Report

v. 1.0

Customer

Collection.xyz

Smart Contract Audit

CollectionSwap

3nd April 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	03.04.23	A. Zveryanskaya	Initial Draft
0.2	03.04.23	A. Zveryanskaya	Minor revision
1.0	03.04.23	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Collection.xyz is an NFT decentralized exchange (DEX) protocol that powers pool creation by grouping tokens using traits, rarity, or other attributes. Users can create pools that automatically buy and sell NFTs based on any logic or strategy, whether through premium traits, socially curated lists, or other metadata. The protocol's core contributor is Gomu, an NFT infrastructure startup.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

bonding-curves/			
	Curve.sol	ExponentialCurve.sol	LinearCurve.sol
	SigmoidCurve.sol	XykCurve.sol	
filter/			
	TokenIDFilter.sol		
lib/			
	TransferLib.sol		
pools/			
	CollectionPool.sol	CollectionPool Enumerable.sol	CollectionPool-ERC20.sol
	CollectionPoolETH.sol	CollectionPool Factory.sol	CollectionPool MissingEnumerable.sol

Then we were asked to review:

- Original Code

Files:

/			
	CollectionPoolCloner.sol	MultiPauser.sol	ReentrancyGuard.sol

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 if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if 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.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 12 major, and a few less important issues. All identified Major issues have been fixed or otherwise addressed in collaboration with the client.

	Active	Fixed
Major	3	9
Moderate	6	10
Minor	16	40

Fixed 59 out of 84 issues

6 Major Issues

CVF-1. FIXED

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

Recommendation As the particular semantics of these variables could be different for different bonding curves, consider making these variables internal, rather than public.

92
```
uint128 public spotPrice;
```

96
```
uint128 public delta;
```

CVF-2. FIXED

- **Category** Procedural
- **Source** CollectionPool.sol

Recommendation This TODO should be resolved.

876
```
// TODO update idSet or not?
```

CVF-3. FIXED

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

Description This function should return the data returned by the inner call.

999
```
function call(address payable target, bytes calldata data) external
    ↪ onlyAuthorized {
```

CVF-4. FIXED

- **Category** Suboptimal
- **Source** CollectionPool.sol

Recommendation This check should be moved out of the loop: if (is2981) { for (...) {...} } else { for (...) {...} }

```
1056 if (is2981) {
```

CVF-5. FIXED

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

Recommendation This is not how ERC-165 is supposed to be used: https://eips.ethereum.org/EIPS/eip-165#how-to-detect-if-a-contract-implements-any-given-interface

```
1124 try IERC165(owner()).supportsInterface(type(IPoolActivityMonitor).
       ↪ interfaceId) returns (bool isMonitored) {
```

CVF-6. FIXED

- **Category** Suboptimal
- **Source** ExponentialCurve.sol

Description Calculating delta^i for every i is suboptimal.

Recommendation Consider just multiplying the previous rowAmount by delta.

```
78 rawAmount = buySpotPrice.fmul(uint256(delta).fpow(i,
     ↪ FixedPointMathLib.WAD), FixedPointMathLib.WAD);
```

CVF-7. FIXED

- **Category** Suboptimal
- **Source** ExponentialCurve.sol

Description Calculating invDelta^i for every i is suboptimal.

Recommendation Consider just multiplying the previous rowAmount by invDelta.

```
150 uint256 invDeltaPowI = invDelta.fpow(i, FixedPointMathLib.WAD);
```

CVF-8. FIXED

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

Description In case the “values” array is longer than the “tokens” array, the remaining values are silently ignored.

Recommendation Consider reverting in such a case.

```
35 uint256 length = tokens.length;
```

CVF-9. FIXED

- **Category** Suboptimal
- **Source** TransferLib.sol

Description In case the “tokenIds” array is longer than the “tokens” array, the remaining token IDs are silently ignored.

Recommendation Consider reverting in such a case.

```
54 uint256 length = tokens.length;
```

CVF-10. INFO

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

Recommendation The carry fee seems redundant, as it could be factored in into the protocol and trade fees.

Client Comment *Acknowledged. Will Not Fix.*

```
87 uint256 carryFee = fees.trade.fmul(feeMultipliers.carry,
   ↪ FEE_DENOMINATOR);
```

CVF-83. INFO

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

Description No access level specified for this variable, so internal access will be used by default. Consider explicitly specifying an access level.

Recommendation Consider explicitly specifying an access level.

```
11 uint256 pauseStates;
```

CVF-84. INFO

- **Category** Suboptimal
- **Source** MultiPauser.sol

Description The index validity checks wouldn't be necessary if pause indexes would be "uint8" rather then "uint256".

```
13 modifier validIndex(uint256 index) {
```

7 Moderate Issues

CVF-11. FIXED

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

Description Underflow is possible here.

Recommendation Consider checking before the loop that there are enough NFTs.

```
35 --lastIndex;
```

CVF-12. INFO

- **Category** Procedural
- **Source** CollectionPoolFactory.sol

Description The actual checks are hidden inside CollectionPoolCloner, which we didn't review. So we cannot tell whether the checks are correct.

```
148 * @notice Checks if an address is a CollectionPool. Uses the fact
      ↪ that the pools are EIP-1167 minimal proxies.
```

CVF-13. INFO

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

Description It is not guaranteed that this check is performed for all non-trade pool, as it is performed only for “token” and “NFT” pools, but there could be pool types other than “trade”, “token”, and “NFT”.

Recommendation Consider moving this check into a separate conditional statement to be executed for all non-trade pools.

Client Comment *There’s only 4 types of pool available NFT, Sell NFT, Buy & Sell NFT, Cannot buy or sell NFT which is trivially useless. The check of token or NFT is thus equivalent to checking for all non-trading pool. It’s also not clear if we want to apply the same logic to other non-trading pools if it exists. We could either. 1. make the change to a simple if else, removing 2 ops from checking if it’s a token or nft pool 2. add an else to throw for any unknown pool types.*

233
```
// Only Trade Pools can have nonzero fee
if (_fee != 0) revert InvalidPoolParams();
```

CVF-14. INFO

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

Description It is not guaranteed that these two arguments are consistent with each other.

Recommendation Consider either calculating the merkleRoot on-chain or publishing the token IDs list off-chain.

262
```
* @param merkleRoot Merkle root representing all allowed IDs
* @param encodedTokenIDs Opaque encoded list of token IDs
```

CVF-15. INFO

- **Category** Procedural
- **Source** CollectionPool.sol

Description We didn't review "CollectionPoolCloner", while it seems to be a sensitive part of the system. Without reviewing it, it is impossible to tell whether immutable parameters schema is safe or not.

```
840 * @dev Used internally to grab pool parameters from calldata, see
      ↪ CollectionPoolCloner for technical details
```

CVF-16. FIXED

- **Category** Suboptimal
- **Source** ExponentialCurve.sol

Description These function calculates the new spot price from the current spot price. Such approach could accumulate rounding errors.

Recommendation Consider calculating the spot price from the total number of tokens ever bough minus the total number of tokens even sold.

```
11 contract ExponentialCurve is Curve, CurveErrorCodes {
```

CVF-17. FIXED

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

Description These functions may send ether to several recipients in a single transaction. This is a bad practice, as in case one recipient declines the transfer, other recipients will not receive ether as well.

Recommendation Consider counting how much ether each recipient is eligible to take, and perform actual transfers later.

Client Comment *We will instead collate royalties and protocol fee and send to factory which never reverts. Recipients can withdraw from there later. This was chosen over keeping it in pools because it's infeasible for recipients to send transactions to withdraw from arbitrary number of pool contracts.*

```
33 function _pullTokenInputAndPayProtocolFee(
```

```
78 function _payRoyalties(RoyaltyDue[] memory royaltiesDue) internal
      ↪ returns (uint256 totalRoyaltiesPaid) {
```

```
97 function _sendTokenOutput(address payable tokenRecipient, uint256
      ↪ outputAmount, RoyaltyDue[] memory royaltiesDue)
```

CVF-18. FIXED

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

Description This makes it possible to avoid paying protocol fees.

Recommendation Consider counting protocol fees that weren't paid to pay then later.

```
68 if (protocolFee > address(this).balance) {
       protocolFee = address(this).balance;
70 }
```

CVF-19. INFO

- **Category** Flaw
- **Source** CollectionPoolEnumerable.sol

Description A malicious user may send large number of worthless NFTs, that are not in the bitmap, to the pool contract, effectively making the contract to iterate through them in attempt to find good ones. This could exceed block gas limit and effectively make the pool useless.

Recommendation Consider maintaining a collection of valid NFTs owned by the pool, and directly taking NFTs from there.

```
40 if (idMap.get(nftId)) {
```

CVF-20. INFO

- **Category** Procedural
- **Source** CollectionPoolERC20.sol

Description We didn't review "CollectionPoolCloner", while it seems to be an sensitive part of the system. Without reviewing it, it is impossible to tell whether immutable parameters schema is safe or not.

```
26 * @dev See CollectionPoolCloner for an explanation on how this works
```

CVF-21. FIXED

- **Category** Suboptimal
- **Source** CollectionPoolERC20.sol

Description These functions may do several token transfers in a single transaction. This is a bad practice, as if transfer to one of the recipients will fail, other recipients won't get tokens as well. A transfer may fail in case the recipient is blacklisted or (in case of ERC-777 tokens) it reverts inside a "tokensReceived" hook.

Recommendation Consider counting how much tokens each recipient is eligible to take and allow performing actual token transfers later.

```
45 function _pullTokenInputAndPayProtocolFee(
```

```
160 function _sendTokenOutput(address payable tokenRecipient, uint256
    ↪ outputAmount, RoyaltyDue[] memory royaltiesDue)
```

CVF-22. FIXED

- **Category** Unclear behavior
- **Source** CollectionPoolERC20.sol

Description This check won't work for ERC-777 tokens in case royaltyRecipient forwards received tokens inside the " tokensReceived" hook.

Recommendation Consider removing this check and implement some other protection if necessary.

84
```
require(
    _token.balanceOf(royaltyRecipient) - royaltyInitBalance ==
        ↪ royaltyAmount,
    "ERC20␣royalty␣not␣transferred␣in"
);
```

CVF-23. FIXED

- **Category** Unclear behavior
- **Source** CollectionPoolERC20.sol

Description This check won't work for ERC-777 tokens in case _assetRecipient forwards received tokens inside the " tokensReceived" hook.

Recommendation Consider removing this check and implement some other protection if necessary.

102
```
require(
    _token.balanceOf(_assetRecipient) - beforeBalance ==
        ↪ amountToAssetRecipient, "ERC20␣not␣transferred␣in"
);
```

CVF-24. FIXED

- **Category** Suboptimal
- **Source** CollectionPoolERC20.sol

Description This makes it possible to avoid paying protocol fees.

Recommendation Consider counting protocol fees that weren't paid to pay then later.

Client Comment *We made transferring the full amount required for swaps.*

```
150 if (protocolFee > poolTokenBalance) {
        protocolFee = poolTokenBalance;
    }
```

CVF-25. FIXED

- **Category** Unclear behavior
- **Source** CollectionPoolERC20.sol

Description These checks don't guarantee that liquidity won't get negative, as it could become negative after paying royalties or protocol fees.

Recommendation Consider performing liquidity check for all outgoing transfers.

Client Comment *The second line of code highlighted is exact (no royalties/protocol fees are paid upon withdrawals).*

```
181 require(liquidity() >= outputAmount, "Too␣little␣ERC20");
```

```
232 require(liquidity() >= amount, "Too␣little␣ERC20");
```

CVF-26. FIXED

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

Description Overflow is possible here.

Recommendation Consider using save conversion.

66
```
newParams.spotPrice = uint128(params.spotPrice +
   ↪ inputValueWithoutFee); // token reserve
newParams.delta = uint128(nftBalance - numItems); // nft reserve
```

118
```
newParams.spotPrice = uint128(params.spotPrice -
   ↪ outputValueWithoutFee); // token reserve
newParams.delta = uint128(nftBalance + numItems); // nft reserve
```

8 Minor Issues

CVF-27. INFO

- **Category** Procedural
- **Source** CollectionPoolMissingEnumerable.sol

Description We didn't review these files.

```
7  import {ICollectionPool} from "../pools/ICollectionPool.sol";
```

```
9  import {CollectionRouter} from "../routers/CollectionRouter.sol";
```

CVF-28. FIXED

- **Category** Suboptimal
- **Source** CollectionPoolMissingEnumerable.sol

Description In case "idSet" is empty, this will produce an assertion error.

Recommendation Consider adding an explicit "require" statement with meaningful error message.

```
27 uint256 lastIndex = idSet.length() - 1;
```

CVF-29. FIXED

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

Description The name sounds odd.

Recommendation Consider renaming to "NFTsCount".

```
91 function NFTsLength() external view returns (uint256) {
```

CVF-30. INFO

- **Category** Procedural
- **Source** CollectionPoolFactory.sol

Description We didn't review these files.

```
17 import {ReentrancyGuard} from "../lib/ReentrancyGuard.sol";

21 import {CollectionRouter} from "../routers/CollectionRouter.sol";

23 import {ICurve} from "../bonding-curves/ICurve.sol";

25 import {CollectionPoolCloner} from "../lib/CollectionPoolCloner.sol"
      ↪ ;
   import {ICollectionPoolFactory} from "./ICollectionPoolFactory.sol";

28 import {CollectionPoolEnumerableERC20} from "./
      ↪ CollectionPoolEnumerableERC20.sol";

30 import {CollectionPoolMissingEnumerableERC20} from "./
      ↪ CollectionPoolMissingEnumerableERC20.sol";
   import {MultiPauser} from "../lib/MultiPauser.sol";
```

CVF-31. FIXED

- **Category** Procedural
- **Source** CollectionPoolFactory.sol

Recommendation Double slashes are redundant.

```
20 import {CollectionPool} from ".//CollectionPool.sol";

24 import {CollectionPoolERC20} from ".//CollectionPoolERC20.sol";
```

CVF-32. FIXED

- **Category** Suboptimal
- **Source** CollectionPoolFactory.sol

Recommendation The “poolAddress” parameters should be indexed.

```
97 event NewPool(address indexed collection, address poolAddress);
   event TokenDeposit(address poolAddress);
```

CVF-33. FIXED

- **Category** Suboptimal
- **Source** CollectionPoolFactory.sol

Recommendation The parameter should be indexed.

```
99 event ProtocolFeeRecipientUpdate(address recipientAddress);
```

CVF-34. FIXED

- **Category** Suboptimal
- **Source** CollectionPoolFactory.sol

Recommendation The “bondingCurve” parameter should be indexed.

```
102 event BondingCurveStatusUpdate(ICurve bondingCurve, bool isAllowed);
```

CVF-35. FIXED

- **Category** Suboptimal
- **Source** CollectionPoolFactory.sol

Recommendation The “target” parameter should be indexed.

```
103 event CallTargetStatusUpdate(address target, bool isAllowed);
```

CVF-36. FIXED

- **Category** Suboptimal
- **Source** CollectionPoolFactory.sol

Recommendation The “router” parameter should be indexed.

```
104 event RouterStatusUpdate(CollectionRouter router, bool isAllowed);
```

CVF-37. FIXED

- **Category** Suboptimal
- **Source** CollectionPoolFactory.sol

Recommendation These checks should be done earlier, before assigning any variables.

```
125 require(_protocolFeeMultiplier <= MAX_PROTOCOL_FEE, "Protocol␣fee␣
    ↪ too␣large");
```

```
128 require(_carryFeeMultiplier <= MAX_CARRY_FEE, "Carry␣fee␣too␣large")
    ↪ ;
```

CVF-38. FIXED

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

Recommendation The return type should be “ICollectionPool”.

```
143 function poolAddressOf(uint256 tokenId) public pure returns (address
    ↪ ) {
```

CVF-39. FIXED

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

Recommendation The type of the “pool” returned value should be “CollectionPoolETH”.

```
195 returns (address pool, uint256 tokenId)
```

```
212 returns (address pool, uint256 tokenId)
```

CVF-40. FIXED

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

Recommendation The type of the “pool” returned value should be “CollectionPool-ERC20”.

```
231 returns (address pool, uint256 tokenId)
```

```
241 returns (address pool, uint256 tokenId)
```

CVF-41. INFO

- **Category** Suboptimal
- **Source** CollectionPoolFactory.sol

Recommendation This check could probably be optimized.

```
271 bool _isPool = isPool(recipient, PoolVariant.ENUMERABLE_ERC20) ||
    ↪ isPool(recipient, PoolVariant.ENUMERABLE_ETH)
     || isPool(recipient, PoolVariant.MISSING_ENUMERABLE_ERC20)
     || isPool(recipient, PoolVariant.MISSING_ENUMERABLE_ETH);
```

CVF-42. FIXED

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

Description These events are emitted even if nothing actually changed.

```
362 emit ProtocolFeeRecipientUpdate(_protocolFeeRecipient);
```

```
372 emit ProtocolFeeMultiplierUpdate(_protocolFeeMultiplier);
```

```
382 emit CarryFeeMultiplierUpdate(_carryFeeMultiplier);
```

```
392 emit BondingCurveStatusUpdate(bondingCurve, isAllowed);
```

```
408 emit CallTargetStatusUpdate(target, isAllowed);
```

CVF-43. FIXED

- **Category** Procedural
- **Source** CollectionPoolFactory.sol

Recommendation The fallback recipient check should be performed before the supports interface check to save gas.

```
445 params.royaltyNumerator == 0 || IERC165(params.nft).
    ↪ supportsInterface(_INTERFACE_ID_ERC2981)
     || params.royaltyRecipientFallback != address(0),
```

CVF-44. FIXED

- **Category** Suboptimal
- **Source** CollectionPoolFactory.sol

Recommendation The conversion to “uint256” is redundant, as Solidity compiler could do it automatically.

```
544 tokenId = uint256(uint160(address(pool)));
```

CVF-45. FIXED

- **Category** Procedural
- **Source** CollectionPoolFactory.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
555 receive() external payable {}
```

CVF-46. FIXED

- **Category** Suboptimal
- **Source** CollectionPoolFactory.sol

Description This override looks redundant.

Recommendation Consider removing it.

```
557 function _beforeTokenTransfer(address from, address to, uint256
       ↪ tokenId) internal override (ERC721) {
        super._beforeTokenTransfer(from, to, tokenId);
    }
```

CVF-47. INFO

- **Category** Procedural
- **Source** CollectionPool.sol

Description We didn't review these files.

```
10 import {ReentrancyGuard} from "../lib/ReentrancyGuard.sol";
```

```
12 import {ICurve} from "../bonding-curves/ICurve.sol";
   import {CollectionRouter} from "../routers/CollectionRouter.sol";
   import {ICollectionPool} from "./ICollectionPool.sol";
   import {ICollectionPoolFactory} from "./ICollectionPoolFactory.sol";
   import {CurveErrorCodes} from "../bonding-curves/CurveErrorCodes.sol
      ↪ ";
```

```
18 import {MultiPauser} from "../lib/MultiPauser.sol";
   import {IPoolActivityMonitor} from "./IPoolActivityMonitor.sol";
```

CVF-48. FIXED

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

Recommendation The type of the “token” parameter could be more specific.

```
119 event TokenDeposit(address indexed collection, address indexed token
    ↪ , uint256 amount);
120 event TokenWithdrawal(address indexed collection, address indexed
    ↪ token, uint256 amount);
    event AccruedTradeFeeWithdrawal(address indexed collection, address
    ↪ indexed token, uint256 amount);
```

CVF-49. FIXED

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

Recommendation The type of the “collection” parameter could be more specific.

```
119 event TokenDeposit(address indexed collection, address indexed token
    ↪ , uint256 amount);
120 event TokenWithdrawal(address indexed collection, address indexed
    ↪ token, uint256 amount);
    event AccruedTradeFeeWithdrawal(address indexed collection, address
    ↪ indexed token, uint256 amount);
    event NFTDeposit(address indexed collection, uint256 numNFTs);
    event NFTWithdrawal(address indexed collection, uint256 numNFTs);
```

CVF-50. FIXED

- **Category** Suboptimal
- **Source** CollectionPool.sol

Recommendation The parameters should be indexed.

```
126 event AssetRecipientChange(address a);
```

```
130 event RoyaltyRecipientFallbackUpdate(address payable newFallback);
```

CVF-51. FIXED

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

Recommendation Events are usually named via nouns, such as “PoolSwapPause”.

```
131 event PoolSwapPaused();
    event PoolSwapUnpaused();
```

CVF-52. FIXED

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

Recommendation The “1e6” value should be a named constant.

```
154 if (_royaltyNumerator >= 1e6) revert RoyaltyNumeratorOverflow();
```

CVF-53. FIXED

- **Category** Readability
- **Source** CollectionPool.sol

Recommendation To avoid these ugly type casts, consider inheriting the “ICollection-PoolFactory” interface from “IERC721”.

```
173 return IERC721(address(factory())).ownerOf(tokenId());
```

```
193 IERC721(address(factory())).safeTransferFrom(msg.sender, newOwner,
      ↪ tokenId());
```

CVF-54. FIXED

- **Category** Suboptimal
- **Source** CollectionPool.sol

Description For an unsigned value, condition “<= 0” looks weird.

Recommendation Consider using “== 0” instead.

```
296 if ((numNFTs <= 0) || (numNFTs > _nft.balanceOf(address(this))))
    ↪ revert InvalidSwapQuantity();
```

```
333     if (nftIds.length <= 0) revert InvalidSwapQuantity();
```

CVF-55. FIXED

- **Category** Suboptimal
- **Source** CollectionPool.sol

Description Returning an error indicator is a bad practice, as it doesn’t guarantee that the state wasn’t changed.

Recommendation Consider throwing a named error instead.

```
427 returns (CurveErrorCodes.Error error, uint256 balance)
```

```
469     CurveErrorCodes.Error error,
```

```
490     CurveErrorCodes.Error error,
```

CVF-56. FIXED

- **Category** Suboptimal
- **Source** CollectionPool.sol

Recommendation The conversion to “uint256” is redundant, as Solidity compiler could make it implicitly.

```
518 _tokenId = uint256(uint160(address(this)));
```

CVF-57. FIXED

- **Category** Readability
- **Source** CollectionPool.sol

Recommendation Should be “else if”.

```
591 if (royaltyRecipientFallback != address(0)) {
```

CVF-58. FIXED

- **Category** Readability
- **Source** CollectionPool.sol

Recommendation Should be “else return”.

```
596 return getAssetRecipient();
```

CVF-59. FIXED

- **Category** Procedural
- **Source** CollectionPool.sol

Description The function “poolVariant” is called on every loop iteration.

Recommendation Consider calling once before the loop.

```
811 router.poolTransferNFTFrom(_nft, routerCaller, _assetRecipient,
      ↪ nftIds[i], poolVariant());
```

CVF-60. FIXED

- **Category** Suboptimal
- **Source** CollectionPool.sol

Recommendation This error should include the data returned by the failed inner call.

```
1004 if (!result) revert CallError();
```

CVF-61. FIXED

- **Category** Suboptimal
- **Source** CollectionPool.sol

Recommendation This function should return the data returned by inner calls.

```
1013 function multicall(bytes[] calldata calls, bool revertOnFail)
        ↪ external onlyAuthorized {
```

CVF-62. FIXED

- **Category** Procedural
- **Source** CollectionPool.sol

Recommendation The error message should include the index of the failed call.

```
1017 revert(_getRevertMsg(result));
```

CVF-63. INFO

- **Category** Procedural
- **Source** ExponentialCurve.sol

Description We didn't review these files.

```
5 import {CurveErrorCodes} from "./CurveErrorCodes.sol";
  import {FixedPointMathLib} from "../lib/FixedPointMathLib.sol";
```

CVF-64. FIXED

- **Category** Suboptimal
- **Source** ExponentialCurve.sol

Recommendation It would be more efficient to calculate as: buySpotPrice * (deltaPowN - FixedPointMathLib.WAD) / (delta - FixedPointMathLib.WAD)

```
68 inputValue = buySpotPrice.fmul(
       (deltaPowN - FixedPointMathLib.WAD).fdiv(delta -
          ↪ FixedPointMathLib.WAD, FixedPointMathLib.WAD),
70     FixedPointMathLib.WAD
   );
```

CVF-65. FIXED

- **Category** Suboptimal
- **Source** ExponentialCurve.sol

Recommendation It would be more efficient to calculate as: uint256(params.spotPrice)*(FixedPointMathLib.WAD - invDeltaPowN)/(FixedPointMathLib.WAD - invDelta)

```
141 outputValue = uint256(params.spotPrice).fmul(
        (FixedPointMathLib.WAD - invDeltaPowN).fdiv(FixedPointMathLib.
            ↪ WAD - invDelta, FixedPointMathLib.WAD),
        FixedPointMathLib.WAD
    );
```

CVF-66. FIXED

- **Category** Suboptimal
- **Source** ExponentialCurve.sol

Recommendation Moving this code to after the loop would allow removing the check.

```
156 if (i == numItems - 1) {
        /// @dev royalty breakdown not needed if fee return value not
            ↪ used
        (lastSwapPrice,) = getOutputValueAndFees(feeMultipliers,
            ↪ rawAmount, new uint256[](0), royaltyAmount);
    }
```

CVF-67. INFO

- **Category** Procedural
- **Source** CollectionPoolETH.sol

Description We didn't review these files.

```
7  import {ICollectionPool} from "./ICollectionPool.sol";
```

```
9  import {ICollectionPoolFactory} from "./ICollectionPoolFactory.sol";
10 import {ICurve} from "../bonding-curves/ICurve.sol";
   import {IPoolActivityMonitor} from "./IPoolActivityMonitor.sol";
```

CVF-68. INFO

- **Category** Procedural
- **Source** CollectionPoolEnumerable.sol

Description We didn't review these files.

```
8  import {CollectionRouter} from "../routers/CollectionRouter.sol";
   import {ICollectionPool} from "./ICollectionPool.sol";

11 import {ICollectionPoolFactory} from "./ICollectionPoolFactory.sol";
```

CVF-69. INFO

- **Category** Procedural
- **Source** CollectionPoolERC20.sol

Description We didn't review these files.

```
7  import {ICollectionPool} from "./ICollectionPool.sol";

9  import {ICollectionPoolFactory} from "./ICollectionPoolFactory.sol";
10 import {CollectionRouter} from "../routers/CollectionRouter.sol";
   import {ICurve} from "../bonding-curves/ICurve.sol";
   import {CurveErrorCodes} from "../bonding-curves/CurveErrorCodes.sol
      ↪ ";
   import {IPoolActivityMonitor} from "./IPoolActivityMonitor.sol";
```

CVF-70. INFO

- **Category** Procedural
- **Source** XykCurve.sol

Description We didn't review these files.

```
5  import {CurveErrorCodes} from "./CurveErrorCodes.sol";

8  import {CollectionPoolCloner} from "../lib/CollectionPoolCloner.sol"
      ↪ ;

10 import {FixedPointMathLib} from "../lib/FixedPointMathLib.sol";
```

CVF-71. INFO

- **Category** Procedural
- **Source** TokenIDFilter.sol

Description We didn't review this file.

```
5 import {ITokenIDFilter} from "./ITokenIDFilter.sol";
```

CVF-72. FIXED

- **Category** Suboptimal
- **Source** TransferLib.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.

```
32 function batchSafeTransferERC20From(IERC20[] calldata tokens,
   ↪ address from, address to, uint256[] calldata values)
```

CVF-73. FIXED

- **Category** Suboptimal
- **Source** TransferLib.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.

```
49 IERC721[] calldata tokens,
```

```
52 uint256[] calldata tokenIds
```

CVF-74. INFO

- **Category** Procedural
- **Source** SigmoidCurve.sol

Description We didn't review these files.

```
5 import {CurveErrorCodes} from "./CurveErrorCodes.sol";
import {FixedPointMathLib} from "../lib/FixedPointMathLib.sol";
```

CVF-75. FIXED

- **Category** Procedural
- **Source** SigmoidCurve.sol

Description The expression “sigmoidParams.deltaN + sign * _numItems” is calculated twice.

Recommendation Consider calculating once and reusing.

```
189 if (!valid64x64Int(sigmoidParams.deltaN + sign * _numItems)) {
```

```
193 newParams.state = encodeState(sigmoidParams.deltaN + sign *
    ↪ _numItems);
```

CVF-76. INFO

- **Category** Suboptimal
- **Source** SigmoidCurve.sol

Description This limits the range of “K” to 0.2^53.

Recommendation Consider calculating as: require (delta » 73 == 0); return int128(delta « 54);

```
288 return ABDKMath64x64.fromUInt(delta) >> K_NORMALIZATION_EXPONENT;
```

CVF-77. INFO

- **Category** Procedural
- **Source** LinearCurve.sol

Description We didn’t review these files.

```
5 import {CurveErrorCodes} from "./CurveErrorCodes.sol";
  import {FixedPointMathLib} from "../lib/FixedPointMathLib.sol";
```

CVF-78. FIXED

- **Category** Readability
- **Source** LinearCurve.sol

Recommendation This value could be calculated as: numItems * (buySpotPrice + newSpotPrice) / 2

```
47 inputValue = numItems * buySpotPrice + (numItems * (numItems - 1) *
      ↪ params.delta) / 2;
```

CVF-79. FIXED

- **Category** Readability
- **Source** LinearCurve.sol

Recommendation This value could be calculated as: numItems * (spotPrice + newSpotPrice + delta) / 2

```
115 outputValue = numItems * params.spotPrice - (numItems * (numItems -
      ↪ 1) * params.delta) / 2;
```

CVF-80. INFO

- **Category** Procedural
- **Source** Curve.sol

Description We didn't review these files.

```
4 import {ICurve} from "./ICurve.sol";
  import {FixedPointMathLib} from "../lib/FixedPointMathLib.sol";
```

CVF-81. INFO

- **Category** Procedural
- **Source** CollectionPoolCloner.sol

Description We didn't review these files.

```
8 import {ICurve} from "../bonding-curves/ICurve.sol";
  import {ICollectionPoolFactory} from "../pools/
     ↪ ICollectionPoolFactory.sol";
```

CVF-82. INFO

- **Category** Readability
- **Source** MultiPauser.sol

Description This line could be simplified by using the "|=" operator.

```
37 pauseStates = pauseStates | (1 << index);
```

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 40 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