SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Lyra

Prepared For:	Lyra
Prepared By:	Sherlock
Lead Security Experts:	Gerhard Wagner, 0xRajeev
Security Researchers:	Riley Holterhus, Mukesh Jaiswal, 0xAsm0d3us, Secureum Bootcamp participants
Prepared On:	June 27th, 2022

Introduction

“Lyra is an options trading protocol accessing the scalability of Layer 2 Ethereum to provide a robust, lightning-fast and reliable trading experience.”

This report is a CASEfile for Lyra Protocol that was prepared by Sherlock Watsons Gerhard Wagner and 0xRajeev, with assistance from Riley Holterhus, Mukesh Jaiswal, 0xAsm0d3us and Secureum participants, in the context of Secureum CASE (Collaborative Assessment & Security Evaluation) during 5-18 May, 2022.

Scope

Branch: Avalon (https://github.com/lyra-finance/lyra-protocol/tree/avalon)
Commit: 6635f6005f68d46dfe60b92e34f372851c536bfd (https://github.com/lyra-finance/lyra-protocol/pull/2/commits/6635f6005f68d46dfe60b92e34f372851c536bfd)
Contracts:

- LiquidityPool.sol
- OptionMarket.sol
- PoolHedger.sol
- LiquidityTokens.sol
- OptionMarketPricer.sol
- ShortCollateral.sol
- OptionGreekCache.sol
- OptionToken.sol
- SynthetixAdapter.sol
- synthetix/AbstractOwned.sol
- synthetix/DecimalMath.sol
- synthetix/Owned.sol
- synthetix/OwnedUpgradeable.sol
- synthetix/SignedDecimalMath.sol
- lib/BlackScholes.sol
- lib/FixedPointMathLib.sol
- lib/GWAV.sol
- lib/SimpleInitializeable.sol
- periphery/Wrapper/BasicOptionMarketWrapper.sol
- periphery/Wrapper/OptionMarketWrapper.sol
- periphery/Wrapper/OptionMarketWrapperWithSwaps.sol

For this review, it’s also worth noting that the Synthetix components were out of scope. And the keeper mechanism along with incentives to keep the protocol state updated were not reviewed.

SHERLOCK

Code Attributes

Code Complexity: Solidity Metrics report

Test Suite

Test coverage: Very good
Quality of tests: The quality of tests were high, although it is recommended that the protocol team extend the test suite to look for edge cases or unusual situations that might reveal flaws in economic assumptions and/or implementation.

Blockchain: Ethereum
L2s: Optimism

Tokens used: sUSD, Synthetix Synths

Findings

Each issue has an assigned severity:

- Informational issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgement as to whether to address such issues.
- Low issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Total Issues

Informational	Low	Medium	High
9	24	9	0

Issue M-01

Failing withdrawals could permanently freeze the queue

Summary
Withdrawals are done in a two-step process. First, a user starts the withdrawal and if the initial validation succeeds, the withdrawal is added to queuedWithdrawals. After a withdrawal delay, a user can then call processWithdrawalQueue. This function does not allow a user to process the withdrawal based on an id but processes withdrawals in chronological order. So if a user wants to withdraw funds, all other withdrawals that were initiated before need to be processed first. This can be problematic if one of the withdrawals fails for an unforeseen reason because then, the withdrawal queue is stuck and no other withdrawals after the failing one can take place.

Deposits have the same two-step process and the design of the queue could lead to a similar issue where a processing failure for a deposit could freeze the deposit queue.

Severity
Medium

Vulnerability Detail
The withdrawal queue returns without processing the withdrawal if totalTokensBurnable is 0. The withdrawal can not be processed until the condition changes. There could be other undiscovered issues that might cause failures or reverts, in which case the withdrawal queue could become stuck as well.

Impact
The withdrawal queue could become permanently stuck and users will not be able to withdraw their funds anymore from the LiquidityPool contract. This will cause a DoS.

Code Snippet
processWithdrawalQueue

Tool used
Manual Review

Recommendation
It is recommended to change the design of the deposit and withdrawal process so that funds can be processed without a queue. Users should be able to process their deposits or withdrawals regardless of a specific order.

Lyra Comment
A queue is necessary due to the potential of the pool filling up blocking withdrawals. In those scenarios, those who came first should be able to withdraw their funds first. We've endeavoured to make sure that the queue processing is as robust as possible.

In the case of some failings in circuit breakers etc. the guardian has the ability to process the queue.

Sherlock Comment
Sounds reasonable.

Issue M-02

Uninitialized Strike is allowed to be used

Summary

A Strike can either be added when a board is added with createOptionBoard or later on with addStrikeToBoard. When a Strike is initialized, its id is derived from the nextStrikeId variable, which is incremented by one for every Strike that is added. nextStrikeId is set to 1 when the OptionMarket is created. Strike ids are checked for their validity by comparing Strike.id with the id from the user input that is used to retrieve the struct from the strikes mapping. This check is insufficient for Strike id 0 because, by default, Strike.id is 0, so the check can be bypassed and the Strike is valid even though it is uninitialized.

Severity

Medium

Vulnerability Detail

The code in _composeTrade performs checks to make sure it only uses a valid Strike. An invalid Strike can be submitted, though with the id 0. The data structure has not been initialized as the nextStrikeId starts at 1. So using the id 0 bypasses the check and then composes TradeParameters based on the uninitialized Strike and subsequently OptionBoard. While during testing, it was not possible to create a position with the uninitialized Strike it might be possible under certain circumstances.

The setStrikeSkew function has a similar issue and the Strike id 0 is not rejected.

Impact

The owner could accidentally call setStrikeSkew with id 0. With the strike skew set, it might be possible to create or update a position with invalid Strike values.

Code Snippet

_composeTrade setStrikeSkew

Tool used

Manual Review

Recommendation

Set nextStrikeId to 0 instead of 1 at contract creation, or add an explicit check to make sure strikeId 0 is considered invalid and rejected in all the functions listed in the Code Snippet section.

Lyra Comment

Checks have been added to ensure strikeId/boardId of 0 cannot be modified even by admins.

Sherlock Comment
Looks reasonable.

Issue M-03

Accurate price at board expiry is not ensured

Summary
The expiration date for a board can be set by settleExpiredBoard function. It queries the spot price from the Synthetix contracts and sets the boardToPriceAtExpiry assuming the price is close to the expiration time. There is no on-chain mechanism currently to ensure that this is the case. The team has indicated that they will initially take on the responsibility of calling the function at board expiration and that they will implement a mechanism to incentivize users (i.e. keepers) to call the function.

Severity
Medium

Vulnerability Detail
N/A

Impact
If no one calls settleExpiredBoard function at board expiration and it’s only later called after a longer period has been passed, then it is possible that the current spot price deviates significantly from the one at board expiration. The settlement of the expired board could lead to financial loss for some users.

Code Snippet
settleExpiredBoard

Tool used
Manual Review

Recommendation
It is recommended to implement an on-chain mechanism that ensures that the boardToPriceAtExpiry is always accurate and as close to the expiration time as possible.

Lyra Comment
Keepers will be run to ensure this function is called as soon as possible after expiry. This feature can be added in the future, there is no simple way to get this data on-chain. In the case of an outage, there most likely will not be a chainlink feed that can be read from to get accurate data.

Sherlock Comment
Sounds reasonable.

SHERLOCK

Issue M-04

Updating BoardCachedGreeks does not check if a board is expired

Summary

A function in the OptionGreekCache contract that updates the cached greeks for an OptionBoardCache misses expiration checks for the board.

Severity

Medium

Vulnerability Detail

The _updateBoardCachedGreeks function only checks if the board id is 0 but misses a check if the board is expired.

Impact

Various parameters of the GlobalCache and the OptionBoardCache can be updated based on the expired board values.

Code Snippet

updateBoardCachedGreeks _updateBoardCachedGreeks

Tool used

Manual Review

Recommendation

Include a check that ensures that a board has not expired before the greeks of the OptionBoardCache are updated.

Lyra Comment

This has been resolved.

https://github.com/lyra-finance/lyra-protocol/blob/avalon/contracts/OptionGreekCache.sol#L727-L729

Sherlock Comment

Looks reasonable.

Issue M-05

Interchanged from-to addresses cause loss of user funds

Summary

The from and to addresses are interchanged in _returnBase and _returnQuote leading to an excess amount of base/quote asset in the contract being transferred again from the user (to contract) instead of being sent back to the user (from the contract).

Severity

Medium

Vulnerability Detail

When a user closes or force closes their position via OptionMarketWrapperWithSwaps, the user's excess base assets in the contract are expected to be sent back to the user via a call to _returnBase. However, when _returnBase makes a call to _transferAsset, it incorrectly uses msg.sender as the from address and address(this) as the to address, instead of the other way around.

A similar issue exists in _returnQuote, which affects both the opening and closing of positions.

Impact

This leads to the loss of user funds whose excess quote/base asset while opening/closing a position is transferred again from the user to the contract leading to twice the amount of the user's initial excess base asset getting stuck in the contract.

Code Snippet

_returnBase _returnQuote _transferAsset _closePosition _openPosition

Tool used

Manual review

Recommendation

Use _transferAsset(baseAsset, address(this), msg.sender, baseBalance) in _returnBase

Use _transferAsset(inputAsset, address(this), msg.sender, quoteBalance) in _returnQuote

Lyra Comment

This has been resolved.

Sherlock Comment

Update: Sounds reasonable.

Note: Protocol team acknowledged this finding which was independently discovered by them while testing on Kovan during the CASE period. It has been apparently fixed in a recent commit post-CASE-start (see https://github.com/lyra-finance/lyra-protocol/blob/0126776cc4061d66ed0400fce21d43e2eea172df/contracts/periphery/Wrapper/OptionMarketWrapperWithSwaps.sol#L458 and https://github.com/lyra-finance/lyra-protocol/blob/0126776cc4061d66ed0400fce21d43e2eea172df/contracts/periphery/Wrapper/OptionMarketWrapperWithSwaps.sol#L476).

Issue M-06

Event emitted with incorrect value

Summary
Event PositionUpdated is always emitted with a value of PositionUpdatedType.ADJUSTED, which is incorrect for positions being opened.

Severity
Medium

Vulnerability Detail
The check for _positionId == 0 in the emission of event PositionUpdated will always return false because _positionId is always updated to a non-zero value earlier.

This would cause the PositionUpdated event emission to always say PositionUpdatedType.ADJUSTED and never PositionUpdatedType.OPENED.

Impact
Frontend or offchain monitoring tools could be affected because they would never see new positions being opened but only positions being adjusted (even when they are being newly opened). This could negatively impact UI & UX to cause confusion and perhaps even a DoS vulnerability.

Code Snippet
PositionUpdated _positionId

Tool used
Manual review

Recommendation
Cache the _positionId value (to track a zero value for later event emission) or use a separate local variable instead of updating the parameter itself.

Lyra Comment
This has been resolved.

Note the "newPosition" variable added

https://github.com/lyra-finance/lyra-protocol/blob/avalon/contracts/OptionToken.sol#L283

Sherlock Comment
Fixes the issue sufficiently.

Issue M-07

Interchanged function arguments

Summary

The positions of arguments totalPoolValue and exchangeParams.spotPrice in the function call _getLiquidity are accidentally interchanged compared to what is expected by the parameters of _getLiquidity.

Severity

Medium

Vulnerability Detail

The first two arguments of _getLiquidity within _getTokenPriceAndStale are interchanged. Instead of:

_getLiquidity(exchangeParams.spotPrice, totalPoolValue,...)

it is incorrectly implemented as:

_getLiquidity(totalPoolValue, exchangeParams.spotPrice,...)

Impact

This will break the liquidityThresholdCrossed constraint in the circuit-breaker _updateCBs and would cause a DoS with the protocol unable to make progress because CBTimestamp check in _canProcess will always fail while processing deposit/withdrawal queues. This would then require the guardian to take over and process everything manually as clarified by the protocol team.

Code Snippet

Call: _getLiquidity(totalPoolValue, exchangeParams.spotPrice,...) Declaration: _getLiquidity(exchangeParams.spotPrice, totalPoolValue,...)

Tool used

Manual review

Lyra Comment

This has been resolved.

https://github.com/lyra-finance/lyra-protocol/blob/avalon/contracts/LiquidityPool.sol#L454

Sherlock Comment

Fixes the issue sufficiently.

Issue M-08

Incorrect collateral transferred leads to loss of user funds

Summary
setCollateralWrapper transfers the entire collateral from the user instead of transferring only the differential amount when there is already some previously deposited collateral.

Severity
Medium

Vulnerability Detail
If setCollateralTo > currentPosition.collateral, then instead of transferring the differential collateral (i.e. setCollateralTo - currentPosition.collateral) amount from the user, setCollateralWrapper transfers the entire collateral amount of setCollateralTo again from the user.

Impact
At a minimum, the user is surprised (could affect their protocol engagement) and the transaction could revert if the user doesn't have the unexpected amount of collateral funds. If transferred, it leads to the loss of the user's additional collateral to the protocol wrapper, which may be claimed by the next user engaging with the wrapper contract.

Code Snippet
setCollateralWrapper

Tool used
Manual review

Recommendation
Transfer only setCollateralTo - currentPosition.collateral instead of setCollateralTo.

Lyra Comment
This has been resolved.

https://github.com/lyra-finance/lyra-protocol/blob/avalon/contracts/LiquidityPool.sol#L454

Sherlock Comment
Looks reasonable.

Issue M-09

CEI pattern violations and cross-function reentrancy

Summary

Checks-effects-interactions (CEI) pattern is violated in a few places. If there is a reentrancy possibility in quoteToken or baseToken, an attacker could steal funds by exploiting cross-function reentrancy.

An attacker could potentially split their OptionToken in the middle of a settleOptions call. The attacker would be fully paid for the settleOptions call, and they would be minted another OptionToken that can be settled for its full value minus one wei. Attacks can be repeated to drain the contract.

Severity

Medium

Vulnerability Detail

Although there are individual nonReentrant guards on the settleOptions (within ShortCollateral.sol) and split (within OptionToken.sol) functions, these functions exist in different contracts, so an attacker might be able to call split in the middle of a settleOptions call (e.g. perhaps _sendLongCallProceeds transfers control flow to the position's owner due to the implementation of quoteToken/baseToken). At the beginning of each iteration of the main for loop in settleOptions, a memory copy of the position is taken. An attacker can simply call split on this position after the memory copy is taken, and this won't affect the memory copy of position.amount. This will create a new OptionToken that can also be settled later on, while the original OptionToken will be burned at the end of the settleOptions call. A small detail here is that the attacker can't call split to create an OptionToken of equal value, but this doesn't prevent the attack, since the attacker can just subtract one wei which is essentially the same value.

Impact

If there is a reentrancy possibility in quoteToken or baseToken, an attacker could steal funds. This is a hypothetical risk at the moment because there are no actual control flow transfers to the user in settleOptions despite the nonReentrant guard suggesting there could be. However, Synthetix tokens are upgradable, and so it is not impossible that quoteToken/baseToken are upgraded by Synthetix in the future and this issue becomes exploitable.

Code Snippet:

reclaimInsolventQuote (CEI pattern violation) settleOptions (CEI pattern violation) split

Tool used

Manual review

Recommendation

Ensure CEI pattern is not violated (even with nonReentrant) and cross-contract reentrancies are considered in the threat model. In this case, settlePositions should probably burn the OptionTokens before transferring anything.

Lyra Comment

The Synthetix contracts can be trusted to not upgrade into a system which would allow this exploit. In the case of Synthetix being exploited, there are much simpler avenues for extracting value out of the lyra contracts (or rather the entire SNX ecosystem). In general, reentrancy checks aren't necessary - however they have been added just in case. As such, this has been resolved.

https://github.com/lyra-finance/lyra-protocol/pull/2/commits/6635f6005f68d46dfe60b92e34f372851c536bfd

Sherlock Comment

Looks reasonable.

Issue L-01

Missing Zero-address Validation

Summary
Lack of zero-address validation on address parameters will lead to reverts and may force contract redeployments in the protocol.

Severity
Low

Vulnerability Detail
Many functions externally accessible by users and owners lack zero-address validation on address parameters. Accidentally using zero addresses will lead to transaction reverts.

Impact
This will lead to transaction reverts, waste gas, require resubmission of transactions and may even force contract redeployments in certain cases within the protocol.

Code Snippet
setLiquidityTracker setAddressResolver setPoolHedger LiquidityTokens.init liquidatePosition

Tool used
Manual review

Recommendation
Add explicit zero-address validation on input parameters of address type.

Lyra Comment
Acknowledged, in the case of init() being called incorrectly a full redeploy will be done.

Sherlock Comment
Noted.

Issue L-02

Missing events

Summary
Critical functions do not emit events.

Severity
Low

Vulnerability Detail
Functions that are access-controlled (onlyOwner) or that change important protocol addresses/parameters should emit events for off-chain tracking of such critical changes to allow users to observe them and decide how/whether to continue engaging with the protocol based on the observed changes.

While many administrative functions emit events, a few critical ones are missing the emission of events.

Impact
External (off-chain) observers will not be able to easily monitor critical on-chain changes to decide how/whether to continue engaging with the protocol. This leads to reduced transparency.

Code Snippet
setLiquidityTracker setPartialCollateralParams setURI setLiquidityToken addCurveStable removeCurveStable addMarket updateMarket

Tool used
Manual review

Recommendation
Add events to functions that change critical parameters.

Lyra Comment
Events have been added to the core contracts. Wrapper has not been updated yet.

Sherlock Comment
Logs are emitted for all core contract functions that are listed in the issue.

Issue L-03

Time-delayed change of critical parameters is absent

Summary

Change of critical parameters should be enforced only after a time delay.

Severity

Low

Vulnerability Detail

When critical parameters of systems need to be changed, it is required to broadcast the change via event emission and recommended to enforce the changes after a time-delay. This is to allow system users to be aware of such critical changes and give them an opportunity to exit or adjust their engagement with the system accordingly.

None of the onlyOwner functions that change critical protocol addresses/parameters have a timelock for a time-delayed change to alert: (1) users and give them a chance to engage/exit protocol if they are not agreeable to the changes (2) team in case of compromised owner(s) and give them a chance to perform incident response.

Impact

Users may be surprised when critical parameters are changed without notice. Furthermore, it can erode users' trust since they can't be sure the protocol rules won't be changed later on.

Compromised owner keys may be used to change protocol addresses/parameters to benefit attackers. Without a time-delay, authorised owners have no time for any planned incident response.

Code Snippet

setOptionMarketParams setLiquidityTracker setPartialCollateralParams setStrikeSkew setLiquidityToken addCurveStable removeCurveStable addMarket updateMarket

Tool used

Manual review

Recommendation

All access-controlled functions that set/change critical addresses/parameters in these contracts should apply a timelock. Consider evaluating the use of OpenZeppelin's TimelockController.

Lyra Comment

“Yes this is in the plans for a future upgrade of the system. I imagine we'll be writing new contracts to handle ownership and allow things like time delays and token holder voting to veto updates to the contracts.”

Sherlock Comment

Noted

Issue L-04

Missing validation and events in init functions

Summary
None of the init functions perform zero-address validation on address parameters or emit events.

Severity
Low

Vulnerability Detail
None of the init functions perform zero-address validation on address parameters or emit events. They all however are onlyOwner callable and have the initializer modifier (from SimpleInitializeable) so that they can be called only once.

Impact
Accidental use of incorrect parameters will require contract redeployment. Offchain monitoring of calls to these critical functions is not possible.

Code Snippet
LiquidityPool LiquidityTokens OptionGreekCache OptionMarket OptionMarketPricer OptionToken PoolHedger ShortCollateral

Tool used
Manual review

Recommendation
It is recommended to perform validation of input parameters and emit events.

Lyra Comment
The issue was acknowledged by the protocol team: "*Very aware a mistake here means a full redeployment.*"

Sherlock Comment
Noted that the choice has been made not to fix.

Issue L-05

Missing validation of asset types

Summary
Lack of input validation in init of OptionMarket to check if the quote and base assets are as expected by the protocol to be sUSD and Synthetix Synths.

Severity
Low

Vulnerability Detail
The protocol assumes that quote and base assets are sUSD and Synthetix Synths. However, there are no checks to enforce that assumption.

Impact
Using any other asset types will break the protocol assumptions and its working.

Code Snippet
init

Tool used
Manual review

Recommendation
Add explicit validation to check that quote asset is sUSD and base asset is one of the recognized Synthetix Synths from https://synthetix.io/synths.

Lyra Comment
This check is covered in deploy scripts.

Sherlock Comment
Noted, but no code reference provided.

Issue L-06

Missing sanity/threshold checks

Summary
Sanity/threshold checks (depending on their types) on input parameters are missing.

Severity
Low

Vulnerability Detail
Functions that update critical protocol parameters are missing sanity/threshold validation on some parameters.

Examples include:

- Missing sanity/threshold check on maxBoardExpiry of optionMarketParams
- Missing sanity/threshold check on vegaFeeCoefficient in setPricingParams
- Missing sanity/threshold checks on all parameters set in setVarianceFeeParams
- Missing sanity/threshold checks on maxStrikesPerBoard in setGreekCacheParameters
- Missing sanity/threshold checks on quoteKey, baseKey and trackingCode in setGlobalsForContract.

Impact
Parameters may accidentally be initialized with invalid values in the context of the protocol or in relation to other parameters. This may lead to incorrect accounting and protocol malfunction.

Code Snippet
setOptionMarketParams setPricingParams setVarianceFeeParams
setGreekCacheParameters setGlobalsForContract

Tool used
Manual review

Recommendation
Add explicit sanity/threshold validation to check that input parameters fall within range or have values as expected in the context of the protocol or in relation to other parameters.

Lyra Comment
For the listed parameters, both 0 and high values are allowed. There’s no reasonable cap as the values are dependent on what price the asset trades at.

In terms of `setGlobalsForContract`, the system won't operate properly until those are corrected. `trackingCode` can be set to zero too.

Sherlock Comment

Noted

Issue L-07

Missing equivalence checks in setters

Summary

Setter functions are missing checks to validate if the new value being set is the same as the current value already set in the contract.

Severity

Low

Vulnerability Detail

Setter functions are missing checks to validate if the new value being set is the same as the current value already set in the contract. Such checks will showcase mismatches between on-chain and off-chain states.

Impact

This may hinder detecting discrepancies between on-chain and off-chain states leading to flawed assumptions of on-chain state and protocol behavior.

Code Snippet

setBoardFrozen

Tool used

Manual review

Recommendation

Add equivalence checks to validate (and revert) if the new value being set is the same as the current value already set in the contract.

Lyra Comment

Acknowledged. Since there is no significant downside besides a wasted transaction, this will not be fixed due to contract size constraints.

Sherlock Comment

Noted

Issue L-08

Board expiry is an open interval

Summary

Option board expiry is defined as “The timestamp when the board expires.” However, the checks enforced on it treat it as an open interval.

Severity

Low

Vulnerability Detail

The checks enforced on option board expiry treat it as an open interval which excludes the (low probability) case where expiry == block.timestamp which is neither treated as expired in _doTrade nor treated as not-expired in settleExpiredBoard.

Impact

Undefined behavior

Code Snippet

expiry _doTrade settleExpiredBoard

Tool used

Manual review

Recommendation

Enforce board expiry as a closed interval to include the value of block.timestamp i.e. board is considered expired (in _doTrade) when board.expiry <= block.timestamp instead of board.expiry < block.timestamp.

Lyra Comment

This has been resolved. Boards are tradable up until the expiry, but not inclusive. Boards can be settled on the same second as expiry.

https://github.com/lyra-finance/lyra-protocol/blob/avalon/contracts/OptionMarket.sol#L1016

https://github.com/lyra-finance/lyra-protocol/blob/avalon/contracts/OptionMarket.sol#L785

Sherlock Comment

The referenced commits fix the issue sufficiently.

SHERLOCK

Issue L-09

Max values lesser than min values

Summary
MaxBaseIV, maxSkew and and maxVol may accidentally be set to values lower than MinBaseIV, minSkew and minVol respectively.

Severity
Low

Vulnerability Detail
While setTradeLimitParams enforces threshold checks on max/min BaseIVs, max/min Skews and maxVol separately, there is no check to ensure that max values are set higher than min values.

Impact
Undefined behavior

Code Snippet
setTradeLimitParams

Tool used
Manual review

Recommendation
Add checks to ensure that max values are set higher than min values.

Lyra Comment
Acknowledged. Impact is zero as trading would just be blocked until parameters are set back to valid ranges, so this will not be fixed.

Sherlock Comment
Noted. Would like to see the tests that confirm this behavior.

Issue L-10

SynthetixAdapter initialize may be front-run

Summary
SynthetixAdapter initialize is susceptible to being initialized by someone other than the deployer.

Severity
Low

Vulnerability Detail
SynthetixAdapter initialize is meant to be used in a proxy setting which leads to a typical front-running scenario where the deployed contract is susceptible to being initialized by someone other than the deployer.

Impact
If an attacker manages to front-run and initialize, without being detected, then they can make the protocol use malicious contracts masquerading to be those responsible for Synthetix, Exchanger, Exchanger Rates, Collateral Shorts and Delegate Approvals by setting addressResolver. They can also set arbitrary values for protocol globals or pause/unpause markets at will.

Code Snippet
initialize

Tool used
Manual review

Recommendation
Front-running can be avoided by atomically deploying and initializing from a proxy or a deploy script.

Lyra Comment
Acknowledged. A frontrun would be detected by deploy scripts, leading to redeploying.

Sherlock Comment
Noted.

Issue L-11

Missing valid option market check in setMarketPaused

Summary

setMarketPaused cannot check if the contract address parameter is indeed a valid option market contract address, in the context of the protocol, to which _isPaused is being applied.

Severity

Low

Vulnerability Detail

Given the absence of a global view of all valid option market contracts created, setMarketPaused has no way to check if the contract address parameter is indeed a valid option market contract address in the context of the protocol. If an incorrect address is accidentally used, setMarketPaused applies the _isPaused boolean to that address while the option market expected to be paused/unpaused is unaffected. This can only be detected offchain and setMarketPaused may need to be executed again during any critical incident response scenario.

Impact

An incorrect address that is not a valid option market contract address, in the context of the protocol, may be paused/unpaused while the actual intended option market remains unaffected. This could affect any critical incident response scenario and, if detected, will require setMarketPaused to be executed again, which could affect the timeliness of response.

Code Snippet

setMarketPaused

Tool used

Manual review

Recommendation

Consider creating a global view of all option markets so that the system can check if they exist and are valid.

Lyra Comment

Acknowledged. “[...] if detected, will require setMarketPaused to be executed again” this would still be the case even with a fix. Impact is minimal so will not be fixed.

Sherlock Comment

The difference is that the suggested fix will detect and revert deterministically to suggest a re-execution. The current implementation may fail silently, requiring an alternative way, which may or may not be in place, to detect failed pausing/unpausing of target market.

Also, pausing is an emergency circuit-breaker mechanism used during high-impact scenarios where time to incident respond is less and valuable. So would contest the "impact is minimal" response.

Issue L-12

Event Spamming may grief system

Summary

The public visibility of updateSynthetixAddresses allows griefing and confusing protocol operations via event spamming.

Severity

Low

Vulnerability Detail

Function updateSynthetixAddresses is called by setAddressResolver when the owner changes addressResolver to cache Synthetix addresses. While updateSynthetixAddresses sets the Synthetix addresses used by protocol via addressResolver (that can only be set by owner), it has public visibility and emits an event SynthetixAddressesUpdated.

Impact

The emission of an event in a public function allows griefing the system via event spamming, which could confuse/overwhelm off-chain monitoring tools about addresses being updated when they are actually not i.e. when addressResolver has not changed or Synthetix addresses themselves have not changed.

Code Snippet

updateSynthetixAddresses setAddressResolver

Tool used

Manual review

Recommendation

Consider changing updateSynthetixAddresses visibility to private, adding onlyOwner modifier or checking for changed addresses before modifying state and emitting an event.

Lyra Comment

The idea of the function being public is that there is less trust placed into the owner. Events being spammed is not a concern, so this has not been fixed.

Sherlock Comment

Noted.

Issue L-13

Inconsistency of access control models

Summary

The protocol uses two models of contract owner access control: OpenZeppelin's Ownable and Synthetix's Owned, which have some differences in capabilities.

Severity

Low

Vulnerability Detail

Contracts in /periphery/Wrapper use OpenZeppelin's Ownable access control model, while the main contracts use Synthetix's Owned access control model. OpenZeppelin's Ownable model allows ownership change in a single step which is dangerous and should be avoided. Synthetix's Owned model uses a 2-step nominateNewOwner and acceptOwnership, which is recommended to avoid accidental ownership transfers to incorrect addresses.

Impact

The two different models of ownership access control may make operations more complicated while changing owners. Wrapper contracts using OpenZeppelin's Ownable access control are susceptible to accidental ownership transfers to incorrect addresses because of single-step change.

Code Snippet

Ownable: BasicOptionMarketWrapper and others

Owned: OptionMarketWrapperWithSwaps

Tool used

Manual review

Recommendation

It is recommended to use only one access control model to avoid complicated security operations. Use Synthetix's Owned model everywhere given its support of 2-step nominateNewOwner and acceptOwnership change process.

Lyra Comment

Fixed, only the Synthetix Owned model is used across all contracts now.

Sherlock Comment

All but LyraAdapter
https://github.com/lyra-finance/lyra-protocol/blob/6635f6005f68d46dfe60b92e34f372851c536bfd/contracts/periphery/LyraAdapter.sol#L32?

Issue L-14

Uninitialized OptionBoard is allowed to be used

Summary

When an option board is created for the first time, id is set to 1 by the OptionMarket contract and id 0 is skipped. For every consequent board, an incremental id is set based on the nextBoardId. Option boards are stored in optionBoards which maps ids to instances of the OptionBoard struct. Throughout the codebase option board ids are checked for their validity by comparing OptionBoard.id with the id provided by the user, which is used to retrieve the OptionBoard struct from the mapping. This check is insufficient for board id 0 because, by default, it is uninitialized and therefore the OptionBoard.id is 0, so the check can be bypassed, and the board is valid.

Severity

Low

Vulnerability Detail

In OptionMarket.sol, there are a variety of places with the following logic:

if (board.id != boardId) {revert InvalidBoardId(address(this), boardId);}

Not rejecting board id 0 may lead to undefined behavior, for example, it is possible to call setBoardFrozen on boardId 0, and the transaction will not revert (even though 0 is not an actual boardId).

Impact

Undefined behavior

Code Snippet

setBoardFrozen setBoardBaseIv addStrikeToBoard forceSettleBoard settleExpiredBoard

Tool used

Manual review

Recommendation

Set nextBoardId to 0 instead of 1 at contract creation, or add an explicit check to make sure boardId 0 is considered invalid and rejected in all the functions listed in the Code Snippet section.

Lyra Comment

Has been resolved by checking the boardId passed in.

Sherlock Comment

SHERLOCK

Looks reasonable.

SHERLOCK

Issue L-15

Missing return value checks on asset transfers

Summary
Token transfers of quote and base assets are missing return value checks in several places.

Severity
Low

Vulnerability Detail
The quote asset in Lyra is always specified to be sUSD while base assets are limited to synths specified by Synthetix at https://synthetix.io/synths. Transfers of these assets within the protocol are checked for return values to revert with QuoteTransferFailed and BaseTransferFailed errors upon failure.

The protocol team clarified that return value checks exist as a defensive programming feature if Synth token contracts (which revert now on failed transfers) were ever upgraded in the future to return success/failure instead of reverting.

However, some asset transfers are missing this return value check.

Impact
If synth token contracts were ever upgraded in the future to return success/failure, then failed transfers would not be detected.

Code Snippet
smClaim reclaimInsolventBase _takeExtraCollateral _returnExcessFunds swap openPosition closePosition forceClosePosition _takeExtraCollateral setCollateralWrapper _openPosition _closePosition _returnBase

Tool used
Manual review

Recommendation
To be consistent, check for return values and revert with QuoteTransferFailed and BaseTransferFailed errors upon failure.

Lyra Comment
This has been fixed in most places. All core contracts should be fixed.

Sherlock Comment
All core contracts are fixed. Periphery contracts are pending a fix.

Issue L-16

Unsafe Casting of uint to int & int to uint

Summary
Explicit casting of uint to int and int to uint is generally unsafe and there are several instances where such type conversions are implemented.

Severity
Low

Vulnerability Detail
Explicit casting of uint to int and int to uint is unsafe when downcast uint values are outside the range of int and when negative int values are cast to uint, because Solidity does not enforce implicit bounds checks on such explicit casting.

Impact
This could lead to incorrect accounting of such values in the protocol.

Code Snippet
_updateExposure _getTotalPoolValueQuote updateStrikeExposureAndGetPrice _updateStrikeExposureAndGetPrice _updateStrikeCachedGreeks _getParity getVarianceFee getCurrentHedgedNetDelta _hedgeDelta updatePosition getCappedExpectedHedge, others in BlackScholes.sol and GWAV.sol libraries.

Tool used
Manual review

Recommendation
Use SafeCast.toInt256 and SafeCast.toUint256 consistently to prevent unsafe casts.

Lyra Comment
This has been resolved across all core contracts, besides maths libraries as the logic there should prevent those cases - and would have a huge impact on gas cost of common operations.

Sherlock Comment
All core contracts are fixed except BlackScholes and GWAV

There is one unfixed cast:
https://github.com/lyra-finance/lyra-protocol/blob/6635f6005f68d46dfe60b92e34f372851c536bfd/contracts/ShortPoolHedger.sol#L442

Confirmed this one is missing.

Issue L-17

removeMarket incorrectly removes first market

Summary
Function removeMarket incorrectly removes the first market if called with a non-existing market identifier.

Severity
Low

Vulnerability Detail
If removeMarket is accidentally called with a non-existing market identifier, then it fails to detect that and instead removes marketIds[0] and marketContracts[0]because the index continues to be 0 in the implementation.

Impact
An incorrect market is removed, which will cause DoS for that market's trading thereafter.

Code Snippet
removeMarket

Tool used
Manual review

Recommendation
Implement logic similar to removeCurveStable where the function reverts if the provided market identifier is not found.

Lyra Comment
Has been fixed.

Sherlock Comment
Fixed sufficiently.

Issue L-18

Incorrect operator * used instead of /

Summary
SignedDecimalMath.divideDecimal incorrectly returns (x * UNIT) * y instead of returning (x * UNIT) / y.

Severity
Low

Vulnerability Detail
SignedDecimalMath appears to have been updated from the corresponding Synthetix version to remove SafeMath in favor of using Solidity's 0.8+ compiler's built-in bound checks (Library @dev comment: "Modified synthetix SafeDecimalMath to include internal arithmetic underflow/overflow"). While doing so, there appears to have been a typographical error introduced in divideDecimal which returns (x * UNIT) * y instead of (x * UNIT) / y.

Impact
While none of Lyra's contracts currently use this function (they use the equivalent function for uint) and therefore are not immediately impacted (hence the Low severity), this needs to be fixed for any potential future/other uses, given that this is a library.

Code Snippet
divideDecimal Synthetix version

Tool used
Manual review

Recommendation
SignedDecimalMath.divideDecimal should return (x * UNIT) / y.

Lyra Comment
Has been fixed.

Sherlock Comment
Is fixed in the reviewed version
https://github.com/lyra-finance/lyra-protocol/blob/6635f6005f68d46dfe60b92e34f372851c536bfd/contracts/synthetix/SignedDecimalMath.sol#L150-L153

Issue L-19

Incorrect implementation of SignedDecimalMath functions

Summary

The implementations of _divideDecimalRound, _multiplyDecimalRound and preciseDecimalToDecimal in SignedDecimalMath miss decrementing resultTimesTen and quotientTimesTen values.

Severity

Low

Vulnerability Detail

SignedDecimalMath functions_divideDecimalRound, _multiplyDecimalRound and preciseDecimalToDecimal miss decrementing resultTimesTen and quotientTimesTen values by 10 when those values % 10 <= -5.

There appear to be copy-paste errors from DecimalMath for negative integers.

Impact

None of Lyra's contracts currently use _divideDecimalRound function (they use the equivalent function for uint, which ultimately gets used in functions of SynthetixAdapter contract and BlackScholes library) and therefore are not immediately impacted.

Lyra's contracts currently use SignedDecimalMath _multiplyDecimalRound, which is called by multiplyDecimalRoundPrecise, which in turn is called by functions in BlackScholes library. Lyra's contracts also currently use SignedDecimalMath preciseDecimalToDecimal, which is called by functions in BlackScholes library. These functions need to be fixed for current and any potential future/other uses, given that this is a library. Current usages may lead to incorrect accounting values for negative integers, which could impact the economic security aspects of the protocol.

Code Snippet

_divideDecimalRound Synthetix version: _divideDecimalRound & _roundDividingByTen

_multiplyDecimalRound Synthetix version: _multiplyDecimalRound & _roundDividingByTen

preciseDecimalToDecimal Synthetix version: preciseDecimalToDecimal & _roundDividingByTen

Tool used
Manual review

Recommendation
Make implementations consistent with Synthetix.

Lyra Comment
Has been fixed.

Sherlock Comment
Is fixed in the reviewed version.

Issue L-20

Users can add collateral and split/merge positions when markets/protocol is paused

Summary
Paused markets/protocol do not prevent users from interacting with the protocol when collateral is added or non-short positions are split or merged.

Severity
Low

Vulnerability Detail
Interaction with paused markets/protocol is prevented by enforcing the notPaused modifier on two SynthetixAdapter functions: getSpotPriceForMarket and getExchangeParams. However, this does not prevent users from calling addCollateral or split/merge non-short positions (under certain conditions) even when paused because those flows do not call either of the two functions that enforce the notPaused modifier.

Impact
Depending on the reasons behind pausing the market/protocol, allowing users to bypass the pause by adding collateral to their positions or split/merge non-short positions (under certain conditions) could affect recovery from the incident that led to the pausing.

Code Snippet
getSpotPriceForMarket getExchangeParams addCollateral

Tool used
Manual review

Recommendation
Evaluate addCollateral, user split/merge of non-short positions and all user flows that modify protocol state to make sure they can be paused when needed. Consider the global pause as an overall pausing mechanism for all protocol state-modifying flows and the market-specific pause for a subset of those flows. Specify and document exactly which flows need to be paused, by which of the two pausing mechanisms and under what conditions.

Lyra Comment
Has been fixed by adding `notGlobalPaused` modifier to several contracts.

Sherlock Comment

Looks reasonable.

Issue L-21

Void constructor

Summary
Calls to base contract constructors that are unimplemented lead to misplaced assumptions.

Severity
Low

Vulnerability Detail
The constructor of OptionToken calls the constructor of ERC721Enumerable which is not implemented. This could lead to misplaced assumptions if the unimplemented constructor were expected to initialize any state.

Impact
Reduced readability and auditability.

Code Snippet:
OptionToken ERC721Enumerable

Tool used
Slither

Recommendation
Remove the call to the unimplemented base constructor.

Lyra Comment
Call to unimplemented constructor has been removed.

Sherlock Comment
Fixed in the reviewed version.

Issue L-22

Event emitted prematurely

Summary
The event PositionUpdated is emitted prematurely before token transfer.

Severity
Low

Vulnerability Detail
The event PositionUpdated is emitted in _beforeTokenTransfer notifying that the option token has been transferred to the new owner indicated by the to address. However, the token transfer technically has not happened yet and is not the owner until that happens.

Impact
This could confuse off-chain monitoring tools.

Code Snippet
_beforeTokenTransfer PositionUpdated

Tool used
Manual review

Recommendation
Move event PositionUpdated from _beforeTokenTransfer to _afterTokenTransfer.

Lyra Comment
Acknowledged. However, as there is no _afterTokenTransfer in the ERC721 implementation to overwrite, the event will remain in the _beforeTokenTransfer event.

Sherlock Comment
There is _afterTokenTransfer in ERC721.sol:
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/450c569d78aa57e8e73547f99ec412409c73d852/contracts/token/ERC721/ERC721.sol#L438-L453

Issue L-23

Max/Infinite approvals are dangerous

Summary
Giving max/infinite approvals to contracts is dangerous.

Severity
Low

Vulnerability Detail
Giving max/infinite approvals to contracts is dangerous because if those contracts are ever exploited then they can remove all the funds from the approving addresses.

Impact
Loss of funds if approved contracts are exploited.

Code Snippet
updateMarket

Tool used
Manual review

Recommendation
Short-term, check allowance and approve only as much as required during each transaction flow. Long-term, design with zero-trust boundaries between components of the system so that even if some get compromised others can function.

Lyra Comment
Acknowledged. However, as the wrapper should never hold any funds, these max approvals are not considered a risk.

Sherlock Comment
Noted.

Issue L-24

Remove private keys from deployment scripts

Summary

There are two locations in the deployment scripts that have hard-coded private keys.

Severity

Low

Vulnerability Detail

Private keys are present in the deployment scripts: deployAndSeedLocal.ts and kovanInteraction.ts. None of the keys contain significant amounts of ETH or other tokens.

Impact

Deployment key leakage can have devastating effects if not noticed early before users are onboarded.

Code Snippet

deployAndSeedLocal.ts kovanInteraction.ts

Tool used

Manual Review

Recommendation

Accidental private key leakage is a serious problem. It is recommended to change the deployment script so that they retrieve the private keys from environment variables for example. Precautions should be taken so that private key leakage does not occur.

Lyra Comment

This private key is the default hardhat private key, and well known. Not a concern.

These scripts are not used for deployment; they are example scripts for integrators - not to be used in production. Deployments use private keys stored in .env files that are not pushed to github.

Sherlock Comment

Noted.

Issue I-01

Inconsistent use of positionId

Summary
closePosition uses params.positionId instead of result.positionId in optionToken.transferFrom which is inconsistent.

Severity
Informational

Vulnerability Detail
While transferring the optionToken back to msg.sender, instead of using positionId returned from optionMarket.closePosition, BasicOptionMarketWrapper.closePosition uses the user provided parameter params.positionId. While the positionId in result should not be different for closePosition (unlike openPosition), unless any underlying logic changes, it is better to use result.positionId for consistency.

The same pattern exists in BasicOptionMarketWrapper.forceClosePosition.

Impact
Reduced readability, auditability and maintainability.

Code Snippet
closePosition forceClosePosition

Tool used
Manual review

Recommendation
Use result.positionId instead of params.positionId in optionToken.transferFrom.

Lyra Comment
This has been fixed.

Sherlock Comment
closePosition appears fixed but not forceClosePosition:
https://github.com/lyra-finance/lyra-protocol/blob/6635f6005f68d46dfe60b92e34f372851c536bfd/contracts/periphery/Wrapper/BasicOptionMarketWrapper.sol#L66

Issue I-02

Unused event declarations

Summary

Unused events may be indicative of unused code or of missed emit/logic.

Severity

Informational

Vulnerability Detail

Events TradingCutoffSet, QuoteKeySet and BaseKeySet are missing emits. This is potentially indicative of missing setters.

Impact

Events that are declared but not used may be indicative of unused declarations leading to reduced readability/maintainability/auditability, or worse, indicative of a missing emit which is bad for monitoring or missing logic that would have emitted that event.

Code Snippet

TradingCutoffSet QuoteKeySet BaseKeySet

Tool used

Manual review

Recommendation

Remove event declarations or add missing setters that will emit these events.

Lyra Comment

These events have been removed.

Sherlock Comment

Fixed in the reviewed version.

Issue I-03

Code, comments & documentation

Summary

Discrepancies in/between or inaccuracies/deficiencies in code, comment and documentation can be misleading and could indicate the presence of inaccurate implementation or documentation.

Severity

Informational

Vulnerability Detail

- Missing detailed specifications and documentation for all contracts. The current documentation has been created for the previous version of the protocol and not the Avalon version being reviewed. This forces reviewers to make assumptions about the functionalities implemented and their intended behaviors.
- Missing documentation of access control specifically about the different privileged roles in the protocol (different owners) and their multisig configuration, status, etc.
- Typo: NatSpec for param strikePrice should say “Price of the Strike”
- Typo: NatSpec for burn function should say “Burns” not “Mints”
- Stale comment “2 eth + 0.2 eth” in getLiquidationFees

Impact

Reduced code comprehension, auditability and maintainability.

Code Snippet

strikePrice burn getLiquidationFees

Tool used

Manual review

Recommendation

Code, comments and documentation should all be complete, accurate and consistent before security review.

Lyra Comment

Recommendations have been resolved, along with more cleanup of comments overall.

Sherlock Comment

Nit: burn says "Burn new tokens and transfers them to `owner`" which is incorrect.

Issue I-04

Missing or Incomplete NatSpec

Summary

Some functions are missing @notice/@dev NatSpec comments for the function, @param for all/some of their parameters and @return for return values.

Severity

Informational

Vulnerability Detail

Given that NatSpec is an important part of code documentation, missing NatSpec comments affects code comprehension, auditability and usability.

Impact

Reduced code comprehension, auditability and usability.

Code Snippet

Examples of functions missing NatSpec: forceSettleBoard setOptionMarketParams smClaim openPosition addCollateral getStrikeAndExpiry

Examples of functions with incomplete NatSpec: _doTrade updateCacheAndGetTradeResult updateContractParams

Tool used

Manual review

Recommendation

Add full NatSpec for all functions.

Lyra Comment

All public facing external functions have been documented properly.

Sherlock Comment

Noted without verifying.

Issue I-05

Unused or inconsistently used named returns

Summary
Functions use a mix of explicit returns and named return variables.

Severity
Informational

Vulnerability Detail
Functions use a mix of explicit returns and named return variables. Some named returns are declared but unused with functions favoring explicit returns. This affects readability.

Impact
Reduced code comprehension and auditability.

Code Snippet
numLiveBoards strikePrice getSettlementParameters timeWeightedFee minVal maxVal

Tool used
Manual review

Recommendation
Consistently use explicit returns or named returns. Remove unused named returns. Favor explicit returns over implicit named returns.

Lyra Comment
Using named returns gives more information to integrators using the platform. As such a combination of named returns with explicit returns has been used as the standard across the contracts.

Sherlock Comment
Noted.

Issue I-06

Inconsistent naming convention

Summary
Function naming convention is inconsistent in a few places.

Severity
Informational

Vulnerability Detail
One of the naming conventions is for internal functions to start with an underscore. While this is followed in many places, there are some functions where this is missing.

Impact
Reduced readability and auditability.

Code Snippet
sendAllQuoteToLP

Tool used
Manual review

Recommendation
Use the naming convention of starting with an underscore for internal functions consistently.

Lyra Comment
All internal functions have been fixed to begin with an underscore.

Sherlock Comment
Noted without verifying.

Issue I-07

Code structure deviates from best-practice

Summary

It is a best practice to order different contract constructs in a certain widely-used layout for better readability and auditability.

Severity

Informational

Vulnerability Detail

The best-practice layout for a contract should follow the following order: state variables, events, modifiers, constructor and functions.

Function ordering helps readers identify which functions they can call and find constructor and fallback functions easier. Functions should be grouped according to their visibility and ordered as: constructor, receive function (if exists), fallback function (if exists), external, public, internal, private.

Some constructs deviate from this recommended best-practice: Modifiers, events and errors are at the end of contracts. External/public functions are mixed with internal/private ones.

Impact

Reduced readability, auditability and maintainability.

Code Snippet

TradeIterationsHasRemainder Modifiers Events and Errors

Tool used

Manual review

Recommendation

Consider adopting recommended best-practice for code structure and layout.

Lyra Comment

In general readability has been a focus of the contracts. Each contract has been split into sections where they can be read from top to bottom to follow code logically. Structs and variables are defined at the top of each file, Errors and Events at the bottom.

Sherlock Comment

Noted.

Issue I-08

Potential gas optimizations

Summary
There are many opportunities to optimize gas usage.

Severity
Informational

Vulnerability Detail
There are various categories of gas optimizations possible across contracts such as:

- Caching of variables in memory instead of reusing storage variables or repeated external calls
- Use of locals/parameters in event emissions instead of their equivalent storage variables
- Avoiding initialisations of loop indices to their default values
- Using unchecked blocks to save more gas on overflow/underflow safe arithmetic operation
- Skipping initializations of variables whose initial values are expected to be the same as default values of their types

Impact
While launching on a Layer-2 (Optimism) implies lesser impact from increased gas usage, the impact is still non-zero and may add up or increase over time.

Code Snippet
Examples

- Caching: quoteAsset
- Event: optionMarketParams
- Loop: createOptionBoard
- unchecked: GWAV
- Initialization: baseInsolventAmount

Tool used
Manual review

Recommendation
Consider optimizing for gas where possible.

Lyra Comment

Acknowledged. Some have been fixed, however as the protocol is run on optimism, readability is more important than minor gas savings.

Sherlock Comment

Noted without verifying.

Issue I-09

Several contracts are stubbed with test helpers

Summary
The SynthetixAdapter contract uses addressResolver which is used to map strings to addresses. This is a helper function that links calls to the right Synthetix contracts. The Resolver that is currently used is a test contract TestAddressResolver.

Severity
Informational

Vulnerability Detail
N/A

Impact
Forgetting to update the deployment script could lead to faulty deployment.

Code Snippet
TestAddressResolver.sol

Tool used
Manual Review

Recommendation
Make sure that the production deployment links to the correct Synthetix contracts.

Lyra Comment
There are full integration tests that run against the synthetix contracts in the local environment. On mainnet the contracts will be linked to the real synthetix contracts.

Sherlock Comment
Noted.