Security Assessment

Lyra Finance

Aug 22nd, 2021

Table of Contents

Summary

This report has been prepared for Lyra Finance to discover issues and vulnerabilities in the source code of the Lyra Finance project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review techniques.

The auditing process pays special attention to the following considerations:

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

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Lyra Finance
Platform	Ethereum
Language	Solidity
Codebase	• https://github.com/lyra-finance/lyra • https://github.com/lyra-finance/lyra-protocol
Commit	• b564c77a7fc8bda8441131cc521b326d7bf56485 • 4961b6771f236478c1c29c171dca8909c7b7f0cb • 41eb27d6b4110ef81f40ccfe35598cd5392157d6 • 38656bca162e70b84bccbce3655400c96a369506

Audit Summary

Delivery Date	Aug 22, 2021
Audit Methodology	Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	4	0	0	4	0	0
Medium	9	0	0	5	0	4
Minor	7	0	0	4	1	2
Informational	13	0	0	6	0	7
Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
ICS	interfaces/ICollateralShort.sol	f92fd1e392854f611ba775a00a168a6bc2c34b95b45c55e7f2e21c77146ca724
IER	interfaces/IExchangeRates.sol	24457743b5fd59125e8f12e1822d9e9a9029d1edfdf604218f682980c398b070
IEE	interfaces/IExchanger.sol	49a7ddaad6185cadc332caf6140e7b4e2c5bc3083a73c3c327af9ba38249248a
ISR	interfaces/ISwapRouter.sol	dcef9e7c6ada5690995debe4dc47fd61d2cbe26e56e2da05caa3922b17707802
ISE	interfaces/ISynthetix.sol	255bd1fa402f01c29dc2e2bbe85ded41e97f59a3ef47a31cdc2c715c0f165d16
ITE	interfaces/ITestERC20.sol	0349123e2247b81547d4c674d54dab1b7b030cb9cc00c270c494ae87c414280b
ADD	openzeppelin-l2/Address.sol	048e01f326f6e5140f4010ea7341fa1a86fe5a8515be06115482b5a070e15d33
ERC	openzeppelin-l2/ERC1155.sol	6b2c891ab3994abe686f8f2fe669411ece10540c4eef90ac20449cd66c3b9d78
ER7	openzeppelin-l2/ERC721.sol	ccc2e0d63e106c2517bea83eae5df5c4c9069cf3c4e2c0d958ea75c2e7f34cf5
MSE	periphery/MultistepSwapper.sol	45caa697773fb0f8a05549f544db4e13e0c9f4a5e7ff27141e4c8e54690ad780
OMS	periphery/OptionMarketSafeSlippage.sol	7dc8cc0f38fbceee6a88048593b20d8302830f9fdd940a9904a3da173af6edce
SDM	synthetix/SafeDecimalMath.sol	9889434e5e4ddd98f63736b77128f8730f73692eac0163935b051327034a2fe3
SSD	synthetix/SignedSafeDecimalMath.sol	715486c62b9c6bf249ba295d3d31b40c58dd0e3b12b189ef217a722909bbc678
BSE	BlackScholes.sol	df45c38cbb86849a0d0c02a808312fffbe94836d1923ffc1c6b1839b6e384558
LCE	LiquidityCertificate.sol	9e25079a4cdb67ac34d1fe777af541d92b6da7b2d9053a9a3d3da5e62838487e
LPE	LiquidityPool.sol	8605a5ba36ae38617e8b0f21b1045a89a74bc04365ad20dc0c68b06cf7fa620e
LGE	LyraGlobals.sol	f57f73921eecae1313c55622b4550c219bd24add781e36639ea86193a5d48957
OGC	OptionGreekCache.sol	1eea6e9de8f14ed234732e3652b4ef74c4a260d60ca857db6808da7e68194e0f
OME	OptionMarket.sol	00e1ad6ccfcf84a80d5718044b61f5deff22731d91928ac4ede21bf8c776c9c6
OMP	OptionMarketPricer.sol	512b6a591dcf41ab916ca46dfd15eac5abd2df182e2dffa33e551bb755dc7dea

ID	File	SHA256 Checksum
OTE	OptionToken.sol	71f0e2fb48ca5e710d4443831c983c6c19dbe8793f7af14132410a2d84bb9f81
PHE	PoolHedger.sol	6a806e02d18796d0ba87c643903f68036f2d186b15be5033982a5494434f1a26
SCE	ShortCollateral.sol	882809b1c42abe153b1b5d658804cead8d2989a041572944284a163c89cecbde

Understandings

Overview

The Lyra Protocol is the collection of smart contracts that allow Liquidity Providers (LPs) to provide capital for traders to buy or sell options from/to the market. When a trader comes in to make a trade with the AMM, the size of the trade is quantified. The mechanism then uses the size of the trade to determine the volatility slippage to charge the trader. The volatility slippage has two components, the baseline IV (implied volatility) and the strike volatility ratio (per strike/expiry combination). The updated IV/strike volatility values are then used to create the volatility input into a Black Scholes model, to get a fair option price.

Dependencies

We assume the contracts `BlackScholes`, `LiquidityCertificate`, `LiquidityPool`, `LyraGlobals`, `OptionGreekCache`, `OptionMarket`, `OptionMarketPricer`, `OptionToken`, `PoolHedger`, `ShortCollateral`, `MultistepSwapper`, `OptionMarketSafeSlippage`, and `OptionMarketViewer` are deployed successfully and triggered correctly within the protocol.

There are a few depending injection contracts or addresses in the current project:

- `quoteAsset`, `baseAsset`, and `short` for the contract `LiquidityPool`;
- `synthetix`, `exchanger`, `exchangeRates`, `short`, and `collateralShort` for the contract `LyraGlobals`;
- `quoteAsset` and `baseAsset` for the contract `optionMarket`;
- `quoteAsset`, `baseAsset`, and `exchangeGlobals.short` for the contract `PoolHedger`;
- `quoteAsset` and `baseAsset` for the contract `ShortCollateral`;
- `swapRouter`, `synthetix`, `tokenIn`, and `swaps[i].tokenOut` for the contract `MultistepSwapper`;
- `quoteAsset` and `baseAsset` for the contract `OptionMarketSafeSlippage`.

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

Privilledged Functions

In the contract `LiquidityCertificate`, the role `liquidityPool` has the authority over the following functions:

- `LiquidityCertificate.mint()` to a new certificate and transfers it to `owner`;
- `LiquidityCertificate.setBurnableAt()` to set information of a burnable liquidity certificate;
- `LiquidityCertificate.burn()` to burn a liquidity certificate.

In the contract `LiquidityPool`, the role `optionMarket` has the authority over the following functions:

- `LiquidityPool.startRound()` to start a round;
- `LiquidityPool.lockQuote()` to lock quote when the system sells a put option;
- `LiquidityPool.lockBase()` to purchase and lock base when the system sells a call option;
- `LiquidityPool.freeQuoteCollateral()` to free quote when the system buys back a put from the user;
- `LiquidityPool.freeBase()` to sell base and free the proceeds of the sale;
- `LiquidityPool.boardLiquidation()` to manage collateral at the time of board liquidation and convert base sent here from the options market;
- `LiquidityPool.sendPremium()` to send the premium to a user who is selling an option to the pool.

The role `shortCollateral` has the authority over the following function:

- `LiquidityPool.sendReservedQuote()` to send reserved quote to the user.

The role `hedge` has the authority over the following function:

- `LiquidityPool.transferQuoteToHedge()` to send quote asset to the pool hedger.

In the contract `LyraGlobals`, the role `owner` has the authority over the following function:

- `LyraGlobals.setGlobals()` to set the globals that for all options markets;
- `LyraGlobals.setGlobalsForContract()` to set the globals that for a specific options market;
- `LyraGlobals.setPaused()` to pause or unpause the `LyraGlobals` contract;
- `LyraGlobals.setTradingCutoff()` to set the time at which the options market will cease trading before expiry;
- `LyraGlobals.setOptionPriceFeeCoefficient()` to set the options price fee coefficient for the options market;
- `LyraGlobals.setSpotPriceFeeCoefficient()` to set the spot price fee coefficient for the options market;
- `LyraGlobals.setVegaFeeCoefficient()` to set the vega fee coefficient for the options market;
- `LyraGlobals.setVegaNormFactor()` to set the vega normalisation factor for the options market;
- `LyraGlobals.setStandardSize()` to set the standard size for the options market;
- `LyraGlobals.setSkewAdjustmentFactor()` to set the skew adjustment factor for the options market;
- `LyraGlobals.setRateAndCarry()` to set the rate for the options market;
- `LyraGlobals.setMinDelta()` to set the minimum Delta that the options market will trade;

- `LyraGlobals.setVolatilityCutoff()` to set the minimum volatility option that the options market will trade;
- `LyraGlobals.setQuoteKey()` to set the `quoteKey` of the options market;
- `LyraGlobals.setBaseKey()` to set the `baseKey` of the options market;

In the contract `OptionGreekCache`, the role `owner` has the authority over the following function:

- `OptionGreekCache.setStaleCacheParameters()` to set cache parameters.

The role `optionMarket` has the authority over the following functions:

- `OptionGreekCache.addBoard()` to add a new options board;
- `OptionGreekCache.removeBoard()` to remove an options board;
- `OptionGreekCache.setBoardIv()` to modify an options board's base IV;
- `OptionGreekCache.setListingSkew()` to modify an options list's skew.

The role `optionPricer` has the authority over the following function:

- `OptionGreekCache.updateListingCacheAndGetPrice()` to upte the options listing cache to reflect the new exposure.

In the contract `OptionMarket`, the role `owner` has the authority over the following function:

- `transferOwnership()` to transfer the ownership of this contract;
- `setBoardFrozen()` to frozen/unfroze the state of an OptionBoard;
- `setBoardBaseIv()` to set the base IV for a board;
- `setListingSkew()` to set the skew for an option listing;
- `createOptionBoard()` to creates a new option board that contains option listings.

In the contract `OptionMarketPricer`, the role `OptionMarket` has the authority over the following function:

- `OptionMarketPricer.updateCacheAndGetTotalCost()` to update the listing cache and calculate the price when a trade is performed.

In the contract `OptionToken`, the role `optionMarket` has the authority over the following function:

- `OptionToken.mint()` to mint new option listing tokens;
- `OptionToken.burn()` to burn option tokens.

In the contract `PoolHedger`, the role `owner` has the authority over the following function:

- `PoolHedger.initShort()` to initialize a short position;

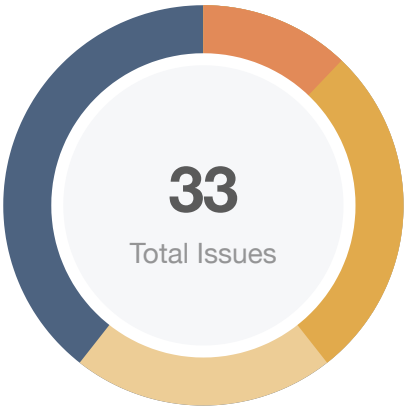
- `PoolHedger.reopenShort()` to reopen a short position;
- `PoolHedger.setShortBuffer()` to set the buffer ratio for short positions.

In the contract `ShortCollateral`, the role `optionMarket` has the authority over the following function:

- `ShortCollateral.sendQuoteCollateral()` to send quote asset collateral;
- `ShortCollateral.sendBaseCollateral()` to send base asset collateral;
- `ShortCollateral.sendToLP` to send base and quote assets to liquidity pool;
- `ShortCollateral.processSettle` to settle options after the board is liquidated.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

Findings



Critical	**0** (0.00%)
Major	**4** (12.12%)
Medium	**9** (27.27%)
Minor	**7** (21.21%)
Informational	**13** (39.39%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Incompatibility With Deflationary Tokens	Logical Issue	Minor	Acknowledged
GLOBAL-02	Lack of Access Control for Initialization	Logical Issue	Minor	Acknowledged
GLOBAL-03	Potential Integer Overflows	Mathematical Operations	Minor	Partially Resolved
GLOBAL-04	Unlocked Compiler Version	Language Specific	Informational	Resolved
GLOBAL-05	Triggering of Essential Functions	Logical Issue	Informational	Acknowledged
ADD-01	Incomplete Contract Check	Logical Issue	Informational	Acknowledged
BSE-01	Risk-free Rate Arbitrage	Logical Issue	Informational	Acknowledged
LCE-01	Centralization Risk	**Centralization / Privilege**	**Medium**	Acknowledged
LCE-02	Incorrect Event Emission	Logical Issue	Medium	Resolved
LGE-01	Centralization Risk	**Centralization / Privilege**	**Major**	Acknowledged
LGE-02	Lack of Input Validation	Volatile Code	Minor	Resolved
LGE-03	Reusable Code	Coding Style	Informational	Resolved
LGE-04	Mechanism of `rateAndInvalid()`	Logical Issue	Informational	Acknowledged

ID	Title	Category	Severity	Status
LPE-01	Centralization Risk	**Centralization / Privilege**	**Medium**	Acknowledged
LPE-02	Lack of Error Message	Coding Style	Informational	Resolved
LPE-03	Lack of Access Control	Logical Issue	Informational	Acknowledged
LPE-04	Not Enough Base Asset As Collateral	Logical Issue	Informational	Acknowledged
LPE-05	Check-Effect-Interaction Pattern Violation	Logical Issue	Minor	Resolved
MSE-01	Potential Reentrancy Risk	Logical Issue	Medium	Resolved
MSE-02	Unhandled Case for Swap	Logical Issue	Informational	Resolved
OGC-01	Centralization Risk	**Centralization / Privilege**	**Major**	Acknowledged
OGC-02	Incorrect Comment	Inconsistency	Informational	Resolved
OGC-03	Lack of Event Emissions for Significant Transactions	Logical Issue	Informational	Resolved
OME-01	Centralization Risk	**Centralization / Privilege**	**Major**	Acknowledged
OME-02	Potential Reentrancy Risk	Logical Issue	Medium	Resolved
OMP-01	Centralization Risk	**Centralization / Privilege**	**Medium**	Acknowledged
OMP-02	Calculation Not Matching The Eq.(17) In Whitepaper	Mathematical Operations	Informational	Resolved
OMS-01	Check-Effect-Interaction Pattern Violation	Logical Issue	Minor	Acknowledged
OTE-01	Centralization Risk	**Centralization / Privilege**	**Medium**	Acknowledged
PHE-01	Centralization Risk	**Centralization / Privilege**	**Major**	Acknowledged
PHE-02	Potential Reentrancy Risk	Logical Issue	Medium	Resolved

ID	Title	Category	Severity	Status
SCE-01	Centralization Risk	**Centralization / Privilege**	**Medium**	Acknowledged
SCE-02	Check-Effect-Interaction Pattern Violation	Logical Issue	Minor	Acknowledged

GLOBAL-01 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Logical Issue	Minor	Global	Acknowledged

Description

When transferring ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fees. For example, if a user deposits 100 deflationary tokens (with 10% transaction fees) to the `LiquidityPool` contract, only 90 tokens actually arrived in the contract. However, the user can still withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

According to our observations, deflationary tokens should not be used as `quoteAsset` and `baseAsset` in this project.

Reference: https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f

Recommendation

We advise the client to regulate the set of `quoteAsset` and `baseAsset` supported and to add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

[Lyra Finance Team]: Acceptable risk, as this version of contracts is only compatible with Synthetix Synths, which are not deflationary. But good to keep in mind.

GLOBAL-02 | Lack of Access Control for Initialization

Category	Severity	Location	Status
Logical Issue	Minor	Global	Acknowledged

Description

In the following contracts, the function `init()` can be called by anyone to initialize the contracts:

- `LiquidityCertificate`
- `LiquidityPool`
- `MultistepSwapper`
- `OptionGreekCache`
- `OptionMarket`
- `OptionMarketPricer`
- `OptionMarketSafeSlippage`
- `OptionToken`
- `PoolHedger`
- `ShortCollateral`

Although the project deployer can discard incorrectly initialized contracts, it might still bring errors if the deployment is not properly processed. One of the possible scenarios is described as below:

1. The deployer writes a script to deploy and initialize the contract.
2. The attacker noticed the deployment and initialized the contract before the initialization by the deployer is committed.
3. The deployment script mistakenly ignores the error of initializing the contract (because `initialized` is `true` now, the transaction of calling `init()` will be reverted), and continues executing other transactions in the script.

In this way, the attacker can inject suspicious addresses into the contracts.

Recommendation

We recommend adding proper access control to the `init()` function in the aforementioned contracts or checking the status of initialization in the deployment process.

Alleviation

[**Lyra Finance Team**]: Accepted risk. Deployment scripts will revert and stop if the transaction is frontrun.

GLOBAL-03 | Potential Integer Overflows

Category	Severity	Location	Status
Mathematical Operations	Minor	Global	Partially Resolved

Description

Checks for integer overflows are not always performed in the project. For example, in OptionMarketPricer.sol L71:

```
71        return (boardBaseIv + orderMoveBaseIv, listing.skew + orderMoveSkew);
```

We understand that in most cases integer overflows will never happen because the parameters will always be within reasonable ranges. However, ranges of some parameters are not explicitly provided in the contracts, so the possibility of integer overflows, especially for operations with inputs of `public` functions, should still be considered.

Recommendation

We recommend always applying libraries like `SafeMath` or checking integer overflows in integer operations unless ranges of relevant parameters are explicitly checked.

Alleviation

The Lyra Finance team heeded our advice and partially resolved this issue in the commits `41eb27d6b4110ef81f40ccfe35598cd5392157d6` and `38656bca162e70b84bccbce3655400c96a369506`.

GLOBAL-04 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	Global	Resolved

Description

The project has an unlocked compiler version `^0.7.6`. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging as compiler-specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.7.6`, the contract should contain the following line:

```
pragma solidity 0.7.6;
```

Alleviation

The Lyra Finance team heeded our advice and partially resolved this issue. The fixing is reflected in the commit `41eb27d6b4110ef81f40ccfe35598cd5392157d6`.

GLOBAL-05 | Triggering of Essential Functions

Category	Severity	Location	Status
Logical Issue	Informational	Global	Acknowledged

Description

Triggering of some essential functions is not implemented within the smart contracts. For example,

- `PoolHedger.hedgeDelta()` to hedge deltas;
- `OptionMarket.liquidateExpiredBoard()` to liquidate an expired board;
- `LiquidityPool.exchangeBase()` to update purchase or sell base assets;
- `LiquidityPool.endRound()` to end a round.

Also, access to these functions is not restricted.

The timing of triggering these functions has a direct influence on the gain/loss of the market participants because some important factors like the spot price are time-dependent.

We assume all essential functions in the trading flow are properly triggered at a reasonable timing.

Alleviation

[**Lyra Finance Team**]: These functions have been intentionally left public, so that anyone can call them whenever it is appropriate.

ADD-01 | Incomplete Contract Check

Category	Severity	Location	Status
Logical Issue	Informational	openzeppelin-l2/Address.sol: 26	Acknowledged

Description

The function `isContract` is supposed to verify if the input address is a contract address or a real user address. As described in the contract comments, this is not a reliable check, because if the input is an address of a contract in construction, `size > 0` could be false and the function will return false, which is not correct.

```
33      assembly {
34        size := extcodesize(account)
35      }
36      return size > 0;
```

Recommendation

It is recommended to check by `msg.sender != tx.origin` to include all the scenarios when function calls are invoked by a contract address.

Alleviation

N/A

BSE-01 | Risk-free Rate Arbitrage

Category	Severity	Location	Status
Logical Issue	Informational	BlackScholes.sol: 267	Acknowledged

Description

As discussed in our last meeting, the risk-free rate and carry cost is set as 0 in default. The discrepancy between the risk-free rate (sUSD) in the project and the real risk-free rate (USD) may create risk-free arbitrage opportunities since these two assets are equivalent.

Alleviation

[**Lyra Finance Team**]: If there is a new consensus about what the risk free rate is, the value can be updated. Besides that, it has a minimal effect on the value for shorter dated options - the contracts will only be operating on < 28 day expiry options to begin with.

LCE-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Medium**	LiquidityCertificate.sol: 111, 132, 154	Acknowledged

Description

In the contract `LiquidityCertificate`, the role `liquidityPool` has the authority over the following functions:

- `LiquidityCertificate.mint()` to mint a new certificate and transfer it to `owner`;
- `LiquidityCertificate.setBurnableAt()` to set information of a burnable liquidity certificate;
- `LiquidityCertificate.burn()` to burn a liquidity certificate.

Any missetting or compromise to the `liquidityPool` account may allow the hacker to take advantage of this and manipulate the options market.

Recommendation

We advise the client to carefully manage the `liquidityPool` account's private key or make sure it is set to the correct contract to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Lyra Finance Team**]: Acceptable risk, as the `liquidityPool` field will always be initialized to the `LiquidityPool` contract for the quote/base pair, and cannot be changed

LCE-02 | Incorrect Event Emission

Category	Severity	Location	Status
Logical Issue	Medium	LiquidityCertificate.sol: 167	Resolved

Description

In the function `LiquidityCertificate.split()`, the event `CertificateDataModified` is emitted twice to notify the public the two liquidity certificate after split. The first `CertificateDataModified` event should log the status of the old certificate after split. However, it uses `certData.liquidity`, which is not updated, as its second parameter:

```
200      emit CertificateDataModified(certificateId, certData.liquidity,
certData.enteredAt, certData.burnableAt);
```

Therefore, this event does not log the correct update.

Recommendation

We recommend using `oldCertLiquidity`, which is the updated liquidity value, as the second parameter of the event in the aforementioned line.

Alleviation

The client heeded our advice and resolved this issue. The fixing is reflected in the commit `4961b6771f236478c1c29c171dca8909c7b7f0cb`.

LGE-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	Major	LyraGlobals.sol: 97, 120, 146, 158, 170, 181, 192, 203, 214, 225, 237, 249, 261, 272, 283	Acknowledged

Description

In the contract `LyraGlobals`, the role `owner` has the authority over the following function:

- `LyraGlobals.setGlobals()` to set the globals that for all options markets;
- `LyraGlobals.setGlobalsForContract()` to set the globals that for a specific options market;
- `LyraGlobals.setPaused()` to pause or unpause the `LyraGlobals` contract;
- `LyraGlobals.setTradingCutoff()` to set the time at which the options market will cease trading before expiry;
- `LyraGlobals.setOptionPriceFeeCoefficient()` to set the options price fee coefficient for the options market;
- `LyraGlobals.setSpotPriceFeeCoefficient()` to set the spot price fee coefficient for the options market;
- `LyraGlobals.setVegaFeeCoefficient()` to set the vega fee coefficient for the options market;
- `LyraGlobals.setVegaNormFactor()` to set the vega normalisation factor for the options market;
- `LyraGlobals.setStandardSize()` to set the standard size for the options market;
- `LyraGlobals.setSkewAdjustmentFactor()` to set the skew adjustment factor for the options market;
- `LyraGlobals.setRateAndCarry()` to set the rate for the options market;
- `LyraGlobals.setMinDelta()` to set the minimum Delta that the options market will trade;
- `LyraGlobals.setVolatilityCutoff()` to set the minimum volatility option that the options market will trade;
- `LyraGlobals.setQuoteKey()` to set the `quoteKey` of the options market;
- `LyraGlobals.setBaseKey()` to set the `baseKey` of the options market;

Any compromise to the `owner` account may allow the hacker to take advantage of this and manipulate the options market.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be

improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multi-signature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Lyra Finance Team**]: `owner` will be set to a MultiSig with multiple signers before any funds enter the contract.

LGE-02 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	LyraGlobals.sol: 170, 181, 192, 203, 214, 261	Resolved

Description

The following functions update important parameters in fee or option price calculations:

- `LyraGlobals.setOptionPriceFeeCoefficient()`
- `LyraGlobals.setSpotPriceFeeCoefficient()`
- `LyraGlobals.setVegaFeeCoefficient()`
- `LyraGlobals.setVegaNormFactor()`
- `LyraGlobals.setStandardSize()`
- `LyraGlobals.setVolatilityCutoff()`

Input validations or checks should be performed to avoid human errors like setting parameters with incorrect decimals.

Recommendation

We recommend adding explicit validation to check the input parameters.

Alleviation

The Lyra Finance team heeded our advice and resolved this issue in the commit `41eb27d6b4110ef81f40ccfe35598cd5392157d6`.

LGE-03 | Reusable Code

Category	Severity	Location	Status
Coding Style	Informational	LyraGlobals.sol: 318, 340, 359, 404	Resolved

Description

The exactly the same code is implemented in the functions `LyraGlobals.getPricingGlobals()` and `LyraGlobals.getGreekCacheGlobals()`, `LyraGlobals.getExchangeGlobals()` and `LyraGlobals.getGlobalsForOptionTrade()` to check if the contract has been paused:

```
    require(!isPaused, "contracts are paused");
```

Recommendation

We recommend implementing a modifier to perform the check and use the modifier in the aforementioned functions.

Alleviation

The Lyra Finance team heeded our advice and resolved this issue in the commit `41eb27d6b4110ef81f40ccfe35598cd5392157d6`.

LGE-04 | Mechanism of `rateAndInvalid()`

Category	Severity	Location	Status
Logical Issue	Informational	LyraGlobals.sol: 307	Acknowledged

Description

The function `exchangeRates.rateAndInvalid()` is an external dependency providing the spot price of an asset to the contract. If the provided price can be manipulated to some extent, attackers might take advantage of this and manipulate the result of liquidation and options settles. Even if the cost of manipulating price is high, because all positions are viewable by the public it should be considered if attackers would pay a high cost to manipulate the price and gain a higher profit from the options market.

We would like to check with the Lyra Finance Team if the result of `exchangeRates.rateAndInvalid()` can always be trusted.

Alleviation

[**Lyra Finance Team**]: This price feed will come from Synthetix, which in turns gets the price feed from Chainlink.

LPE-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	Medium	LiquidityPool.sol: 334, 407, 420, 461, 484, 498, 520, 544, 618	Acknowledged

Description

In the contract `LiquidityPool`, the role `optionMarket` has the authority over the following functions:

- `LiquidityPool.startRound()` to start a round;
- `LiquidityPool.lockQuote()` to lock quote when the system sells a put option;
- `LiquidityPool.lockBase()` to purchase and lock base when the system sells a call option;
- `LiquidityPool.freeQuoteCollateral()` to free quote when the system buys back a put from the user;
- `LiquidityPool.freeBase()` to sell base and free the proceeds of the sale;
- `LiquidityPool.boardLiquidation()` to manage collateral at the time of board liquidation and convert base sent here from the options market;
- `LiquidityPool.sendPremium()` to send the premium to a user who is selling an option to the pool.

The role `shortCollateral` has the authority over the following function:

- `LiquidityPool.sendReservedQuote()` to send reserved quote to the user.

The role `poolHedger` has the authority over the following function:

- `LiquidityPool.transferQuoteToHedge()` to send quote asset to the pool hedger.

Any missetting or compromise to the `optionMarket`, `shortCollateral` or `poolHedger` account may allow the hacker to take advantage of this and manipulate the options market.

Recommendation

We advise the client to carefully manage the `optionMarket`, `shortCollateral` and `poolHedger` accounts' private keys or set them to the correct contracts to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Lyra Finance Team**]: All three roles (`liquidityPool`, `shortCollateral` and `poolHedger`) will be initialized to smart contracts, and cannot be changed.

LPE-02 | Lack of Error Message

Category	Severity	Location	Status
Coding Style	Informational	LiquidityPool.sol: 408, 485	Resolved

Description

Error messages in the following `require` checks can indicate the desired operation failures to users or relay essential warnings:

- `require(amount <= freeCollatLiq);`
- `require(amountBase <= lockedCollateral.base);`

Recommendation

We recommend providing error message strings for the aforementioned `require` checks.

Alleviation

The Lyra Finance team heeded our advice and resolved this issue. The fixing is reflected in the commit `4961b6771f236478c1c29c171dca8909c7b7f0cb`.

LPE-03 | Lack of Access Control

Category	Severity	Location	Status
Logical Issue	Informational	LiquidityPool.sol: 362	Acknowledged

Description

The function `LiquidityPool.exchangeBase()` can be called by anyone to trigger `exchangeGlobals.synthetix.exchange()` when `baseAsset.balanceOf(address(this)) != lockedCollateral.base`.

If `exchangeGlobals.synthetix.exchange()` requires exchange fees, the contract might lose assets if this function is frequently triggered.

We would like to confirm with the client if this would be an issue in the project.

Alleviation

[Lyra Finance Team]: This is intentional. We want the contract to always be fully collateralized. However, the gas limit on Optimism does not allow the exchange to occur in the same transaction as opening/closing a long call position. Thus this function can be public.

LPE-04 | Not Enough Base Asset As Collateral

Category	Severity	Location	Status
Logical Issue	Informational	LiquidityPool.sol: 391~393	Acknowledged

Description

In the function `LiquidityPool.exchangeBase()`, when the `currentBaseBalance` of the `LiquidityPool` is less than `lockedCollateral.base`, the pool would exchange quote asset to base asset to bring up the collateral to its required amount. However, when there is not enough quote asset, it is no longer possible to acquire enough base asset to cover the collateral since the newly deposited quote asset belongs to the category of `queuedQuoteFunds` which is not available to `LiquidityPool.exchangeBase()`. In a situation where there is no adequate base asset to be reserved as collateral, the AMM, in a sense, is naked shorting the base asset. Furthermore, such a position of the liquidity pool will also impact the delta hedging strategy.

We would like to check with the team whether this risk is acceptable.

Alleviation

[**Lyra Finance Team**]: We acknowledge and accept the risk. There will be keeper bots constantly running to run this function.

LPE-05 | Check-Effect-Interaction Pattern Violation

Category	Severity	Location	Status
Logical Issue	Minor	LiquidityPool.sol: 152, 227	Resolved

Description

The Solidity documentation suggests that a smart contract should follow the `Checks-Effects-Interactions` pattern. However, the functions at the aforementioned lines violate the `Checks-Effects-Interactions` pattern by having external calls (Interactions) before event emissions (Effects).

Recommendation

We recommend adopting the `Checks-Effects-Interactions` pattern in the aforementioned functions by, for example, emitting events before processing external calls.

Alleviation

The Lyra Finance team heeded our advice and resolved this issue in the commit `41eb27d6b4110ef81f40ccfe35598cd5392157d6`.

MSE-01 | Potential Reentrancy Risk

Category	Severity	Location	Status
Logical Issue	Medium	periphery/MultistepSwapper.sol: 52	Resolved

Description

The function `MultistepSwapper.swap()` has state updates after the external calls, so it is vulnerable to potential reentrancy attacks.

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned function to prevent reentrancy attack.

Alleviation

The Lyra Finance team heeded our advice and resolved this issue in the commit `41eb27d6b4110ef81f40ccfe35598cd5392157d6`.

MSE-02 | Unhandled Case for Swap

Category	Severity	Location	Status
Logical Issue	Informational	periphery/MultistepSwapper.sol: 89	Resolved

Description

The function `MultistepSwapper.swap()` swaps one token for another token based on `swaps[i].swapType`. However, it is possible that `swaps[i].swapType` is neither `SwapType.Synthetix` nor `SwapType.Uniswap` due to an incorrect input.

It does not affect the transaction result because the user is guaranteed to receive `amountOutMinimum` `tokenOut` because, otherwise, the transaction will be reverted, but it is still recommended to handle the corner case. For example,

1. User A sent 10 CToken to the contract by mistake.
2. User B triggers `MultistepSwapper.swap()` and is planning to swap 100 AToken to 1000 CToken with path AToken -> BToken -> CToken and `amountOutMinimum` 0 (many people did this).
3. 100 AToken is swapped for 10 BToken.
4. 10 BToken should be swapped for 1000 CToken, but the swap is not performed because `swaps[1].swapType` is mistakenly set. However, `tokenIn` is still set to CToken while `amountOut` remains the value in the previous swap, which is 10.
5. The contract sends 10 CToken to user B because it happens to have 10 CToken.

Although the transaction is successfully committed, the result is not expected.

Recommendation

We recommend checking `swaps[i].swapType` to make sure it is either `SwapType.Synthetix` or `SwapType.Uniswap`.

Alleviation

The Lyra Finance team heeded our advice and resolved this issue in the commit `41eb27d6b4110ef81f40ccfe35598cd5392157d6`.

OGC-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	Major	OptionGreekCache.sol: 111, 138, 164, 190, 202, 214~216, 327	Acknowledged

Description

In the contract `OptionGreekCache`, the role `owner` has the authority over the following function:

- `OptionGreekCache.setStaleCacheParameters()` to set cache parameters.

The role `optionMarket` has the authority over the following functions:

- `OptionGreekCache.addBoard()` to add a new options board;
- `OptionGreekCache.removeBoard()` to remove an options board;
- `OptionGreekCache.setBoardIv()` to modify an options board's base IV;
- `OptionGreekCache.setListingSkew()` to modify an options list's skew;
- `OptionGreekCache.addListingToBoard()` to add a new `listing` to the `listingCaches` and the `listingId` to the `boardCache`.

The role `optionPricer` has the authority over the following function:

- `OptionGreekCache.updateListingCacheAndGetPrice()` to upte the options listing cache to reflect the new exposure.

Any missetting or compromise to the `owner`, `optionMarket` and `optionPricer` accounts may allow the hacker to take advantage of this and manipulate the options market.

Recommendation

We advise the client to carefully manage the `owner`, `optionMarket` and `optionPricer` accounts' private key or make sure they are set to the correct contracts to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g. 48 hours, for awareness on privileged operations;

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Lyra Finance Team**]: `owner` will be set to a MultiSig with multiple signers before any funds enter the contract. `optionMarket` and `optionPricer` will be initialized to smart contracts, and cannot be changed.

OGC-02 | Incorrect Comment

Category	Severity	Location	Status
Inconsistency	Informational	OptionGreekCache.sol: 202	Resolved

Description

The comment that accompanies the `OptionGreekCache.setListingSkew()` function states that the function modifies an options board's base IV yet it updates an options listing's skew instead.

Recommendation

We advise that the comment is properly adjusted to reflect the true functionality of the `OptionGreekCache.setListingSkew()` function.

Alleviation

The Lyra Finance team heeded our advice and resolved this issue. The fixing is reflected in the commit `4961b6771f236478c1c29c171dca8909c7b7f0cb`.

OGC-03 | Lack of Event Emissions for Significant Transactions

Category	Severity	Location	Status
Logical Issue	Informational	OptionGreekCache.sol: 111, 138, 190, 202	Resolved

Description

The following functions updates options or relevant parameters of the options market:

- `OptionGreekCache.setStaleCacheParameters()`
- `OptionGreekCache.addBoard()`
- `OptionGreekCache.setBoardIv()`
- `OptionGreekCache.setListingSkew()`

Considering updates made by these functions are important to the market participants, events logging the updates should be emitted to notify the public.

Recommendation

We recommend emitting events to log the updates when the aforementioned functions are triggered.

Alleviation

[**Lyra Finance Team**]: Added `StaleCacheParametersUpdated` for `OptionGreekCache.setStaleCacheParameters()` in the commit `41eb27d6b4110ef81f40ccfe35598cd5392157d6`.

For other functions, when they are updated there are events emitted in the `OptionMarket` contract which can be mapped exactly to the values in the `OptionGreekCache` contract.

OME-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Major**	OptionMarket.sol: 145, 156, 168, 181, 200, 242	Acknowledged

Description

In the contract `OptionMarket`, the role `owner` has the authority over the following function:

- `OptionMarket.transferOwnership()` to transfer the ownership of this contract;
- `OptionMarket.setBoardFrozen()` to freeze/unfreeze the state of an OptionBoard;
- `OptionMarket.setBoardBaseIv()` to set the base IV for a board;
- `OptionMarket.setListingSkew()` to set the skew for an option listing;
- `OptionMarket.createOptionBoard()` to creates a new option board that contains option listings;
- `OptionMarket.addListingToBoard()` to add a listing to an existing board.

Any missetting or compromise to the `owner` account may allow the hacker to take advantage of this and manipulate the options market.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Lyra Finance Team**]: `owner` will be set to a MultiSig with multiple signers before any funds enter the contract.

OME-02 | Potential Reentrancy Risk

Category	Severity	Location	Status
Logical Issue	Medium	OptionMarket.sol: 308, 379	Resolved

Description

The functions `OptionMarket.openPosition()` and `OptionMarket.closePosition()` have state updates and event emissions after the external calls, so they are vulnerable to potential reentrancy attacks.

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[**Lyra Finance Team**]: As the only external contract that could feasibly do a re-entrancy attack on `openPosition`/`closePosition` are trusted Synthetix contracts, this is an accepted risk. We are also protected by gasLimits on Optimism where the contracts will be deployed.

OMP-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Medium**	OptionMarketPricer.sol: 85	Acknowledged

Description

In the contract `OptionMarketPricer`, the role `OptionMarket` has the authority over the following function:

- `OptionMarketPricer.updateCacheAndGetTotalCost()` to update the listing cache and calculate the price when a trade is performed.

Any missetting or compromise to the `OptionMarket` account may allow the hacker to take advantage of this and manipulate the options market.

Recommendation

We advise the client to carefully manage the `OptionMarket` account's private key or set it to the correct contract to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Lyra Finance Team**]: The `optionMarket` address will be initialized to a smart contract, and cannot be changed.

OMP-02 | Calculation Not Matching The Eq.(17) In Whitepaper

Category	Severity	Location	Status
Mathematical Operations	Informational	OptionMarketPricer.sol: 169~172	Resolved

Description

In the function `OptionMarketPricer.getVegaUtil()`, the calculation of `normVol` differs from the Eq(17) in the whitepaper by the coefficients `100` and `pricingGlobals.vegaNormFactor`. Since we don't know the exact value of `pricingGlobals.vegaNormFactor` we would like to check with the team whether the calculation is correct.

Alleviation

[**Lyra Finance Team**]: This is correct, and has to do with the representation of the number in solidity as compared to the whitepaper. `pricingGlobals.vegaNormFactor` will be initialized to 0.2, which is the 20% figure in the whitepaper.

OMS-01 | Check-Effect-Interaction Pattern Violation

Category	Severity	Location	Status
Logical Issue	Minor	periphery/OptionMarketSafeSlippage.sol: 69, 114	Acknowledged

Description

The Solidity documentation suggests that a smart contract should follow the `Checks-Effects-Interactions` pattern. However, the functions at the aforementioned lines violate the `Checks-Effects-Interactions` pattern by having external calls (Interactions) before event emissions (Effects).

Recommendation

We recommend adopting the `Checks-Effects-Interactions` pattern in the aforementioned functions by, for example, emitting events before processing external calls.

Alleviation

N/A

OTE-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Medium**	OptionToken.sol: 35, 46, 62	Acknowledged

Description

In the contract `OptionToken`, the role `optionMarket` has the authority over the following function:

- `OptionToken.mint()` to mint new option listing tokens;
- `OptionToken.burn()` to burn option tokens.

The role `owner` has the authority over the function `OptionToken.setURI()` to set a new URI.

Any missetting or compromise to the `optionMarket` or `owner` account may allow the hacker to take advantage of this and manipulate the options market.

Recommendation

We advise the client to carefully manage the `optionMarket` and `owner` accounts' private keys or set them to the correct contracts to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Lyra Finance Team**]: The `optionMarket` address will be initialized to a smart contract, and cannot be changed.

PHE-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Major**	PoolHedger.sol: 55, 97, 111	Acknowledged

Description

In the contract `PoolHedger`, the role `owner` has the authority over the following function:

- `PoolHedger.initShort()` to initialize a short position;
- `PoolHedger.reopenShort()` to reopen a short position;
- `PoolHedger.setShortBuffer()` to set the buffer ratio for short positions.

Any compromise to the `owner` account may allow the hacker to take advantage of this and manipulate the options market.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Lyra Finance Team**]: `owner` will be set to a MultiSig with multiple signers before any funds enter the contract.

PHE-02 | Potential Reentrancy Risk

Category	Severity	Location	Status
Logical Issue	Medium	PoolHedger.sol: 111	Resolved

Description

The functions `PoolHedger.reopenShort()` has state updates and event emissions after the external calls, so they are vulnerable to potential reentrancy attacks.

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[**Lyra Finance Team**]: The `loans()` in the interface `ICollateralShort` is the same as the mapping. The `Loan` object contains the same fields as what is returned by the `loans()` function, so they behave the same way. The intention is to use the public `loans` mapping.

Reference: https://github.com/Synthetixio/synthetix/blob/meb-short-refactor/contracts/Collateral.sol#L40

SCE-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Medium**	ShortCollateral.sol: 63, 78, 93, 124	Acknowledged

Description

In the contract `ShortCollateral`, the role `optionMarket` has the authority over the following function:

- `ShortCollateral.sendQuoteCollateral()` to send quote asset collateral;
- `ShortCollateral.sendBaseCollateral()` to send base asset collateral;
- `ShortCollateral.sendToLP()` to send base and quote assets to liquidity pool;
- `ShortCollateral.processSettle()` to settle options after the board is liquidated.

Any missetting or compromise to the `optionMarket` account may allow the hacker to take advantage of this and manipulate the options market.

Recommendation

We advise the client to carefully manage the `optionMarket` account's private key or set it to the correct contract to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**Lyra Finance Team**]: The `optionMarket` address will be initialized to a smart contract, and cannot be changed.

SCE-02 | Check-Effect-Interaction Pattern Violation

Category	Severity	Location	Status
Logical Issue	Minor	ShortCollateral.sol: 63, 78, 93, 124	Acknowledged

Description

The Solidity documentation suggests that a smart contract should follow the `Checks-Effects-Interactions` pattern. However, the functions at the aforementioned lines violate the `Checks-Effects-Interactions` pattern by having external calls (Interactions) before event emissions (Effects).

Recommendation

We recommend adopting the `Checks-Effects-Interactions` pattern in the aforementioned functions by, for example, emitting events before processing external calls.

Alleviation

N/A

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

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

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.