Report

v. 1.0

Customer

XO Market

Smart Contract Audit

EVM Contracts

2nd December 2025

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	02.12.25	A. Zveryanskaya	Initial Draft
0.2	02.12.25	A. Zveryanskaya	Minor revision
1.0	02.12.25	A. Zveryanskaya	Release

2 Summary

This document presents the results of a smart contract audit performed by ABDK Consulting for XO and the System XOMarketV1. The review ran from 27th October to 1st December 2025 and was executed by Mikhail Vladimirov and Dmitry Khovratovich. Our goal was to validate the LS-LMSR pricing math, fee-flow fairness, and governance safeguards highlighted in the provided issue log. System components such as XOMarketFactoryV1, XOFeeHandler, XOMarketNFT, and XOMarketOutcome stitch together upgradeable markets that rely on OpenZeppelin access control, UUPS proxies, ERC-721/ERC-1155 tokens, and ABDK quad-precision math while enforcing resolver vetting, NFT-bound ownership, non-transferable outcome shares, and Achievement-gated deployment.

Our conclusion is that System exhibits medium maturity and generally clean modular boundaries, yet the dynamic alpha math and share accounting still need attention before mainnet launch. The architecture correctly isolates per-market state, applies resolver-controlled pausing, calculates configurable creation costs, and routes creator, protocol, and user fees through XOFeeHandler with share-based accrual, which aligns with the intended business logic of collateralized prediction markets.

The most pressing risks are captured in CVF-2 and CVF-3. CVF-2 details an Arithmetic overflow/underflow class issue in LSLMSRDynamic where products of alphaBps, c*_fp, and total supply are cast into int256, so a large T window can corrupt slope calculations and destabilize pricing convexity. CVF-3 exposes an input-validation gap inside calculatePurchaseCost that allows duplicate outcome entries, making total purchase costs depend on array ordering and enabling griefing when propagated through buy and sell. Both findings directly threaten collateral accounting and undermine the fairness goals of the fee/price subsystem.

We recommend hardening System by replacing risky multiplications with checked mulDiv helpers, caching per-market precision factors, enforcing strictly ordered outcome arrays at factory and market layers, and expanding tests that cover resolver default/closing windows, parked-fee redistribution, and NFT-owner refunds. Final verdict: proceed to broader deployment only after the Major items are demonstrably remediated and regression-tested, while continuing periodic reviews as System features evolve toward production readiness.

It is important to note that a security audit is not a guarantee of absolute security but rather a snapshot of the system's security posture at a specific point in time. While we strive to uncover all potential issues, the evolving nature of blockchain technology and DeFi means new vulnerabilities can emerge.

3 System overview

This section provides a high-level overview of the System for creating, operating, and settling decentralized prediction markets where each market is represented by upgradeable contracts and tokenized ownership records. We were asked to review:

- Original Code
- Code with Fixes

And the following files:

/			
	XOMarketConfiguration Manager.sol	XOMarketFactoryV1.sol	XOMarketV1.sol
	XOFeeHandler.sol		
utils/			
	LSLMSRDynamic.sol		
tokens/			
	XOMarketOutcome.sol	XOMarketNFT.sol	
storage/			
	XOMarketV1Storage.sol	XOMarketOutcome Storage.sol	XOMarketNFT Storage.sol
	XOMarketConfiguration ManagerStorage.sol	XOMarketFactory V1Storage.sol	XOFeeHandler Storage.sol

Our team assessed how `XOMarketFactoryV1` deploys market instances, how `XOMarketV1` governs individual markets, and how surrounding modules orchestrate collateral intake, resolver-driven lifecycle changes, and payout mechanics. The System allows a creator who possesses gating credentials to launch a market with predefined outcomes, deposit ERC20 collateral, and mint an ERC721 market NFT that identifies ownership. Participants acquire ERC1155 outcome shares through automated market maker pricing driven by the `LSLMSRDynamic` library, while the same library underpins sales and redemptions, ensuring that pricing stays linked to total supply and configured alpha parameters. The System also defines comprehensive post-resolution flows: resolvers can pause, approve, or cancel markets, and once a result or void decision is recorded, the contracts compute per share redemption values so that holders of winning or voided outcome tokens can reclaim collateral. Fee routing is explicit; portions of each trade are split among creators, the protocol treasury, and a specialized fee handler that tracks user entitlements before transferring rewards. Overall, the System aims to guarantee collateral safety, enforce resolver authority, and provide transparent token accounting so that each market can begin in a pending state, move to active trading, and

eventually conclude with either payout or refund paths depending on the resolver's decision.

The technology stack relies entirely on Solidity version 0.8.28, which supplies built-in overflow checks and supports the upgradeable patterns present across the project. Every core contract inherits from OpenZeppelin upgradeable base classes such as `AccessManagedUpgradeable`, `UUPSUpgradeable`, `ReentrancyGuardUpgradeable`, and the ERC token standards, ensuring standardized access control, proxy upgrade authorization, and guardrails against reentrancy during stateful operations. Token interfaces include ERC20 for collateral, ERC721 for market ownership, and ERC1155 for outcome shares, each integrated through OpenZeppelin's implementations. Mathematical operations that require higher precision, such as cost curve exponentials, use the ABDK `ABDKMathQuad` library to deliver quad precision floating point support within the `LSLMSRDynamic` module, enabling smooth price curves even when supplies fluctuate dramatically. Storage layouts for upgradeable contracts follow the EIP-7201 namespacing technique, visible in files like `XOMarketV1Storage` and `XOFeeHandlerStorage`, preventing slot collisions across upgrades. The repository itself contains Solidity sources without auxiliary build files, so our review references the standard Ethereum development toolchain used alongside OpenZeppelin upgrades rather than a bespoke IDE; this minimalistic structure nonetheless demonstrates consistent pragma directives, modular directory separation (core contracts, token contracts, storage definitions, utilities), and dependency management via interface imports and library references. External connectors such as `IXOAchievements` are included via interface references under a periphery folder, indicating that the System expects to be compiled within an environment where those interfaces are resolved, typically through a package manager that mirrors OpenZeppelin's structure.

Core components revolve around a layered governance and market lifecycle design that can be explained without delving into low level Solidity details. At the top sits `XOMarketConfigurationManager`, a controller that administrators use to set protocol fees, treasury addresses, allowable collateral tokens, resolver permissions, pause status, and timing knobs such as resolution windows. Whenever a market is created, `XOMarketFactoryV1` consults this manager, verifies that the creator owns the necessary achievement credential, and deploys a new market instance behind a beacon proxy so upgrades to logic can be rolled out uniformly. Ownership of each market instance is captured by `XOMarketNFT`, an ERC721 collection where each token ID deterministically encodes the market address and chain ID; this NFT is used to check who may respond to update requests or cancel a pending market. Trading relies on `XOMarketOutcome`, an ERC1155 token contract that mints non-transferable outcome shares for each market, ensuring that only the market itself can move tokens when unlocking creator stakes after closure. The System further integrates `XOFeeHandler`, which tracks how many outcome shares each trader holds and distributes the user fee portion as additional collateral claims that can be withdrawn once accrued. External interactions include reading ERC20 balances for collateral transfers, querying ERC20 metadata to derive precision factors, communicating with resolvers specified in the manager, and, when applicable, referencing the achievements contract that gates factory access. Finally, the `LSLMSRDynamic` library glues the experience together by calculating purchase costs, sale returns, prices, and creation costs as functions of total supply and customizable alpha curves, allowing the System to offer adaptive liquidity that can be tuned per market through configuration parameters without rewriting market logic.

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check whether the code actually does what it is supposed to do, whether the algorithms are optimal and correct, and whether proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Recommendations** cover code style, best practices and general improvements.

5 Our findings

We found 1 major, and a few less important issues. All identified Major issues have been fixed.

Fixed 1 out of 1 issues

6 Major Issues

CVF-3 FIXED

- **Category** Unclear behavior
- **Source** LSLMSRDynamic.sol

Description There is no check to ensure elements or “outcomes” array are unique.

Recommendation Require element of the “outcomes” array to go in strictly ascending order.

```
239 function calculatePurchaseCost(uint256[] memory supplies, uint8[]
    ↪ memory outcomes, uint256[] memory amounts, AlphaCurveConfig
    ↪ memory config)
```

7 Moderate Issues

CVF-1 FIXED

- **Category** Unclear behavior
- **Source** XOFeeHandler.sol

Description All these fees will be distributed to future shareholders (probably to the very first future shareholder), who didn't actually hold shares at the time these fees were earned. This is unfair.

Recommendation Consider giving these fees to the protocol owners or just don't charge fees unless there are some shareholders.

Client Comment *We have changed the mechanism.*

```
89 marketData.undistributedFees += feeAmount; // park until someone
   ↪ holds shares
```

CVF-2 FIXED

- **Category** Overflow/Underflow
- **Source** LSLMSRDynamic.sol

Description Overflow is possible when converting to "int256".

Recommendation Use safe conversion.

```
96 if (totalSupply <= config.T1) return -int256(alphaBps * config.c1_fp
   ↪ ) / int256(1_000_000 * totalSupply);
   return -int256(alphaBps * config.c2_fp) / int256(1_000_000 *
   ↪ totalSupply);
```

```
107 int256 bPrimeBpsValue = int256(alphaBps) + int256(totalSupply) *
    ↪ alphaPrimeBps;
```

CVF-4 INFO

- **Category** Suboptimal
- **Source** XOMarketV1.sol

Description Calculating this value on each invocation is suboptimal.

Recommendation Calculate once per market and store in market data.

335
```
uint256 precisionFactor = _getPrecisionFactor($.marketData.
    ↪ collateralToken);
```

450
```
uint256 precisionFactor = _getPrecisionFactor($.marketData.
    ↪ collateralToken);
```

619
```
uint256 precisionFactor = _getPrecisionFactor($.marketData.
    ↪ collateralToken);
```

CVF-5 INFO

- **Category** Suboptimal
- **Source** LSLMSRDynamic.sol

Description Converting configuration parameters from uint into quad-precision floating point numbers on every function invocation is suboptimal.

Recommendation Store configuration parameters as quad-precision floating point number to avoid excessive conversions.

80
```
bytes16 c1 = ABDKMathQuad.div(ABDKMathQuad.fromUInt(config.c1_fp),
    ↪ ABDKMathQuad.fromUInt(1_000_000));
bytes16 ratio = ABDKMathQuad.div(ABDKMathQuad.fromUInt(config.T0),
    ↪ ABDKMathQuad.fromUInt(totalSupply));
```

83
```
return ABDKMathQuad.toUInt(ABDKMathQuad.mul(ABDKMathQuad.fromUInt(
    ↪ config.alpha0Bps), power));
```

85
```
bytes16 c2 = ABDKMathQuad.div(ABDKMathQuad.fromUInt(config.c2_fp),
    ↪ ABDKMathQuad.fromUInt(1_000_000));
bytes16 ratio = ABDKMathQuad.div(ABDKMathQuad.fromUInt(config.T1),
    ↪ ABDKMathQuad.fromUInt(totalSupply));
```

88
```
return ABDKMathQuad.toUInt(ABDKMathQuad.mul(ABDKMathQuad.fromUInt(
    ↪ config.alpha1Bps), power));
```

CVF-6 INFO

- **Category** Suboptimal
- **Source** LSLMSRDynamic.sol

Description Converting constant uint values into quad-precision floating point numbers on-chain is waste of gas.

Recommendation Precompute constant quad-precision numbers.

Client Comment *B is not a constant so we can't store it.*

```
80    bytes16 c1 = ABDKMathQuad.div(ABDKMathQuad.fromUInt(config.c1_fp
         ↪ ), ABDKMathQuad.fromUInt(1_000_000));
```

```
85    bytes16 c2 = ABDKMathQuad.div(ABDKMathQuad.fromUInt(config.c2_fp
         ↪ ), ABDKMathQuad.fromUInt(1_000_000));
```

```
150 bytes16 liquidityQuad = ABDKMathQuad.fromUInt(B);
bytes16 maxBpsQuad = ABDKMathQuad.fromUInt(MAX_BPS);
```

```
153 bytes16 zero = ABDKMathQuad.fromUInt(0);
```

```
156 bytes16 sumExponentials = ABDKMathQuad.fromUInt(0);
bytes16 sumQuantityExponentials = ABDKMathQuad.fromUInt(0);
```

```
172   ABDKMathQuad.mul(sumQuantityExponentials, ABDKMathQuad.div(
         ↪ ABDKMathQuad.fromUInt(1), ABDKMathQuad.mul(liquidityQuad,
         ↪ sumExponentials)));
```

CVF-7 INFO

- **Category** Suboptimal
- **Source** XOMarketV1.sol

Description Executing a heavy operation just to calculate event parameters is a bad practice, as these calculations could be performed off-chain.

Recommendation Refactor the code to avoid excessive heavy calculations on-chain.

```
163 // heavy op, we will remove later once we do better indexing
uint256[] memory outcomePrices = getPrices();
emit MarketPriceChanged(outcomePrices);
```

```
199 // heavy op, we will remove later once we do better indexing
200 uint256[] memory outcomePrices = getPrices();
emit MarketPriceChanged(outcomePrices);
```

CVF-8 INFO

- **Category** Suboptimal
- **Source** XOMarketV1.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

132
```
function buy(uint8[] calldata outcomes, uint256[] calldata amounts,
    ↪ uint256 maxCost) external nonReentrant marketActive
    ↪ whenNotPaused {
```

169
```
function sell(uint8[] calldata outcomes, uint256[] calldata amounts,
    ↪  uint256 minReturn) external nonReentrant marketActive
    ↪ whenNotPaused {
```

CVF-9 FIXED

- **Category** Suboptimal
- **Source** LSLMSRDynamic.sol

Recommendation It would be more efficient to use the “pow_2” and “log_2” functions instead of “exp” and “ln”.

73
```
return ABDKMathQuad.exp(ABDKMathQuad.mul(expo, ABDKMathQuad.ln(base)
    ↪ ));
```

CVF-10 FIXED

- **Category** Suboptimal
- **Source** XOFeeHandler.sol

Recommendation Parked fees should be distributed not when more fees are received (which could take a long time to happen), but rather when the first shareholder appears. This would also make processing incoming fees cheaper, as this check wouldn’t be required.

94
```
// fold in parked fees
```

96
```
if (marketData.undistributedFees != 0) {
    totalFeesDistributed += marketData.undistributedFees;
```

CVF-11 INFO

- **Category** Suboptimal
- **Source** LSLMSRDynamic.sol

Description Performing this division every time is suboptimal.

Recommendation Store the "c1" configuration parameter already scaled down by 1 million.

```
80 bytes16 c1 = ABDKMathQuad.div(ABDKMathQuad.fromUInt(config.c1_fp),
      ↪ ABDKMathQuad.fromUInt(1_000_000));
```

CVF-12 FIXED

- **Category** Overflow/Underflow
- **Source** LSLMSRDynamic.sol

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type while certain intermediary calculation overflows.

Recommendation Use the "mulDiv" function.

```
96 if (totalSupply <= config.T1) return -int256(alphaBps * config.c1_fp
      ↪ ) / int256(1_000_000 * totalSupply);
   return -int256(alphaBps * config.c2_fp) / int256(1_000_000 *
      ↪ totalSupply);
```

CVF-13 FIXED

- **Category** Overflow/Underflow
- **Source** LSLMSRDynamic.sol

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type while certain intermediary calculation overflows.

Recommendation Use the "mulDiv" functipn.

```
102 return (alphaBps * totalSupply) / MAX_BPS;
```

CVF-14 FIXED

- **Category** Overflow/Underflow
- **Source** XOMarketConfigurationManager.sol

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, but certain intermediary calculation overflows.

Recommendation Use the “mulDiv” function.

```
181 uint256 numerator = uint256(config.maxCreatorFeeBps - config.
    ↪ minCreatorFeeBps) * uint256(initialAmount - config.
    ↪ minInitialCollateralAmount);
    uint256 denominator = uint256(config.maxFeeCollateralAmount - config
    ↪ .minInitialCollateralAmount);
```

```
184 return config.minCreatorFeeBps + uint16(numerator / denominator);
```

CVF-15 FIXED

- **Category** Unclear behavior
- **Source** XOMarketV1.sol

Description There is no check to ensure the “outcomes” and “amounts” arrays are of the same length. If the “amounts” array is longer, extra elements are silently ignored.

Recommendation Implement a proper length check.

Client Comment *The functions rely on the calculatePurchaseCost and calculateSaleReturn from LSLMSRDynamic which does the check so no need to replicate it.*

```
132 function buy(uint8[] calldata outcomes, uint256[] calldata amounts,
    ↪ uint256 maxCost) external nonReentrant marketActive
    ↪ whenNotPaused {
```

```
169 function sell(uint8[] calldata outcomes, uint256[] calldata amounts,
    ↪  uint256 minReturn) external nonReentrant marketActive
    ↪ whenNotPaused {
```

CVF-16 FIXED

- **Category** Suboptimal
- **Source** XOFeeHandler.sol

Description This function duplicates logic of the “previewClaimable” function. This increase the code size and also two implementation of the same logic could easily get inconsistent with each other.

Recommendation Use the “previewClaimable” function inside the “_settleUserFees” function to calculate the new accrued fees amount.

```
154 function _settleUserFees(address marketAddress, address userAddress)
      ↪  internal {
```

CVF-17 FIXED

- **Category** Procedural
- **Source** XOMarketConfiguration-ManagerStorage.sol

Recommendation This TODO should be resolved or removed.

```
12 // TODO update the storage slot here
```

8 Recommendations

CVF-18 FIXED

- **Category** Readability
- **Source** LSLMSRDynamic.sol

Recommendation Brackets are redundant.

```
102 return (alphaBps * totalSupply) / MAX_BPS;
```

CVF-19 FIXED

- **Category** Readability
- **Source** XOMarketV1.sol

Recommendation Brackets are redundant.

```
308 MarketStatus closedStatus = (status == MarketStatus.VOIDED) ?
    ↪ MarketStatus.VOIDED_CLOSED : MarketStatus.CLOSED;
```

CVF-20 INFO

- **Category** Procedural
- **Source** LSLMSRDynamic.sol

Description Consider specifying as "^0.8.0' unless there is something special regarding this particular version.

Recommendation See also: XOMarketNFT.sol, XOMarketOutcome.sol, XOFeeHandlerStorage.sol, XOMarketConfigurationManagerStorage.sol, XOMarketFactoryV1Storage.sol, XOMarketNFTStorage.sol, XOMarketOutcomeStorage.sol, XOMarketV1Storage.sol, XOFeeHandler.sol, XOMarketConfigurationManager.sol, XOMarketFactoryV1.sol, XOMarketV1.sol.

```
2 pragma solidity ^0.8.28;
```

CVF-21 INFO

- **Category** Readability
- **Source** LSLMSRDynamic.sol

Description Declaring a top-level structure in a file named after a library makes it harder navigating through code.

Recommendation Move the structure declaration into the library or move it into a separate file.

```
42 struct AlphaCurveConfig {
```

CVF-22 INFO

- **Category** Procedural
- **Source** XOMarketConfigurationManager.sol

Description In ERC-20 the “decimals” property is used by UI to render token amounts in a human-readable way. Using this property in smart contracts is discouraged.

Recommendation Treat all token amounts as integers.

Client Comment *It’s an extra harmless check.*

```
143 uint8 decimals = IERC20Metadata(token).decimals();
    require(decimals > 0 && decimals <= 18, InvalidPrecisionFactor());
```

CVF-23 FIXED

- **Category** Procedural
- **Source** XOMarketV1.sol

Description In ERC-20 the “decimals” property is used by UI to render token amounts in a human-readable way. Using this property in smart contracts is discouraged.

Recommendation Threat all token amounts as integers.

Client Comment *We need to rely on this. We can probably set it as part of xomarketconfig manager collateralconfig at some point.*

```
500 return 10 ** IERC20Metadata(token).decimals();
```

CVF-24 FIXED

- **Category** Unclear behavior
- **Source** XOMarketConfigurationManager.sol

Description It is unclear, why decimals=0 is forbidden, while decimals=1 is not.

Recommendation Consider allowing decimals=0.

Client Comment *Its to prevent any tokens that dont set clear decimals.*

```
144 require(decimals > 0 && decimals <= 18, InvalidPrecisionFactor());
```

CVF-25 INFO

- **Category** Suboptimal
- **Source** XOFeeHandlerStorage.sol

Recommendation It would be more efficient to merge these two mappings into a single mapping whose keys are markets and values are structs encapsulating the values of the original mappings.

```
14 mapping(address => IXOFeeHandler.MarketInfo) marketInfo; // market
      ↪ -> info
   mapping(address => mapping(address => IXOFeeHandler.UserFeeState))
      ↪ userFeeState; // market -> user -> state
```

CVF-26 INFO

- **Category** Suboptimal
- **Source** LSLMSRDynamic.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

```
239 function calculatePurchaseCost(uint256[] memory supplies, uint8[]
      ↪ memory outcomes, uint256[] memory amounts, AlphaCurveConfig
      ↪ memory config)
```

```
275 function calculateSaleReturn(uint256[] memory supplies, uint8[]
      ↪ memory outcomes, uint256[] memory amounts, AlphaCurveConfig
      ↪ memory config)
```

CVF-27 INFO

- **Category** Procedural
- **Source** XOFeeHandlerStorage.sol

Description Solidity compiler is smart enough to precompute constant hash expressions.
Recommendation Use hash expression instead of a hardcoded hash value.

```
10 // keccak256(abi.encode(uint256(keccak256("xo.storage.XOFeeHandler")
      ↪ ) - 1)) & ~bytes32(uint256(0xff))
   bytes32 private constant _XO_FEE_HANDLER_STORAGE = 0
      ↪ x160370c7e284d0827b77fcc3d240a4d074282593ca841d0d94b9eb72da6ea200
      ↪ ;
```

CVF-28 INFO

- **Category** Procedural
- **Source** XOMarketConfiguration-ManagerStorage.sol

Description Solidity compiler is smart enough to precompute constant hash expressions.
Recommendation Use hash expression instead of a hardcoded hash value.

```
11 // keccak256(abi.encode(uint256(keccak256("xo.storage.
   ↪ XOMarketConfigurationManager")) - 1)) & ~bytes32(uint256(0xff)
   ↪ )

13 bytes32 private constant _MARKET_CONFIG_STORAGE =
0x5cbcaf8ebf78e743328d584c6a2f123bdeabff0870c1f64d063cfc37bb8bbb00;
```

CVF-29 INFO

- **Category** Procedural
- **Source** XOMarketFactoryV1Storage.sol

Description Solidity compiler is smart enough to precompute constant hash expressions.
Recommendation Use hash expression instead of a hardcoded hash value.

```
11 // keccak256(abi.encode(uint256(keccak256("xo.storage.
   ↪ XOMarketFactoryV1")) - 1)) & ~bytes32(uint256(0xff))
bytes32 private constant FACTORY_STORAGE_LOCATION =
0x4f78989477789e6f561a02ca95cbb365eaa62df76dfa49fbcd6a47dbabc0a500;
```

CVF-30 INFO

- **Category** Procedural
- **Source** XOMarketV1Storage.sol

Description Solidity compiler is smart enough to precompute constant hash expressions.
Recommendation Use hash expression instead of a hardcoded hash value.

```
15 // keccak256(abi.encode(uint256(keccak256("xo.storage.XOMarketV1"))
   ↪ - 1)) & ~bytes32(uint256(0xff))
bytes32 private constant _XO_MARKET_V1_STORAGE =
0x569c277e858c3078813a7329c153fd3911089693d00b9126033cc60b603d5d00;
```

CVF-31 FIXED

- **Category** Suboptimal
- **Source** XOMarketConfigurationManager.sol

Description Specifying constant values in comments is a bad practice as values in comments could easily get out of sync with actual values in the code.

Recommendation Don't specify constant values in comments.

```
16 /// @dev One percent in basis points (100 = 1%)
```

```
86     require(feeBps <= _ONE_PERCENT_BPS * 5, FeeExceedsMaximum()); //
          ↪  Max 5%
```

```
98     require(feeBps <= _ONE_PERCENT_BPS * 50, FeeExceedsMaximum());
          ↪ // Max 50% of the creator fee
```

```
158    require(config.feeBps <= _ONE_PERCENT_BPS, FeeExceedsMaximum());
          ↪  // Max 1%
```

CVF-32 FIXED

- **Category** Bad naming
- **Source** XOMarketV1.sol

Recommendation The "_xoMarkets" and "_xOutcomes" arguments should be named as singulars: "_xoMarket" and "_xOutcome".

```
42 constructor(address _config, address _xoMarkets, address _xOutcomes,
      ↪  address _xoFeeHandler) {
```

CVF-33 FIXED

- **Category** Suboptimal
- **Source** XOMarketOutcome.sol

Description The "from" and "to" addresses are compared to zero twice.

Recommendation Refactor the code to compare only once.

```
91 if (from != address(0) && to != address(0)) {
```

```
102 require(from == address(0) || to == address(0) || isMarketTransfer,
      ↪ "XOMarketOutcome:␣token␣transfers␣are␣not␣allowed");
```

CVF-34 FIXED

- **Category** Bad naming
- **Source** XOMarketV1.sol

Description The argument names doesn't match corresponding internal variable names.

Recommendation Make argument names and internal variable names consistent with each other.

```
43 MARKET_CONFIG_MANAGER = IXOMarketConfigurationManager(_config);
   MARKET_NFT = IXOMarketNFT(_xoMarkets);
   MARKET_OUTCOME = IXOMarketOutcome(_xOutcomes);
```

CVF-35 INFO

- **Category** Bad datatype
- **Source** XOMarketConfigurationManager.sol

Recommendation The argument type should be "IERC20".

```
75 function collateralConfig(address token) external view returns (
      ↪ IXOMarketConfigurationManager.CollateralConfig memory) {
```

CVF-36 FIXED

- **Category** Bad datatype
- **Source** XOMarketFactoryV1.sol

Recommendation The argument type should be more specific.

```
153 function marketExists(address market) external view returns (bool) {
```

CVF-37 FIXED

- **Category** Bad naming
- **Source** XOMarketFactoryV1.sol

Description The immutable variables names and the corresponding argument names doesn't match each other.

Recommendation Make the matching names.

```
53 MARKET_CONFIG_MANAGER = IXOMarketConfigurationManager(config);
```

```
55 MARKET_NFT = IXOMarketNFT(xoMarkets);
   MARKET_OUTCOME = IXOMarketOutcome(xOutcomes);
```

CVF-38 FIXED

- **Category** Bad datatype
- **Source** XOMarketFactoryV1Storage.sol

Recommendation The key type for this mapping should be more specific.

```
15 mapping(address => IXOMarketFactoryV1.MarketRegistryData)
      ↪ marketRegistry; // market address => market registry data
```

CVF-39 FIXED

- **Category** Bad datatype
- **Source** XOMarketConfigurationManager.sol

Recommendation The maximum fee value should be a named constant.

```
86 require(feeBps <= _ONE_PERCENT_BPS * 5, FeeExceedsMaximum()); // Max
      ↪  5%
```

CVF-40 FIXED

- **Category** Bad datatype
- **Source** XOMarketConfigurationManager.sol

Recommendation The maximum user fee value should be a named constant.

```
98 require(feeBps <= _ONE_PERCENT_BPS * 50, FeeExceedsMaximum()); //
      ↪ Max 50% of the creator fee
```

CVF-41 FIXED

- **Category** Bad naming
- **Source** XOMarketFactoryV1.sol

Recommendation The names “xoMarkets” and “xOutcomes” should be singular: “xoMarket” and “xOutcome”.

```
52 constructor(address config, address marketBeacon, address xoMarkets,
      ↪  address xOutcomes, address xoAchievements, address
      ↪ xoFeeHandler) {
```

CVF-42 FIXED

- **Category** Documentation
- **Source** XOMarketConfiguration-ManagerStorage.sol

Description The semantics of the keys in these mappings is unclear.

Recommendation Give descriptive names to the keys and/or explain in a documentation comment.

```
25 mapping(address => IXOMarketConfigurationManager.CollateralConfig)
      ↪ collateralTokens;
   mapping(address => IXOMarketConfigurationManager.ResolverConfig)
      ↪ resolvers;
```

CVF-43 INFO

- **Category** Bad datatype
- **Source** XOMarketV1.sol

Recommendation The type for the "_config" argument should be "IXOMarketConfigurationManager".

```
42 constructor(address _config, address _xoMarkets, address _xOutcomes,
      ↪  address _xoFeeHandler) {
```

CVF-44 INFO

- **Category** Bad datatype
- **Source** XOMarketV1.sol

Recommendation The type for the "_xoFeeHandler" argument should be "IXOFeeHandler".

```
42 constructor(address _config, address _xoMarkets, address _xOutcomes,
      ↪  address _xoFeeHandler) {
```

CVF-45 INFO

- **Category** Bad datatype
- **Source** XOMarketV1.sol

Recommendation The type for the "_xoMarkets" argument should be IXOMarketNFT".

```
42 constructor(address _config, address _xoMarkets, address _xOutcomes,
      ↪  address _xoFeeHandler) {
```

CVF-46 INFO

- **Category** Bad datatype
- **Source** XOMarketV1.sol

Recommendation The type for the "_xOutcomes" argument should be "IXOMarketOutcome".

```
42 constructor(address _config, address _xoMarkets, address _xOutcomes,
   ↪  address _xoFeeHandler) {
```

CVF-47 INFO

- **Category** Bad datatype
- **Source** XOFeeHandler.sol

Recommendation The type for the "collateralToken" should be "IERC20".

```
56     returns (bool isEnabled, address collateralToken, uint256
       ↪ accumulatedFeePerShareX18, uint256 totalOutstandingShares,
       ↪  uint256 undistributedFees)
```

```
71 function registerMarket(address marketAddress, address
   ↪ collateralToken) external onlyFactory {
```

CVF-48 INFO

- **Category** Bad datatype
- **Source** XOMarketFactoryV1.sol

Recommendation The type for the "config" argument should be "IXOMarketConfigurationManager".

```
52 constructor(address config, address marketBeacon, address xoMarkets,
   ↪  address xOutcomes, address xoAchievements, address
   ↪ xoFeeHandler) {
```

CVF-49 INFO

- **Category** Bad datatype
- **Source** XOMarketConfigurationManager.sol

Recommendation The type for the "market" argument should be "IXOMarketV1".

```
92 function setProtocolFeeOverride(address market, bool overrideEnabled
   ↪ , uint16 protocolFeeBpsOverride) external restricted {
```

CVF-50 INFO

- **Category** Bad datatype
- **Source** XOFeeHandler.sol

Recommendation The type for the “marketAddress” should be more specific.

```
64  function userFeeState(address marketAddress, address userAddress)
       ↪ external view returns (uint256 lastFeeIndexX18, uint256
       ↪ accruedFees, uint256 userShares) {
```

```
71  function registerMarket(address marketAddress, address
       ↪ collateralToken) external onlyFactory {
```

```
126 function claim(address marketAddress, address recipient) external
       ↪ nonReentrant restricted returns (uint256 claimedAmount) {
```

```
139 function previewClaimable(address marketAddress, address userAddress
       ↪ ) external view returns (uint256) {
```

```
154 function _settleUserFees(address marketAddress, address userAddress)
       ↪  internal {
```

CVF-51 INFO

- **Category** Bad datatype
- **Source** XOMarketFactoryV1.sol

Recommendation The type for the “marketBeacon” argument should be more specific.

```
52  constructor(address config, address marketBeacon, address xoMarkets,
       ↪  address xOutcomes, address xoAchievements, address
       ↪ xoFeeHandler) {
```

CVF-52 INFO

- **Category** Bad datatype
- **Source** XOMarketConfigurationManager.sol

Recommendation The type for the “token” argument should be “IERC20”.

```
141 function setCollateralTokenConfig(address token,
       ↪ IXOMarketConfigurationManager.CollateralConfig memory config)
       ↪ external restricted {
```

CVF-53 INFO

- **Category** Bad datatype
- **Source** XOMarketConfigurationManager.sol

Recommendation The type for the “token” argument should be “IERC20”.

```
165 function calculateAllowedCreatorFeeBps(address token, uint128
    ↪ initialAmount) external view returns (uint16 allowedFeeBps) {
```

CVF-54 INFO

- **Category** Bad datatype
- **Source** XOMarketFactoryV1.sol

Recommendation The type for the “xoAchievements” argument should be “IXOAchievements”.

```
52 constructor(address config, address marketBeacon, address xoMarkets,
    ↪  address xOutcomes, address xoAchievements, address
    ↪ xoFeeHandler) {
```

CVF-55 INFO

- **Category** Bad datatype
- **Source** XOMarketFactoryV1.sol

Recommendation The type for the “xoFeeHandler” argument should be “IXOFeeHandler”.

```
52 constructor(address config, address marketBeacon, address xoMarkets,
    ↪  address xOutcomes, address xoAchievements, address
    ↪ xoFeeHandler) {
```

CVF-56 INFO

- **Category** Bad datatype
- **Source** XOMarketFactoryV1.sol

Recommendation The type for the “xoMarkets” argument should be “IXOMarketNFT”.

```
52 constructor(address config, address marketBeacon, address xoMarkets,
    ↪  address xOutcomes, address xoAchievements, address
    ↪ xoFeeHandler) {
```

CVF-57 INFO

- **Category** Bad datatype
- **Source** XOMarketFactoryV1.sol

Recommendation The type for the “xOutcomes” argument should be “IXOMarketOutcome”.

```
52 constructor(address config, address marketBeacon, address xoMarkets,
   ↪  address xOutcomes, address xoAchievements, address
   ↪ xoFeeHandler) {
```

CVF-58 INFO

- **Category** Bad datatype
- **Source** XOFeeHandlerStorage.sol

Recommendation The type for the first keys on these mappings should be more specific.

```
14 mapping(address => IXOFeeHandler.MarketInfo) marketInfo; // market
   ↪ -> info
   mapping(address => mapping(address => IXOFeeHandler.UserFeeState))
   ↪ userFeeState; // market -> user -> state
```

CVF-59 INFO

- **Category** Bad datatype
- **Source** XOMarketFactoryV1.sol

Recommendation The type for this argument should be “IERC20”.

```
77 address collateralToken,
```

CVF-60 INFO

- **Category** Bad datatype
- **Source** XOMarketFactoryV1.sol

Recommendation The type for this variable should be “IXOMarketV1”.

```
89 address marketAddress = address(proxy);
```

CVF-61 INFO

- **Category** Bad datatype
- **Source** XOMarketFactoryV1.sol

Recommendation The type for this variable should be more specific.

```
32 address public immutable MARKET_BEACON;
```

CVF-62 FIXED

- **Category** Bad datatype
- **Source** LSLMSRDynamic.sol

Recommendation The value “1_000_000” should be a named constant.

```
80 bytes16 c1 = ABDKMathQuad.div(ABDKMathQuad.fromUInt(config.c1_fp),
   ↪ ABDKMathQuad.fromUInt(1_000_000));
```

CVF-63 INFO

- **Category** Suboptimal
- **Source** XOFeeHandler.sol

Description These checks are redundant, as it is anyway possible to pass dead addresses.
Recommendation Remove this check.
Client Comment *We prefer to keep them, harmless checks.*

```
72 require(marketAddress != address(0) && collateralToken != address(0)
   ↪ , "ZeroAddr");
```

```
129 require(recipient != address(0), "ZeroRecipient");
```

```
140 require(marketAddress != address(0) && userAddress != address(0), "
   ↪ ZeroAddr");
```

CVF-64 FIXED

- **Category** Suboptimal
- **Source** LSLMSRDynamic.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
64 error UnsafeSchedule_BPrimeNegative();
```

```
67 error InvalidOutcome();
   error InsufficientSupply();
   error InvalidAmount();
```

CVF-65 INFO

- **Category** Unclear behavior
- **Source** XOMarketConfigurationManager.sol

Description These events are emitted even if nothing actually changed.

```
88 emit ProtocolFeeSet(feeBps);
```

```
100 emit UserFeeSet(feeBps);
```

```
107 emit TreasurySet(_treasury);
```

```
113 emit PrivateMarketsEnabledSet(enabled);
```

```
119 emit ResolutionDefaultPeriodSet(period);
```

```
125 emit ResolutionClosingPeriodSet(period);
```

```
131 emit ResolutionPeriodSet(period);
```

```
137 emit PauseStatusSet(paused);
```

CVF-66 INFO

- **Category** Suboptimal
- **Source** XOMarketV1.sol

Recommendation This check is redundant, as it is anyway possible to pass a dead resolver address.

```
81 require(p.resolver != address(0), InvalidParameter());
```

CVF-67 INFO

- **Category** Suboptimal
- **Source** XOMarketConfigurationManager.sol

Recommendation This check is redundant, as it is anyway possible to pass a dead treasury address.

```
105 require(_treasury != address(0), InvalidAddress());
```

```
142 require(token != address(0), InvalidAddress());
```

CVF-68 INFO

- **Category** Suboptimal
- **Source** XOMarketConfigurationManager.sol

Description This check seems redundant, as it is anyway possible to pass a dead resolver address.
Recommendation Remove this check.

```
157 require(resolver != address(0), InvalidAddress());
```

CVF-69 INFO

- **Category** Procedural
- **Source** XOFeeHandler.sol

Recommendation This check should be done earlier.

```
84 if (feeAmount == 0) return;
```

CVF-70 INFO

- **Category** Procedural
- **Source** XOFeeHandler.sol

Recommendation This check should be performed earlier.

```
119 require(userState.userShares >= shareAmount, "InsufficientShares");
```

CVF-71 INFO

- **Category** Procedural
- **Source** XOFeeHandler.sol

Recommendation This check should be performed earlier.

```
132 if (claimedAmount == 0) return 0;
```

CVF-72 FIXED

- **Category** Suboptimal
- **Source** XOMarketV1.sol

Description This condition was already checked and is guaranteed to be true here.
Recommendation Refactor the code to not check this condition again.

```
346 } else if ($.marketData.status == MarketStatus.VOIDED_CLOSED) {
```

CVF-73 FIXED

- **Category** Suboptimal
- **Source** XOMarketOutcome.sol

Recommendation This could be replaced with: require (FACTORY.marketExists(from)) Such replacement would make the require statement below unnecessary.

```
95  isMarketTransfer = FACTORY.marketExists(from);
    if (!isMarketTransfer) {
        break;
    }
```

CVF-74 INFO

- **Category** Unclear behavior
- **Source** XOMarketV1.sol

Description This event is emitted even if nothing actually changed.

```
642 emit MarketStatusUpdated(newStatus);
```

CVF-75 FIXED

- **Category** Readability
- **Source** XOMarketConfigurationManager.sol

Recommendation This value could be rendered as “0.01e4”.

```
17 uint256 internal constant _ONE_PERCENT_BPS = 1e2;
```

CVF-76 INFO

- **Category** Bad naming
- **Source** XOMarketFactoryV1.sol

Description UPPER_CASE identifiers are commonly used for constants, not for immutable variables.

Recommendation Use camelCase instead.

Client Comment *We use upper case for immutables and upper case starting with _ for constants.*

```
31 IXOMarketConfigurationManager public immutable MARKET_CONFIG_MANAGER
   ↪ ;
address public immutable MARKET_BEACON;
IXOMarketNFT public immutable MARKET_NFT;
IXOMarketOutcome public immutable MARKET_OUTCOME;
IXOAchievements public immutable XO_ACHIEVEMENTS;
IXOFeeHandler public immutable XO_FEE_HANDLER;
```

CVF-77 INFO

- **Category** Bad naming
- **Source** XOMarketV1.sol

Description UPPER_CASE identifiers are commonly used for constants, rather than for immutable variables.

Recommendation Use camelCase instead.

Client Comment *We use upper case for immutables and upper case starting with _ for constants.*

```
36 IXOMarketConfigurationManager public immutable MARKET_CONFIG_MANAGER
   ↪ ;
IXOMarketNFT public immutable MARKET_NFT;
IXOMarketOutcome public immutable MARKET_OUTCOME;
IXOFeeHandler public immutable XO_FEE_HANDLER;
```

CVF-78 INFO

- **Category** Procedural
- **Source** XOMarketNFT.sol

Description We didn't review these files.

```
9  import { IXOMarketNFT } from "../interfaces/IXOMarketNFT.sol";
10 import { IXOMarketFactoryV1 } from "../interfaces/IXOMarketFactoryV1
   ↪ .sol";
```

CVF-79 INFO

- **Category** Procedural
- **Source** XOMarketOutcome.sol

Description We didn't review these files.

```
8  import { IXOMarketOutcome } from "../interfaces/IXOMarketOutcome.sol
   ↪ ";
   import { IXOMarketFactoryV1 } from "../interfaces/IXOMarketFactoryV1
   ↪ .sol";
```

CVF-80 INFO

- **Category** Procedural
- **Source** XOMarketV1Storage.sol

Description We didn't review these files.

```
4  import { IXOMarketV1 } from "../interfaces/IXOMarketV1.sol";
   import { IXOMarketConfigurationManager } from "../interfaces/
   ↪ IXOMarketConfigurationManager.sol";
   import { IXOMarketOutcome } from "../interfaces/IXOMarketOutcome.sol
   ↪ ";
   import { IXOMarketNFT } from "../interfaces/IXOMarketNFT.sol";
```

CVF-81 INFO

- **Category** Procedural
- **Source** XOFeeHandler.sol

Description We didn't review these files.

```
10 import { IXOFeeHandler } from "./interfaces/IXOFeeHandler.sol";
   import { IXOMarketFactoryV1 } from "./interfaces/IXOMarketFactoryV1.
   ↪ sol";
```

CVF-82 INFO

- **Category** Procedural
- **Source** XOMarketFactoryV1.sol

Description We didn't review these files.

```
12 import { IXOMarketFactoryV1 } from "./interfaces/IXOMarketFactoryV1.
      ↪ sol";
   import { IXOMarketV1 } from "./interfaces/IXOMarketV1.sol";
   import { IXOMarketConfigurationManager } from "./interfaces/
      ↪ IXOMarketConfigurationManager.sol";
   import { IXOMarketNFT } from "./interfaces/IXOMarketNFT.sol";
   import { IXOMarketOutcome } from "./interfaces/IXOMarketOutcome.sol"
      ↪ ;
```

```
19 import { IXOAchievements } from "../periphery/interfaces/
      ↪ IXOAchievements.sol";
20 import { IXOFeeHandler } from "./interfaces/IXOFeeHandler.sol";
```

CVF-83 INFO

- **Category** Procedural
- **Source** XOMarketV1.sol

Description We didn't review these files.

```
13 import { IXOMarketV1 } from "./interfaces/IXOMarketV1.sol";
   import { IXOMarketConfigurationManager } from "./interfaces/
      ↪ IXOMarketConfigurationManager.sol";
   import { IXOMarketOutcome } from "./interfaces/IXOMarketOutcome.sol"
      ↪ ;
   import { IXOMarketNFT } from "./interfaces/IXOMarketNFT.sol";
   import { IXOFeeHandler } from "./interfaces/IXOFeeHandler.sol";
```

CVF-84 INFO

- **Category** Procedural
- **Source** XOFeeHandlerStorage.sol

Description We didn't review this file.

```
4 import { IXOFeeHandler } from "../interfaces/IXOFeeHandler.sol";
```

CVF-85 INFO

- **Category** Procedural
- **Source** XOMarketConfiguration-ManagerStorage.sol

Description We didn't review this file.

```
4 import { IXOMarketConfigurationManager } from "../interfaces/
    ↪ IXOMarketConfigurationManager.sol";
```

CVF-86 INFO

- **Category** Procedural
- **Source** XOMarketFactoryV1Storage.sol

Description We didn't review this file.

```
4 import { IXOMarketFactoryV1 } from "../interfaces/IXOMarketFactoryV1
    ↪ .sol";
```

CVF-87 INFO

- **Category** Procedural
- **Source** XOMarketConfigurationManager.sol

Description We didn't review this file.

```
9 import { IXOMarketV1 } from "./interfaces/IXOMarketV1.sol";
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 300 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting