Competitive Security Assessment

Pulsar

Nov 17th, 2022

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	Pulsar
Platform & Language	Solidity
Codebase	• repo - https://github.com/PulsarSwap/TWAMM-Contracts/ • audit commit - f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b • final commit - 9e1b42eb5f7a2fe4a859fb383714e499fe438ff6
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

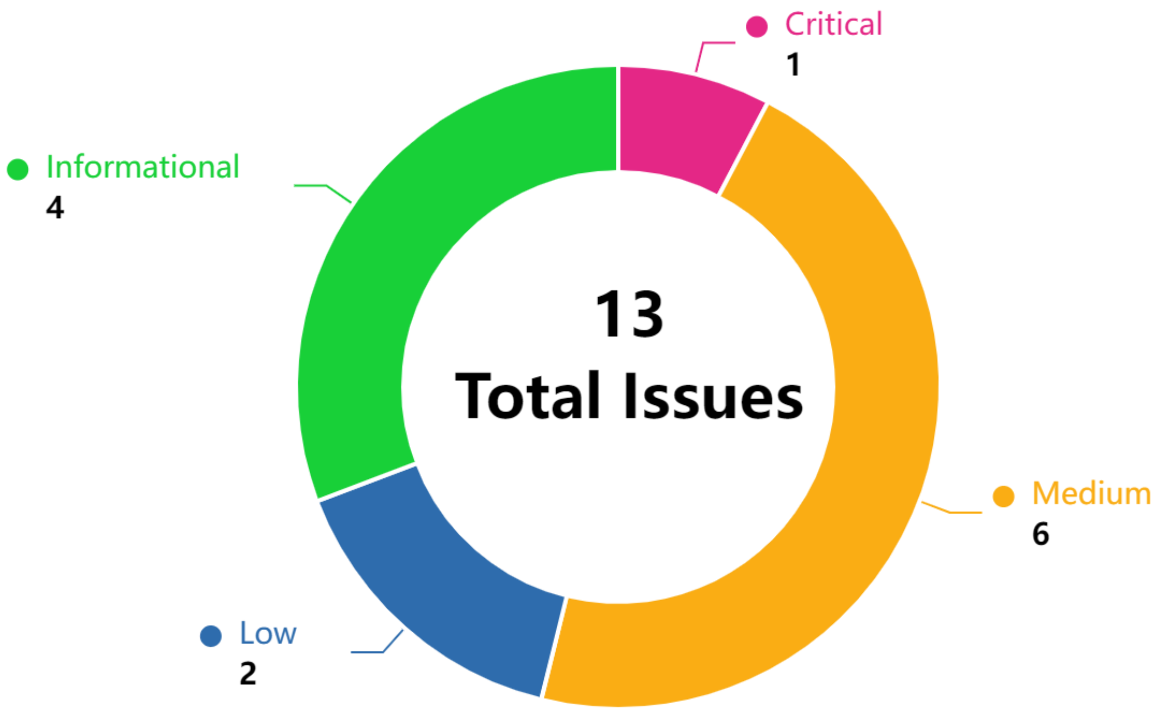
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	1	0	0	1	0	0
Medium	6	0	0	3	1	2
Low	2	0	0	1	0	1
Informational	4	0	3	1	0	0

Audit Scope

File	Commit Hash
contracts/TWAMM.sol	f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b
contracts/Pair.sol	f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b
contracts/libraries/LongTermOrders.sol	f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b
contracts/libraries/BinarySearchTree.sol	f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b
contracts/interfaces/IPair.sol	f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b
contracts/interfaces/ITWAMM.sol	f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b
contracts/libraries/OrderPool.sol	f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b
contracts/libraries/Library.sol	f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b
contracts/Factory.sol	f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b
contracts/libraries/TransferHelper.sol	f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b
contracts/interfaces/IFactory.sol	f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b
contracts/libraries/SafeMath.sol	f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b
contracts/interfaces/IWETH.sol	f5cc7b0ea35f9e9a6872cdff62fb9c740ef7da5b

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
PUL-1	Functions `cancelTermSwapTokenToToken`, `cancelTermSwapTokenToETH` and `cancelTermSwapETHToToken` should check whether token0 and token1 match the order of tokens in `LongTermOrders`	Logical	Medium	Fixed	alansh
PUL-2	Invalid check on orderIdStatusMap	Logical	Low	Fixed	thereksfour
PUL-3	No Upper Limit for the `fee`	Privilege Related	Low	Declined	Hellobloc

PUL-4	No need to use SafeMath in solidity version 0.8+	Gas Optimization	Informational	Fixed	0xxm
PUL-5	No slippage control when providing or removing liquidity	Logical	Medium	Fixed	thereksfour
PUL-6	The Gaslimit Dos in `executeVirtualOrdersUntilSpecifiedBlock`	DOS	Medium	Declined	Hellobloc
PUL-7	Tokens received after LongTermSwap may be smaller than expected due to precision loss	Language Specific	Medium	Fixed	thereksfour
PUL-8	Unsupported fee-on-transfer tokens	Logical	Medium	Mitigated	thereksfour
PUL-9	Wrong usage of sortAmounts	Logical	Medium	Declined	alansh
PUL-10	`RemoveLiquidity` has the Potential to Completely Empty the `Pair`, which would Lead to A Potential Fraud Risk.	Logical	Critical	Fixed	Hellobloc
PUL-11	`safeTransferFrom` lacks `isContract` check	Logical	Informational	Acknowledged	Hellobloc
PUL-12	no need to re-calculate k	Gas Optimization	Informational	Acknowledged	alansh
PUL-13	redundant check	Logical	Informational	Acknowledged	alansh

PUL-1:Functions `cancelTermSwapTokenToToken`, `cancelTermSwapTokenToETH` and `cancelTermSwapETHToToken` should check whether token0 and token1 match the order of tokens in `LongTermOrders`

Category	Severity	Code Reference	Status	Contributor
Logical	**Medium**	**code/twamm/contracts/TWAMM.sol#L491** **code/twamm/contracts/TWAMM.sol#L518** **code/twamm/contracts/TWAMM.sol#L543**	**Fixed**	**alansh**

Code

```
491:        (unsoldAmount, purchasedAmount) = IPair(pair).cancelLongTermSwap(

518:            .cancelLongTermSwap(msg.sender, orderId);

543:            .cancelLongTermSwap(msg.sender, orderId);
```

Description

alansh : Here it's assuming `token0` is the selling token, but it's not check. It's safer for `cancelLongTermSwap` to also return the `sellTokenId` and check it in the calling side.

Recommendation

alansh : Add a `sellTokenId` return value to `cancelLongTermSwap` and check `token0` == `sellTokenId` (the same issue exists in other functions that calls `cancelLongTermSwap`, won't repeat)

Client Response

Fixed. Added `require(tokenSell == token0, "Wrong Sell Token");` to make sure the `token0` is the selling token.

PUL-2:Invalid check on orderIdStatusMap

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	**code/twamm/contracts/libraries/LongTermOrders.sol#L180-L181** **code/twamm/contracts/libraries/LongTermOrders.sol#L231-L232**	**Fixed**	**thereksfour**

Code

```
180:        require(self.orderIdStatusMap[orderId] = true, "Order Invalid");
181:        require(order.owner == sender, "Sender Must Be Order Owner");

231:        require(self.orderIdStatusMap[orderId] = true, "Order Invalid");
232:        require(order.owner == sender, "Sender Must Be Order Owner");
```

Description

thereksfour : In the withdrawProceedsFromLongTermSwap and cancelLongTermSwap functions, when orderIdStatusMap is set to false, the check is invalid because the check on orderIdStatusMap is `orderIdStatusMap = true` instead of `orderIdStatusMap == true`. Although this invalid check will not cause high risks because other checks are sufficient, it may cause front-end display errors due to inaccurate revert messages. For example, calling withdrawProceeds on a cancelled order will prompt `Sales Rate Amount Must Be Positive` instead of `Order Invalid`.

Recommendation

thereksfour : Change to

```
-     require(self.orderIdStatusMap[orderId] = true, "Order Invalid");
+    require(self.orderIdStatusMap[orderId], "Order Invalid");
```

Client Response

Fixed to correct `require(self.orderIdStatusMap[orderId] == true, "Order Invalid");` statement.

PUL-3:No Upper Limit for the `fee`

Category	Severity	Code Reference	Status	Contributor
Privilege Related	**Low**	**code/twamm/contracts/Factory.sol#L64-L67**	**Declined**	**Hellobloc**

Code

```
64:    function setFeeArg(uint32 _feeArg) external override {
65:        require(msg.sender == feeToSetter, "Factory: Forbidden");
66:        feeArg = _feeArg;
67:    }
```

Description

Hellobloc : In the `Factory` contract of the `TWAMM` project, the project owner can set the `fee` for its `pairs`. This allows the project owner to set a very high fee before the user's transaction is on-chain, thus causing a loss to the user.

```
function setFeeArg(uint32 _feeArg) external override {
        require(msg.sender == feeToSetter, "Factory: Forbidden");
        feeArg = _feeArg;
    }
```

Recommendation

Hellobloc : We propose to constrain the setting of the `fee` to ensure that there is a check on its upper limit, and to use multiple signatures for privileged users.

Client Response

The fee sent to `feeTo` equals to `1/(feeArg+1)`. As the total fee decreases as the `feeArg` increases and it is bound to 1.

PUL-4:No need to use SafeMath in solidity version 0.8+

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	**Informational**	**code/twamm/contracts/libraries/SafeMath.sol#L7**	**Fixed**	**0xxm**

Code

```
7:library SafeMath {
```

Description

0xxm : Solidity provides the overflow checking for version above 0.8. The contract does not need to import the SafeMath library for overflow checking, which can save gas.

Recommendation

0xxm : Remove SafeMath to save gas.

Client Response

Fixed and removed SafeMath use.

PUL-5:No slippage control when providing or removing liquidity

Category	Severity	Code Reference	Status	Contributor
Logical	**Medium**	**code/twamm/contracts/TWAMM.sol#L176-L180** **code/twamm/contracts/TWAMM.sol#L207-L210** **code/twamm/contracts/TWAMM.sol#L230-L233** **code/twamm/contracts/TWAMM.sol#L262-L265**	**Fixed**	**thereksfour**

Code

```
176:        amountIn0 = (lpTokenAmount * reserve0) / totalSupplyLP;
177:        amountIn1 = (lpTokenAmount * reserve1) / totalSupplyLP;
178:
179:        IERC20(token0).safeTransferFrom(msg.sender, pair, amountIn0);
180:        IERC20(token1).safeTransferFrom(msg.sender, pair, amountIn1);

207:        IERC20(token).safeTransferFrom(msg.sender, pair, amountTokenIn);
208:        IWETH(WETH).deposit{value: amountETHIn}();
209:        IERC20(WETH).safeTransfer(pair, amountETHIn);
210:        IPair(pair).provideLiquidity(msg.sender, lpTokenAmount);

230:        (uint256 amountOutA, uint256 amountOutB) = IPair(pair).removeLiquidity(
231:            msg.sender,
232:            lpTokenAmount
233:        );

262:        (uint256 amountOutA, uint256 amountOutB) = IPair(pair).removeLiquidity(
263:            msg.sender,
264:            lpTokenAmount
265:        );
```

Description

thereksfour : In the addLiquidity* and withdrawLiquidity* functions, the number of tokens the user needs to provide or receive is affected by the number of reserved tokens the contract currently has. Since the number of reserved tokens in the contract changes after each swap, the actual number of tokens spent by the user is not the same as expected. Consider the following scenario, the number of tokens A in the current Pair contract is 100, the number of tokens B is 10,000, and the number of LPs is 1,000. User A is ready to provide liquidity for 50 LPs and is expected to spend 50 tokens A and 500 tokens B. But in the same block, a swap transaction occurred before user A provided liquidity, and user B exchanged 50 tokens A for 3333 tokens B. At this time, the number of tokens A in the Pair contract is 150, the number of tokens B is 6667, and the number of LPs is 1000. User A needs to provide 7.5 tokens A and 333 tokens B to obtain 50 LPs.

Recommendation

thereksfour : Consider adding amountInAmax/amountInBmax parameters to the addLiquidity* function to allow users to control the tokens spent and adding amountOutAmin/amountOutBmin parameters in withdrawLiquidity* function to allow users to control received tokens

Client Response

Fixed by adding `amountOut0Min`, `amountOut1Min`, `amountTokenOutMin`, `amountETHOutMin` parameters to guarantee the minimum value required for the output amount.

PUL-6:The Gaslimit Dos in `executeVirtualOrdersUntilSpecifiedBlock`

Category	Severity	Code Reference	Status	Contributor
DOS	**Medium**	**code/twamm/contracts/libraries/LongTermOrders.sol#353-369**	**Declined**	**Hellobloc**

Code

```
353:        for (uint256 i = 0; i < expiriesList.length; i++) {
354:            if (
355:                (OrderPoolA.salesRateEndingPerBlock[expiriesList[i]] > 0 ||
356:                    OrderPoolB.salesRateEndingPerBlock[expiriesList[i]] > 0) &&
357:                (expiriesList[i] > self.lastVirtualOrderBlock &&
358:                    expiriesList[i] < blockNumber)
359:            ) {
360:                executeVirtualTradesAndOrderExpiries(
361:                    self,
362:                    reserveMap,
363:                    expiriesList[i]
364:                );
365:            }
366:        }
367:
368:        executeVirtualTradesAndOrderExpiries(self, reserveMap, blockNumber);
369:    }
```

Description

Hellobloc : `executeVirtualOrdersUntilSpecifiedBlock` has a traversal of the `expireblocklist`, which will cause the loop to traverse too many times when a large number of `expireblocks` are squeezed without updates, thus reaching the transaction `gaslimit`.

Eventually, the project is permanently deactivated.

```
function executeVirtualOrdersUntilSpecifiedBlock(
        LongTermOrders storage self,
        mapping(address => uint256) storage reserveMap,
        uint256 blockNumber
    ) public {
        ...
        for (uint256 i = 0; i < expiriesList.length; i++) {
            if (
                (OrderPoolA.salesRateEndingPerBlock[expiriesList[i]] > 0 ||
                    OrderPoolB.salesRateEndingPerBlock[expiriesList[i]] > 0) &&
                (expiriesList[i] > self.lastVirtualOrderBlock &&
                    expiriesList[i] < blockNumber)
            ) {
                executeVirtualTradesAndOrderExpiries(
                    self,
                    reserveMap,
                    expiriesList[i]
                );
            }
        }
        ...
    }
```

Given that `removeLiquidity` emptying `Pair` will result in the risk of `executeVirtualOrdersUntilSpecifiedBlock` revert. The possibility of a large number of `expireblock`s not being updated becomes greater.

Recommendation

Hellobloc : We recommend providing update removal methods for individual expired blocks and adding a upper limit check for the corresponding data push operations.

Client Response

BinarySearchTree contract creates a tree structure from all the expired blocks. It cannot manually delete expired block.

PUL-7:Tokens received after LongTermSwap may be smaller than expected due to precision loss

Category	Severity	Code Reference	Status	Contributor
Language Specific	**Medium**	**code/twamm/contracts/libraries/LongTermOrders.sol#L139-L140**	**Fixed**	**thereksfour**

Code

```
139:        uint256 sellingRate = amount / (orderExpiry - currentBlock);
140:
```

Description

thereksfour : There is an precision loss when calculating the sellingRate in the performLongTermSwap function Consider the following scenario, where self.orderBlockInterval = 500, block.number = 999, and the user plans to swap 10000 tokens in 2000 blocks After the following calculation, sellingRate = 9.990, due to the precision loss, sellingRate = 9 So the actual number of tokens swapped by the user is 9 * 1001 = 9009, not 10000

```
        uint256 currentBlock = block.number;   //   999
        uint256 lastExpiryBlock = currentBlock -    // 999 - 999 % 500 = 500
            (currentBlock % self.orderBlockInterval);
        uint256 orderExpiry = self.orderBlockInterval * // 500 * (2+1) + 500 = 2000
            (numberOfBlockIntervals + 1) +
            lastExpiryBlock;
        uint256 sellingRate = amount / (orderExpiry - currentBlock); // 10000 / (2000 - 999) = 10000 / 1001
```

Considering the decimals of most tokens, 10000 is a very small amount, but for some special tokens (Gemini dollar: https://etherscan.io/token/0x056Fd409E1d7A124BD7017459dFEa2F387b6d5Cd, with a decimal of 2), this means that $10 is lost when swapping a $100 token. So to be compatible with different tokens, it should be considered to keep three digits in the calculation of the sellingRate.

Recommendation

thereksfour :

```
- 139          uint256 sellingRate = amount / (orderExpiry - currentBlock);
+ 139          uint256 sellingRate = amount * 1000 / (orderExpiry - currentBlock);


- 197:         order.sellAmount = (block.number - order.submitBlock) * order.saleRate;
+ 197:         order.sellAmount = (block.number - order.submitBlock) * order.saleRate / 1000;
            order.sellAmount =
                (order.expirationBlock - order.submitBlock) *
- 247:          order.saleRate;
+ 247:          order.saleRate / 1000;
        } else {
            order.sellAmount =
                (block.number - order.submitBlock) *
- 251:          order.saleRate;
+ 251:          order.saleRate / 1000;
```

Client Response

Fixed. Multiply by 10000 to reduce precision loss

PUL-8:Unsupported fee-on-transfer tokens

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	code/twamm/contracts/Pair.sol#L215-L224 code/twamm/contracts/TWAMM.sol#L240-L245	Mitigated	thereksfour

Code

```
215:        IERC20(tokenA).safeTransfer(twamm, amountAOut);
216:        IERC20(tokenB).safeTransfer(twamm, amountBOut);
217:
218:        if (feeOn)
219:            rootKLast = reserveMap[tokenA]
220:                .fromUint()
221:                .sqrt()
222:                .mul(reserveMap[tokenB].fromUint().sqrt())
223:                .toUint();
224:        emit LiquidityRemoved(to, lpTokenAmount, amountAOut, amountBOut);


240:
241:        require(
242:            IERC20(token0).balanceOf(address(this)) >= amountOut0 &&
243:                IERC20(token1).balanceOf(address(this)) >= amountOut1,
244:            "Inaccurate Amount for Tokens."
245:        );
```

Description

thereksfour : According to https://github.com/d-xo/weird-erc20/#fee-on-transfer, there are ERC20 tokens that charge fee for every transfer() or transferFrom(). However, the current implementation does not support fee-on-transfer tokens.

For example, consider a user creates a pair with fee-on-transfer tokens and provides liquidity, then if the user withdraws the liquidity, Pair.removeLiquidity sends the tokens to the twamm contract and returns amountAOut/amountBOut, and since fees are charged in the process, the amount of tokens received by the twamm contract will be less than amountAOut/amountBOut , and the following check will fail, resulting in the user not being able to remove the liquidity.

```
            require(
                IERC20(token0).balanceOf(address(this)) >= amountOut0 &&
                    IERC20(token1).balanceOf(address(this)) >= amountOut1,
                "Inaccurate Amount for Tokens."
            );
```

Recommendation

thereksfour : Consider limiting the tokens that can be used to create the pair. Or add support for such tokens.

Client Response

The issues is mitigated by the front end app restricts the creation of fee-on-transfer tokens pair.

PUL-9:Wrong usage of sortAmounts

Category	Severity	Code Reference	Status	Contributor
Logical	**Medium**	**code/twamm/contracts/TWAMM.sol#L234** **code/twamm/contracts/TWAMM.sol#L266**	**Declined**	**alansh**

Code

```
234:        (amountOut0, amountOut1) = Library.sortAmounts(

266:        (amountTokenOut, amountETHOut) = Library.sortAmounts(
```

Description

alansh : `sortAmounts` expect `amountOutA` to match `token`, `amountOutB` to match `WETH`, here it doesn't necessarily hold.
alansh : `sortAmounts` expect `amountOutA` to match `token0`, `amountOutB` to match `token1`, here it doesn't necessarily hold.

Recommendation

alansh : Find tokenA and tokenB by sortTokens(token0, token1) instead.
alansh : Find tokenA and tokenB by sortTokens(token0, token1) instead.

Client Response

The output `amountOutA` and `amountOutB` from the `IPair::removeLiquidity()` function is already sorted based on the token address, i.e. `amountOutA` corresponds to the lower address `tokenA`. The function `sortAmounts` finds the correct amount for each token, and results `amountOut0` is the amount associated with the input parameter `token0` and same goes to `amountOut1` and `token1`. Hence it is not an issue.

PUL-10: `RemoveLiquidity` has the Potential to Completely Empty the `Pair`, which would Lead to A Potential Fraud Risk.

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	code/twamm/contracts/Pair.sol#L129-L158 code/twamm/contracts/Pair.sol#L202-L244 code/twamm/contracts/libraries/LongTermOrders.sol#L395-L402	Fixed	Hellobloc

Code

```
            .fromUint()
            .sqrt()
            .mul(amountB.fromUint().sqrt())
            .toUint();

        bool feeOn = mintFee(0, 0);
        _mint(to, lpTokenAmount);

        if (feeOn) rootKLast = lpTokenAmount;
        emit InitialLiquidityProvided(to, lpTokenAmount, amountA, amountB);
    }

    ///@notice provide liquidity to the AMM
    ///@param lpTokenAmount number of lp tokens to mint with new liquidity
    function provideLiquidity(address to, uint256 lpTokenAmount)
        external
        override
        checkCaller
        nonReentrant
        returns (uint256 amountAIn, uint256 amountBIn)
    {
        //execute virtual orders
        longTermOrders.executeVirtualOrdersUntilSpecifiedBlock(
            reserveMap,
            block.number
        );

        require(lpTokenAmount > 0, "Invalid Amount");
        require(totalSupply() != 0, "No Liquidity Has Been Provided Yet");


        uint256 reserveA = reserveMap[tokenA];
        uint256 reserveB = reserveMap[tokenB];

        //the ratio between the number of underlying tokens and the number of lp tokens must remain invariant after burn
        amountAOut = (reserveA * lpTokenAmount) / totalSupply();
        amountBOut = (reserveB * lpTokenAmount) / totalSupply();

        reserveMap[tokenA] -= amountAOut;
        reserveMap[tokenB] -= amountBOut;

        bool feeOn = mintFee(reserveA, reserveB);
        _burn(to, lpTokenAmount);

        IERC20(tokenA).safeTransfer(twamm, amountAOut);
        IERC20(tokenB).safeTransfer(twamm, amountBOut);

        if (feeOn)
            rootKLast = reserveMap[tokenA]
                .fromUint()
                .sqrt()
                .mul(reserveMap[tokenB].fromUint().sqrt())
                .toUint();
        emit LiquidityRemoved(to, lpTokenAmount, amountAOut, amountBOut);
    }

    ///@notice instant swap a given amount of tokenA against embedded amm
    function instantSwapFromAToB(address sender, uint256 amountAIn)
        external
        override
        checkCaller
        nonReentrant
        returns (uint256 amountBOut)
    {
        require(
            reserveMap[tokenA] > 0 && reserveMap[tokenB] > 0,
            "Insufficient Liquidity"
        );
        require(amountAIn > 0, "Invalid Amount");
        amountBOut = performInstantSwap(tokenA, tokenB, amountAIn);

        emit InstantSwapAToB(sender, amountAIn, amountBOut);
    }


            tokenAOut =
                ((tokenAStart + tokenAIn) * tokenBIn) /
                (tokenBStart + tokenBIn);
            tokenBOut =
                ((tokenBStart + tokenBIn) * tokenAIn) /
                (tokenAStart + tokenAIn);
            ammEndTokenA = tokenAStart + tokenAIn - tokenAOut;
            ammEndTokenB = tokenBStart + tokenBIn - tokenBOut;
```

Description

Hellobloc : The first provision of liquidity in `uniswapv2:pair` locks `MINIMUM_LIQUIDITY` amount of liquidity tokens, thus ensuring that no liquidity provider can completely `empty` the `Pair`.

```
    if (_totalSupply == 0) {
            liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
           _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY tokens
      }
```

However, this design has not been extended to the `Pulsar Protocol`, which leads to the possibility that the provider of all liquidity has the ability to empty the `Pair`, resulting in `reserve` and `totalsupply` becoming zero.

This will result in the following risks.

1. The result is that the `executeVirtualOrdersUntilSpecifiedBlock` operation cannot be executed when there is only some long-term buy orders from A to B. This eventually causes a denial of service for `Pair`. This further prevents the user from invoking operations such as `cancelLongTermSwap`.

Reason The reason why `executeVirtualOrdersUntilSpecifiedBlock` cannot be executed is that `computeVirtualBalances` will be run in it. And in `computeVirtualBalances`, a division operation with `(tokenBStart + tokenBIn)` as the denominator is performed.

And since there are only buy orders from A to B in a long term order, i.e. `tokenBIn = 0`, and `tokenBStart = reserve = 0`, the transaction is eventually revert. **Code**

```
   function computeVirtualBalances(
          uint256 tokenAStart,
          uint256 tokenBStart,
          uint256 tokenAIn,
          uint256 tokenBIn
      )
      {        ...
             tokenAOut =
                 ((tokenAStart + tokenAIn) * tokenBIn) /
                 (tokenBStart + tokenBIn);
             tokenBOut =
                 ((tokenBStart + tokenBIn) * tokenAIn) /
                 (tokenAStart + tokenAIn);
             ammEndTokenA = tokenAStart + tokenAIn - tokenAOut;
             ammEndTokenB = tokenBStart + tokenBIn - tokenBOut;
      }
```

2. This leads to the possibility of calling `provideInitialLiquidity` twice to set an unreasonable price at low cost.

Reason The basis for the first liquidity addition is determined by `totalSupply == 0`. However, when the `liquidity` is completely removed, the value will be `0`. This will lead to malicious liquidity provider being able to set an `unreasonable price` for `Pair` at low cost, which will eventually lead to `longTermSwap` users being unable to cancel their orders and having to accept the unreasonable price for swap, resulting in fraudulent attacks. **Code**

```
function provideLiquidity(address to, uint256 lpTokenAmount)
        ...
    {
        ...
        reserveMap[tokenA] += amountAIn;
        reserveMap[tokenB] += amountBIn;
    }
function provideInitialLiquidity(
        address to,
        uint256 amountA,
        uint256 amountB
    )
    ...
    {
        require(amountA > 0 && amountB > 0, "Invalid Amount");
        require(totalSupply() == 0, "Liquidity Has Already Been Provided");

        reserveMap[tokenA] = amountA;
        reserveMap[tokenB] = amountB;

        //initial LP amount is the geometric mean of supplied tokens
        lpTokenAmount = amountA
            .fromUint()
            .sqrt()
            .mul(amountB.fromUint().sqrt())
            .toUint();

        bool feeOn = mintFee(0, 0);
        _mint(to, lpTokenAmount);

        if (feeOn) rootKLast = lpTokenAmount;
        emit InitialLiquidityProvided(to, lpTokenAmount, amountA, amountB);
    }
```

The following restrictions may be required for the above attack.

1. the pool only exists from A to B or B to A in one of the buy orders.
2. feeto is `0x0`, otherwise it needs `feeto` address to cooperate with the evil
3. first in the `removeLiquidity` after a period of block time to `provideInitialLiquidity` call.ecommendation:

Recommendation

Hellobloc : We recommend locking in a certain amount of liquidity tokens at the time of the `provideInitialLiquidity` to ensure that liquidity cannot be completely emptied.

Client Response

Fixed. We acknowledged this issue and fixed by turning on `feeOn` flag and set the `feeArg` equal to be `1000`. When the `feeOn` is True, there will always be some lpToken in the protocol (not burned) and hence the pool will never be zero.

PUL-11: `safeTransferFrom` lacks `isContract` check

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	code/banana/contracts/libraries/TransferHelper.sol#L32-L44 code/twamm/contracts/libraries/TransferHelper.sol#L32-L44	Acknowledged	Hellobloc

Code

```
    function safeTransferFrom(
        address token,
        address from,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, value));
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper::transferFrom: transferFrom failed"
        );
    }

            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper::safeTransfer: transfer failed"
        );
    }

    function safeTransferFrom(
        address token,
        address from,
        address to,
        uint256 value
    ) public {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(
```

Description

Hellobloc : The `token` address `isContract` check is missing in `TransferHelper`. We should ensure that the `token` address exists code to ensure that an invalid call to the EOA's address is not executed.

```
function safeTransferFrom(
        address token,
        address from,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, value));
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper::transferFrom: transferFrom failed"
        );
    }
```

Recommendation

Hellobloc : We recommend adding the `isContract()` check for `safeTransferFrom`

Client Response

We acknowledged the submission and choose not to enhance.

PUL-12:no need to re-calculate k

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	**Informational**	**code/twamm/contracts/libraries/LongTermOrders.sol#L457**	**Acknowledged**	**alansh**

Code

```
        int256 eDenominator = aStart.sqrt().mul(bStart.sqrt()).inv();
```

Description

alansh : `k` is already calculated and passed in as a parameter.

Recommendation

alansh : `int256 eDenominator = k.sqrt().inv();`

Client Response

We acknowledged the submission and choose not to enhance.

PUL-13:redundant check

Category	Severity	Code Reference	Status	Contributor
Logical	**Informational**	**code/twamm/contracts/libraries/BinarySearchTree.sol#L185**	**Acknowledged**	**alansh**

Code

```
185:        } else if (curNode.left == 0 && curNode.right != 0) {
```

Description

alansh : In this branch, curNode.left == 0 is guaranteed

Recommendation

alansh : remove the curNode.left == 0 check

Client Response

We acknowledged the submission and choose not to enhance.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.