BLOCKSEC

Security Audit

Report for Bubbly Finance Contracts

Date: September 27, 2024 **Version:** 1.1

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**

1.1 About Target Contracts . . . 1
1.2 Procedure of Auditing . . . 2
1.2.1 Software Security . . . 2
1.2.2 DeFi Security . . . 2
1.2.3 NFT Security . . . 3
1.2.4 Additional Recommendation . . . 3
1.3 Security Model . . . 3

Chapter 2 Findings **5**

2.1 Software Security . . . 6
2.1.1 Incorrect transfer logic in function `Deliver()` . . . 6
2.1.2 Incorrect calculation of `poolReserve[pool]` in function `DeliverforX()` . . . 7
2.1.3 Incorrect assignment of `vtokenToToken` in function `setToken()` . . . 8
2.1.4 Incorrect usages of `add()` and `sub()` in function `Deliver()` . . . 9
2.1.5 Lack of deliver methods in the CPM . . . 10
2.2 DeFi Security . . . 12
2.2.1 Incorrect record handling in function `DeliverlistX()` and `DeliverlistY()` 12
2.2.2 Lack of access control on closing positions . . . 14
2.2.3 Potential incorrect update for collateral positions' `token1Amount` . . . 16
2.2.4 Incorrect value of `burnParams.lpcollateral` when invoking function `burn()` 18
2.2.5 Incorrect time check in function `DeliverforX()` . . . 20
2.2.6 Incorrect price check in function `DeliverforX()` . . . 21
2.2.7 Inconsistent calculation for `amountY` in function `DeliverforX()` . . . 22
2.2.8 Incorrect fee accounting logic in function `swap()` . . . 22
2.2.9 Potential inconsistent calculation for the `amount0inUSD` . . . 24
2.2.10 Incorrect check on `amountOutReceived` in function `exactOutputInternal()` 26
2.2.11 Lack of forced liquidation to defend against bad debts . . . 27
2.3 Additional Recommendation . . . 28
2.3.1 Redundant function `createAndInitializePoolIfNecessary()` . . . 28
2.3.2 Redundant variables . . . 29
2.3.3 Wrong comment in function `exactOutputInternal()` . . . 30
2.3.4 Use `safemath` in logic calculations that differ from UniswapV3 . . . 31
2.3.5 Add a check on `params.quoteToken` . . . 31
2.4 Note . . . 33
2.4.1 Potential centralization risks . . . 33

Report Manifest

Item	Description
Client	Bubbly
Target	Bubbly Finance Contracts

Version History

Version	Date	Description
1.0	July 30, 2024	First release
1.1	September 27, 2024	Second release

Signature

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The focus of this audit is on the Bubbly Finance Contracts[1] of the Bubbly. The Bubbly introduces a decentralized exchange (DEX) that offers a Uniswap-like trading experience for pre-market assets such as points and pre-launch tokens. It uses a Margin Liquidity Market Maker (MLMM) model, blending Concentrated Liquidity Market Makers (CLMM) with margin trading principles to enhance liquidity and price discovery. This approach aims to improve trading efficiency and liquidity for assets that lack traditional market solutions.

Please note that the audit scope is limited to the following smart contracts [2]:

```
1 contracts/BubblyFactory.sol
2 contracts/BubblyPool.sol
3 contracts/CollateralPositionManager.sol
4 contracts/NonfungiblePositionManager.sol
5 contracts/Delivery.sol
```

Listing 1.1: Audit Scope for this Report

Other files are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Please note that the new functions added `Version 3` are not within the scope of the audit.

Project	Version	Commit Hash
Bubbly Finance Contracts	`Version 1`	`8968a5a9bc60f348459a06674f370fa92138a554`
	`Version 2`	`02d7bea659f7b668aff982865ba24886e60baabd`
	`Version 3`	`84189fbbe8bdab168cb7d0d567c863fa10766e22`

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset.

[1] https://github.com/bubbly-finance/bubbly-core

[2] File Delivery.sol is renamed as newDelivery.sol in version 2

Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.2 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.2.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.2.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic

- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.2.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.2.4 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.3 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [3] and Common Weakness Enumeration [4]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

[3] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[4] https://cwe.mitre.org/

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **sixteen** potential security issues. Besides, we have **five** recommendations and **one** note.

- High Risk: 12
- Medium Risk: 3
- Low Risk: 1
- Recommendation: 5
- Note: 1

ID	Severity	Description	Category	Status
1	High	Incorrect transfer logic in function `Deliver()`	Software Security	Fixed
2	High	Incorrect calculation of `poolReserve[pool]` in function `DeliverforX()`	Software Security	Fixed
3	High	Incorrect assignment of `vtokenToToken` in function `setToken()`	Software Security	Fixed
4	High	Incorrect usages of `add()` and `sub()` in function `Deliver()`	Software Security	Fixed
5	Medium	Lack of deliver methods in the CPM	Software Security	Fixed
6	High	Incorrect record handling in function `DeliverlistX()` and `DeliverlistY()`	DeFi Security	Fixed
7	High	Lack of access control on closing positions	DeFi Security	Fixed
8	High	Potential incorrect update for collateral positions' `token1Amount`	DeFi Security	Fixed
9	High	Incorrect value of `burnParams.lpcollateral` when invoking function `burn()`	DeFi Security	Fixed
10	Medium	Incorrect time check in function `DeliverforX()`	DeFi Security	Fixed
11	High	Incorrect price check in function `DeliverforX()`	DeFi Security	Fixed
12	High	Inconsistent calculation for `amountY` in function `DeliverforX()`	DeFi Security	Fixed
13	High	Incorrect fee accounting logic in function `swap()`	DeFi Security	Fixed
14	Low	Potential inconsistent calculation for the `amount0inUSD`	DeFi Security	Fixed

15	Medium	Incorrect check on `amountOutReceived` in function `exactOutputInternal()`	DeFi Security	Fixed
16	High	Lack of forced liquidation to defend against bad debts	DeFi Security	Fixed
17	-	Redundant function `createAndInitializePoolIfNecessary()`	Recommendation	Fixed
18	-	Redundant variables	Recommendation	Confirmed
19	-	Wrong comment in function `exactOutputInternal()`	Recommendation	Fixed
20	-	Use `safemath` in logic calculations that differ from UniswapV3	Recommendation	Fixed
21	-	Add a check on `params.quoteToken`	Recommendation	Fixed
22	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Software Security

2.1.1 Incorrect transfer logic in function `Deliver()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Delivery`, the function `Deliver()` allows sellers to deliver tokens to buyers. Specifically, the contract should receive `token0` (i.e., `vtokenToToken`) from sellers. However, the function invokes `TransferHelper.safeTransfer(vtokenToToken[pool], address(this), amount0)` to transfer the `token0` to itself, which is incorrect.

```
53  function Deliver(address pool, bool forX) external returns (uint256 amount0,uint256 amount1 ,
        bool getToken){
54      //pool flag check
55      require(DeliverBeginTime[pool] != uint32(0),'Time not set');
56      require(_blockTimestamp() >= DeliverBeginTime[pool], 'Delivery not start');
57      require(DeliverEpoch != uint32(0));
58      //check overflow
59      require(_blockTimestamp() <= DeliverBeginTime[pool] + DeliverEpoch, 'Delivery after 24hs');
60      require(IBubblyPool(pool).deliveryflag());
61      // check token address
62      require(vtokenToToken[pool]!= address(0));
63      getToken = forX;
64      (amount0, amount1) = _getPosition(pool, forX);
65      //check tokenBalance
66      if(!forX){
67          //seller delivery
68          require(!DeliverlistY[msg.sender], 'already delivered' );
69          DeliverlistY[msg.sender] = true;
```

```
70          //transfer real tokenY from msg.sender to contract for delivery
71          TransferHelper.safeTransfer(vtokenToToken[pool], address(this), amount0);
72          TransferHelper.safeTransferFrom(IBubblyPool(pool).token1(), pool, msg.sender, 2 *
                amount1);
73          //change state
74          poolTotalToken[pool].add(amount0);
75          poolTotalQuoteToken[pool].add(amount1);
76          poolReserve[pool].add(amount0);
77      }
78      else{
79          //buyer delivery
80          require(!DeliverlistX[msg.sender], 'already delivered' );
81          DeliverlistX[msg.sender] = true;
82          require(poolReserve[pool] >= amount0 , 'not enough balance');
83          //use library
84          require(FullMath.mulDiv(poolTotalQuoteToken[pool], uint256(1), poolTotalToken[pool]) <=
                 FullMath.mulDiv(amount1, uint256(1), amount0), 'price too low');
85          TransferHelper.safeTransferFrom(vtokenToToken[pool], address(this), msg.sender, amount0
                );
86          poolReserve[pool].sub(amount0);
87      }
88      emit DeliverBeforeDeadline(msg.sender, amount0, amount1, forX, pool);
89
90  }
```

Listing 2.1: contracts/Delivery.sol

Impact The delivery cannot function as intended.

Suggestion Revise the transfer method and use `TransferHelper.safeTransferFrom(vtokenToToken[pool], msg.sender, address(this), amount0)` instead.

2.1.2 Incorrect calculation of `poolReserve[pool]` in function `DeliverforX()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `Delivery` contract, the calculation of the `poolReserve[pool]` in the function `DeliverforX()` is wrong. Specifically, the function currently fails to subtract `amountX` from `poolReserve[pool]` after executing the token transfer from `address(this)` to `msg.sender`. This oversight can lead to a scenario where if the balance of `address(this)` falls below `amountX`, the invocation of `TransferHelper.safeTransferFrom(vtokenToToken[pool], address(this), msg.sender, amountX)` will fail, preventing buyers from completing their deliveries.

```
92  function DeliverforX(address pool) external returns (uint256 amountX,uint256 amountY){
93      //pool flag check
94      require(DeliverBeginTime[pool] != uint32(0),'Time not set');
95      require(DeliverEpoch != uint32(0));
96      require(_blockTimestamp() >= DeliverBeginTime[pool] + DeliverEpoch, 'Delivery before 24hs')
            ;
97      require(IBubblyPool(pool).deliveryflag());
```

```
98      // check token address
99      require(vtokenToToken[pool]!= address(0));
100     (uint256 amount0, uint256 amount1) = _getPosition(pool, true);
101     //frontend control delivery amount = amount0
102     if(poolReserve[pool] == 0 || FullMath.mulDiv(poolTotalQuoteToken[pool], uint256(1),
            poolTotalToken[pool]) <= FullMath.mulDiv(amount1, uint256(1), amount0)){
103         //get double collateral
104         amountY = 2 * amount1;
105         TransferHelper.safeTransferFrom(IBubblyPool(pool).token1(), pool, msg.sender, amountY);
106     }
107     else{
108         //get tokenX
109         amountX = amount0 > poolReserve[pool] ? poolReserve[pool] : amount0;
110         TransferHelper.safeTransferFrom(vtokenToToken[pool], address(this), msg.sender, amountX
                );
111         amountY = amount1 - FullMath.mulDiv(amountX , amount1 , amount0);
112         if(amountY != 0) TransferHelper.safeTransferFrom(IBubblyPool(pool).token1(), pool, msg.
                sender, amountY);
113     }
114     //buyer delivery
115     require(!DeliverlistX[msg.sender], 'already delivered' );
116     DeliverlistX[msg.sender] = true;
117     emit DeliverAfterDeadline(msg.sender, amountX, amountY, pool);
118 }
```

Listing 2.2: contracts/Delivery.sol

Impact Buyers may not be delivered.

Suggestion The `poolReserve[pool]` should subtract `amountX` after transferring the corresponding tokens from `address(this)` to `msg.sender`.

2.1.3 Incorrect assignment of `vtokenToToken` in function `setToken()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `Delivery` contract, the assignment `vtokenToToken[vtoken] = token` is incorrect in the function `setToken()`. It should be `vtokenToToken[pool] = token`.

```
120 function setToken(
121     address pool,
122     address token,
123     address vtoken
124 ) external onlyFactoryOwner {
125     require(IBubblyPool(pool).deliveryflag());
126     require(vtokenToToken[pool] == address(0));
127     vtokenToToken[vtoken] = token;
128     emit SetDeliveryToken(pool, vtoken ,token);
129 }
```

Listing 2.3: contracts/Delivery.sol

Impact The functions `Deliver()` and `DeliverforX()` cannot function as intended.

Suggestion Change the assignment to `vtokenToToken[pool] = token`.

2.1.4 Incorrect usages of `add()` and `sub()` in function `Deliver()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Delivery`, the function `Deliver()` incorrectly uses functions in the `LowGasSafeMath` library to update the state variables. Specifically, the function does not assign the return values to the variables.

```
7   /// @notice Returns x + y, reverts if sum overflows uint256
8   /// @param x The augend
9   /// @param y The addend
10  /// @return z The sum of x and y
11  function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
12      require((z = x + y) >= x);
13  }
14
15  /// @notice Returns x - y, reverts if underflows
16  /// @param x The minuend
17  /// @param y The subtrahend
18  /// @return z The difference of x and y
19  function sub(uint256 x, uint256 y) internal pure returns (uint256 z) {
20      require((z = x - y) <= x);
21  }
```

Listing 2.4: libraries/LowGasSafeMath.sol

```
53  function Deliver(address pool, bool forX) external returns (uint256 amount0,uint256 amount1 ,
        bool getToken){
54      //pool flag check
55      require(DeliverBeginTime[pool] != uint32(0),'Time not set');
56      require(_blockTimestamp() >= DeliverBeginTime[pool], 'Delivery not start');
57      require(DeliverEpoch != uint32(0));
58      //check overflow
59      require(_blockTimestamp() <= DeliverBeginTime[pool] + DeliverEpoch, 'Delivery after 24hs');
60      require(IBubblyPool(pool).deliveryflag());
61      // check token address
62      require(vtokenToToken[pool]!= address(0));
63      getToken = forX;
64      (amount0, amount1) = _getPosition(pool, forX);
65      //check tokenBalance
66      if(!forX){
67          //seller delivery
68          require(!DeliverlistY[msg.sender], 'already delivered' );
69          DeliverlistY[msg.sender] = true;
70          //transfer real tokenY from msg.sender to contract for delivery
71          TransferHelper.safeTransfer(vtokenToToken[pool], address(this), amount0);
```

```
72          TransferHelper.safeTransferFrom(IBubblyPool(pool).token1(), pool, msg.sender, 2 *
                amount1);
73          //change state
74          poolTotalToken[pool].add(amount0);
75          poolTotalQuoteToken[pool].add(amount1);
76          poolReserve[pool].add(amount0);
77      }
78      else{
79          //buyer delivery
80          require(!DeliverlistX[msg.sender], 'already delivered' );
81          DeliverlistX[msg.sender] = true;
82          require(poolReserve[pool] >= amount0 , 'not enough balance');
83          //use library
84          require(FullMath.mulDiv(poolTotalQuoteToken[pool], uint256(1), poolTotalToken[pool]) <=
                 FullMath.mulDiv(amount1, uint256(1), amount0), 'price too low');
85          TransferHelper.safeTransferFrom(vtokenToToken[pool], address(this), msg.sender, amount0
                );
86          poolReserve[pool].sub(amount0);
87      }
88      emit DeliverBeforeDeadline(msg.sender, amount0, amount1, forX, pool);
89
90  }
```

Listing 2.5: contracts/Delivery.sol

Impact The state variables cannot be properly updated, incapacitating the delivery process.

Suggestion Revise the function usage.

2.1.5 Lack of deliver methods in the CPM

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Currently, the CPM will set the position recipient as itself if `params.recipient` is zero. However, the CPM does not provide any method to deliver its positions and get the assets, which means the corresponding assets will be locked in the pool.

```
138 function exactInputInternal(
139     internalExactInputSingleParams memory params,
140     SwapCallbackData memory data
141 ) private returns (uint256 amountOut) {
142     // allow swapping to the router address with address 0
143     if (params.recipient == address(0)) params.recipient = address(this);
144
145     (address tokenIn, address tokenOut, uint24 fee) = data.path.decodeFirstPool();
146
147     bool zeroForOne = tokenIn != params.quoteToken;
148     if(params.isOpen == false){
149         //tokenIn is always basetoken when close a position
150         require(tokenIn != params.quoteToken);
151     }
```

```
152     (int256 amount0, int256 amount1 ,uint256 totalFeeInQuoteToken) =
153         getPool(params.quoteToken, tokenIn, tokenOut, fee).swap(
154             IBubblyPoolActions.SwapParams({
155                 recipient : params.recipient,
156                 zeroForOne : zeroForOne,
157                 amountSpecified : params.amountIn.toInt256(),
158                 isOpen : params.isOpen,
159                 collateralamount : params.collateralamount,
160                 sqrtPriceLimitX96 : params.sqrtPriceLimitX96 == 0
161                 ? (zeroForOne ? TickMath.MIN_SQRT_RATIO + 1 : TickMath.MAX_SQRT_RATIO - 1)
162                 : params.sqrtPriceLimitX96
163             }),
164             abi.encode(data)
165         );
166     //long or short
167     bool long = params.isOpen != zeroForOne;
168     address pool = address(getPool(params.quoteToken, tokenIn, tokenOut, fee));
169     int256 amount1OnPosition = amount1 > 0 ? amount1 - int256(totalFeeInQuoteToken) : amount1 +
          int256(totalFeeInQuoteToken) ;
170     _updateposition(
171         UpdatePositionParams({
172             isOpen : params.isOpen,
173             long : long,
174             recipient : params.recipient,
175             pool : pool,
176             amount0 : uint256((amount0 > 0) ? amount0 : (-amount0)) ,
177             amount1 : uint256((amount1OnPosition > 0)? amount1OnPosition:(-amount1OnPosition))
178         })
179     );
180
181     return uint256(-(zeroForOne ? amount1 : amount0));
182 }
```

Listing 2.6: contracts/CollateralPositionManager.sol

```
212 function exactOutputInternal(
213     internalExactOutputSingleParams memory params,
214     SwapCallbackData memory data
215 ) private returns (uint256 amountIn) {
216     // allow swapping to the router address with address 0
217     if (params.recipient == address(0)) params.recipient = address(this);
218
219     (address tokenOut, address tokenIn, uint24 fee) = data.path.decodeFirstPool();
220
221     bool zeroForOne = tokenIn != params.quoteToken;
222     if(params.isOpen == false){
223         //tokenIn is always basetoken when close a position
224         require(tokenIn == params.quoteToken);
225     }
226     (int256 amount0Delta, int256 amount1Delta, uint256 totalFeeInQuoteToken) =
227         getPool(params.quoteToken, tokenIn, tokenOut, fee).swap(
228             IBubblyPoolActions.SwapParams({
229                 recipient : params.recipient,
```

```
230                 zeroForOne : zeroForOne,
231                 amountSpecified : -params.amountOut.toInt256(),
232                 isOpen : params.isOpen,
233                 collateralamount : params.collateralamount,
234                 sqrtPriceLimitX96: params.sqrtPriceLimitX96 == 0
235                 ? (zeroForOne ? TickMath.MIN_SQRT_RATIO + 1 : TickMath.MAX_SQRT_RATIO - 1)
236                 : params.sqrtPriceLimitX96
237             }),
238             abi.encode(data)
239         );
240     //stack too deep
241     {
242         uint256 amountOutReceived;
243         (amountIn, amountOutReceived) = zeroForOne
244             ? (uint256(amount0Delta), uint256(-amount1Delta))
245             : (uint256(amount1Delta), uint256(-amount0Delta));
246
247         // it's technically possible to not receive the full output amount,
248         // so if no price limit has been specified, require this possibility away
249         if (params.sqrtPriceLimitX96 == 0) require(amountOutReceived == params.amountOut);
250     }
251     address pool = address(getPool(params.quoteToken, tokenIn, tokenOut, fee));
252     bool long = params.isOpen != zeroForOne;
253     int256 amount1OnPosition = amount1Delta > 0 ? amount1Delta - int256(totalFeeInQuoteToken) :
          amount1Delta + int256(totalFeeInQuoteToken) ;
254     _updateposition(
255         UpdatePositionParams({
256             isOpen : params.isOpen,
257             long : long,
258             recipient : params.recipient,
259             pool : pool,
260             amount0 : uint256((amount0Delta > 0) ? amount0Delta : (-amount0Delta)),
261             amount1 : uint256((amount1OnPosition > 0) ? amount1OnPosition : (-amount1OnPosition
                ))
262         })
263     );
264 }
```

Listing 2.7: contracts/CollateralPositionManager.sol

Impact The CPM's assets will be locked in the pool.

Suggestion If there is no need to open positions for the CPM, it should revert when `params.recipient` is zero.

2.2 DeFi Security

2.2.1 Incorrect record handling in function `DeliverlistX()` and `DeliverlistY()`

Severity High

Status Fixed in `Version 2`

Introduced by Version 1

Description In the contract `Delivery`, the functions `Deliver()` and `DeliverforX()` utilize `DeliverlistX[msg.sender]` and `DeliverlistY[msg.sender]` to check if buyers and sellers have already made deliveries. This implementation does not account for different pool indices. As a result, buyers and sellers are restricted to making a single delivery across all pools rather than per pool. This oversight can lead to the potential loss of users' assets in pools other than the first one they interact with.

```
23  mapping(address => bool) public DeliverlistX;
24  mapping(address => bool) public DeliverlistY;
```

Listing 2.8: contracts/Delivery.sol

```
53  function Deliver(address pool, bool forX) external returns (uint256 amount0,uint256 amount1 ,
        bool getToken){
54      //pool flag check
55      require(DeliverBeginTime[pool] != uint32(0),'Time not set');
56      require(_blockTimestamp() >= DeliverBeginTime[pool], 'Delivery not start');
57      require(DeliverEpoch != uint32(0));
58      //check overflow
59      require(_blockTimestamp() <= DeliverBeginTime[pool] + DeliverEpoch, 'Delivery after 24hs');
60      require(IBubblyPool(pool).deliveryflag());
61      // check token address
62      require(vtokenToToken[pool]!= address(0));
63      getToken = forX;
64      (amount0, amount1) = _getPosition(pool, forX);
65      //check tokenBalance
66      if(!forX){
67          //seller delivery
68          require(!DeliverlistY[msg.sender], 'already delivered' );
69          DeliverlistY[msg.sender] = true;
70          //transfer real tokenY from msg.sender to contract for delivery
71          TransferHelper.safeTransfer(vtokenToToken[pool], address(this), amount0);
72          TransferHelper.safeTransferFrom(IBubblyPool(pool).token1(), pool, msg.sender, 2 *
                amount1);
73          //change state
74          poolTotalToken[pool].add(amount0);
75          poolTotalQuoteToken[pool].add(amount1);
76          poolReserve[pool].add(amount0);
77      }
78      else{
79          //buyer delivery
80          require(!DeliverlistX[msg.sender], 'already delivered' );
81          DeliverlistX[msg.sender] = true;
82          require(poolReserve[pool] >= amount0 , 'not enough balance');
83          //use library
84          require(FullMath.mulDiv(poolTotalQuoteToken[pool], uint256(1), poolTotalToken[pool]) <=
                FullMath.mulDiv(amount1, uint256(1), amount0), 'price too low');
85          TransferHelper.safeTransferFrom(vtokenToToken[pool], address(this), msg.sender, amount0
                );
86          poolReserve[pool].sub(amount0);
87      }
```

```
88        emit DeliverBeforeDeadline(msg.sender, amount0, amount1, forX, pool);
89
90    }
91
92    function DeliverforX(address pool) external returns (uint256 amountX,uint256 amountY){
93        //pool flag check
94        require(DeliverBeginTime[pool] != uint32(0),'Time not set');
95        require(DeliverEpoch != uint32(0));
96        require(_blockTimestamp() >= DeliverBeginTime[pool] + DeliverEpoch, 'Delivery before 24hs')
              ;
97        require(IBubblyPool(pool).deliveryflag());
98        // check token address
99        require(vtokenToToken[pool]!= address(0));
100       (uint256 amount0, uint256 amount1) = _getPosition(pool, true);
101       //frontend control delivery amount = amount0
102       if(poolReserve[pool] == 0 || FullMath.mulDiv(poolTotalQuoteToken[pool], uint256(1),
              poolTotalToken[pool]) <= FullMath.mulDiv(amount1, uint256(1), amount0)){
103           //get double collateral
104           amountY = 2 * amount1;
105           TransferHelper.safeTransferFrom(IBubblyPool(pool).token1(), pool, msg.sender, amountY);
106       }
107       else{
108           //get tokenX
109           amountX = amount0 > poolReserve[pool] ? poolReserve[pool] : amount0;
110           TransferHelper.safeTransferFrom(vtokenToToken[pool], address(this), msg.sender, amountX
                  );
111           amountY = amount1 - FullMath.mulDiv(amountX , amount1 , amount0);
112           if(amountY != 0) TransferHelper.safeTransferFrom(IBubblyPool(pool).token1(), pool, msg.
                  sender, amountY);
113       }
114       //buyer delivery
115       require(!DeliverlistX[msg.sender], 'already delivered' );
116       DeliverlistX[msg.sender] = true;
117       emit DeliverAfterDeadline(msg.sender, amountX, amountY, pool);
118   }
```

Listing 2.9: contracts/Delivery.sol

Impact Users' assets in other pools will be lost when delivering.

Suggestion Change the type of `DeliverlistX` and `DeliverlistY` from `mapping(address => bool)` to `mapping(address => mapping(address => bool))` and use `DeliverlistX[pool][msg.sender]` and `DeliverlistY[pool][msg.sender]` instead.

2.2.2 Lack of access control on closing positions

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Currently, anyone can call the function `exactInputSingle()` or `exactOutputSingle()` with arbitrary `params.recipient`, which means anyone can close others' positions.

```
185   function exactInputSingle(ExactInputSingleParams calldata params)
186       external
187       payable
188       override
189       checkDeadline(params.deadline)
190       returns (uint256 amountOut)
191   {
192       uint256 collateralamount = _getcollateral(params.isOpen, params.tokenIn != params.
              quoteToken, address(getPool(params.quoteToken, params.tokenIn, params.tokenOut, params
              .fee)), params.recipient, params.amountIn);
193       amountOut = exactInputInternal(internalExactInputSingleParams({
194           amountIn : params.amountIn,
195           recipient : params.recipient,
196           sqrtPriceLimitX96: params.sqrtPriceLimitX96,
197           isOpen: params.isOpen,
198           collateralamount : collateralamount,
199           quoteToken : params.quoteToken
200           }),
201           SwapCallbackData({path: abi.encodePacked(params.tokenIn, params.fee, params.tokenOut),
                  payer: msg.sender})
202       );
203       require(amountOut >= params.amountOutMinimum, 'Too little received');
204   }
```

Listing 2.10: contracts/CollateralPositionManager.sol

```
267   function exactOutputSingle(ExactOutputSingleParams calldata params)
268       external
269       payable
270       override
271       checkDeadline(params.deadline)
272       returns (uint256 amountIn)
273   {
274       // avoid an SLOAD by using the swap return data
275       uint256 collateralamount = _getcollateral(params.isOpen, params.tokenIn != params.
              quoteToken, address(getPool(params.quoteToken, params.tokenIn, params.tokenOut, params
              .fee)), params.recipient, params.amountOut);
276       amountIn = exactOutputInternal(internalExactOutputSingleParams({
277           amountOut : params.amountOut,
278           recipient : params.recipient,
279           sqrtPriceLimitX96 : params.sqrtPriceLimitX96,
280           isOpen : params.isOpen,
281           collateralamount : collateralamount,
282           quoteToken : params.quoteToken
283           }),
284           SwapCallbackData({path: abi.encodePacked(params.tokenOut, params.fee, params.tokenIn),
                  payer: msg.sender})
285       );
286
287       require(amountIn <= params.amountInMaximum, 'Too much requested');
288       // has to be reset even though we don't use it in the single hop case
289       amountInCached = DEFAULT_AMOUNT_IN_CACHED;
290   }
```

Listing 2.11: contracts/CollateralPositionManager.sol

Impact Users lose assets.

Suggestion Add restriction that the `params.recipient` must be the `msg.sender`.

2.2.3 Potential incorrect update for collateral positions' `token1Amount`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `CollateralPositionManager` contract, the `_updatePosition()` function is designed to modify users' collateral positions. This function decreases `token0Amount` by a specified `amount0` and conditionally adjusts `token1Amount` based on whether the `token0Amount` is equal to zero upon closing a position. This conditional update can lead to discrepancies between the states of `token0Amount` and `token1Amount`. A malicious seller can partially close a short position while leaving a minimal residual of `token0`. This can enable the seller to redeem a large quantity of `token1` while delivering only a small amount of `token0`.

```
286   function _updateposition(UpdatePositionParams memory params) internal{
287       CollateralPosition memory oldposition = _checkposition(params.long, params.recipient,
              params.pool);
288       CollateralPosition memory newposition;
289       if(params.isOpen){
290           newposition = CollateralPosition({
291                                           token0amount:oldposition.token0amount.add(params.
                                                amount0) ,
292                                           token1amount:oldposition.token1amount.add(params.
                                                amount1)
293                                           });
294       }else{
295           require(oldposition.token0amount >= params.amount0,'pne');
296           newposition = CollateralPosition({
297                                           token0amount:oldposition.token0amount.sub(params.
                                                amount0) ,
298                                           token1amount:(oldposition.token0amount == params.
                                                amount0) ? 0 : oldposition.token1amount
299                                           });
300       }
301
302       if (params.long) {
303           _longpositions[params.pool][params.recipient] = newposition;
304       }
305       else{
306           _shortpositions[params.pool][params.recipient] = newposition;
307       }
308
309       emit updatePosition(params.recipient, params.pool, newposition.token0amount, newposition.
              token1amount, params.long, params.isOpen);
```

```
310 }
```

Listing 2.12: contracts/CollateralPositionManager.sol

```
53  function burn(
54      int24 tickLower,
55      int24 tickUpper,
56      uint128 amount,
57      burnParams calldata burnparams
58  ) external override lock onlyNPM returns (uint256 amount0, uint256 amount1,uint256 amount0used
        , uint256 amount1used,uint256 amountowed) {
59
60      (Position.Info storage position, int256 amount0Int, int256 amount1Int) =
61          _modifyPosition(
62              ModifyPositionParams({
63                  owner: msg.sender,
64                  tickLower: tickLower,
65                  tickUpper: tickUpper,
66                  liquidityDelta: -int256(amount).toInt128()
67              })
68          );
69
70      amount0 = uint256(-amount0Int);
71      amount1 = uint256(-amount1Int);
72      uint256 amount0inUSD;
73      if(amount0 > 0){
74          uint160 tickUpperSqrtPriceX96 = TickMath.getSqrtRatioAtTick(tickUpper);
75          uint160 tickLowerSqrtPriceX96 = TickMath.getSqrtRatioAtTick(tickLower);
76
77          if ( slot0.sqrtPriceX96 < tickLowerSqrtPriceX96 || slot0.sqrtPriceX96 >
                tickUpperSqrtPriceX96){
78              amount0inUSD = SqrtPriceMath.getAmount1Delta(tickLowerSqrtPriceX96,
                    tickUpperSqrtPriceX96, amount, false);
79          }else{
80              amount0inUSD = SqrtPriceMath.getAmount1Delta(slot0.sqrtPriceX96,
                    tickUpperSqrtPriceX96, amount, false);
81          }
82
83      }
84
85      if(burnparams.liquidity > 0){
86
87          amount0used = FullMath.mulDivRoundingUp(uint256(amount), burnparams.lpamount0, uint256(
                burnparams.liquidity));
88          amount1used = FullMath.mulDivRoundingUp(uint256(amount), burnparams.lpamount1, uint256(
                burnparams.liquidity));
89
90          uint256 amount1delta = amount1used > amount1 ? (amount1used - amount1) : (amount1 -
                amount1used);
91
92          amountowed = burnparams.lpcollateral - amount1delta;
93
94          if(amountowed > 0){
```

```
95              position.tokensOwed += uint128(amountowed);
96          }
97      }
98      emit Burn(msg.sender, tickLower, tickUpper, amount, amount0, amount1);
99  }
```

Listing 2.13: contracts/BubblyPool.sol

Impact Potential steal of funds due to the inconsistent update.

Suggestion Revise the update logic for the `token1Amount`.

2.2.4 Incorrect value of `burnParams.lpcollateral` when invoking function `burn()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `decreaseLiquidity()` of contract `NonfungiblePositionManager`, the provided value of parameter (i.e., `burnParams.lpcollateral`) to function `burn()` is incorrect. Specifically, the value should be collateral corresponding to the proportion of the amount in the total liquidity of the position instead of the total collateral supplied by the liquidity provider.

```
431  function burn(
432      int24 tickLower,
433      int24 tickUpper,
434      uint128 amount,
435      burnParams calldata burnparams
436  ) external override lock onlyNPM returns (uint256 amount0, uint256 amount1,uint256 amount0used
         , uint256 amount1used,uint256 amountowed) {
437
438      (Position.Info storage position, int256 amount0Int, int256 amount1Int) =
439          _modifyPosition(
440              ModifyPositionParams({
441                  owner: msg.sender,
442                  tickLower: tickLower,
443                  tickUpper: tickUpper,
444                  liquidityDelta: -int256(amount).toInt128()
445              })
446          );
447
448
449      amount0 = uint256(-amount0Int);
450      amount1 = uint256(-amount1Int);
451      uint256 amount0inUSD;
452      if(amount0 > 0){
453          uint160 tickUpperSqrtPriceX96 = TickMath.getSqrtRatioAtTick(tickUpper);
454          uint160 tickLowerSqrtPriceX96 = TickMath.getSqrtRatioAtTick(tickLower);
455
456          if ( slot0.sqrtPriceX96 < tickLowerSqrtPriceX96 || slot0.sqrtPriceX96 >
                 tickUpperSqrtPriceX96){
457              amount0inUSD = SqrtPriceMath.getAmount1Delta(tickLowerSqrtPriceX96,
                     tickUpperSqrtPriceX96, amount, false);
```

```
458             }else{
459                 amount0inUSD = SqrtPriceMath.getAmount1Delta(slot0.sqrtPriceX96,
                        tickUpperSqrtPriceX96, amount, false);
460             }
461
462         }
463
464         if(burnparams.liquidity > 0){
465
466             amount0used = FullMath.mulDivRoundingUp(uint256(amount), burnparams.lpamount0, uint256(
                    burnparams.liquidity));
467             amount1used = FullMath.mulDivRoundingUp(uint256(amount), burnparams.lpamount1, uint256(
                    burnparams.liquidity));
468
469
470             uint256 amount1delta = amount1used > amount1 ? (amount1used - amount1) : (amount1 -
                    amount1used);
471
472             amountowed = burnparams.lpcollateral - amount1delta;
473
474
475             if(amountowed > 0){
476                 position.tokensOwed += uint128(amountowed);
477             }
478         }
479         emit Burn(msg.sender, tickLower, tickUpper, amount, amount0, amount1);
480     }
```

Listing 2.14: contracts/BubblyPool.sol

```
260     function decreaseLiquidity(DecreaseLiquidityParams calldata params)
261         external
262         override
263         isAuthorizedForToken(params.tokenId)
264         checkDeadline(params.deadline)
265         returns (uint256 amount0, uint256 amount1)
266     {
267         require(params.liquidity > 0);
268         Position storage position = _positions[params.tokenId];
269
270         uint128 positionLiquidity = position.liquidity;
271         require(positionLiquidity >= params.liquidity);
272         uint256 amount0used;
273         uint256 amount1used;
274         uint256 amountowed;
275         PoolAddress.PoolKey memory poolKey = _poolIdToPoolKey[position.poolId];
276         IBubblyPool pool = IBubblyPool(PoolAddress.computeAddress(factory, poolKey));
277
278         (amount0, amount1,amount0used,amount1used,amountowed) = pool.burn(
279                                     position.tickLower,
280                                     position.tickUpper,
281                                     params.liquidity,
282                                     IBubblyPoolActions.burnParams({
```

```
283                                     liquidity:position.liquidity,
284                                     lpamount0:position.lpamount0,
285                                     lpamount1:position.lpamount1,
286                                     lpcollateral:position.lpcollateral
287                                 })
288                             );
```

Listing 2.15: contracts/NonfungiblePositionManager.sol

Impact If the liquidity provider removes only partial liquidity from the pool, this discrepancy can result in a larger withdrawable collateral than intended.

Suggestion Revise the value of `burnParams.lpcollateral`.

2.2.5 Incorrect time check in function `DeliverforX()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Function `DeliverforX()` checks that `_blockTimestamp() >= DeliverBeginTime[pool] + DeliverEpoch`. Additionally, function() `Deliver()` checks that `_blockTimestamp() <= DeliverBeginTime[pool] + DeliverEpoch`. However, when `_blockTimestamp() == DeliverBeginTime[pool] + DeliverEpoch`, both `Deliver()` and `DeliverforX()` can be invoked which is against the protocol design.

```
53    function Deliver(address pool, bool forX) external returns (uint256 amount0,uint256 amount1 ,
          bool getToken){
54        //pool flag check
55        require(DeliverBeginTime[pool] != uint32(0),'Time not set');
56        require(_blockTimestamp() >= DeliverBeginTime[pool], 'Delivery not start');
57        require(DeliverEpoch != uint32(0));
58        //check overflow
59        require(_blockTimestamp() <= DeliverBeginTime[pool] + DeliverEpoch, 'Delivery after 24hs');
```

Listing 2.16: contracts/Delivery.sol

```
92    function DeliverforX(address pool) external returns (uint256 amountX,uint256 amountY){
93        //pool flag check
94        require(DeliverBeginTime[pool] != uint32(0),'Time not set');
95        require(DeliverEpoch != uint32(0));
96        require(_blockTimestamp() >= DeliverBeginTime[pool] + DeliverEpoch, 'Delivery before 24hs')
              ;
```

Listing 2.17: contracts/Delivery.sol

Impact Function `DeliverforX()` can be invoked before `Deliver()`, which results in DoS.

Suggestion Ensure `_blockTimestamp() > DeliverBeginTime[pool] + DeliverEpoch` in function `DeliverforX()`.

2.2.6 Incorrect price check in function `DeliverforX()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `DeliverforX()`, when the buyer's position price is higher than the average price of the sellers' delivery price, vToken (i.e., `token0`), instead of collateral (i.e., `token1`), should be delivered to the buyer if there is. Otherwise, if buyers whose position prices are lower invoke the function `DeliverforX()` earlier, the collateral in the pool may not be sufficient for some buyers.

```
92  function DeliverforX(address pool) external returns (uint256 amountX,uint256 amountY){
93      //pool flag check
94      require(DeliverBeginTime[pool] != uint32(0),'Time not set');
95      require(DeliverEpoch != uint32(0));
96      require(_blockTimestamp() >= DeliverBeginTime[pool] + DeliverEpoch, 'Delivery before 24hs')
            ;
97      require(IBubblyPool(pool).deliveryflag());
98      // check token address
99      require(vtokenToToken[pool]!= address(0));
100     (uint256 amount0, uint256 amount1) = _getPosition(pool, true);
101     //frontend control delivery amount = amount0
102     if(poolReserve[pool] == 0 || FullMath.mulDiv(poolTotalQuoteToken[pool], uint256(1),
            poolTotalToken[pool]) <= FullMath.mulDiv(amount1, uint256(1), amount0)){
103         //get double collateral
104         amountY = 2 * amount1;
105         TransferHelper.safeTransferFrom(IBubblyPool(pool).token1(), address(this), msg.sender,
                amountY);
106     }
107     else{
108         //get tokenX
109         amountX = amount0 > poolReserve[pool] ? poolReserve[pool] : amount0;
110         TransferHelper.safeTransferFrom(vtokenToToken[pool], address(this), msg.sender, amountX
                );
111         amountY = amount1 - FullMath.mulDiv(amountX , amount1 , amount0);
112         if(amountY != 0) TransferHelper.safeTransferFrom(IBubblyPool(pool).token1(), address(
                this), msg.sender, amountY);
113     }
114     //buyer delivery
115     require(!DeliverlistX[msg.sender], 'already delivered' );
116     DeliverlistX[msg.sender] = true;
117     emit DeliverAfterDeadline(msg.sender, amountX, amountY, pool);
118 }
```

Listing 2.18: contracts/Delivery.sol

Impact The collateral in the pool may not be sufficient for some buyers.

Suggestion Revise the logic for the function `DeliverforX()`.

2.2.7 Inconsistent calculation for `amountY` in function `DeliverforX()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `Delivery` contract, the function `DeliverforX()` refunds double the initial principal when the `poolReserve` is zero or the buyer purchased at a higher price. However, in other scenarios, it only refunds the remaining collateral `amountY` (on Line 111) to buyers instead of doubling the amount as compensation.

```
92  function DeliverforX(address pool) external returns (uint256 amountX,uint256 amountY){
93      //pool flag check
94      require(DeliverBeginTime[pool] != uint32(0),'Time not set');
95      require(DeliverEpoch != uint32(0));
96      require(_blockTimestamp() >= DeliverBeginTime[pool] + DeliverEpoch, 'Delivery before 24hs')
            ;
97      require(IBubblyPool(pool).deliveryflag());
98      // check token address
99      require(vtokenToToken[pool]!= address(0));
100     (uint256 amount0, uint256 amount1) = _getPosition(pool, true);
101     //frontend control delivery amount = amount0
102     if(poolReserve[pool] == 0 || FullMath.mulDiv(poolTotalQuoteToken[pool], uint256(1),
            poolTotalToken[pool]) <= FullMath.mulDiv(amount1, uint256(1), amount0)){
103         //get double collateral
104         amountY = 2 * amount1;
105         TransferHelper.safeTransferFrom(IBubblyPool(pool).token1(), address(this), msg.sender,
                amountY);
106     }
107     else{
108         //get tokenX
109         amountX = amount0 > poolReserve[pool] ? poolReserve[pool] : amount0;
110         TransferHelper.safeTransferFrom(vtokenToToken[pool], address(this), msg.sender, amountX
                );
111         amountY = amount1 - FullMath.mulDiv(amountX , amount1 , amount0);
112         if(amountY != 0) TransferHelper.safeTransferFrom(IBubblyPool(pool).token1(), address(
                this), msg.sender, amountY);
113     }
114     //buyer delivery
115     require(!DeliverlistX[msg.sender], 'already delivered' );
116     DeliverlistX[msg.sender] = true;
117     emit DeliverAfterDeadline(msg.sender, amountX, amountY, pool);
118 }
```

Listing 2.19: contracts/Delivery.sol

Impact Some buyers may not receive the correct amount of collateral as expected.

Suggestion Revise the calculation for `amountY` to be doubled.

2.2.8 Incorrect fee accounting logic in function `swap()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Currently, fees will be charged during the swap process in two different directions (i.e., zeroForOne and OneForZero). However, there are flaws in the fee accounting logic for both directions. First, when the swap direction is OneForZero, the fees are already added to the `amount1` variable and shouldn't be added again before the function `BubblySwapCallback()` is invoked. Second, when the direction is zeroForOne and swap is closed, the `totalFeeInQuoteToken` should be added to the `amount1` (negative) instead of subtracting from it.

```
718   // do the transfers and collect payment
719   if (swapParams.zeroForOne) {
720
721       if(!swapParams.isOpen){
722           //0->1 close long position
723           //sub swap fee
724           if (amount1 < 0) {
725               amount1 = amount1 - int256(totalFeeInQuoteToken);
726               TransferHelper.safeTransfer(token1, swapParams.recipient, uint256(-amount1));
727           }
728           //for quoter
729           IBubblySwapCallback(msg.sender).BubblySwapCallback(token1, swapParams.isOpen, amount0,
                  amount1, data);
730       }
731       else{
732           //open short position
733           uint256 balance1Before = balance1();
734           //add swap fee
735           amount1 = amount1 - int256(totalFeeInQuoteToken);
736           IBubblySwapCallback(msg.sender).BubblySwapCallback(token1, swapParams.isOpen, amount0,
                  amount1, data);
737           require(balance1Before.add(uint256(-amount1)) <= balance1(), 'IIA');
738       }
739
740   }
741   else {
742       if(!swapParams.isOpen){
743           //close short position
744           //ensure collateralamount > 0 ,confirm by periphery
745           //add fee for position record
746           amount1 += int256(totalFeeInQuoteToken);
747           uint256 amountToPay = 2 * swapParams.collateralamount > uint256(amount1) ? (2 *
                  swapParams.collateralamount).sub(uint256(amount1)):0;
748           if (amount1 > 0) TransferHelper.safeTransfer(token1, swapParams.recipient, amountToPay)
                  ;
749           //for quoter
750           IBubblySwapCallback(msg.sender).BubblySwapCallback(token1, swapParams.isOpen, amount0,
                  amount1, data);
751       }
752       else{
753           //1->0 open long position
754           //add fee for position record
```

```
755             amount1 += int256(totalFeeInQuoteToken);
756             uint256 balance1Before = balance1();
757             IBubblySwapCallback(msg.sender).BubblySwapCallback(token1, swapParams.isOpen, amount0,
                    amount1, data);
758             require(balance1Before.add(uint256(amount1)) <= balance1(), 'IIA');
759         }
760
761     }
```

Listing 2.20: contracts/BubblyPool.sol

Impact Users will get more tokens when they close positions or deliver.

Suggestion Revise the fee accounting logic. Further, it is suggested that an overflow check should be added on the calculation result of `amount1 = amount1 + int256(totalFeeInQuoteToken)`.

2.2.9 Potential inconsistent calculation for the `amount0inUSD`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `BubblyPool` contract, there exists a discrepancy in how position ranges are determined between the `mint()` and `_modifyPosition()` functions. This inconsistency could potentially lead to unexpected behaviors, deviating from the original UniswapV3 design. In the `_modifyPosition()` function, a position is considered to be in-range if it meets the condition specified on line 233: `tickLower <= current tick < tickUpper`. This implies an open interval for the upper bound of the tick range. However, in the `mint()` function, a position is defined as in-range using a closed interval for both bounds. It utilizes prices converted from ticks to set the in-range condition, diverging from the approach used in `_modifyPosition()`.

```
202     function _modifyPosition(ModifyPositionParams memory params)
203     private
204     noDelegateCall
205     returns (
206         Position.Info storage position,
207         int256 amount0,
208         int256 amount1
209     )
210     {
211
212     checkTicks(params.tickLower, params.tickUpper);
213
214     Slot0 memory _slot0 = slot0; // SLOAD for gas optimization
215
216     position = _updatePosition(
217         params.owner,
218         params.tickLower,
219         params.tickUpper,
220         params.liquidityDelta,
```

```
221         _slot0.tick
222     );
223
224     if (params.liquidityDelta != 0) {
225         if (_slot0.tick < params.tickLower) {
226             // current tick is below the passed range; liquidity can only become in range by
                    crossing from left to
227             // right, when we'll need _more_ token0 (it's becoming more valuable) so user must
                    provide it
228             amount0 = SqrtPriceMath.getAmount0Delta(
229                 TickMath.getSqrtRatioAtTick(params.tickLower),
230                 TickMath.getSqrtRatioAtTick(params.tickUpper),
231                 params.liquidityDelta
232             );
233         } else if (_slot0.tick < params.tickUpper) {
234             // current tick is inside the passed range
235             uint128 liquidityBefore = liquidity; // SLOAD for gas optimization
```

Listing 2.21: contracts/BubblyPool.sol

```
353     function mint(
354         address recipient,
355         int24 tickLower,
356         int24 tickUpper,
357         uint128 amount,
358         bytes calldata data
359     ) external override lock onlyNPM returns (uint256 amount0, uint256 amount1,uint256 collateral)
            {
360         require(!deliveryflag);
361         require(amount > 0);
362
363         (, int256 amount0Int, int256 amount1Int) =
364             _modifyPosition(
365                 ModifyPositionParams({
366                     owner: recipient,
367                     tickLower: tickLower,
368                     tickUpper: tickUpper,
369                     liquidityDelta: int256(amount).toInt128()
370                 })
371             );
372
373         amount0 = uint256(amount0Int);
374         amount1 = uint256(amount1Int);
375         uint256 amount0inUSD;
376         //stack too deep
377         {
378             uint160 tickUppersqrtPriceX96 = TickMath.getSqrtRatioAtTick(tickUpper);
379             uint160 tickLowersqrtPriceX96 = TickMath.getSqrtRatioAtTick(tickLower);
380
381             if ( slot0.sqrtPriceX96 < tickLowersqrtPriceX96 || slot0.sqrtPriceX96 >
                    tickUppersqrtPriceX96){
382                 amount0inUSD = SqrtPriceMath.getAmount1Delta(
383                     tickLowersqrtPriceX96,
```

```
384                 tickUppersqrtPriceX96,
385                 amount,
386                 true
387             );
```

Listing 2.22: contracts/BubblyPool.sol

Impact The inconsistency could potentially lead to unexpected behaviors, deviating from the original UniswapV3 design

Suggestion Revise the logic to be consistent between the functions.

2.2.10 Incorrect check on `amountOutReceived` in function `exactOutputInternal()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `CollateralPositionManager` contract, it requires `amountOutReceived == params.amountOut` when `params.sqrtPriceLimitX96 == 0` in the function `exactOutputInternal()`. However, when `tokenOut == params.quoteToken`, `amountOutReceived` contains `totalFeeInQuoteToken` while `params.amountOut` does not, which might lead to a revert.

```
212   function exactOutputInternal(
213       internalExactOutputSingleParams memory params,
214       SwapCallbackData memory data
215   ) private returns (uint256 amountIn) {
216       // allow swapping to the router address with address 0
217       if (params.recipient == address(0)) params.recipient = address(this);
218
219       (address tokenOut, address tokenIn, uint24 fee) = data.path.decodeFirstPool();
220
221       bool zeroForOne = tokenIn != params.quoteToken;
222       if(params.isOpen == false){
223           //tokenIn is always basetoken when close a position
224           require(tokenIn == params.quoteToken);
225       }
226       (int256 amount0Delta, int256 amount1Delta, uint256 totalFeeInQuoteToken) =
227           getPool(params.quoteToken, tokenIn, tokenOut, fee).swap(
228               IBubblyPoolActions.SwapParams({
229                   recipient : params.recipient,
230                   zeroForOne : zeroForOne,
231                   amountSpecified : -params.amountOut.toInt256(),
232                   isOpen : params.isOpen,
233                   collateralamount : params.collateralamount,
234                   sqrtPriceLimitX96: params.sqrtPriceLimitX96 == 0
235                   ? (zeroForOne ? TickMath.MIN_SQRT_RATIO + 1 : TickMath.MAX_SQRT_RATIO - 1)
236                   : params.sqrtPriceLimitX96
237               }),
238               abi.encode(data)
239           );
240       //stack too deep
```

```
241         {
242             uint256 amountOutReceived;
243             (amountIn, amountOutReceived) = zeroForOne
244                 ? (uint256(amount0Delta), uint256(-amount1Delta))
245                 : (uint256(amount1Delta), uint256(-amount0Delta));
246
247             // it's technically possible to not receive the full output amount,
248             // so if no price limit has been specified, require this possibility away
249             if (params.sqrtPriceLimitX96 == 0) require(amountOutReceived == params.amountOut);
250         }
251         address pool = address(getPool(params.quoteToken, tokenIn, tokenOut, fee));
252         bool long = params.isOpen != zeroForOne;
253         int256 amount1OnPosition = amount1Delta > 0 ? amount1Delta - int256(totalFeeInQuoteToken) :
                amount1Delta + int256(totalFeeInQuoteToken) ;
254         _updateposition(
255             UpdatePositionParams({
256                 isOpen : params.isOpen,
257                 long : long,
258                 recipient : params.recipient,
259                 pool : pool,
260                 amount0 : uint256((amount0Delta > 0) ? amount0Delta : (-amount0Delta)),
261                 amount1 : uint256((amount1OnPosition > 0) ? amount1OnPosition : (-amount1OnPosition
                    ))
262             })
263         );
264     }
```

Listing 2.23: contracts/CollateralPositionManager.sol

Impact This might lead to a revert.

Suggestion Require `amountOutReceived == params.amountOut + totalFeeInQuoteToken` when `params.sqrtPriceLimitX96 == 0 && tokenOut == params.quoteToken` and `amountOutReceived == params.amountOut` when `params.sqrtPriceLimitX96 == 0 && tokenOut != params.quoteToken`.

2.2.11 Lack of forced liquidation to defend against bad debts

Severity High

Status Fixed in `Version 3`

Introduced by `Version 1`

Description Currently, there is no forced liquidation for short positions, which may bring bad debt risks for the protocol. For example, an attacker can open a short position to manipulate the price, followed by adding a large amount of liquidity with only a small amount of collateral (e.g., USDC). This keeps the price low when closing the short position, resulting in profits for the short position. This form of attack drains the liquidity of other liquidity providers (LPs), leaving behind significant bad debts. The liquidity is added at depressed prices, and the LPs, now acting as sellers, are unable to repurchase `token0`, exacerbating the situation. Furthermore, attackers can also first open a short position to manipulate the price, followed by opening a short position again, and then close the first short position to gain profit.

Additionally, the attacker can open a short position followed by buying a large amount of `token0` at a low price, and close part of the long position (selling) after other users buy the `token0` later. In this attack, the profit comes from other buyers, leaving the attacker's short position a bad debt.

Impact Attackers can leverage this vulnerability to drain the pool or hold a large amount of the `token0`.

Suggestion Disable the position closing function.

2.3 Additional Recommendation

2.3.1 Redundant function `createAndInitializePoolIfNecessary()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `NonfungiblePositionManager` contract, there is an external function `createAndInitializePoolIfNecessary()` since the contract inherits the `PoolInitializer` contract. However, the invocation of the external function `createAndInitializePoolIfNecessary` will revert since the function `createPool` and `initialize` requires the authority of the factory's owner.

```
13  function createAndInitializePoolIfNecessary(
14      address token0,
15      address token1,
16      uint24 fee,
17      uint160 sqrtPriceX96
18  ) external payable override returns (address pool) {
19      //require(token0 < token1);
20      pool = IBubblyFactory(factory).getPool(token0, token1, fee);
21
22      if (pool == address(0)) {
23          pool = IBubblyFactory(factory).createPool(token0, token1, fee);
24          IBubblyPool(pool).initialize(sqrtPriceX96);
25      } else {
26          (uint160 sqrtPriceX96Existing, , , , , , ) = IBubblyPool(pool).slot0();
27          if (sqrtPriceX96Existing == 0) {
28              IBubblyPool(pool).initialize(sqrtPriceX96);
29          }
30      }
31  }
```

Listing 2.24: contracts/base/PoolInitializer.sol

```
38  function createPool(
39      address tokenA,
40      address tokenB,
41      uint24 fee
42  ) external override noDelegateCall returns (address pool) {
43      require(msg.sender == owner);
44      //check NPM&&CPM initialized
```

```
45      require(NPM != address(0),'NPM0');
46      require(CPM != address(0),'CPM0');
47      require(tokenA != tokenB);
48      //token1 always quoteToken
49      (address token0, address token1) = (tokenA, tokenB) ;
50      require(token0 != address(0));
51      int24 tickSpacing = feeAmountTickSpacing[fee];
52      require(tickSpacing != 0);
53      require(getPool[token0][token1][fee] == address(0));
54      pool = deploy(CPM, NPM, address(this), token0, token1, fee, tickSpacing);
55      getPool[token0][token1][fee] = pool;
56      // populate mapping in the reverse direction, deliberate choice to avoid the cost of
            comparing addresses
57      getPool[token1][token0][fee] = pool;
58      emit PoolCreated(token0, token1, fee, tickSpacing, pool);
59  }
```

Listing 2.25: contracts/BubblyFactory.sol

```
167 function initialize(uint160 sqrtPriceX96) external override onlyFactoryOwner{
168     require(slot0.sqrtPriceX96 == 0, 'AI');
169
170     int24 tick = TickMath.getTickAtSqrtRatio(sqrtPriceX96);
171
172     (uint16 cardinality, uint16 cardinalityNext) = observations.initialize(_blockTimestamp());
173
174     slot0 = Slot0({
175         sqrtPriceX96: sqrtPriceX96,
176         tick: tick,
177         observationIndex: 0,
178         observationCardinality: cardinality,
179         observationCardinalityNext: cardinalityNext,
180         feeProtocol: 0,
181         unlocked: true
182     });
183
184     emit Initialize(sqrtPriceX96, tick);
185 }
```

Listing 2.26: contracts/BubblyPool.sol

Suggestion Remove the function `createAndInitializePoolIfNecessaryggestion()`.

2.3.2 Redundant variables

Status Confirmed

Introduced by `Version 1`

Description In the contract `BubblyPool`, the `amount0inUSD` variable is unused in the function `burn()`. Therefore, the variable and corresponding arithmetic operations can be safely removed for simplicity.

```
431 function burn(
```

```
432       int24 tickLower,
433       int24 tickUpper,
434       uint128 amount,
435       burnParams calldata burnparams
436   ) external override lock onlyNPM returns (uint256 amount0, uint256 amount1,uint256 amount0used
          , uint256 amount1used,uint256 amountowed) {
437
438       (Position.Info storage position, int256 amount0Int, int256 amount1Int) =
439           _modifyPosition(
440               ModifyPositionParams({
441                   owner: msg.sender,
442                   tickLower: tickLower,
443                   tickUpper: tickUpper,
444                   liquidityDelta: -int256(amount).toInt128()
445               })
446           );
447
448
449       amount0 = uint256(-amount0Int);
450       amount1 = uint256(-amount1Int);
451       uint256 amount0inUSD;
452       if(amount0 > 0){
453           uint160 tickUpperSqrtPriceX96 = TickMath.getSqrtRatioAtTick(tickUpper);
454           uint160 tickLowerSqrtPriceX96 = TickMath.getSqrtRatioAtTick(tickLower);
455
456           if ( slot0.sqrtPriceX96 < tickLowerSqrtPriceX96 || slot0.sqrtPriceX96 >
                  tickUpperSqrtPriceX96){
457               amount0inUSD = SqrtPriceMath.getAmount1Delta(tickLowerSqrtPriceX96,
                      tickUpperSqrtPriceX96, amount, false);
458           }else{
459               amount0inUSD = SqrtPriceMath.getAmount1Delta(slot0.sqrtPriceX96,
                      tickUpperSqrtPriceX96, amount, false);
460           }
461
462       }
```

Listing 2.27: contracts/BubblyPool.sol

In the contract `CollateralPositionManager`, the value of the `amountInCached` variable is not used and this variable can be removed.

```
58   /// @dev Transient storage variable used for returning the computed amount in for an exact
         output swap.
59   uint256 private amountInCached = DEFAULT_AMOUNT_IN_CACHED;
```

Listing 2.28: contracts/CollateralPositionManager.sol

Suggestion Remove the redundant variables and related codes.

2.3.3 Wrong comment in function `exactOutputInternal()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The comment lined out below is wrong (line 223):

```
212   function exactOutputInternal(
213       internalExactOutputSingleParams memory params,
214       SwapCallbackData memory data
215   ) private returns (uint256 amountIn) {
216       // allow swapping to the router address with address 0
217       if (params.recipient == address(0)) params.recipient = address(this);
218
219       (address tokenOut, address tokenIn, uint24 fee) = data.path.decodeFirstPool();
220
221       bool zeroForOne = tokenIn != params.quoteToken;
222       if(params.isOpen == false){
223           //tokenIn is always basetoken when close a position
224           require(tokenIn == params.quoteToken);
225       }
```

Listing 2.29: contracts/CollateralPositionManager.sol

Suggestion Revise the comment.

2.3.4 Use `safemath` in logic calculations that differ from UniswapV3

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The solidity version of the project is 0.7.6 which does not check on overflow/underflow natively. So it is recommended to use `safemath` in logic calculations that differ from UniswapV3.

Suggestion Use `safemath` in logic calculations that differ from UniswapV3.

2.3.5 Add a check on `params.quoteToken`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `CollateralPositionManager`, the functions `exactInputInternal()` and `exactOutputInternal()` does not check whether `params.quoteToken` equals `IBubblyPool(pool).token1()`.

```
138   function exactInputInternal(
139       internalExactInputSingleParams memory params,
140       SwapCallbackData memory data
141   ) private returns (uint256 amountOut) {
142       // allow swapping to the router address with address 0
143       if (params.recipient == address(0)) params.recipient = address(this);
144
145       (address tokenIn, address tokenOut, uint24 fee) = data.path.decodeFirstPool();
146
147       bool zeroForOne = tokenIn != params.quoteToken;
148       if(params.isOpen == false){
149           //tokenIn is always basetoken when close a position
```

```
150             require(tokenIn != params.quoteToken);
151         }
152         (int256 amount0, int256 amount1 ,uint256 totalFeeInQuoteToken) =
153             getPool(params.quoteToken, tokenIn, tokenOut, fee).swap(
154                 IBubblyPoolActions.SwapParams({
155                     recipient : params.recipient,
156                     zeroForOne : zeroForOne,
157                     amountSpecified : params.amountIn.toInt256(),
158                     isOpen : params.isOpen,
159                     collateralamount : params.collateralamount,
160                     sqrtPriceLimitX96 : params.sqrtPriceLimitX96 == 0
161                     ? (zeroForOne ? TickMath.MIN_SQRT_RATIO + 1 : TickMath.MAX_SQRT_RATIO - 1)
162                     : params.sqrtPriceLimitX96
163                 }),
164                 abi.encode(data)
165             );
166         //long or short
167         bool long = params.isOpen != zeroForOne;
168         address pool = address(getPool(params.quoteToken, tokenIn, tokenOut, fee));
169         int256 amount1OnPosition = amount1 > 0 ? amount1 - int256(totalFeeInQuoteToken) : amount1 +
                int256(totalFeeInQuoteToken) ;
170         _updateposition(
171             UpdatePositionParams({
172                 isOpen : params.isOpen,
173                 long : long,
174                 recipient : params.recipient,
175                 pool : pool,
176                 amount0 : uint256((amount0 > 0) ? amount0 : (-amount0)) ,
177                 amount1 : uint256((amount1OnPosition > 0)? amount1OnPosition:(-amount1OnPosition))
178             })
179         );
180
181         return uint256(-(zeroForOne ? amount1 : amount0));
182     }
```

Listing 2.30: contracts/CollateralPositionManager.sol

```
212     function exactOutputInternal(
213         internalExactOutputSingleParams memory params,
214         SwapCallbackData memory data
215     ) private returns (uint256 amountIn) {
216         // allow swapping to the router address with address 0
217         if (params.recipient == address(0)) params.recipient = address(this);
218
219         (address tokenOut, address tokenIn, uint24 fee) = data.path.decodeFirstPool();
220
221         bool zeroForOne = tokenIn != params.quoteToken;
222         if(params.isOpen == false){
223             //tokenIn is always basetoken when close a position
224             require(tokenIn == params.quoteToken);
225         }
226         (int256 amount0Delta, int256 amount1Delta, uint256 totalFeeInQuoteToken) =
227             getPool(params.quoteToken, tokenIn, tokenOut, fee).swap(
```

```
228             IBubblyPoolActions.SwapParams({
229                 recipient : params.recipient,
230                 zeroForOne : zeroForOne,
231                 amountSpecified : -params.amountOut.toInt256(),
232                 isOpen : params.isOpen,
233                 collateralamount : params.collateralamount,
234                 sqrtPriceLimitX96: params.sqrtPriceLimitX96 == 0
235                 ? (zeroForOne ? TickMath.MIN_SQRT_RATIO + 1 : TickMath.MAX_SQRT_RATIO - 1)
236                 : params.sqrtPriceLimitX96
237             }),
238             abi.encode(data)
239         );
240     //stack too deep
241     {
242         uint256 amountOutReceived;
243         (amountIn, amountOutReceived) = zeroForOne
244             ? (uint256(amount0Delta), uint256(-amount1Delta))
245             : (uint256(amount1Delta), uint256(-amount0Delta));
246 
247         // it's technically possible to not receive the full output amount,
248         // so if no price limit has been specified, require this possibility away
249         if (params.sqrtPriceLimitX96 == 0) require(amountOutReceived == params.amountOut);
250     }
251     address pool = address(getPool(params.quoteToken, tokenIn, tokenOut, fee));
252     bool long = params.isOpen != zeroForOne;
253     int256 amount1OnPosition = amount1Delta > 0 ? amount1Delta - int256(totalFeeInQuoteToken) :
          amount1Delta + int256(totalFeeInQuoteToken) ;
254     _updateposition(
255         UpdatePositionParams({
256             isOpen : params.isOpen,
257             long : long,
258             recipient : params.recipient,
259             pool : pool,
260             amount0 : uint256((amount0Delta > 0) ? amount0Delta : (-amount0Delta)),
261             amount1 : uint256((amount1OnPosition > 0) ? amount1OnPosition : (-amount1OnPosition
                ))
262         })
263     );
264 }
```

Listing 2.31: contracts/CollateralPositionManager.sol

Suggestion Ensure `params.quoteToken` equals `IBubblyPool(pool).token1()`.

2.4 Note

2.4.1 Potential centralization risks

Introduced by `Version 1`

Description There are several important functions like `setDeliveryTime()`, `setToken()`, `setDeliverEpoch()`, `setNPM()`, `setCPM()`, etc., which are only callable by the factory's owner.

If the owner’s private key is lost or compromised, it could lead to losses for the protocol and users.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS