As a result, unless `address(this)` is also whitelisted as an `Operator` , `this.swap(_swapData)` will revert the whole transaction.

https://github.com/unbound-finance/defiedge-core/blob/f3f544a287a6495d5bc6c01236ef73601b580f77/contracts/personal/DefiEdgePrivateManager.sol#L98-L164

```
function rebalance(
    bytes calldata _swapData,
    PartialTick[] memory _existingTicks,
    Tick[] memory _newTicks,
    bool _burnAll
) external onlyOperator {
    if (_burnAll) {
        require(_existingTicks.length == 0, "IA");
        onHold = true;
        burnAllLiquidity();
        delete ticks;
        emit Hold();
    }
    //swap from 1inch if needed
    if (_swapData.length > 0) {
        this.swap(_swapData);
    }
```

https://github.com/unbound-finance/defiedge-core/blob/f3f544a287a6495d5bc6c01236ef73601b580f77/contracts/personal/UniswapV3PrivateLiquidityManager.sol#L317-L355

```
function swap(bytes calldata data) external onlyOperator {
```

https://github.com/unbound-finance/defiedge-core/blob/f3f544a287a6495d5bc6c01236ef73601b580f77/contracts/personal/UniswapV3PrivateLiquidityManager.sol#L38-L41

```
modifier onlyOperator() {
    require(hasRole(ADMIN_ROLE, msg.sender), "N");
    _;
}
```

Recommendation

Consider creating an internal function for `swap()` and call the internal function in `rebalance()` and the public `swap()` method.

Status

✓ **Fixed**

WP-H11: A malicious early user/attacker can manipulate the Strategy's pricePerShare to take an unfair share of future users' deposits from precision loss

High

Issue Description

https://github.com/unbound-finance/defiedge-core/blob/f3f544a287a6495d5bc6c01236ef73601b580f77/contracts/twap/libraries/TwapShareHelper.sol#L26-L74

```
function calculateShares(
    FeedRegistryInterface _registry,
    IUniswapV3Pool _pool,
    ITwapStrategyManager _manager,
    bool[2] memory _useTwap,
    uint256 _amount0,
    uint256 _amount1,
    uint256 _totalAmount0,
    uint256 _totalAmount1,
    uint256 _totalShares
) public view returns (uint256 share) {

    require(_amount0 > 0 && _amount1 > 0, 'INSUFFICIENT_AMOUNT');

    _amount0 = TwapOracleLibrary.normalise(_pool.token0(), _amount0);
    _amount1 = TwapOracleLibrary.normalise(_pool.token1(), _amount1);
    _totalAmount0 = TwapOracleLibrary.normalise(_pool.token0(), _totalAmount0);
    _totalAmount1 = TwapOracleLibrary.normalise(_pool.token1(), _totalAmount1);

    // price in USD
    uint256 token0Price = TwapOracleLibrary.getPriceInUSD(
        _pool,
        _registry,
        _pool.token0(),
        _useTwap,
        _manager
    );

54          uint256 token1Price = TwapOracleLibrary.getPriceInUSD(
55              _pool,
56              _registry,
57              _pool.token1(),
58              _useTwap,
59              _manager
60          );
61
62          if (_totalShares > 0) {
63
64              if(_amount0 < _amount1){
65                  share = FullMath.mulDiv(_amount1, _totalShares, _totalAmount1);
66              } else {
67                  share = FullMath.mulDiv(_amount0, _totalShares, _totalAmount0);
68              }
69
70          } else {
71              share = ((token0Price.mul(_amount0)).add(token1Price.mul(_amount1)))
72                  .div(DIVISOR);
73          }
74      }
```

When `totalshares = 0` , the total USD value of the initial liquidity will be used as the initial supply of `share` (L71-72).

If the attacker (as the first user) `mint()` with precise amounts of `token0` and `token1` , making `share = 1` , they can make the initital total supply of the Strategy shares to be as small as `1 wei` .

The attacker can then transfer tokens directly to the Strategy, the price per share can be inflated to a very high value, say if the attacker inflated the price per share to `1,000,000 USD` worth of tokens for 1 wei of `share` .

When the next user `mint()` with tokens worth `1,999,999 USD` , will only receive 1 wei of share due to precision loss.

As a result, the attacker can net a profit from the precision loss of the later users.

PoC

Given:

- `managementFee` = 0%
- token0 is `DAI`
- token1 is `LUSD`

1. Attacker is the first user, called `mint()` with `_amount0` = 50 and `_amount1` = 50, received `1 share`

- `token0Price` = 1e18
- `_amount0` = 50
- `token1Price` = 1e18
- `_amount1` = 50
- `(token0Price.mul(_amount0)).add(token1Price.mul(_amount1))` = 100e18
- `((token0Price.mul(amount0)).add(token1Price.mul(amount1))).div(DIVISOR)` = 100e18 / 100e18 = 1

1. Attacker sent `1,000,000e18-50` `DAI` and `1,000,000e18-50` `LUSD` to contract

- `1 share` now worth `1,000,000e18` `DAI` and `1,000,000e18` `LUSD`

1. Bob called `mint()` with `_amount0` = `1,999,999e18` and `_amount1` = `1,999,999e18`, received `1 share`

- `totalAmount0` = `1,000,000e18`
- `totalAmount1` = `1,000,000e18`
- `share` = _amount0 * _totalShares / _totalAmount0 = 1,999,999e18 * 1 / 1,000,000e18 = 1

1. Attacker called `burn()` with `_shares` = 1, received `1,499,999.5e18` `DAI` and `1,499,999.5e18` `LUSD` back, netted a profit of `499,999.5e18` `DAI` and `LUSD`

- `collect0` = FullMath.mulDiv(collect0, _shares, _totalSupply) = 2,999,999e18 * 1 / 2 = 1,499,999.5e18
- `collect1` = FullMath.mulDiv(collect1, _shares, _totalSupply) = 2,999,999e18 * 1 / 2 = 1,499,999.5e18

Recommendation

Consider requiring a minimal amount of share tokens to be minted for the first minter, and send a portion of the initial shares as a permanent reserve to the DAO/factory/manager address so that the pricePerShare can be more resistant to manipulation.

Status

✓ **Fixed**

WP-L12: Redundant code

Low

Issue Description

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/libraries/OracleLibrary.sol#L29-L45

```
29  function normalise(address _token, uint256 _amount)
30      internal
31      view
32      returns (uint256 normalised)
33  {
34      // return uint256(_amount) * (10**(18 - IERC20Minimal(_token).decimals()));
35      normalised = _amount;
36      uint256 _decimals = IERC20Minimal(_token).decimals();
37
38      if (_decimals < 18) {
39          uint256 missingDecimals = uint256(18).sub(_decimals);
40          normalised = uint256(_amount).mul(10**(missingDecimals));
41      } else if (_decimals > 18) {
42          uint256 extraDecimals = _decimals.sub(uint256(18));
43          normalised = uint256(_amount).div(10**(extraDecimals));
44      }
45  }
```

L35 is unnecessary, `normalised` will be overridden at L40 or L43.

Status

✓ Fixed

WP-I13: Misleading varibale names

Informational

Issue Description

1. `*fee` looks like amounts, but they are actually shares/percentage

`managementFee` , `performanceFee` , `protocolFee` , `protocolPerformanceFee` , `MAX_PROTOCOL_PERFORMANCE_FEES` are percentage in `PRECISION` .

`accManagementFee` is accumulated `managershare` , in shares.

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/DefiEdgeStrategyFactory.sol#L107-L117

```
        address manager = address(
            new StrategyManager(
                IStrategyFactory(address(this)),
                params.operator,
                params.feeTo,
                params.managementFee,
                params.performanceFee,
                params.limit,
                allowedDeviation
            )
        );
```

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/DefiEdgeStrategyFactory.sol#L162-L166

```
    function changeFee(uint256 _fee) external onlyGovernance {
        require(_fee <= 1e7, "IA"); // should be less than 10%
        protocolFee = _fee;
        emit ChangeProtocolFee(protocolFee);
    }
```

https://github.com/unbound-finance/defiedge-core/blob/

de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/DefiEdgeStrategyFactory.sol#L172-L176

```
    function changeProtocolPerformanceFee(uint256 _fee) external onlyGovernance {
        require(_fee <= MAX_PROTOCOL_PERFORMANCE_FEES, "IA"); // should be less than 20%
        protocolPerformanceFee = _fee;
        emit ChangeProtocolPerformanceFee(protocolPerformanceFee);
    }
```

https://github.com/unbound-finance/defiedge-core/blob/
de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/base/StrategyBase.sol#L87-L119

```
function issueShare(
        uint256 _amount0,
        uint256 _amount1,
        uint256 _totalAmount0,
        uint256 _totalAmount1,
        address _user
    ) internal returns (uint256 share) {
        // calculate number of shares
        share = ShareHelper.calculateShares(
            chainlinkRegistry,
            pool,
            manager,
            useTwap,
            _amount0,
            _amount1,
            _totalAmount0,
            _totalAmount1,
            totalSupply()
        );

        require(share > 0, "IS");

        uint256 managerShare;
        uint256 managementFee = manager.managementFee();
        // strategy owner fees
        if (managementFee > 0) {
            managerShare = share.mul(managementFee).div(FEE_PRECISION);
            accManagementFee = accManagementFee.add(managerShare);
        }

        // issue shares
        _mint(_user, share.sub(managerShare));
    }
```

Recommendation

Consider changing to:

- `managementFee` -> `managementFeeRate`
- `performanceFee` -> `performanceFeeRate`
- `protocolFee` -> `protocolFeeRate`
- `protocolPerformanceFee` -> `protocolPerformanceFeeRate`
- `MAX_PROTOCOL_PERFORMANCE_FEES` -> `MAX_PROTOCOL_PERFORMANCE_FEES_RATE`
- `accManagementFee` -> `accManagementFeeShares`

2. `*Share` looks like they are in shares, but they are actually amounts

https://github.com/unbound-finance/defiedge-core/blob/de7c8fed13a46ff6749feefbed3b8ff5d0d08798/contracts/base/UniswapV3LiquidityManager.sol#L146-L189

```
    function _transferPerformanceFees(uint256 _fee0, uint256 _fee1) internal {
        (
            address managerFeeTo,
            address protocolFeeTo,
            uint256 managerToken0Share,
            uint256 managerToken1Share,
            uint256 protocolToken0Share,
            uint256 protocolToken1Share
        ) = ShareHelper.calculateFeeTokenShares(factory, manager, _fee0, _fee1);

        if (managerToken0Share > 0) {
            TransferHelper.safeTransfer(
                address(token0),
                managerFeeTo,
                managerToken0Share
            );
        }
```

```
163
164         if (managerToken1Share > 0) {
165             TransferHelper.safeTransfer(
166                 address(token1),
167                 managerFeeTo,
168                 managerToken1Share
169             );
170         }
171
172         if (protocolToken0Share > 0) {
173             TransferHelper.safeTransfer(
174                 address(token0),
175                 protocolFeeTo,
176                 protocolToken0Share
177             );
178         }
179
180         if (protocolToken1Share > 0) {
181             TransferHelper.safeTransfer(
182                 address(token1),
183                 protocolFeeTo,
184                 protocolToken1Share
185             );
186         }
187
188         emit FeesClaim(address(this), _fee0, _fee1);
189     }
```

According to L160, L168, L176, L184, `managerToken0Share` , `managerToken1Share` , `protocolToken0Share` , `protocolToken1Share` are not in shares but in amounts.

Recommendation

Consider changing to:

- `managerToken0Share` -> `managerToken0Amount`
- `managerToken1Share` -> `managerToken1Amount`
- `protocolToken0Share` -> `protocolToken0Amount`
- `protocolToken1Share` -> `protocolToken1Amount`

https://github.com/unbound-finance/defiedge-core/blob/f3f544a287a6495d5bc6c01236ef73601b580f77/contracts/twap/libraries/TwapShareHelper.sol#L122-L168

```
    function calculateFeeTokenShares(
        ITwapStrategyFactory _factory,
        ITwapStrategyManager _manager,
        uint256 _fee0,
        uint256 _fee1
    )
        public
        view
        returns (
            address managerFeeTo,
            address protocolFeeTo,
            uint256 managerToken0Share,
            uint256 managerToken1Share,
            uint256 protocolToken0Share,
            uint256 protocolToken1Share
        )
    {
        // protocol fees
        uint256 protocolFee = _factory.protocolFee();

        // performance fee to manager
        uint256 performanceFee = _manager.performanceFee();

        // protocol performance fee
        uint256 _protocolPerformanceFee = _factory.protocolPerformanceFee();

        // calculate the fees for protocol and manager from performance fees
        uint256 performanceToken0Share = FullMath.mulDiv(_fee0, performanceFee, 1e8);
        uint256 performanceToken1Share = FullMath.mulDiv(_fee1, performanceFee, 1e8);

        if(performanceToken0Share > 0){
            protocolToken0Share = FullMath.mulDiv(performanceToken0Share, protocolFee, 1e8);
            managerToken0Share = performanceToken0Share.sub(protocolToken0Share);
        }

        if(performanceToken1Share > 0){
            protocolToken1Share = FullMath.mulDiv(performanceToken1Share,
protocolFee, 1e8);
            managerToken1Share = performanceToken1Share.sub(protocolToken1Share);
        }

        protocolToken0Share = protocolToken0Share.add(FullMath.mulDiv(_fee0,
_protocolPerformanceFee, 1e8));
        protocolToken1Share = protocolToken1Share.add(FullMath.mulDiv(_fee1,
_protocolPerformanceFee, 1e8));

        // moved here for saving bytecode
        managerFeeTo = _manager.feeTo();
        protocolFeeTo = _factory.feeTo();
    }
```

Recommendation

- `managerToken0Share` -> `managerToken0Amount`
- `managerToken1Share` -> `managerToken1Amount`
- `protocolToken0Share` -> `protocolToken0Amount`
- `protocolToken1Share` -> `protocolToken1Amount`
- `performanceToken0Share` -> `performanceToken0Amount`
- `performanceToken1Share` -> `performanceToken1Amount`
- `protocolFee` -> `protocolFeeRate`
- `performanceFee` -> `performanceFeeRate`
- `protocolPerformanceFee` -> `protocolPerformanceFeeRate`

Status

✓ Fixed

WP-H14: The amount of shares issued in `DefiEdgeStrategy.mint()` is inaccurate due to oracle price deviation

High

Issue Description

In `DefiEdgeStrategy.mint()` , the amount of shares issued to the user is calculated based on the USD price from the Oracle (Chainlink).

Because there will be a deviation from the oracle price to the actual market price.

Per to Chainlink's docs:

> Chainlink Price Feeds do not provide streaming data. Rather, the aggregator updates its latestAnswer when the value deviates beyond a specified threshold or when the heartbeat idle time has passed. You can find the heartbeat and deviation values for each data feed at data.chain.link or in the Contract Addresses lists.

And the `Deviation` and `Heartbeat` on Ethereum Mainnet can be quite large and long, usually `2%` for Deviation and `24h` as Heartbeat.

Source: https://docs.chain.link/docs/ethereum-addresses/

We believe this can cause the amount of newly minted shares to be fewer or larger than expected.

https://github.com/unbound-finance/defiedge-core/blob/2402275b3d45277616e2fa5811243df20877f4a8/contracts/DefiEdgeStrategy.sol#L73-L154

```
Tick storage tick = ticks[0];
// index 0 will always be an primary tick
(amount0, amount1) = mintLiquidity(
    tick.tickLower,
    tick.tickUpper,
    _amount0,
    _amount1,
    msg.sender
);

// update data in the tick
tick.amount0 = tick.amount0.add(amount0);
tick.amount1 = tick.amount1.add(amount1);
```

```
// issue share based on the liquidity added
share = issueShare(
    amount0,
    amount1,
    totalAmount0,
    totalAmount1,
    msg.sender
);
```

`DefiEdgeStrategy.mint()` will call `ShareHelper.calculateShares()` in `issueShare()` :

https://github.com/unbound-finance/defiedge-core/blob/2402275b3d45277616e2fa5811243df20877f4a8/contracts/libraries/ShareHelper.sol#L27-L76

```
address _token0 = _pool.token0();
address _token1 = _pool.token1();

_amount0 = OracleLibrary.normalise(_token0, _amount0);
_amount1 = OracleLibrary.normalise(_token1, _amount1);
_totalAmount0 = OracleLibrary.normalise(_token0, _totalAmount0);
_totalAmount1 = OracleLibrary.normalise(_token1, _totalAmount1);

// price in USD
uint256 token0Price = OracleLibrary.getPriceInUSD(
    _factory,
    _registry,
    _token0,
    _isBase[0]
);

uint256 token1Price = OracleLibrary.getPriceInUSD(
    _factory,
    _registry,
    _token1,
    _isBase[1]
);

if (_totalShares > 0) {
    uint256 numerator = (token0Price.mul(_amount0)).add(
        token1Price.mul(_amount1)
    );

    uint256 denominator = (token0Price.mul(_totalAmount0)).add(
        token1Price.mul(_totalAmount1)
    );

    share = FullMath.mulDiv(numerator, _totalShares, denominator);
} else {
    share = ((token0Price.mul(_amount0)).add(token1Price.mul(_amount1)))
        .div(DIVISOR);
}
```

PoC

Given:

- The pair is WBTC/WETH;
- Current totalShares = `100000`
- The actual market price of WBTC is `$20k` ;
- The actual market price of WETH is `$2k` ;
- The Chainlink price of WBTC is `$22k` ; (+10% deviation for easier demonstration)
- The Chainlink price of WETH is `$1.8k` ; (-10% deviation for easier demonstration)

1. The total amounts of the pool are: `10 WBTC` and `100 WETH` ;
2. Alice `mint()` and added `0.1 WETH` and `10 WBTC` ; (the primary tick is unbalanced with the total amounts)
3. In `calculateShares()` :
   - `numerator` : `0.1 * 1.8k + 10 * 22k == 220180`
   - `denominator` : `100 * 1.8k + 10 * 22k == 400000`
   - `share` : `220180*100000/400000 == 55045`

Alice now holds `55045/155045 ~= 0.355` of the pool, while she actualy only added `0.1 * 2k + 10 * 20k / 100 * 2k + 10 * 20k ~= 0.3335` of the pool.

Recommendation

Consider using the same method in `DefiEdgeTwapStrategy.mint()` to calculate shares, which will make it more accurate, gas efficient, and consistent.

Status

ⓘ Acknowledged

WP-H15: `amount0` , `amount1` returned from `UniswapV3TwapLiquidityManager.getAUMWithFees()` is larger than the actual amounts, causing fewer shares being minted

High

Issue Description

At L468-469 in `UniswapV3TwapLiquidityManager.getAUMWithFees()` , all the `totalFee0` , `totalFee1` from the underlying pool are added to `amount0` , `amount1` directly.

However, not all the `totalFee0` , `totalFee1` belongs to the share holders. There is a portion of the fees belongs to the `manager` and `protocol` as `PerformanceFees` .

At L472, the PerformanceFees will be transferred to the `managerFeeTo` and `protocolFeeTo` .

As a result, the `amount0` and `amount1` returned from `UniswapV3TwapLiquidityManager.getAUMWithFees()` is larger than the actual amounts.

https://github.com/unbound-finance/defiedge-core/blob/2402275b3d45277616e2fa5811243df20877f4a8/contracts/twap/base/UniswapV3TwapLiquidityManager.sol#L409-L477

```
453  // collect fees
454  if (_includeFee && currentLiquidity > 0) {
455
456      // update fees earned in Uniswap pool
457      // Uniswap recalculates the fees and updates the variables when amount is
     passed as 0
458      pool.burn(tick.tickLower, tick.tickUpper, 0);
459
460      (totalFee0, totalFee1) = pool.collect(
461          address(this),
462          tick.tickLower,
463          tick.tickUpper,
464          type(uint128).max,
465          type(uint128).max
466      );
467
```

```
    amount0 = amount0.add(totalFee0);
    amount1 = amount1.add(totalFee1);

    // transfer performance fees
    _transferPerformanceFees(totalFee0, totalFee1);

    emit FeesClaim(address(this), totalFee0, totalFee1);
}
```

https://github.com/unbound-finance/defiedge-core/blob/2402275b3d45277616e2fa5811243df20877f4a8/contracts/twap/base/UniswapV3TwapLiquidityManager.sol#L143-L191

```
/**
* @notice Splits and stores the performance feees in the local variables
* @param _fee0 Amount of accumulated fee for token0
* @param _fee1 Amount of accumulated fee for token1
*/
function _transferPerformanceFees(uint256 _fee0, uint256 _fee1) internal {
    (
        address managerFeeTo,
        address protocolFeeTo,
        uint256 managerToken0Amount,
        uint256 managerToken1Amount,
        uint256 protocolToken0Amount,
        uint256 protocolToken1Amount
    ) = TwapShareHelper.calculateFeeTokenShares(factory, manager, _fee0, _fee1);

    if (managerToken0Amount > 0) {
        TransferHelper.safeTransfer(
            address(token0),
            managerFeeTo,
            managerToken0Amount
        );
    }

    if (managerToken1Amount > 0) {
        TransferHelper.safeTransfer(
            address(token1),
            managerFeeTo,
            managerToken1Amount
            );
    }

    if (protocolToken0Amount > 0) {
        TransferHelper.safeTransfer(
            address(token0),
            protocolFeeTo,
            protocolToken0Amount
        );
    }

    if (protocolToken1Amount > 0) {
        TransferHelper.safeTransfer(
            address(token1),
            protocolFeeTo,
            protocolToken1Amount
        );
    }

    emit FeesClaim(address(this), _fee0, _fee1);
}
```

In `DefiEdgeTwapStrategy.mint()` , `totalAmount0` , `totalAmount1` are wrong and larger than the actual amounts, hence the `shares` calculated in `issueShare()` is fewer than expected.

Recommendation

Consider moving L423-424 getting the balance of `token0` and `token1` to the end of the loop, and return the actual `position0 + balance0` as `amount0` and `position1 + balance1` as `amount1` .

Status

✓ Fixed

WP-H16: The amounts of `PerformanceFeeShares` are calculated wrong when `addPerformanceFees()` is called in the for loop

High

Issue Description

When there are more than 1 tick, `getAUMWithFees()` will do a for loop and call `addPerformanceFees()` at the end of each iteration.

For all the ticks before the last one, the `totalAmount0` and `totalAmount1` , which will be used for calculating the amounts of `PerformanceFeeShares` , will be lower than the actual amounts, because the accumulated fees in the later ticks are not included.

As a result, the amount of `PerformanceFeeShares` will be larger than expected.

https://github.com/unbound-finance/defiedge-core/blob/2402275b3d45277616e2fa5811243df20877f4a8/contracts/base/UniswapV3LiquidityManager.sol#L145-L191

```
/**
* @notice Splits and stores the performance feees in the local variables
* @param _fee0 Amount of accumulated fee for token0
* @param _fee1 Amount of accumulated fee for token1
*/
function addPerformanceFees(uint256 _fee0, uint256 _fee1) internal {
    // transfer performance fee to manager
    uint256 performanceFeeRate = manager.performanceFeeRate();
    // address feeTo = manager.feeTo();

    // get total amounts with fees
    (uint256 totalAmount0, uint256 totalAmount1, ,) = this
        .getAUMWithFees(false);

    accPerformanceFeeShares = accPerformanceFeeShares.add(
        ShareHelper.calculateShares(
            factory,
            chainlinkRegistry,
            pool,
            usdAsBase,
            FullMath.mulDiv(_fee0, performanceFeeRate, FEE_PRECISION),
            FullMath.mulDiv(_fee1, performanceFeeRate, FEE_PRECISION),
            totalAmount0,
            totalAmount1,
            totalSupply()
        )
    );

    // protocol performance fee
    uint256 _protocolPerformanceFee = factory.protocolPerformanceFeeRate();

    accProtocolPerformanceFeeShares = accProtocolPerformanceFeeShares.add(
        ShareHelper.calculateShares(
            factory,
            chainlinkRegistry,
            pool,
            usdAsBase,
            FullMath.mulDiv(_fee0, _protocolPerformanceFee, FEE_PRECISION),
            FullMath.mulDiv(_fee1, _protocolPerformanceFee, FEE_PRECISION),
            totalAmount0,
            totalAmount1,
            totalSupply()
        )
    );

    emit FeesClaim(address(this), _fee0, _fee1);
}
```

https://github.com/unbound-finance/defiedge-core/blob/2402275b3d45277616e2fa5811243df20877f4a8/contracts/base/UniswapV3LiquidityManager.sol#L395-L460

```
    // collect fees
    if(_includeFee && currentLiquidity > 0){

        // update fees earned in Uniswap pool
        // Uniswap recalculates the fees and updates the variables when amount is passed as 0
        pool.burn(tick.tickLower, tick.tickUpper, 0);

        (totalFee0, totalFee1) = pool.collect(
            address(this),
            tick.tickLower,
            tick.tickUpper,
            type(uint128).max,
            type(uint128).max
        );

        amount0 = amount0.add(totalFee0);
        amount1 = amount1.add(totalFee1);

        // mint performance fees
        addPerformanceFees(totalFee0, totalFee0);

        emit FeesClaim(address(this), totalFee0, totalFee1);
    }
```

Status

✓ Fixed

WP-M17: `getAUMWithFees()` returns wrong `totalFee0`, `totalFee1`

Medium

Issue Description

In the current implementation, `totalFee0` and `totalFee1` are not the sum of all the fees from all ticks, but instead the `fee0` and `fee1` from the last tick.

https://github.com/unbound-finance/defiedge-core/blob/2402275b3d45277616e2fa5811243df20877f4a8/contracts/twap/base/UniswapV3TwapLiquidityManager.sol#L409-L477

```
// collect fees
if (_includeFee && currentLiquidity > 0) {

    // update fees earned in Uniswap pool
    // Uniswap recalculates the fees and updates the variables when amount is passed as 0
    pool.burn(tick.tickLower, tick.tickUpper, 0);

    (totalFee0, totalFee1) = pool.collect(
        address(this),
        tick.tickLower,
        tick.tickUpper,
        type(uint128).max,
        type(uint128).max
    );

    amount0 = amount0.add(totalFee0);
    amount1 = amount1.add(totalFee1);

    // transfer performance fees
    _transferPerformanceFees(totalFee0, totalFee1);

    emit FeesClaim(address(this), totalFee0, totalFee1);
}
```

Status

✓ Fixed

WP-M18: `mint()` will revert when `totalAmount0` = 0

Medium

Issue Description

https://github.com/unbound-finance/defiedge-core/blob/2402275b3d45277616e2fa5811243df20877f4a8/contracts/twap/DefiEdgeTwapStrategy.sol#L74-L146

```
function mint(
    uint256 _amount0,
    uint256 _amount1,
    uint256 _amount0Min,
    uint256 _amount1Min,
    uint256 _minShare
)
    external
    onlyValidStrategy
    returns (
        uint256 amount0,
        uint256 amount1,
        uint256 share
    )
{
    require(manager.isUserWhiteListed(msg.sender), "UA");

    // get total amounts with fees
    (uint256 totalAmount0, uint256 totalAmount1, ,) = this
        .getAUMWithFees(true);

    // calculate optimal token0 & token1 amount for mint
    (_amount0, _amount1) = TwapShareHelper.getOptimalAmounts(
        _amount0,
        _amount1,
        _amount0Min,
        _amount1Min,
        totalAmount0,
        totalAmount1
    );
```

https://github.com/unbound-finance/defiedge-core/blob/2402275b3d45277616e2fa5811243df20877f4a8/contracts/twap/libraries/TwapShareHelper.sol#L160-L192

```
function getOptimalAmounts(
    uint256 _amount0,
    uint256 _amount1,
    uint256 _amount0Min,
    uint256 _amount1Min,
    uint256 _totalAmount0,
    uint256 _totalAmount1
)
    public
    pure
    returns(
        uint256 amount0,
        uint256 amount1
    )
{
    require(_amount0 > 0 && _amount1 > 0, 'INSUFFICIENT_AMOUNT');

    if (_totalAmount0 == 0 && _totalAmount1 == 0) {
        (amount0, amount1) = (_amount0, _amount1);
    } else {
        uint amount1Optimal = _amount0.mul(_totalAmount1).div(_totalAmount0);
        if (amount1Optimal <= _amount1) {
            require(amount1Optimal >= _amount1Min, 'INSUFFICIENT_AMOUNT_1');
            (amount0, amount1) = (_amount0, amount1Optimal);
        } else {
            uint amount0Optimal = _amount1.mul(_totalAmount0).div(_totalAmount1);
            assert(amount0Optimal <= _amount0);
            require(amount0Optimal >= _amount0Min, 'INSUFFICIENT_AMOUNT_0');
            (amount0, amount1) = (amount0Optimal, _amount1);
        }
    }

}
```

In the current implementation, `getOptimalAmounts()` assumes `_totalAmount0` will never be `0` . Unless both _totalAmount0 and _totalAmount1 are 0 (`_totalAmount0 == 0 && _totalAmount1 == 0`).

However, for a Uniswap V3 position, when the price is out of range, `_totalAmount0` can be `0`.

When `_totalAmount0 = 0`, `getOptimalAmounts()` will revert due to `divide by 0`, and the whole `mint()` transaction reverts.

Recommendation

Change to:

```
if (_totalAmount0 == 0) {
    require(_amount0Min == 0, 'INSUFFICIENT_AMOUNT_0');
    (amount0, amount1) = (0, _amount1);
    return;
}
if (_totalAmount1 == 0) {
    require(_amount1Min == 0, 'INSUFFICIENT_AMOUNT_1');
    (amount0, amount1) = (_amount0, 0);
    return;
}
```

Status

✓ Fixed

WP-I19: `MINIMUM_LIQUIDITY` is too small

Informational

Issue Description

While there is a `MIN_SHARE` to ensure each mint will at least issue `1e16` shares, the permanent reserve amount is ultra low, which is insufficient for manipulation resistance.

`MIN_SHARE` won't help, as they can also burn as much as they want.

https://github.com/unbound-finance/defiedge-core/blob/2402275b3d45277616e2fa5811243df20877f4a8/contracts/base/StrategyBase.sol#L18-L19

```
uint256 public constant MIN_SHARE = 1e16;
uint256 public constant MINIMUM_LIQUIDITY = 1000;
```

https://github.com/unbound-finance/defiedge-core/blob/2402275b3d45277616e2fa5811243df20877f4a8/contracts/base/StrategyBase.sol#L109-L150

```
function issueShare(
    uint256 _amount0,
    uint256 _amount1,
    uint256 _totalAmount0,
    uint256 _totalAmount1,
    address _user
) internal returns (uint256 share) {

    uint256 _shareTotalSupply = totalSupply();
    // calculate number of shares
    share = ShareHelper.calculateShares(
        factory,
        chainlinkRegistry,
        pool,
        usdAsBase,
        _amount0,
        _amount1,
        _totalAmount0,
        _totalAmount1,
        _shareTotalSupply
    );

    require(share > MIN_SHARE, "IS");

    uint256 managerShare;
    uint256 managementFeeRate = manager.managementFeeRate();

    if(_shareTotalSupply == 0){
        share = share.sub(MINIMUM_LIQUIDITY);
        _mint(address(0), MINIMUM_LIQUIDITY);
    }

    // strategy owner fees
    if (managementFeeRate > 0) {
        managerShare = share.mul(managementFeeRate).div(FEE_PRECISION);
        accManagementFeeShares = accManagementFeeShares.add(managerShare);
        share = share.sub(managerShare);
    }

    // issue shares
    _mint(_user, share);
}
```

Recommendation

Consider changing `MINIMUM_LIQUIDITY` to `1e12` .

Status

✓ Fixed

WP-I20: `burnLiquidity()` Returning `tokensBurned0` and `tokensBurned1` as `collect0` / `collect1` is misleading

Informational

Issue Description

The amount `collected` has a different meaning than the amount `burned` .

Therefore, returning `tokensBurned0` and `tokensBurned1` as `collect0` / `collect1` in `burnLiquidity()` can be misleading.

https://github.com/unbound-finance/defiedge-core/blob/2402275b3d45277616e2fa5811243df20877f4a8/contracts/twap/base/UniswapV3TwapLiquidityManager.sol#L79-L141

```
function burnLiquidity(
    int24 _tickLower,
    int24 _tickUpper,
    uint256 _shares,
    uint128 _currentLiquidity
)
    internal
    returns (
        uint256 collect0,
        uint256 collect1,
        uint256 fee0,
        uint256 fee1
    )
{
    uint256 tokensBurned0;
    uint256 tokensBurned1;

    if (_shares > 0) {
        (_currentLiquidity, , , , ) = pool.positions(
            PositionKey.compute(address(this), _tickLower, _tickUpper)
        );
        if (_currentLiquidity > 0) {
            uint256 liquidity = FullMath.mulDiv(
                _currentLiquidity,
                _shares,
                totalSupply()
            );

            (tokensBurned0, tokensBurned1) = pool.burn(
                _tickLower,
                _tickUpper,
                liquidity.toUint128()
            );
        }
    } else {
        (tokensBurned0, tokensBurned1) = pool.burn(
            _tickLower,
            _tickUpper,
            _currentLiquidity
        );
    }
    // collect fees
    (collect0, collect1) = pool.collect(
        address(this),
        _tickLower,
        _tickUpper,
        type(uint128).max,
        type(uint128).max
    );

    fee0 = collect0 > tokensBurned0
        ? uint256(collect0).sub(tokensBurned0)
        : 0;
    fee1 = collect1 > tokensBurned1
        ? uint256(collect1).sub(tokensBurned1)
        : 0;

    collect0 = tokensBurned0;
    collect1 = tokensBurned1;

    // transfer performance fees
    _transferPerformanceFees(fee0, fee1);
}
```

Status

✓ Fixed

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Disclaimer