

REPORT 6448A5976662E0001AAC4FAE

Created	Wed Apr 26 2023 04:16:23 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	644779608288ab6760a063fb

## REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
<a href="#">37ef7221-2412-473b-8951-6132eeb529a6</a>	core/OrderBook.sol	10

Started	Wed Apr 26 2023 04:16:32 GMT+0000 (Coordinated Universal Time)
Finished	Wed Apr 26 2023 05:04:15 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.7.3
Main Source File	Core/OrderBook.sol

## DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	10

## ISSUES

LOW

A floating pragma is set.

SWC-103

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

core/OrderBook.sol

Locations

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.0;
4
5 import "../libraries/math/SafeMath.sol";
```

LOW

Read of persistent state following external call

SWC-107

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

core/OrderBook.sol

Locations

```
825 _transferInETH();
826
827 require(msg.value > minExecutionFee, "OrderBook: insufficient execution fee");
828
829 _createDecreaseOrder(
```

LOW

### Write to persistent state following external call

SWC-107

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

core/OrderBook.sol

Locations

```
864 | blocknum
865 | );
866 | decreaseOrdersIndex[_account] = _orderIndex.add(1);
867 | decreaseOrders[_account][_orderIndex] = order;
```

LOW

### Write to persistent state following external call

SWC-107

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

libraries/Utils/ReentrancyGuard.sol

Locations

```
58 | // By storing the original value once again, a refund is triggered (see
59 | // https://eips.ethereum.org/EIPS/eip-2200)
60 | status = _NOT_ENTERED;
61 | }
62 | }
```

LOW

### Read of persistent state following external call

SWC-107

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

core/OrderBook.sol

Locations

```
849 | bool _triggerAboveThreshold
850 | ) private {
851 | uint256 _orderIndex = decreaseOrdersIndex[_account];
852 |
853 | uint256 blocknum = Chain.currentBlockNumber();
```

## LOW

### Write to persistent state following external call

SWC-107

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

core/OrderBook.sol

Locations

```
865 | );  
866 | decreaseOrdersIndex[_account] = _orderIndex.add(1);  
867 | decreaseOrders[_account][_orderIndex] = order;  
868 |  
869 | emit CreateDecreaseOrder(
```

## LOW

### Multiple calls are executed in the same transaction.

SWC-113

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

libraries/chain/Chain.sol

Locations

```
24 | function currentBlockNumber() internal view returns (uint256) {  
25 | if (block.chainid == ARBITRUM_CHAIN_ID || block.chainid == ARBITRUM_GOERLI_CHAIN_ID) {  
26 | return arbSys.arbBlockNumber();  
27 | }
```

## LOW

### Potential use of "blockhash" as source of randomness.

SWC-120

The environment variable "blockhash" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

libraries/chain/Chain.sol

Locations

```
37 | }  
38 |  
39 | return blockhash(blockNumber);  
40 | }  
41 | }
```

## LOW

### Potential use of "block.number" as source of randomness.

SWC-120

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

libraries/chain/Chain.sol

Locations

```
27 | }  
28 |  
29 | return block.number;  
30 | }
```

## LOW Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

SWC-123

Source file

core/OrderBook.sol

Locations

```
545 | } public view returns (uint256, bool) {
546 |     uint256 currentPrice = _maximizePrice
547 |     ? IOracle(_oracle).getMaxPrice(_indexToken) : IOracle(_oracle).getMinPrice(_indexToken);
548 |     bool isPriceValid = _triggerAboveThreshold ? currentPrice > _triggerPrice : currentPrice < _triggerPrice;
549 |     if (_raise) {
```

Source file

core/OrderBook.sol

Locations

```
16 | import "../oracle/interfaces/IOracle.sol";
17 |
18 | contract OrderBook is ReentrancyGuard, IOrderBook {
19 |     using SafeMath for uint256;
20 |     using SafeERC20 for IERC20;
21 |     using Address for address payable;
22 |
23 |     uint256 public constant BASIS_POINTS_DIVISOR = 10000;
24 |     uint256 public constant PRICE_PRECISION = 1e30;
25 |     uint256 public constant USDG_PRECISION = 1e18;
26 |
27 |     struct IncreaseOrder {
28 |         address account;
29 |         address purchaseToken;
30 |         uint256 purchaseTokenAmount;
31 |         address collateralToken;
32 |         address indexToken;
33 |         uint256 sizeDelta;
34 |         bool isLong;
35 |         uint256 triggerPrice;
36 |         bool triggerAboveThreshold;
37 |         uint256 executionFee;
38 |         uint256 updatedAtBlock;
39 |     }
40 |     struct DecreaseOrder {
41 |         address account;
42 |         address collateralToken;
43 |         uint256 collateralDelta;
44 |         address indexToken;
45 |         uint256 sizeDelta;
46 |         bool isLong;
47 |         uint256 triggerPrice;
48 |         bool triggerAboveThreshold;
49 |         uint256 executionFee;
50 |         uint256 updatedAtBlock;
51 |     }
52 |     struct SwapOrder {
53 |         address account;
54 |         address[] path;
55 |         uint256 amountIn;
56 |         uint256 minOut;
57 |         uint256 triggerRatio;
58 |         bool triggerAboveThreshold;
59 |         bool shouldUnwrap;
60 |         uint256 executionFee;
```

```

61     uint256 updatedAtBlock;
62 }
63
64 mapping (address => bool) public isPositionManager;
65
66 mapping (address => mapping(uint256 => IncreaseOrder)) public increaseOrders;
67 mapping (address => uint256) public increaseOrdersIndex;
68 mapping (address => mapping(uint256 => DecreaseOrder)) public decreaseOrders;
69 mapping (address => uint256) public decreaseOrdersIndex;
70 mapping (address => mapping(uint256 => SwapOrder)) public swapOrders;
71 mapping (address => uint256) public swapOrdersIndex;
72
73 address public gov;
74 address public weth;
75 address public usdg;
76 address public router;
77 address public vault;
78 uint256 public minExecutionFee;
79 uint256 public minPurchaseTokenAmountUsd;
80 bool public isInitialized = false;
81
82 event CreateIncreaseOrder(
83     address indexed account,
84     uint256 orderIndex,
85     address purchaseToken,
86     uint256 purchaseTokenAmount,
87     address collateralToken,
88     address indexToken,
89     uint256 sizeDelta,
90     bool isLong,
91     uint256 triggerPrice,
92     bool triggerAboveThreshold,
93     uint256 executionFee
94 );
95 event CancelIncreaseOrder(
96     address indexed account,
97     uint256 orderIndex,
98     address purchaseToken,
99     uint256 purchaseTokenAmount,
100     address collateralToken,
101     address indexToken,
102     uint256 sizeDelta,
103     bool isLong,
104     uint256 triggerPrice,
105     bool triggerAboveThreshold,
106     uint256 executionFee
107 );
108 event ExecuteIncreaseOrder(
109     address indexed account,
110     uint256 orderIndex,
111     address purchaseToken,
112     uint256 purchaseTokenAmount,
113     address collateralToken,
114     address indexToken,
115     uint256 sizeDelta,
116     bool isLong,
117     uint256 triggerPrice,
118     bool triggerAboveThreshold,
119     uint256 executionFee,
120     uint256 executionPrice
121 );
122 event UpdateIncreaseOrder(
123     address indexed account,

```

```
124 uint256 orderIndex,
125 address collateralToken,
126 address indexToken,
127 bool isLong,
128 uint256 sizeDelta,
129 uint256 triggerPrice,
130 bool triggerAboveThreshold,
131 )
132 event CreateDecreaseOrder(
133     address indexed account,
134     uint256 orderIndex,
135     address collateralToken,
136     uint256 collateralDelta,
137     address indexToken,
138     uint256 sizeDelta,
139     bool isLong,
140     uint256 triggerPrice,
141     bool triggerAboveThreshold,
142     uint256 executionFee,
143 )
144 event CancelDecreaseOrder(
145     address indexed account,
146     uint256 orderIndex,
147     address collateralToken,
148     uint256 collateralDelta,
149     address indexToken,
150     uint256 sizeDelta,
151     bool isLong,
152     uint256 triggerPrice,
153     bool triggerAboveThreshold,
154     uint256 executionFee,
155 )
156 event ExecuteDecreaseOrder(
157     address indexed account,
158     uint256 orderIndex,
159     address collateralToken,
160     uint256 collateralDelta,
161     address indexToken,
162     uint256 sizeDelta,
163     bool isLong,
164     uint256 triggerPrice,
165     bool triggerAboveThreshold,
166     uint256 executionFee,
167     uint256 executionPrice,
168 )
169 event UpdateDecreaseOrder(
170     address indexed account,
171     uint256 orderIndex,
172     address collateralToken,
173     uint256 collateralDelta,
174     address indexToken,
175     uint256 sizeDelta,
176     bool isLong,
177     uint256 triggerPrice,
178     bool triggerAboveThreshold,
179 )
180 event CreateSwapOrder(
181     address indexed account,
182     uint256 orderIndex,
183     address[] path,
184     uint256 amountIn,
185     uint256 minOut,
186     uint256 triggerRatio,
```

```

187     bool triggerAboveThreshold;
188     bool shouldUnwrap;
189     uint256 executionFee;
190     //
191     event CancelSwapOrder;
192     address indexed account;
193     uint256 orderIndex;
194     address[] path;
195     uint256 amountIn;
196     uint256 minOut;
197     uint256 triggerRatio;
198     bool triggerAboveThreshold;
199     bool shouldUnwrap;
200     uint256 executionFee;
201     //
202     event UpdateSwapOrder;
203     address indexed account;
204     uint256 orderIndex;
205     address[] path;
206     uint256 amountIn;
207     uint256 minOut;
208     uint256 triggerRatio;
209     bool triggerAboveThreshold;
210     bool shouldUnwrap;
211     uint256 executionFee;
212     //
213     event ExecuteSwapOrder;
214     address indexed account;
215     uint256 orderIndex;
216     address[] path;
217     uint256 amountIn;
218     uint256 minOut;
219     uint256 amountOut;
220     uint256 triggerRatio;
221     bool triggerAboveThreshold;
222     bool shouldUnwrap;
223     uint256 executionFee;
224     //
225
226     event Initialize;
227     address router;
228     address vault;
229     address weth;
230     address usdg;
231     uint256 minExecutionFee;
232     uint256 minPurchaseTokenAmountUsd;
233     //
234     event UpdateMinExecutionFee(uint256 minExecutionFee);
235     event UpdateMinPurchaseTokenAmountUsd(uint256 minPurchaseTokenAmountUsd);
236     event UpdateGov(address gov);
237     event SetPositionManager(address indexed account, bool isActive);
238
239     modifier onlyGov() {
240         require(msg.sender == gov, "OrderBook: forbidden");
241     }
242
243
244     modifier onlyPositionManager() {
245         require(isPositionManager[msg.sender], "OrderBook: forbidden");
246     }
247
248
249     constructor() {

```



```

250 gov = msg.sender;
251
252
253 function initialize()
254 address _router;
255 address _vault;
256 address _weth;
257 address _usdg;
258 uint256 _minExecutionFee;
259 uint256 _minPurchaseTokenAmountUsd;
260 external onlyGov {
261 require(!isInitialized, "OrderBook: already initialized");
262 isInitialized = true;
263
264 router = _router;
265 vault = _vault;
266 weth = _weth;
267 usdg = _usdg;
268 minExecutionFee = _minExecutionFee;
269 minPurchaseTokenAmountUsd = _minPurchaseTokenAmountUsd;
270
271 emit Initialize(router, vault, weth, usdg, minExecutionFee, minPurchaseTokenAmountUsd);
272 }
273
274 receive() external payable {
275 require(msg.sender == weth, "OrderBook: invalid sender");
276 }
277
278 function setPositionManager(address _account, bool _isActive) external onlyGov {
279 isPositionManager[_account] = _isActive;
280 emit SetPositionManager(_account, _isActive);
281 }
282
283 function setMinExecutionFee(uint256 _minExecutionFee) external onlyGov {
284 minExecutionFee = _minExecutionFee;
285
286 emit UpdateMinExecutionFee(_minExecutionFee);
287 }
288
289 function setMinPurchaseTokenAmountUsd(uint256 _minPurchaseTokenAmountUsd) external onlyGov {
290 minPurchaseTokenAmountUsd = _minPurchaseTokenAmountUsd;
291
292 emit UpdateMinPurchaseTokenAmountUsd(_minPurchaseTokenAmountUsd);
293 }
294
295 function setGov(address _gov) external onlyGov {
296 gov = _gov;
297
298 emit UpdateGov(_gov);
299 }
300
301 function getSwapOrder(address _account, uint256 _orderIndex) override public view returns (
302 address path0
303 address path1
304 address path2
305 uint256 amountIn
306 uint256 minOut
307 uint256 triggerRatio
308 bool triggerAboveThreshold
309 bool shouldUnwrap
310 uint256 executionFee
311 ) {
312 SwapOrder memory order = swapOrders[_account][_orderIndex];

```

```

313 return
314 order.path.length > 0 ? order.path[0].address(0)
315 order.path.length > 1 ? order.path[1].address(0)
316 order.path.length > 2 ? order.path[2].address(0)
317 order.amountIn
318 order.minOut
319 order.triggerRatio
320 order.triggerAboveThreshold
321 order.shouldUnwrap
322 order.executionFee
323 {}
324 {}
325
326 function createSwapOrder(
327 address[] memory _path
328 uint256 _amountIn
329 uint256 _minOut
330 uint256 _triggerRatio, // tokenB / tokenA
331 bool _triggerAboveThreshold
332 uint256 _executionFee
333 bool _shouldWrap
334 bool _shouldUnwrap
335 ) external payable nonReentrant {
336 require(_path.length == 2 || _path.length == 3, "OrderBook: invalid _path.length");
337 require(_path[0] != _path[_path.length - 1], "OrderBook: invalid _path");
338 require(_amountIn > 0, "OrderBook: invalid _amountIn");
339 require(_executionFee >= minExecutionFee, "OrderBook: insufficient execution fee");
340
341 // always need this call because of mandatory executionFee user has to transfer in ETH
342 transferInETH();
343
344 if (!_shouldWrap)
345 require(_path[0] == weth, "OrderBook: only weth could be wrapped");
346 require(msg.value == _executionFee.add(_amountIn), "OrderBook: incorrect value transferred");
347 } else {
348 require(msg.value == _executionFee, "OrderBook: incorrect execution fee transferred");
349 Router(router).pluginTransfer(_path[0], msg.sender, address(this), _amountIn);
350 }
351
352 createSwapOrder(msg.sender, _path, _amountIn, _minOut, _triggerRatio, _triggerAboveThreshold, _shouldUnwrap, _executionFee);
353 }
354
355 function _createSwapOrder(
356 address _account
357 address[] memory _path
358 uint256 _amountIn
359 uint256 _minOut
360 uint256 _triggerRatio
361 bool _triggerAboveThreshold
362 bool _shouldUnwrap
363 uint256 _executionFee
364 ) private {
365 uint256 _orderId = swapOrdersIndex[_account];
366
367 uint256 blocknum = Chain.currentBlockNumber();
368 SwapOrder memory order = SwapOrder(
369 _account
370 _path
371 _amountIn
372 _minOut
373 _triggerRatio
374 _triggerAboveThreshold
375 _shouldUnwrap

```

```

376     _executionFee
377     blocknum
378 }
379 swapOrdersIndex[_account] = _orderIndex.add(1);
380 swapOrders[_account][_orderIndex] = order;
381
382 emit CreateSwapOrder(
383     _account,
384     _orderIndex,
385     _path,
386     _amountIn,
387     _minOut,
388     _triggerRatio,
389     _triggerAboveThreshold,
390     _shouldUnwrap,
391     _executionFee
392 );
393
394
395 function cancelMultiple()
396     uint256[] memory _swapOrderIndexes,
397     uint256[] memory _increaseOrderIndexes,
398     uint256[] memory _decreaseOrderIndexes
399     external {
400     for (uint256 i = 0; i < _swapOrderIndexes.length; i++) {
401         cancelSwapOrder(_swapOrderIndexes[i]);
402     }
403     for (uint256 i = 0; i < _increaseOrderIndexes.length; i++) {
404         cancelIncreaseOrder(_increaseOrderIndexes[i]);
405     }
406     for (uint256 i = 0; i < _decreaseOrderIndexes.length; i++) {
407         cancelDecreaseOrder(_decreaseOrderIndexes[i]);
408     }
409 }
410
411 function cancelSwapOrder(uint256 _orderIndex) public nonReentrant {
412     SwapOrder memory order = swapOrders[msg.sender][_orderIndex];
413     require(order.account != address(0), "OrderBook: non-existent order");
414
415     delete swapOrders[msg.sender][_orderIndex];
416
417     if (order.path[0] == weth) {
418         transferOutETH(order.executionFee.add(order.amountIn), payable(msg.sender));
419     } else {
420         IERC20(order.path[0]).safeTransfer(msg.sender, order.amountIn);
421         transferOutETH(order.executionFee, payable(msg.sender));
422     }
423
424     emit CancelSwapOrder(
425         msg.sender,
426         _orderIndex,
427         order.path,
428         order.amountIn,
429         order.minOut,
430         order.triggerRatio,
431         order.triggerAboveThreshold,
432         order.shouldUnwrap,
433         order.executionFee
434     );
435 }
436
437 function validateSwapOrderPriceWithTriggerAboveThreshold(
438     address[] memory _path,

```

```

439 uint256 _triggerRatio
440 address _oracle
441 public view returns (bool) {
442     require(_path.length == 2 || _path.length == 3, "OrderBook: invalid _path.length");
443
444     // limit orders don't need this validation because minOut is enough
445     // so this validation handles scenarios for stop orders only
446     // when a user wants to swap when a price of tokenB increases relative to tokenA
447     address tokenA = _path[0];
448     address tokenB = _path[_path.length - 1];
449     require(tokenA != usdg, "tokenA is usdg, not permit");
450     require(tokenB != usdg, "tokenB is usdg, not permit");
451
452     uint256 tokenAPrice;
453     uint256 tokenBPrice;
454
455     tokenAPrice = IOracle(_oracle).getMinPrice(tokenA);
456     tokenBPrice = IOracle(_oracle).getMaxPrice(tokenB);
457
458     uint256 currentRatio = tokenBPrice.mul(PRICE_PRECISION).div(tokenAPrice);
459
460     bool isValid = currentRatio > _triggerRatio;
461     return isValid;
462 }
463
464 function updateSwapOrder(uint256 _orderIndex, uint256 _minOut, uint256 _triggerRatio, bool _triggerAboveThreshold) external nonReentrant {
465     SwapOrder storage order = swapOrders[msg.sender][_orderIndex];
466     require(order.account != address(0), "OrderBook: non-existent order");
467
468     order.minOut = _minOut;
469     order.triggerRatio = _triggerRatio;
470     order.triggerAboveThreshold = _triggerAboveThreshold;
471     order.updatedAtBlock = Chain.currentBlockNumber();
472
473     emit UpdateSwapOrder(
474         msg.sender,
475         _orderIndex,
476         order.path,
477         order.amountIn,
478         minOut,
479         triggerRatio,
480         triggerAboveThreshold,
481         order.shouldUnwrap,
482         order.executionFee
483     );
484 }
485
486 function executeSwapOrder(address _account, uint256 _orderIndex, address payable _feeReceiver, address _oracle) override external nonReentrant onlyPositionManager {
487     SwapOrder memory order = swapOrders[_account][_orderIndex];
488     require(order.account != address(0), "OrderBook: non-existent order");
489
490     for (uint i = 0; i < order.path.length; i++) {
491         if (order.path[i] == usdg) {
492             continue;
493         }
494         if (order.updatedAtBlock > IOracle(_oracle).minOracleBlockNumbers[order.path[i]]) {
495             revert("order.updatedAtBlock > oracle.minOracleBlockNumbers");
496         }
497     }
498
499     if (order.triggerAboveThreshold) {
500
501         // gas optimisation

```

```

502 // order_minAmount should prevent wrong price execution in case of simple limit order
503 require
504 validateSwapOrderPriceWithTriggerAboveThreshold(order.path, order.triggerRatio, _oracle);
505 "OrderBook: invalid price for execution"
506 }
507 }
508
509 delete swapOrders[_account][_orderIndex];
510
511 IERC20(order.path[0]).safeTransfer(vault, order.amountIn);
512
513 uint256 _amountOut;
514 if (order.path.order.path.length - 1) == weth && order.shouldUnwrap {
515     amountOut = _swap(order.path, order.minOut, address(this));
516     transferOutETH(_amountOut, payable(order.account));
517 } else {
518     _amountOut = _swap(order.path, order.minOut, order.account);
519 }
520
521 // pay executor
522 transferOutETH(order.executionFee, _feeReceiver);
523
524 emit ExecuteSwapOrder(
525     _account,
526     _orderIndex,
527     order.path,
528     order.amountIn,
529     order.minOut,
530     _amountOut,
531     order.triggerRatio,
532     order.triggerAboveThreshold,
533     order.shouldUnwrap,
534     order.executionFee
535 );
536 }
537
538 function validatePositionOrderPrice(
539     bool _triggerAboveThreshold,
540     uint256 _triggerPrice,
541     address _indexToken,
542     bool _maximizePrice,
543     bool _raise,
544     address _oracle
545 ) public view returns (uint256, bool) {
546     uint256 currentPrice = _maximizePrice
547     ? IOracle(_oracle).getMaxPrice(_indexToken) : IOracle(_oracle).getMinPrice(_indexToken);
548     bool isPriceValid = _triggerAboveThreshold ? currentPrice > _triggerPrice : currentPrice < _triggerPrice;
549     if (_raise) {
550         require(isPriceValid, "OrderBook: invalid price for execution");
551     }
552     return (currentPrice, isPriceValid);
553 }
554
555 function getDecreaseOrder(address _account, uint256 _orderIndex) override public view returns (
556     address collateralToken,
557     uint256 collateralDelta,
558     address indexToken,
559     uint256 sizeDelta,
560     bool isLong,
561     uint256 triggerPrice,
562     bool triggerAboveThreshold,
563     uint256 executionFee
564 ) {

```

```

565 DecreaseOrder memory order = decreaseOrders[_account][_orderIndex];
566 return
567 order.collateralToken;
568 order.collateralDelta;
569 order.indexToken;
570 order.sizeDelta;
571 order.isLong;
572 order.triggerPrice;
573 order.triggerAboveThreshold;
574 order.executionFee;
575 }
576 }
577
578 function getIncreaseOrder(address _account, uint256 _orderIndex) override public view returns (
579 address purchaseToken;
580 uint256 purchaseTokenAmount;
581 address collateralToken;
582 address indexToken;
583 uint256 sizeDelta;
584 bool isLong;
585 uint256 triggerPrice;
586 bool triggerAboveThreshold;
587 uint256 executionFee;
588 ) {
589 IncreaseOrder memory order = increaseOrders[_account][_orderIndex];
590 return
591 order.purchaseToken;
592 order.purchaseTokenAmount;
593 order.collateralToken;
594 order.indexToken;
595 order.sizeDelta;
596 order.isLong;
597 order.triggerPrice;
598 order.triggerAboveThreshold;
599 order.executionFee;
600 }
601 }
602
603 function createIncreaseOrder(
604 address[] memory _path;
605 uint256 _amountIn;
606 address _indexToken;
607 uint256 _minOut;
608 uint256 _sizeDelta;
609 address _collateralToken;
610 bool _isLong;
611 uint256 _triggerPrice;
612 bool _triggerAboveThreshold;
613 uint256 _executionFee;
614 bool _shouldWrap;
615 ) external payable nonReentrant {
616 // always need this call because of mandatory executionFee user has to transfer in ETH
617 transferInETH();
618
619 require(_executionFee >= minExecutionFee, "OrderBook: insufficient execution fee");
620 if (_shouldWrap {
621 require(_path[0] == weth, "OrderBook: only weth could be wrapped");
622 require(msg.value == _executionFee + _amountIn, "OrderBook: incorrect value transferred");
623 } else {
624 require(msg.value == _executionFee, "OrderBook: incorrect execution fee transferred");
625 IRouter(router).pluginTransfer(_path[0], msg.sender, address(this), _amountIn);
626 }
627

```

```

628 address _purchaseToken = _path[_path.length - 1];
629 uint256 _purchaseTokenAmount;
630 if (_path.length > 1) {
631     require(_path[0] != _purchaseToken, "OrderBook: invalid _path");
632     IERC20(_path[0]).safeTransfer(vault, _amountIn);
633     _purchaseTokenAmount = _swap(_path, _minOut, address(this));
634 } else {
635     _purchaseTokenAmount = _amountIn;
636 }
637
638
639 uint256 _purchaseTokenAmountUsd = IVault(vault).tokenToUsdMin(_purchaseToken, _purchaseTokenAmount);
640 require(_purchaseTokenAmountUsd >= minPurchaseTokenAmountUsd, "OrderBook: insufficient collateral");
641
642
643 createIncreaseOrder
644 msg.sender
645 _purchaseToken
646 _purchaseTokenAmount
647 _collateralToken
648 _indexToken
649 _sizeDelta
650 _isLong
651 _triggerPrice
652 _triggerAboveThreshold
653 _executionFee
654
655
656
657 function createIncreaseOrder
658 address _account
659 address _purchaseToken
660 uint256 _purchaseTokenAmount
661 address _collateralToken
662 address _indexToken
663 uint256 _sizeDelta
664 bool _isLong
665 uint256 _triggerPrice
666 bool _triggerAboveThreshold
667 uint256 _executionFee
668 private {
669     uint256 _orderIndex = increaseOrdersIndex[msg.sender];
670
671     uint256 blocknum = Chain.currentBlockNumber();
672     IncreaseOrder memory order = IncreaseOrder(
673         _account,
674         _purchaseToken,
675         _purchaseTokenAmount,
676         _collateralToken,
677         _indexToken,
678         _sizeDelta,
679         _isLong,
680         _triggerPrice,
681         _triggerAboveThreshold,
682         _executionFee,
683         blocknum
684     );
685     increaseOrdersIndex[_account] = _orderIndex.add(1);
686     increaseOrders[_account][_orderIndex] = order;
687
688     emit CreateIncreaseOrder(
689         _account,
690         _orderIndex,

```

```

691     _purchaseToken
692     _purchaseTokenAmount
693     _collateralToken
694     _indexToken
695     _sizeDelta
696     _isLong
697     _triggerPrice
698     _triggerAboveThreshold
699     _executionFee
700 }
701
702
703 function updateIncreaseOrder(uint256 _orderId, uint256 _sizeDelta, uint256 _triggerPrice, bool _triggerAboveThreshold) external nonReentrant {
704     IncreaseOrder storage order = increaseOrders[msg.sender][_orderId];
705     require(order.account != address(0), "OrderBook: non-existent order");
706
707     order.triggerPrice = _triggerPrice;
708     order.triggerAboveThreshold = _triggerAboveThreshold;
709     order.sizeDelta = _sizeDelta;
710     order.updatedAtBlock = Chain.currentBlockNumber();
711
712     emit UpdateIncreaseOrder(
713         msg.sender,
714         _orderId,
715         order.collateralToken,
716         order.indexToken,
717         order.isLong,
718         _sizeDelta,
719         _triggerPrice,
720         _triggerAboveThreshold
721     );
722 }
723
724 function cancelIncreaseOrder(uint256 _orderId) public nonReentrant {
725     IncreaseOrder memory order = increaseOrders[msg.sender][_orderId];
726     require(order.account != address(0), "OrderBook: non-existent order");
727
728     delete increaseOrders[msg.sender][_orderId];
729
730     if (order.purchaseToken == weth) {
731         transferOutETH(order.executionFee.add(order.purchaseTokenAmount), payable(msg.sender));
732     } else {
733         IERC20(order.purchaseToken).safeTransfer(msg.sender, order.purchaseTokenAmount);
734         transferOutETH(order.executionFee, payable(msg.sender));
735     }
736
737     emit CancelIncreaseOrder(
738         order.account,
739         _orderId,
740         order.purchaseToken,
741         order.purchaseTokenAmount,
742         order.collateralToken,
743         order.indexToken,
744         order.sizeDelta,
745         order.isLong,
746         order.triggerPrice,
747         order.triggerAboveThreshold,
748         order.executionFee
749     );
750 }
751
752 function executeIncreaseOrder(address _address, uint256 _orderId, address payable _feeReceiver, address _oracle) override external nonReentrant onlyPositionManager {
753     IncreaseOrder memory order = increaseOrders[_address][_orderId];

```



```

754 require(order.account != address(0), "OrderBook: non-existent order");
755
756
757 if (order.updatedAtBlock > IOracle(_oracle).minOracleBlockNumbers(order.purchaseToken)) {
758     revert("order.updatedAtBlock > oracle.minOracleBlockNumbers1");
759 }
760 if (order.updatedAtBlock > IOracle(_oracle).minOracleBlockNumbers(order.collateralToken)) {
761     revert("order.updatedAtBlock > oracle.minOracleBlockNumbers2");
762 }
763 if (order.updatedAtBlock > IOracle(_oracle).minOracleBlockNumbers(order.indexToken)) {
764     revert("order.updatedAtBlock > oracle.minOracleBlockNumbers3");
765 }
766
767
768 // increase long should use max price
769 // increase short should use min price
770 uint256 currentPrice;
771     = validatePositionOrderPrice
772 order.triggerAboveThreshold
773 order.triggerPrice
774 order.indexToken
775 order.isLong
776 true
777 _oracle
778
779 delete increaseOrders[_address][_orderIndex];
780
781 IERC20(order.purchaseToken).safeTransfer(vault, order.purchaseTokenAmount);
782
783 if (order.purchaseToken != order.collateralToken) {
784     address[] memory path = new address[](2);
785     path[0] = order.purchaseToken;
786     path[1] = order.collateralToken;
787
788     uint256 amountOut = swap(path, 0, address(this));
789     IERC20(order.collateralToken).safeTransfer(vault, amountOut);
790 }
791
792 IVault(vault).increasePosition(order.account, order.collateralToken, order.indexToken, order.sizeDelta, order.isLong);
793
794 // pay executor
795 transferOutETH(order.executionFee, _feeReceiver);
796 emitExecuteIncreaseOrder(order, _orderIndex, currentPrice);
797
798
799 function emitExecuteIncreaseOrder(IncreaseOrder memory order, uint256 _orderIndex, uint256 currentPrice) private {
800     emit ExecuteIncreaseOrder(
801         order.account,
802         _orderIndex,
803         order.purchaseToken,
804         order.purchaseTokenAmount,
805         order.collateralToken,
806         order.indexToken,
807         order.sizeDelta,
808         order.isLong,
809         order.triggerPrice,
810         order.triggerAboveThreshold,
811         order.executionFee,
812         currentPrice
813     );
814 }
815
816 function createDecreaseOrder(

```

```

817 address _indexToken,
818 uint256 _sizeDelta,
819 address _collateralToken,
820 uint256 _collateralDelta,
821 bool _isLong,
822 uint256 _triggerPrice,
823 bool _triggerAboveThreshold
824 } external payable nonReentrant {
825     .transferInETH();
826
827     require(msg.value > minExecutionFee, "OrderBook: insufficient execution fee");
828
829     .createDecreaseOrder(
830         msg.sender,
831         _collateralToken,
832         _collateralDelta,
833         _indexToken,
834         _sizeDelta,
835         _isLong,
836         _triggerPrice,
837         _triggerAboveThreshold
838     );
839 }
840
841 function .createDecreaseOrder(
842     address _account,
843     address _collateralToken,
844     uint256 _collateralDelta,
845     address _indexToken,
846     uint256 _sizeDelta,
847     bool _isLong,
848     uint256 _triggerPrice,
849     bool _triggerAboveThreshold
850 ) private {
851     uint256 _orderIndex = decreaseOrdersIndex[_account];
852
853     uint256 blocknum = Chain.currentBlockNumber();
854     DecreaseOrder memory order = DecreaseOrder(
855         _account,
856         _collateralToken,
857         _collateralDelta,
858         _indexToken,
859         _sizeDelta,
860         _isLong,
861         _triggerPrice,
862         _triggerAboveThreshold,
863         msg.value,
864         blocknum
865     );
866     decreaseOrdersIndex[_account] = _orderIndex.add(1);
867     decreaseOrders[_account][_orderIndex] = order;
868
869     emit CreateDecreaseOrder(
870         _account,
871         _orderIndex,
872         _collateralToken,
873         _collateralDelta,
874         _indexToken,
875         _sizeDelta,
876         _isLong,
877         _triggerPrice,
878         _triggerAboveThreshold,
879         msg.value

```

```

880
881
882
883 function executeDecreaseOrder(address _address, uint256 _orderIndex, address payable _feeReceiver, address _oracle override external nonReentrant onlyPositionManager
884 DecreaseOrder memory order = decreaseOrders[_address][_orderIndex];
885 require(order.account != address(0), "OrderBook: non-existent order");
886
887
888 if (order.updatedAtBlock > IOracle(_oracle).minOracleBlockNumbers(order.collateralToken)) {
889 revert("order.updatedAtBlock > oracle.minOracleBlockNumbers2");
890 }
891
892 if (order.updatedAtBlock > IOracle(_oracle).minOracleBlockNumbers(order.indexToken)) {
893 revert("order.updatedAtBlock > oracle.minOracleBlockNumbers");
894 }
895
896
897 // decrease long should use min price
898 // decrease short should use max price
899 uint256 currentPrice, = validatePositionOrderPrice
900 order.triggerAboveThreshold,
901 order.triggerPrice,
902 order.indexToken,
903 !order.isLong,
904 true,
905 _oracle
906 );
907 delete decreaseOrders[_address][_orderIndex];
908
909 uint256 amountOut = IVault(vault).decreasePosition(
910 order.account,
911 order.collateralToken,
912 order.indexToken,
913 order.collateralDelta,
914 order.sizeDelta,
915 order.isLong,
916 address(this),
917 );
918
919 // transfer released collateral to user
920 if (order.collateralToken == weth) {
921 transferOutETH(amountOut, payable(order.account));
922 } else {
923 IERC20(order.collateralToken).safeTransfer(order.account, amountOut);
924 }
925
926 // pay executor
927 transferOutETH(order.executionFee, _feeReceiver);
928 emitExecuteDecreaseOrder(order, _orderIndex, currentPrice);
929
930
931 function emitExecuteDecreaseOrder(DecreaseOrder memory order, uint256 _orderIndex, uint256 currentPrice) private {
932 emit ExecuteDecreaseOrder(
933 order.account,
934 order.index,
935 order.collateralToken,
936 order.collateralDelta,
937 order.indexToken,
938 order.sizeDelta,
939 order.isLong,
940 order.triggerPrice,
941 order.triggerAboveThreshold,
942 order.executionFee,

```

```

943     currentPrice
944 }
945 }
946
947 function cancelDecreaseOrder(uint256 _orderIndex) public nonReentrant {
948     DecreaseOrder memory order = decreaseOrders[msg.sender][_orderIndex];
949     require(order.account != address(0), "OrderBook: non-existent order");
950
951     delete decreaseOrders[msg.sender][_orderIndex];
952     .transferOutETH(order.executionFee, payable(msg.sender));
953
954     emit CancelDecreaseOrder(
955         order.account,
956         orderIndex,
957         order.collateralToken,
958         order.collateralDelta,
959         order.indexToken,
960         order.sizeDelta,
961         order.isLong,
962         order.triggerPrice,
963         order.triggerAboveThreshold,
964         order.executionFee
965     );
966 }
967
968 function updateDecreaseOrder(
969     uint256 _orderIndex,
970     uint256 _collateralDelta,
971     uint256 _sizeDelta,
972     uint256 _triggerPrice,
973     bool _triggerAboveThreshold
974 ) external nonReentrant {
975     DecreaseOrder storage order = decreaseOrders[msg.sender][_orderIndex];
976     require(order.account != address(0), "OrderBook: non-existent order");
977
978     order.triggerPrice = _triggerPrice;
979     order.triggerAboveThreshold = _triggerAboveThreshold;
980     order.sizeDelta = _sizeDelta;
981     order.collateralDelta = _collateralDelta;
982     order.updatedAtBlock = Chain.currentBlockNumber();
983
984     emit UpdateDecreaseOrder(
985         msg.sender,
986         orderIndex,
987         order.collateralToken,
988         collateralDelta,
989         order.indexToken,
990         sizeDelta,
991         order.isLong,
992         triggerPrice,
993         triggerAboveThreshold
994     );
995 }
996
997 function transferInETH() private {
998     if (msg.value != 0) {
999         IWEth(weth).deposit{value: msg.value}();
1000     }
1001 }
1002
1003 function transferOutETH(uint256 _amountOut, address payable _receiver) private {
1004     IWEth(weth).withdraw(_amountOut);
1005     _receiver.sendValue(_amountOut);

```

```

1006 |
1007 |
1008 | function _swap(address[] memory _path, uint256 _minOut, address _receiver) private returns (uint256) {
1009 |     if (_path.length == 2) {
1010 |         return _vaultSwap(_path[0], _path[1], _minOut, _receiver);
1011 |     }
1012 |     if (_path.length == 3) {
1013 |         uint256 midOut = _vaultSwap(_path[0], _path[1], 0, address(this));
1014 |         IERC20(_path[1]).safeTransfer(vault, midOut);
1015 |         return _vaultSwap(_path[1], _path[2], _minOut, _receiver);
1016 |     }
1017 |
1018 |     revert("OrderBook: invalid _path.length");
1019 | }
1020 |
1021 | function _vaultSwap(address _tokenIn, address _tokenOut, uint256 _minOut, address _receiver) private returns (uint256) {
1022 |     uint256 amountOut;
1023 |
1024 |     if (_tokenOut == usdg) { // buyUSDG
1025 |         amountOut = IVault(vault).buyUSDG(_tokenIn, _receiver);
1026 |     } else if (_tokenIn == usdg) { // sellUSDG
1027 |         amountOut = IVault(vault).sellUSDG(_tokenOut, _receiver);
1028 |     } else { // swap
1029 |         amountOut = IVault(vault).swap(_tokenIn, _tokenOut, _receiver);
1030 |     }
1031 |
1032 |     require(amountOut >= _minOut, "OrderBook: insufficient amountOut");
1033 |     return amountOut;
1034 | }
1035 |

```