Lyra Periphery Smart Contract Audit

Lyra, 10 March 2023

1. Introduction

iosiro was commissioned by Lyra (https://www.lyra.finance/) to conduct a smart contract audit of Lyra's periphery smart contracts. The audit was performed by 2 auditors between 27 February and 10 March 2023, using 20 resource days.

This report is organized into the following sections.

- Section 2 - Executive summary: A high-level description of the findings of the audit.
- Section 3 - Audit details: A description of the scope and methodology of the audit.
- Section 4 - Design specification: An outline of the intended functionality of the smart contracts.
- Section 5 - Detailed findings: Detailed descriptions of the findings of the audit.

The information in this report should be used to understand the smart contracts' risk exposure better and as a guide to improving the security posture of the smart contracts by remediating the issues identified. The results of this audit reflect the in-scope source code reviewed at the time of the audit.

The purpose of this audit was to achieve the following:

- Identify potential security flaws.
- Ensure that the smart contracts function according to the documentation provided.

Assessing the off-chain functionality associated with the contracts, for example, backend web application code, was outside of the scope of this audit.

Due to the unregulated nature and ease of transfer of cryptocurrencies, operations that store or interact with these assets are considered high risk from cyber attacks. As such, the highest level of security should be observed when interacting with these assets. This requires a forward-thinking approach, which takes into account the new and experimental nature of blockchain technologies. Strategies that should be used to encourage secure code development include:

- Security should be integrated into the development lifecycle, and the level of perceived security should not be limited to a single code audit.
- Defensive programming should be employed to account for unforeseen circumstances.
- Current best practices should be followed where possible.

2. Executive summary

This report presents the findings of an audit performed by iosiro on specific peripheral smart contracts for the Lyra Protocol.

Audit findings

iosiro noted one high risk issue that was swiftly remediated by the Lyra team before the conclusion of the audit. The issue was due to the original reference implementation assuming a fixed total supply of tokens. This could have been abused by users to inflate their proportional governance voting power.

Two low-risk and some information issues relating to event signatures were identified. The first low-risk issue relates to inconsistent behavior during specific edge cases. The second low-risk issue is due to an undocumented failure condition in a library function. The vulnerable segment of code was unused at the time of the audit. However, as it can introduce a vulnerable case in the future, it should be preemptively fixed.

Overall, the source code of the smart contracts in scope was found to be of a high standard.

Recommendations

At a high level, the security posture of the Lyra Protocol could be further strengthened by:

- Remediating the issues identified in this report and performing a review to ensure that the issues were correctly addressed.
- Performing additional audits at regular intervals, as security best practices, tools, and knowledge change over time. Additional audits throughout the project's lifespan ensure the longevity of the codebase.
- Continue their bug bounty program encouraging the responsible disclosure of security vulnerabilities in the system.

3. Audit details

3.1 Scope

The source code considered in-scope for the assessment is described below. Code from all other files was considered to be out-of-scope. Out-of-scope code that interacts with in-scope code was assumed to function as intended and not introduce any functional or security vulnerabilities for the purposes of this audit.

3.1.1 Smart contracts

- **Project name:** Utilities
- **Commits:** 36b9d4
- **Files:** MemoryBinarySearch.sol, QuickSort.sol, UnorderedMemoryArray.sol, ConvertDecimals.sol, DecimalMath.sol, SignedDecimalMath.sol, OptionEncoding.sol, Black76.sol, FixedPointMathLib.sol, IntLib.sol, AbstractOwned.sol, Owned.sol, OwnedUpgradeable.sol

- **Project Name:** ListingManager
- **Commits:** b46a14e
- **Files:** ListingManager.sol, ListingManagerLibrarySettings.sol, ExpiryGenerator.sol, StrikePriceGenerator.sol, VolGenerator.sol

- **Project name:** Governance Wrappers
- **Commits:** f53841
- **Files:** BaseGovernanceWrapper.sol, LiquidityPoolGovernanceWrapper.sol, OptionGreekCacheGovernanceWrapper.sol, OptionMarketGovernanceWrapper.sol, OptionMarketPricerGovernanceWrapper.sol, OptionTokenGovernanceWrapper.sol, GMXAdapterGovernanceWrapper.sol, GMXHedgerGovernanceWrapper.sol

- **Project Name:** Synthetix Adapter V2
- **Commits:** e4b9056
- **Files:** SynthetixAdapter2.sol

- **Project Name:** Lyra Governance Strategy
- **Commits:** eecf62
- **Files:** LyraGovernanceStrategy.sol

- **Project Name:** Lyra Distributor
- **Commits:** 25b583
- **Files:** MultiDistributor.sol

3.2 Methodology

The audit was conducted using a variety of techniques described below.

3.2.1 Code review

The source code was manually inspected to identify potential security flaws. Code review is a useful approach for detecting security flaws, discrepancies between the specification and implementation, design improvements, and high-risk areas of the system.

3.2.2 Dynamic analysis

The contracts were compiled, deployed, and tested in a test environment, both manually and through the test suite provided. Manual analysis was used to confirm that the code was functional and discover security issues that could be exploited.

3.2.3 Automated analysis

Tools were used to automatically detect the presence of several types of security vulnerabilities, including reentrancy, timestamp dependency bugs, and transaction-ordering dependency bugs. Static analysis results were reviewed manually and any false positives were removed. Any true positive results are included in this report.
Static analysis tools commonly used include Slither, Securify, and MythX. Tools such as the Remix IDE, compilation output, and linters could also be used to identify potential areas of concern.

3.3 Risk ratings

Each issue identified during the audit has been assigned a risk rating. The rating is determined based on the criteria outlined below.

- **High risk:** The issue could result in a loss of funds for the contract owner or system users.
- **Medium risk:** The issue resulted in the code specification being implemented incorrectly.
- **Low risk:** A best practice or design issue that could affect the security of the contract.

- **Informational:** A lapse in best practice or a suboptimal design pattern that has a minimal risk of affecting the security of the contract.
- **Closed:** The issue was identified during the audit and has since been satisfactorily addressed, removing the risk it posed.

4. Design specification

The following section outlines the intended functionality of the system at a high level. This specification is based on the implementation in the codebase. Any perceived points of conflict should be highlighted with the auditing team to determine the source of the discrepancy.

4.1 Synthetix Adapter

The Synthetix Adapter used by the Lyra Protocol's Avalon release was modified to implement access control on all exchange functionality. A basic set of functions were added to allow the owner of the contract to whitelist specific addresses. For each of the exchange functions, code was introduced to verify that the caller is included in the whitelist.

4.2 Lyra Governance Strategy

The governance model of Aave was forked and modified to make use of Staked Lyra instead of Aave and Staked Aave. The Governance Strategy allows users who stake their Lyra tokens and receive Staked Lrya to participate in governance votes and delegate their voting power to other users. The underlying implementation makes use of snapshots of the users' voting power that is queried when voting and determining the outcome of voting.

4.3 Governance Wrappers

The governance wrappers provide a standard to configure different parts of the Lyra ecosystem, either by the risk council or the governance wrapper owners. All wrappers follow the same design, allowing the governance wrapper owner to configure additional settings and perform strict validation of changes when the risk council makes changes.

4.4 Listing Manager

The listing manager introduces an abstraction layer to queue options boards and strikes to be added to the market. Boards can be queued for specific expiration dates, as long as the expiration complies with some predefined conditions and no other board with the given expiry exists. Furthermore, the strike generation is automated and the prices are calculated as a function of the associated board’s expiration and the spot price of the market.

4.5 Multi-Distributor

The multi-distributor contract allows whitelisted addresses to create batches of new claims. Addresses can only be added to the whitelist by the contract owner, after which the whitelisted address can submit a batch of claims that contains the token to be distributed, the addresses that are eligible to claim, as well as the amount of tokens each address is allowed to claim. After submitting the batch, the batch first needs to be approved by the contract owner.

4.6 Utilities

The utilities repository contains various libraries for the purpose of doing math, handling decimal numbers, handling arrays and encoding option details into IDs. A brief explanation of each directory is provided below.

- **decimals:** Utilities for converting numbers between different levels of precision, as well as performing basic operations such as multiplication and division.
- **math:** Utilities for performing more complex mathematical operations, such as calculating exponents, square roots, ln and converting signed values to absolute values. The math directory also contains the Black76 library, used to price options according to the Black-Scholes options pricing model.
- **arrays:** Utilities for sorting arrays, finding specific values in sorted arrays, as well as manipulating arrays that contain unique values.
- **encoding:** The `OptionEncoding` library provides a convenient way to pack all details of a strike into a `uint96`. The packed value contains the strike price, the expiration and whether it is a call or not. The encoded value can then be used as an ID.
- **ownership:** This folder contains contracts that are used for two-step ownership, and supports upgradeability.

5. Detailed findings

The following section details the findings of the audit.

5.1 High risk

No identified high-risk issues were open at the conclusion of the review.

5.2 Medium risk

No identified medium-risk issues were open at the conclusion of the review.

5.3 Low risk

5.3.1 Invalidate queues when circuit breakers are triggered

ListingManager.sol#L150

Description

The `ListingManager` contract validates that the Market's circuit breakers have not been triggered when enqueuing and executing strikes. However, it does not account for the scenario where the circuit breakers are triggered after the strikes are queued but reset before execution. This could result in stale strikes with invalid skews being added to boards.

Recommendation

The `ListingManager` should validate that the `queuedTime` of the strike or board is greater than the circuit breaker's latest timestamp when processing the queues:

```diff
diff --git a/src/ListingManager.sol b/src/ListingManager.sol
index 2fd5b44..d980f41 100644
--- a/src/ListingManager.sol
+++ b/src/ListingManager.sol
@@ -147,7 +147,7 @@ contract ListingManager is ListingManagerLibrarySettings,
Ownable2Step {
   ////////////////////////////

   function executeQueuedStrikes(uint boardId, uint executionLimit) public {
-      if (isCBActive()) {
+      if (isCBActive(queuedStrikes[boardId].queuedTime)) {
       emit LM_CBClearQueuedStrikes(msg.sender, boardId);
       delete queuedStrikes[boardId];
       return;
@@ -191,7 +191,7 @@ contract ListingManager is ListingManagerLibrarySettings,
Ownable2Step {
   //////////////////////////

   function executeQueuedBoard(uint expiry) public {
-      if (isCBActive()) {
+      if (isCBActive(queuedBoards[expiry].queuedTime)) {
       emit LM_CBClearQueuedBoard(msg.sender, expiry);
       delete queuedBoards[expiry];
       return;
@@ -244,7 +244,7 @@ contract ListingManager is ListingManagerLibrarySettings,
Ownable2Step {
   // if so; request the skews from the libraries
   // and then add to queue
   function findAndQueueStrikesForBoard(uint boardId) external {
-      if (isCBActive()) {
+      if (isCBActive(block.timestamp)) {
       revert LM_CBActive(block.timestamp);
       }

@@ -294,7 +294,7 @@ contract ListingManager is ListingManagerLibrarySettings,
Ownable2Step {
   /////////////////////
   function queueNewBoard(uint newExpiry) external {
-       if (isCBActive()) {
+       if (isCBActive(block.timestamp)) {
        revert LM_CBActive(block.timestamp);
        }

@@ -507,8 +507,8 @@ contract ListingManager is ListingManagerLibrarySettings,
Ownable2Step {
        return exchangeAdapter.getSpotPriceForMarket(address(optionMarket),
IBaseExchangeAdapter.PriceType.REFERENCE);
   }

-  function isCBActive() internal view returns (bool) {
-       return liquidityPool.CBTimestamp() > block.timestamp;
+  function isCBActive(uint256 timestamp) internal view returns (bool) {
+       return liquidityPool.CBTimestamp() > timestamp;
   }

   ///////////
```

5.3.2 `UnorderedMemoryArray.addUniqueToArray()` fails when arrays contain zero as an element

[UnorderedMemoryArray.sol#L65]

Description

`addUniqueToArray()` allows adding an element to an array as long as the element does not already exist, which is obtained by calling `findInArray()`. If `findInArray()` returns -1, it is an indication that the element does not exist and `addUniqueToArray()` can write the new element to the provided `arrayLen`.

`findInArray()` will return -1 either when the array has been fully searched and the new element has not been found, or when an element contains a zero value. As a result, if a legitimate zero value occurs somewhere in the array before an index that already contains the new element, `addUniqueToArray()` will add the new element to the array, breaking the array's uniqueness.

An example test to illustrate this issue is provided below, where an attempt is made to add an element with a value of 5 to an array that already contains the value of 5, but occurs after a value of 0.

```
JavaScript
diff --git a/test/arrays/UnorderedMemoryArray.t.sol
b/test/arrays/UnorderedMemoryArray.t.sol
index 5a6e871..2cb6e6f 100644
--- a/test/arrays/UnorderedMemoryArray.t.sol
+++ b/test/arrays/UnorderedMemoryArray.t.sol
@@ -102,6 +102,20 @@ contract UnorderedMemoryArrayTest is Test {
     assertEq(index, 1);
   }

+  function testAddToArrayThatContainsZero() public {
+    uint[] memory arr = new uint[](4);
+    arr[0] = 0;
+    arr[1] = 5;
+    arr[2] = 10;
+    arr[3] = 1337;
+
+    (uint[] memory newArray, uint newArrayLen, uint len) =
tester.addUniqueToArray(arr, 5, 2);
+    assertEq(newArray[0], 0);
+    assertEq(newArray[1], 5);
+    assertEq(newArray[2], 10);
+    assertEq(newArray[3], 1337);
+  }
+
   /* -------------------------- *
    *   address array functions   *
    * -------------------------- */
```

Recommendation

The zero check in `findInArray()` could be removed in order to allow the function to search through the entire array.

5.4 Informational risk

5.4.1 Design comments

Actions to improve the functionality and readability of the codebase are outlined below.

Incorrect event signature

ListingManager.sol#L234

When emitting the `LM_QueuedBoardExecuted` event, `boardId` is passed as the second parameter, but the event signature expects `expiry`. The event signature should be updated to reflect the implementation.

Event indexing

SynthetixAdapter2.sol#L492

The `contractAddress` parameter of the `ExchangeWhitelistSet` event should be marked as `indexed` to allow efficient querying of changes to the whitelist.

5.5 Closed

5.5.1 Total supply snapshotting not supported (high risk)

LyraGovernanceStrategy.sol#L40

Description

The `totalSupply` of the `STK_LYRA` token is not fixed, which could be abused to influence the results of governance votes. This differs from the reference implementation that uses the total supply of `Aave`, instead of `stkAave,` to determine the quorum thresholds for governance votes.

The `getTotalPropositionSupplyAt(...)` and `getTotalPropositionSupplyAt(...)` functions are used by the `ProposalValidator` to determine whether a quorum has been achieved. Although the `STK_LYRA` token implements the `getTotalSupplyAt(...)` function, giving the impression that the historic total supply can be queried, the underlying implementation does not support snapshotting and always returns the current total supply. This could allow malicious actors to manipulate the number of votes required to overturn or pass governance votes.

Consider the following scenario:

1. A staker with significant voting power submits their vote at the start of the voting period. Their token supply and all delegate votes are added to the proposal.
2. The staker redeems all their staked tokens, reducing the total supply.
3. This reduced total supply is used to determine the quorum when evaluating the proposal's outcome.
4. The attacker successfully manipulates the outcome of the vote.

Recommendation

The `LyraGovernanceStategy` should return the `Lyra` token's total supply in `getTotalPropositionSupplyAt(...)` and `getTotalPropositionSupplyAt(...)`. Since the token's total supply is fixed, implementing snapshotting is unnecessary.

Update

The issue was resolved per the recommendation in commit eecf62f2.