Lyra Finance

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **May 23rd, 2022 - June 15th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/23/2022	Luis Quispe Gonzales
0.2	Document Update	06/10/2022	Luis Quispe Gonzales
0.3	Draft Version	06/16/2022	Luis Quispe Gonzales
0.4	Draft Review	06/16/2022	Gabi Urrutia
1.0	Remediation Plan	06/22/2022	Luis Quispe Gonzales
1.1	Remediation Plan Review	06/22/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Luis Quispe Gonzales	Halborn	Luis.QuispeGonzales@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Lyra Finance engaged Halborn to conduct a security audit on their smart contracts beginning on May 23rd, 2022 and ending on June 15th, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided three weeks and a half for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the Lyra team. The main ones were the following:

- Update the calculus of pending delta liquidity to achieve a more accurate delta hedging.
- Revert transactions that try to update collateral in positions with settled boards.
- Limit the amount of sUSD that liquidity pool can swap to sETH when closing a position.
- Cap the differences between sETH balance and locked collateral when opening a position.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the solidity code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs (MythX).
- Static Analysis of security for scoped contract, and imported functions (Slither).
- Testnet deployment (Remix IDE).

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

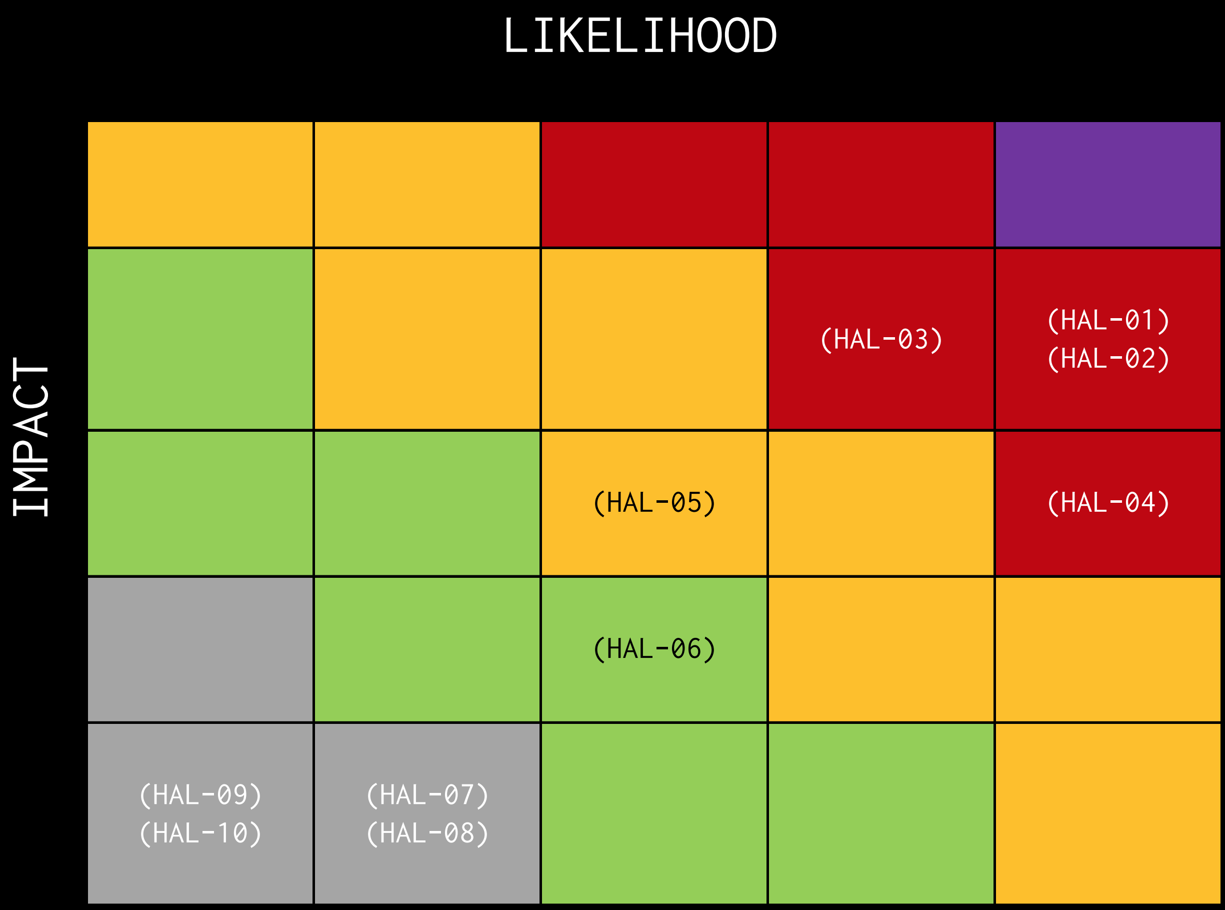
1.4 SCOPE

1. Solidity Smart Contracts
   (a) Repository: lyra-protocol
   (b) Commit ID: a834eb7b2dc044fc26964072f8e0d3cd414faa61
   (c) Contracts in scope:
      i. LiquidityPool.sol
      ii. LiquidityTokens.sol
      iii. OptionGreekCache.sol
      iv. OptionMarket.sol
      v. OptionMarketPricer.sol
      vi. OptionToken.sol
      vii. PoolHedger.sol
      viii. ShortCollateral.sol
      ix. SynthetixAdapter.sol
      x. Contracts in interfaces, lib and synthetix folders

Out-of-scope: External libraries and financial related attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	4	1	1	4



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MISCALCULATION OF PENDING DELTA LIQUIDITY LEADS TO AN INCORRECT HEDGE	High	SOLVED - 06/14/2022
(HAL-02) COLLATERAL CAN BE UPDATED IN SETTLED BOARDS	High	SOLVED - 06/20/2022
(HAL-03) LIQUIDITY POOL COULD RUN OUT OF SUSD TOKENS	High	SOLVED - 06/14/2022
(HAL-04) DIFFERENCES BETWEEN LOCKED COLLATERAL AND SETH BALANCE ARE NOT ADEQUATELY CAPPED	High	PARTIALLY SOLVED
(HAL-05) SKEW UPDATE COULD CREATE DATA INCONSISTENCIES WITH GWAV ORACLE	Medium	RISK ACCEPTED
(HAL-06) WITHDRAWALS GET TEMPORARILY BLOCKED WHEN CREATING BOARDS	Low	SOLVED - 06/21/2022
(HAL-07) INACCURATE VALIDITY CHECKS	Informational	ACKNOWLEDGED
(HAL-08) CHANGES CAN BE MADE IN EXPIRED BOARDS	Informational	ACKNOWLEDGED
(HAL-09) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS	Informational	SOLVED - 06/21/2022
(HAL-10) CACHING ARRAY LENGTH IN FOR LOOPS CAN SAVE GAS	Informational	SOLVED - 06/21/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISCALCULATION OF PENDING DELTA LIQUIDITY LEADS TO AN INCORRECT HEDGE - HIGH

Description:

The _getLiquidity function in the **LiquidityPool** contract miscalculates the value of pendingDeltaLiquidity when **pendingDelta > freeLiquidity**.

Consequently, in the aforementioned situation, the liquidity pool cannot be used for withdrawals nor use the available liquidity for delta hedging, as shown in the following comparison table:

Scenario 1: Using pendingDelta
Scenario 2: Correct calculus

Parameter	Scenario 1	Scenario 2
totalQuote	100	100
usedQuote	80	80
reservedTokenValue	15	15
pendingDelta	30	30
liquidity.pendingDeltaLiquidity	5	20
liquidity.freeLiquidity	0	0
liquidity.burnableLiquidity	0	0

Scenario 2 shows that **pendingDeltaLiquidity**, the amount is used for delta hedging, could be up to 20 instead of just 5. This latter value is the incorrect amount proposed by Scenario 1.

Code Location:

Listing 1: LiquidityPool.sol (Line 863)

```
854 uint usedQuote = totalOutstandingSettlements + totalQueuedDeposits
    ↳ + lockedCollateral.quote + pendingBaseValue;
855
856 uint totalQuote = quoteAsset.balanceOf(address(this));
857
858 liquidity.freeLiquidity = totalQuote > (usedQuote +
```

```
reservedTokenValue)
  ? totalQuote - (usedQuote + reservedTokenValue)
  : 0;

// ensure pendingDelta <= liquidity.freeLiquidity
liquidity.pendingDeltaLiquidity = liquidity.freeLiquidity > pendingDelta ? pendingDelta : liquidity.freeLiquidity;
liquidity.freeLiquidity -= liquidity.pendingDeltaLiquidity;

liquidity.burnableLiquidity = totalQuote > (usedQuote + pendingDelta) ? totalQuote - (usedQuote + pendingDelta) : 0;
```

Risk Level:

Likelihood - 5
Impact - 4

Recommendation:

Update the pendingDeltaLiquidity calculation to take advantage of current balance in the liquidity pool for more accurate delta hedging.

Remediation plan:

SOLVED: The issue was fixed in commit d8d2e902c6d368313d9e04bec40c21fbf70b870b.

3.2 (HAL-02) COLLATERAL CAN BE UPDATED IN SETTLED BOARDS - HIGH

Description:

The _doTrade and addCollateral functions from **OptionMarket** contract allow updating collateral on positions, even if the board is **settled**. As a consequence, unexpected situations may happen:

- The added collateral could be forever stuck in **ShortCollateral** contract and the liquidity pool would never get it.
- If the result is not favorable for users once a board is settled, they can reduce their collateral and negatively affect the liquidity of the protocol.
- Even in some edge scenarios of volatility, the collateral on positions can be reduced to less than expected.

Code Location:

addCollateral function will update position's collateral without previously verifying if board is **settled**:

Listing 2: OptionMarket.sol (Line 511)

```
508 function addCollateral(uint positionId, uint amountCollateral) external nonReentrant notGlobalPaused {
509   int pendingCollateral = SafeCast.toInt256(amountCollateral);
510   OptionType optionType = optionToken.addCollateral(positionId, amountCollateral);
511   _routeUserCollateral(optionType, pendingCollateral);
512 }
```

When opening / closing a position, if trade amount is 0, _doTrade function won't verify is board is **settled** and will return earlier, which allows updating position's collateral:

Listing 3: OptionMarket.sol (Lines 728,732)

```
727 // don't engage AMM if only collateral is added/removed
728 if (trade.amount == 0) {
729   if (expectedAmount != 0) {
730     revert TradeIterationsHasRemainder(address(this), iterations,
      ↳ expectedAmount, 0, 0);
731   }
732   return (0, 0, 0, new OptionMarketPricer.TradeResult[](0));
733 }
734
735 if (board.frozen) {
736   revert BoardIsFrozen(address(this), board.id);
737 }
738 if (board.expiry < block.timestamp) {
739     revert BoardExpired(address(this), board.id, board.expiry,
      ↳ block.timestamp);
```

Risk Level:

Likelihood - 5
Impact - 4

Recommendation:

Update the logic of addCollateral and _doTrade functions to revert if a board is **settled**.

Remediation plan:

SOLVED: The issue was fixed in the following commits:

- 056017961c89f0800eec2dfd5bc559a591b68aae
- d8d2e902c6d368313d9e04bec40c21fbf70b870b

3.3 (HAL-03) LIQUIDITY POOL COULD RUN OUT OF SUSD TOKENS - HIGH

Description:

When closing a position **long call**, the _maybeExchangeBase function is called with the argument revertBuyOnInsufficientFunds set to **false**. As a consequence, the liquidity pool will be able to swap sUSD for sETH without limits and could eventually run out of sUSD tokens.

This situation could affect some relevant operations such as withdrawal, premium payment, settlement, etc.

Proof of Concept:

Initial liquidity info for the test:

```
LIQUIDITY INFO at the beginning ==>

lockedCollateral.quote: 9501.000000000000014400
lockedCollateral.base: 54.543000000000000021
sETH for LP: 54.543000000000000020
sUSD for LP: 1840627.851083896890880753

Liquidity info:
1831126.851083896890864533
1831126.851083896890866353
108822.041039109990052640
0
65428.033812641801566895
2011840.657743795295625665
```

The attacker opens a long call position of 120 sETH. There is a big difference between lockedCollateral.base and sETH balance because of (HAL-04) DIFFERENCES BETWEEN LOCKED COLLATERAL AND SETH BALANCE ARE NOT ADEQUATELY CAPPED:

```
Attacker opens position: 120 sETH long call

Transaction sent: 0xa0aed23bb877c42e64f7356629f3df159e388a0edd5f9538737621ab71ad0988
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  OptionMarket.openPosition confirmed   Block: 3873752   Gas used: 1641660 (13.68%)
```

```
LIQUIDITY INFO after opening position ==>

lockedCollateral.quote: 9501.000000000000014400
lockedCollateral.base: 174.543000000000000021
sETH for LP: 54.543000000000000020
sUSD for LP: 1883448.436249456900578499

Liquidity info:
1634948.399434191862033186
1853464.522945791862035006
327338.164550709990052640
20482.913303665038529093
65428.033812641801566895
2013353.350435351843563691
```

Because of the difference, the owner decides to set maxFeePaid = **MAX_UINT** to enable sETH repurchase.

On the other hand, a user withdraws sUSD from liquidity pool. The image shows liquidity info after withdrawal:

```
A user withdraws from LP...

Transaction sent: 0x635e9a8860554b0427a085199806a7cf99a2b561b76ad6449a05e0ea2dae5ec7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 52559
  LiquidityPool.processWithdrawalQueue confirmed   Block: 3873755   Gas used: 346462 (2.89%)

LIQUIDITY INFO after the withdraw ==>

lockedCollateral.quote: 9501.000000000000014400
lockedCollateral.base: 174.543000000000000021
sETH for LP: 54.543000000000000020
sUSD for LP: 91070.045664819640578499

Liquidity info:
0
61086.132361154602035006
327338.164550709990052640
0
65428.033812641801566895
220974.959850714583563691
```

Finally, the attacker closes the 82.6 sETH long call position:

```
Attacker closes position: 82.6 sETH long call

Transaction sent: 0x4326fa66a5fb145fefcbbac686ed65b5ff3256bf3c902bf135bb1688d0f563b2
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  OptionMarket.closePosition confirmed   Block: 3873756   Gas used: 1925908 (16.05%)
```

Since the swap is not limited by any parameter, the protocol uses all available sUSD. In the end, the new sUSD balance is almost 0 (0.29037... in the example):

```
LIQUIDITY INFO after closing position ==>

lockedCollateral.quote: 9501.000000000000014400
lockedCollateral.base: 91.943000000000000021
sETH for LP: 91.943000000000000021
sUSD for LP: 0.290376586180518102

Liquidity info:
0
0
176926.232866891990052640
0
65428.033812641801566895
226442.995709868839683055
```

Code Location:

Listing 4: LiquidityPool.sol (Line 917)

```
913 (uint quoteSpent, uint baseReceived) = synthetixAdapter.
    ↳ exchangeToExactBaseWithLimit(
914   exchangeParams,
915   address(optionMarket),
916   amountBase,
917   revertBuyOnInsufficientFunds ? freeLiquidity : type(uint).max
918 );
919 emit BasePurchased(quoteSpent, baseReceived);
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

It is recommended to only allow the use of freeLiquidity for swapping when closing a position. Another alternative could be to allow an amount greater than freeLiquidity but within a predefined threshold.

Remediation plan:

SOLVED: The issue was fixed in commit d8d2e902c6d368313d9e04bec40c21fbf70b870b. With the update to the _getLiquidity function in the **LiquidityPool** contract, this attack vector is not feasible.

3.4 (HAL-04) DIFFERENCES BETWEEN LOCKED COLLATERAL AND SETH BALANCE ARE NOT ADEQUATELY CAPPED - HIGH

Description:

The differences between lockedCollateral.base and sETH balance in the liquidity pool are not limited to opening **long call** positions. These differences could create distorted liquidity values with the following consequences:

- In some scenarios, the liquidity pool can run out of sUSD, which could affect some operations such as withdrawals, payment of premiums, settlement, etc. See (HAL-03) LIQUIDITY POOL COULD RUN OUT OF SUSD TOKENS for more details.
- Distorted liquidity values will erroneously affect protocol decisions, for example: more liquidity to withdraw, less amount to hedge, etc. See Proof of Concept below for more details.

Proof of Concept:

Initial situation for the test:

```
Running 'scripts/attack1.py::main'...
Transaction sent: 0xb37574c084717667be73043a76ccbf01a1a2ed78402aaf065fe1758d92535b04
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  LiquidityPool.setLiquidityPoolParameters confirmed   Block: 3861937   Gas used: 57939 (0.48%)

Positions for attacker: ()

lockedCollateral.base: 54.543000000000000021
sETH for LP: 54.543000000000000021
sUSD for LP: 49840627.851083896890880753
```

Scenario 1: quoteBaseFeeRate <= maxFeePaid

The user opens a long call position of 100 sETH. Due to the swap, **sETH balance** and **lockedCollateral.base** have the same value:

```
Attacker will open a position: 100 sETH long call

Transaction sent: 0x5b1fe47a425e2a5bcafcebd6d971e5e666b09a9132aedf0a525a589303f75e4e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  OptionMarket.openPosition confirmed   Block: 3861940   Gas used: 1868954 (15.57%)

Positions for attacker: ((292, 1, 0, 100000000000000000000, 0, 1),)

lockedCollateral.base: 154.543000000000000021
sETH for LP: 154.543000000000000021
sUSD for LP: 49707841.519693199889784837

Liquidity info:
49625898.735556274855150323
49625898.735556274855150323
290918.810632109990052640
72441.784136925034620114
65428.033812641801566895
50026696.302049777684252500
```

Scenario 2: quoteBaseFeeRate > maxFeePaid

The user opens a long call position 100 sETH. Because there is no swap, differences between **sETH balance** and **lockedCollateral.base** will increase:

```
Attacker will open a position: 100 sETH long call

Transaction sent: 0x5b1fe47a425e2a5bcafcebd6d971e5e666b09a9132aedf0a525a589303f75e4e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  OptionMarket.openPosition confirmed   Block: 3861890   Gas used: 1632091 (13.60%)

Positions for attacker: ((292, 1, 0, 100000000000000000000, 0, 1),)

lockedCollateral.base: 154.543000000000000021
sETH for LP: 54.543000000000000021
sUSD for LP: 49876283.902989623663638291

Liquidity info:
49684686.133396623663623891
49866782.902989623663623891
290918.810632109990052640
0
65428.033812641801566895
50013041.915753201458105954
```

Comparative table of liquidity info between both scenarios:

Scenario 1: quoteBaseFeeRate <= maxFeePaid
Scenario 2: quoteBaseFeeRate > maxFeePaid

Parameter	Scenario 1	Scenario 2
freeLiquidity	49,625,898.74	49,684,686.13
burnableLiquidity	49,625,898.74	49,866,782.90
usedCollatLiquidity	290,918.81	290,918.81
pendingDeltaLiquidity	72,441.78	0.00
usedDeltaLiquidity	65,428.03	65,428.03
NAV	50,026,696.30	50,013,041.92

Scenario 2 shows that **freeLiquidity** and **burnableLiquidity** have increased, but **pendingDeltaLiquidity** has decreased.

These **distorted** liquidity values (compared to Scenario 1) will erroneously affect protocol decisions, e.g: more liquidity to withdraw, less amount to hedge, etc.

Code Location:

lockBase function increases the value of lockedCollateral.base and then calls _maybeExchangeBase:

Listing 5: LiquidityPool.sol (Lines 554,555)

```
549 function lockBase(
550   uint amount,
551   SynthetixAdapter.ExchangeParams memory exchangeParams,
552   uint freeLiquidity
553 ) external onlyOptionMarket {
554   lockedCollateral.base += amount;
555   _maybeExchangeBase(exchangeParams, freeLiquidity, true);
556   emit BaseLocked(amount, lockedCollateral.base);
557 }
```

_maybeExchangeBase function could return earlier without swapping, which creates a big difference between lockedCollateral.base and sETH balance:

Listing 6: LiquidityPool.sol (Lines 898,901,911)

```
898 } else if (currentBaseBalance < lockedCollateral.base) {
899   // Buy base for quote
900   uint amountBase = lockedCollateral.base - currentBaseBalance;
901   if (exchangeParams.quoteBaseFeeRate > lpParams.maxFeePaid) {
902     uint estimatedExchangeCost = synthetixAdapter.
  ↳ estimateExchangeToExactBase(exchangeParams, amountBase);
903     if (revertBuyOnInsufficientFunds && estimatedExchangeCost >
  ↳ freeLiquidity) {
904       revert InsufficientFreeLiquidityForBaseExchange(
905         address(this),
906         amountBase,
907         estimatedExchangeCost,
```

```
        freeLiquidity
      );
    }
    return;
  }
```

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

It is recommended not to allow opening positions if the liquidity pool cannot get enough sETH, even after trying to swap. Another alternative could be to allow these transactions as long as the difference between sETH balance and lockedCollateral.base does not exceed a predefined threshold.

Remediation plan:

PARTIALLY SOLVED: Commit d8d2e902c6d368313d9e04bec40c21fbf70b870b partially fixes this security issue by not allowing the liquidity pool to run out of sUSD.

3.5 (HAL-05) SKEW UPDATE COULD CREATE DATA INCONSISTENCIES WITH GWAV ORACLE - MEDIUM

Description:

The _addNewStrikeToStrikeCache and _updateStrikeSkew functions of the **OptionGreekCache** contract update strikeSkewGWAV with the value of a new skew.

If this new skew is outside the **gwavSkewFloor / gwavSkewCap** range, strikeSkewGWAV will store a capped skew (not the actual value), which feeds the GWAV oracle inconsistent data.

Code Location:

Listing 7: OptionGreekCache.sol (Lines 408,420)

```
404 function _addNewStrikeToStrikeCache(
405   OptionBoardCache storage boardCache,
406   uint strikeId,
407   uint strikePrice,
408   uint skew
409 ) internal {
410   // This is only called when a new board or a new strike is added
    ↳ , so exposure values will be 0
411   StrikeCache storage strikeCache = strikeCaches[strikeId];
412   strikeCache.id = strikeId;
413   strikeCache.strikePrice = strikePrice;
414   strikeCache.skew = skew;
415   strikeCache.boardId = boardCache.id;
416
417   emit StrikeCacheUpdated(strikeCache);
418
419   strikeSkewGWAV[strikeId]._initialize(
420     _max(_min(skew, greekCacheParams.gwavSkewCap),
    ↳ greekCacheParams.gwavSkewFloor),
421     block.timestamp
422   );
```

Listing 8: OptionGreekCache.sol (Lines 862,865)

```
857 function _updateStrikeSkew(
858   OptionBoardCache storage boardCache,
859   StrikeCache storage strikeCache,
860   uint newSkew
861 ) internal {
862 strikeCache.skew = newSkew;
863
864 strikeSkewGWAV[strikeCache.id]._write(
865   _max(_min(newSkew, greekCacheParams.gwavSkewCap),
  ↳ greekCacheParams.gwavSkewFloor),
866   block.timestamp
867 );
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to revert transactions if the new value of skew is less than **gwavSkewFloor** or greater than **gwavSkewCap**.

Remediation plan:

RISK ACCEPTED: The Lyra team accepted the risk of this finding and stated that feeding the GWAV oracle with a capped skew value (different from the one cached) in edge cases is intentional behavior of the protocol.

3.6 (HAL-06) WITHDRAWALS GET TEMPORARILY BLOCKED WHEN CREATING BOARDS - LOW

Description:

When the owner creates a new board, the addBoard function from **OptionGreekCache** contract is called. This function triggers _updateGlobalLastUpdatedAt, which sets the value of minUpdatedAtPrice to 0.

As a consequence, users will not be able to withdraw from the liquidity pool (processWithdrawalQueue) because _canProcess will always return **false** until someone explicitly calls the updateBoardCachedGreeks function to update the cache with the actual values.

Code Location:

addBoard function triggers _updateGlobalLastUpdatedAt:

Listing 9: OptionGreekCache.sol (Line 336)

```
323 boardCache.expiry = board.expiry;
324 boardCache.iv = board.iv;
325 boardCache.updatedAt = block.timestamp;
326 emit BoardCacheUpdated(boardCache);
327 boardIVGWAV[board.id]._initialize(board.iv, block.timestamp);
328 emit BoardIvUpdated(boardCache.id, board.iv, globalCache.maxIvVariance);
329
330 liveBoards.push(board.id);
331
332 for (uint i = 0; i < strikes.length; i++) {
333   _addNewStrikeToStrikeCache(boardCache, strikes[i].id, strikes[i].strikePrice, strikes[i].skew);
334 }
335
336 _updateGlobalLastUpdatedAt();
```

_updateGlobalLastUpdatedAt function sets minUpdatedAtPrice to 0:

Listing 10: OptionGreekCache.sol (Lines 822,836)

```
816 for (uint i = 1; i < liveBoards.length; i++) {
817   boardCache = boardCaches[liveBoards[i]];
818   if (boardCache.updatedAt < minUpdatedAt) {
819     minUpdatedAt = boardCache.updatedAt;
820   }
821   if (boardCache.updatedAtPrice < minUpdatedAtPrice) {
822     minUpdatedAtPrice = boardCache.updatedAtPrice;
823   }
824   if (boardCache.updatedAtPrice > maxUpdatedAtPrice) {
825     maxUpdatedAtPrice = boardCache.updatedAtPrice;
826   }
827   if (boardCache.maxSkewVariance > maxSkewVariance) {
828     maxSkewVariance = boardCache.maxSkewVariance;
829   }
830   if (boardCache.ivVariance > maxIvVariance) {
831     maxIvVariance = boardCache.ivVariance;
832   }
833 }
834
835 globalCache.minUpdatedAt = minUpdatedAt;
836 globalCache.minUpdatedAtPrice = minUpdatedAtPrice;
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended to change the visibility of the updateBoardCachedGreeks function to **public** and update addBoard to call this function instead of _updateGlobalLastUpdatedAt.

Remediation plan:

SOLVED: The issue was fixed in commit 1e04d54b12c4faf0378b54b67f93d5de2b7c6e68.

3.7 (HAL-07) INACCURATE VALIDITY CHECKS - INFORMATIONAL

Description:

The updateCacheAndGetTradeResult function in the **OptionMarketPricer** contract contains the following inaccurate validity checks:

- newSkew includes **min / max** values in its validity check. As a consequence, the function will revert incorrectly when dealing with edge values.
- pricing.callDelta does not include **min / max** values in its validity check. As a consequence, the function will not revert when dealing with edge values, as it should.

This issue is categorized as informational because it could cause the aforementioned function to not work as expected in edge cases.

Code Location:

Listing 11: OptionMarketPricer.sol (Line 262)

```
259 // If it is a force close and skew ends up outside the "abs min/
    ↳ max" thresholds
260 if (
261   trade.tradeDirection != OptionMarket.TradeDirection.LIQUIDATE &&
262   (newSkew <= tradeLimitParams.absMinSkew || newSkew >=
    ↳ tradeLimitParams.absMaxSkew)
263 ) {
264    revert ForceCloseSkewOutOfRange(
```

Listing 12: OptionMarketPricer.sol (Lines 328-329)

```
326 // delta must fall BELOW the min or ABOVE the max to allow for
    ↳ force closes
327 if (
328   pricing.callDelta > tradeLimitParams.minForceCloseDelta &&
```

```
  pricing.callDelta < (int(DecimalMath.UNIT) - tradeLimitParams.minForceCloseDelta)
) {
  revert ForceCloseDeltaOutOfRange(
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

It is recommended to correct validity checks as mentioned above to avoid unexpected behavior in the updateCacheAndGetTradeResult function when dealing with edge values.

Remediation plan:

ACKNOWLEDGED: The Lyra team acknowledged this finding.

3.8 (HAL-08) CHANGES CAN BE MADE IN EXPIRED BOARDS - INFORMATIONAL

Description:

The following operations in **OptionMarket** contract affect boards, even if they are already expired:

- Base IV can be set on an expired board
- Skew can be set on a strike from an expired board
- Strikes can be added on an expired board

It is worth noting that this issue is classified as informational because it does not affect the settlement process, but could cause the owner to spend more gas unnecessarily if they mistakenly interact with an expired board.

Code Location:

Listing 13: OptionMarket.sol (Lines 264,276,277)

```
264 function setBoardBaseIv(uint boardId, uint baseIv) external
    ↳ onlyOwner {
265   OptionBoard storage board = optionBoards[boardId];
266   if (board.id != boardId) {
267     revert InvalidBoardId(address(this), boardId);
268   }
269   if (baseIv == 0) {
270     revert ExpectedNonZeroValue(address(this), NonZeroValues.
    ↳ BASE_IV);
271   }
272   if (!board.frozen) {
273     revert BoardNotFrozen(address(this), boardId);
274   }
275
276   board.iv = baseIv;
277   greekCache.setBoardIv(boardId, baseIv);
278   emit BoardBaseIvSet(boardId, baseIv);
```

Listing 14: OptionMarket.sol (Lines 286,300,301)

```
286 function setStrikeSkew(uint strikeId, uint skew) external
  ↳ onlyOwner {
287   Strike storage strike = strikes[strikeId];
288   if (strike.id != strikeId) {
289     revert InvalidStrikeId(address(this), strikeId);
290   }
291   if (skew == 0) {
292     revert ExpectedNonZeroValue(address(this), NonZeroValues.SKEW);
293   }
294 
295   OptionBoard memory board = optionBoards[strike.boardId];
296   if (!board.frozen) {
297     revert BoardNotFrozen(address(this), board.id);
298   }
299 
300   strike.skew = skew;
301   greekCache.setStrikeSkew(strikeId, skew);
302   emit StrikeSkewSet(strikeId, skew);
303 }
```

Listing 15: OptionMarket.sol (Lines 312,319,320)

```
312 function addStrikeToBoard(
313   uint boardId,
314   uint strikePrice,
315   uint skew
316 ) external onlyOwner {
317   OptionBoard storage board = optionBoards[boardId];
318   if (board.id != boardId) revert InvalidBoardId(address(this),
  ↳ boardId);
319   Strike memory strike = _addStrikeToBoard(board, strikePrice,
  ↳ skew);
320   greekCache.addStrikeToBoard(boardId, strike.id, strikePrice,
  ↳ skew);
321 }
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

It is recommended that the functions mentioned above be reverted if they are called for expired boards.

Remediation plan:

ACKNOWLEDGED: The Lyra team acknowledged this finding.

3.9 (HAL-09) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS - INFORMATIONAL

Description:

In the following loops, the `i` variable is incremented using `i++`. It is known that, in loops, using `++i` costs less gas per iteration than `i++`.

Proof of Concept:

For example, based on the following test contract:

Listing 16: Test.sol

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.9;

contract test {
    function postiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; i++) {
        }
    }
    function preiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; ++i) {
        }
    }
}
```

We can see the difference in the gas costs:

```
>>> test_contract.postiincrement(1)
Transaction sent: 0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 44
  test.postiincrement confirmed   Block: 13622335   Gas used: 21620 (0.32%)

<Transaction '0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b'>
>>> test_contract.preiincrement(1)
Transaction sent: 0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 45
  test.preiincrement confirmed   Block: 13622336   Gas used: 21593 (0.32%)

<Transaction '0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a'>
```

```
>>> test_contract.postiincrement(10)
Transaction sent: 0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 46
  test.postiincrement confirmed   Block: 13622337   Gas used: 22673 (0.34%)

<Transaction '0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033'>
>>> test_contract.preiincrement(10)
Transaction sent: 0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 47
  test.preiincrement confirmed   Block: 13622338   Gas used: 22601 (0.34%)

<Transaction '0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05'>
```

Code Location:

Listing 17: LiquidityPool.sol (Line 313)

```
310 function processDepositQueue(uint limit) external nonReentrant {
311     (uint tokenPrice, bool stale, ) = _getTokenPriceAndStale();
312
313     for (uint i = 0; i < limit; i++) {
314       QueuedDeposit storage current = queuedDeposits[
  ↳ queuedDepositHead];
315       if (!_canProcess(current.depositInitiatedTime, lpParams.
  ↳ depositDelay, stale, queuedDepositHead)) {
316         return;
317       }
```

Listing 18: LiquidityPool.sol (Line 341)

```
340 function processWithdrawalQueue(uint limit) external nonReentrant
  ↳ {
341     for (uint i = 0; i < limit; i++) {
342       (uint totalTokensBurnable, uint tokenPriceWithFee, bool
  ↳ stale) = _getTotalBurnableTokens();
```

Listing 19: Other resources affected

```
GWAV: L#136
OptionGreekCache: L#332, 348, 352, 726, 816, 878, 913, 922, 1000
OptionMarket: L#236, 393, 415, 458, 744, 972, 1009
OptionToken: L#300, 509, 591, 605, 616
ShortCollateral: L#172
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use ++i instead of i++ to increment the value of a uint variable inside a loop. This is not applicable outside of loops.

Remediation plan:

SOLVED: The issue was fixed in commit 1e04d54b12c4faf0378b54b67f93d5de2b7c6e68.

3.10 (HAL-10) CACHING ARRAY LENGTH IN FOR LOOPS CAN SAVE GAS - INFORMATIONAL

Description:

Reading the length of the array at each iteration of the loop requires 6 gas (3 for mload and 3 to place memory_offset) onto the stack. Caching the length of the array on the stack saves about 3 gas per iteration.

Code Location:

Listing 20: OptionGreekCache.sol (Line 332)

```
330 liveBoards.push(board.id);
331
332 for (uint i = 0; i < strikes.length; i++) {
333   _addNewStrikeToStrikeCache(boardCache, strikes[i].id, strikes[i].strikePrice, strikes[i].skew);
334 }
335
336 _updateGlobalLastUpdatedAt();
```

Listing 21: Other resources affected

```
GWAV: L#136
OptionGreekCache: L#348, 352, 726, 816, 878, 913, 922, 1000
OptionMarket: L#236, 393, 415, 458, 972, 1009
OptionToken: L#300, 509, 591, 605
ShortCollateral: L#172
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider caching the length of the array. A sample code can be seen below:

Listing 22: OptionGreekCache.sol (Lines 878,879)

```
876 function _updateMaxIvVariance() internal {
877   uint maxIvVariance = boardCaches[liveBoards[0]].ivVariance;
878   uint256 boardlength = liveBoards.length;
879   for (uint i = 1; i < boardlength; i++) {
880     if (boardCaches[liveBoards[i]].ivVariance > maxIvVariance) {
```

Remediation plan:

SOLVED: The issue was fixed in commit 1e04d54b12c4faf0378b54b67f93d5de2b7c6e68.

THANK YOU FOR CHOOSING

HALBORN