Report

v. 1.0

Customer

Collection.xyz

Smart Contract Audit

Vaults

2nd August 2023

Report prepared by
ABDK
Consulting

Contents

1 Changelog 5

2 Introduction 6

3 Project scope 7

4 Methodology 8

5 Our findings 9

6 Critical Issues 10
CVF-1. INFO 10

7 Major Issues 11
CVF-2. FIXED 11
CVF-3. FIXED 11
CVF-4. FIXED 11
CVF-5. FIXED 12
CVF-6. FIXED 12
CVF-7. INFO 12
CVF-8. INFO 13
CVF-9. INFO 13
CVF-10. FIXED 13
CVF-11. INFO 14
CVF-12. FIXED 14
CVF-13. INFO 14
CVF-14. FIXED 15
CVF-15. FIXED 15
CVF-16. INFO 15
CVF-17. FIXED 16
CVF-18. INFO 16
CVF-19. INFO 16
CVF-20. FIXED 17
CVF-21. FIXED 17
CVF-22. FIXED 17
CVF-23. FIXED 18
CVF-24. FIXED 18
CVF-25. FIXED 18
CVF-26. FIXED 19
CVF-27. FIXED 19
CVF-28. FIXED 19
CVF-29. FIXED 20
CVF-30. FIXED 20

8 Moderate Issues **21**

CVF-31. FIXED . . . 21
CVF-32. INFO . . . 21
CVF-33. FIXED . . . 21
CVF-34. FIXED . . . 22
CVF-35. INFO . . . 22
CVF-36. INFO . . . 22

9 Minor Issues **23**

CVF-37. INFO . . . 23
CVF-38. FIXED . . . 23
CVF-39. INFO . . . 24
CVF-40. FIXED . . . 24
CVF-41. FIXED . . . 24
CVF-42. INFO . . . 25
CVF-43. INFO . . . 25
CVF-44. INFO . . . 25
CVF-45. INFO . . . 26
CVF-46. FIXED . . . 26
CVF-47. INFO . . . 26
CVF-48. INFO . . . 27
CVF-49. FIXED . . . 27
CVF-50. FIXED . . . 27
CVF-51. INFO . . . 28
CVF-52. FIXED . . . 28
CVF-53. INFO . . . 28
CVF-54. INFO . . . 29
CVF-55. INFO . . . 29
CVF-56. FIXED . . . 29
CVF-57. INFO . . . 30
CVF-58. FIXED . . . 30
CVF-59. INFO . . . 31
CVF-60. INFO . . . 31
CVF-61. FIXED . . . 31
CVF-62. INFO . . . 32
CVF-63. INFO . . . 32
CVF-64. INFO . . . 32
CVF-65. FIXED . . . 33
CVF-66. FIXED . . . 33
CVF-67. FIXED . . . 33
CVF-68. FIXED . . . 34
CVF-69. INFO . . . 34
CVF-70. FIXED . . . 34
CVF-71. INFO . . . 35
CVF-72. INFO . . . 35
CVF-73. INFO . . . 36

CVF-74. INFO 36
CVF-75. INFO 37
CVF-76. FIXED 37
CVF-77. INFO 38
CVF-78. FIXED 38
CVF-79. INFO 38
CVF-80. INFO 39
CVF-81. INFO 39
CVF-82. INFO 39
CVF-83. INFO 40
CVF-84. INFO 40
CVF-85. INFO 41
CVF-86. INFO 42
CVF-87. INFO 42
CVF-88. INFO 43
CVF-89. INFO 43
CVF-90. INFO 44
CVF-91. INFO 44
CVF-92. INFO 44
CVF-93. INFO 44
CVF-94. INFO 45
CVF-95. INFO 45
CVF-96. INFO 45
CVF-97. INFO 45
CVF-98. INFO 46
CVF-99. INFO 46
CVF-100. INFO 46
CVF-101. INFO 46
CVF-102. INFO 47
CVF-103. INFO 47
CVF-104. INFO 47
CVF-105. INFO 48
CVF-106. INFO 48
CVF-107. INFO 48
CVF-108. INFO 49
CVF-109. INFO 49
CVF-110. INFO 49
CVF-111. INFO 50
CVF-112. INFO 50
CVF-113. INFO 50
CVF-114. INFO 51
CVF-115. INFO 51
CVF-116. INFO 51

1 Changelog

#	Date	Author	Description
0.1	02.08.23	A. Zveryanskaya	Initial Draft
0.2	02.08.23	A. Zveryanskaya	Minor revision
1.0	02.08.23	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:

/			
	Registry.sol		
lib/			
	SortitionSumTreeFactory.sol	TransferLib.sol	
rewards/			
	DrawRewardProgram.sol	TimeWeightedContributionRewardProgram.sol	
rng/			
	RNGChainlinkV2.sol		
validators/			
	MonotonicIncreasingValidator.sol		
vaults/			
	AdvancedVault.sol	Collectionstaker.sol	Vault.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 1 critical, 29 major, and a few less important issues.

Severity	Active	Fixed
Critical	1	0
Major	8	21
Moderate	3	3
Minor	63	17

Fixed 41 out of 116 issues

6 Critical Issues

CVF-1. INFO

- **Category** Suboptimal
- **Source** DrawRewardProgram.sol

Description This loop doesn't scale. A malicious user may spam the contract with large number of interactors to make this loop exceed the block gas limit, effectively blocking sortition tree recalculating.

Recommendation Consider implementing a more efficient approach.

Client Comment *This is acceptable for the use case we expect of vaults. In order to add a new interactor to the list, a malicious user would need to either 1. Own multiple LP tokens with multiple wallets and stake all of them. This would cost them at least 200k + 55k gas per interactor from creating the LP token and transferring the LP token to the vault 2. Have a single LP token, but repeatedly stake, unstake, and transfer it to a new wallet to add to the interactor list. This would cost them 3 ∗ 55k gas per interactor for the 3 NFT transfers incurred. As such, it's not a scalable attack vector. Especially when you consider that the only result of this is that it makes it more expensive/impossible to recharge the draw vault, but won't affect the draw in progress (the suboptimal function is only called by 1 external function). However, we do note that it might be a desirable feature to limit the max number of interactors per draw so that draw creators won't be in a situation where their draw becomes unrechargeable, which we will leave as an initialization parameter.*

```
609 for (uint256 i = length; i > 0;) {
```

7 Major Issues

CVF-2. FIXED

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

Description There is not range check for this argument.

Recommendation Consider adding appropriate check.

```
72 inertia = advancedParams.inertia;
```

CVF-3. FIXED

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

Description This allows adding the same reward program several times and emitting several events for it.

Recommendation Consider forbidding this.

```
108 emit RewardProgramAdded(_rewardPrograms[i]);
    rewardProgramIsOwned[_rewardPrograms[i]] = true;
```

CVF-4. FIXED

- **Category** Suboptimal
- **Source** Vault.sol

Description Here linear searches are performed agains in-storage arrays. This could consume lots of gas.

Recommendation Consider using mappings or other efficient structures, to optimize searches.

Client Comment *Added additional index parameter to avoid 1 scan, defaultRewardPrograms is not expected to be large and is not permissionless so that scan is allowed.*

```
163 for (uint256 i; i < length;) {
```

```
167         for (uint256 j; j < length;) {
```

CVF-5. FIXED

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

Description There is no check to ensure these arrays are of the same length.

Recommendation Consider adding such check.

```
uint256[] memory tokenIds,
bytes[][] calldata contexts
```

CVF-6. FIXED

- **Category** Suboptimal
- **Source** Vault.sol

Recommendation Binary search would be much more efficient.

```
while (j < innerLength) {
```

CVF-7. INFO

- **Category** Suboptimal
- **Source** Vault.sol

Description Mandatory including default reward programs into “targetedRewards” is redundant. The contract could just always subscribe to them.

Recommendation Consider not including them and removing this check.

Client Comment *Working as intended.*

```
/// @dev Ensure all default RPs are included in input
enforceDefaultRewardPrograms(targetedRewards);
```

CVF-8. INFO

- **Category** Suboptimal
- **Source** Vault.sol

Description Copying the existing array every time is expensive.

Recommendation Consider using a more efficient data structure such as tree or heap.

```
445 while (oldIndex < existingLength && newIndex < newRewardsLength) {
```

CVF-9. INFO

- **Category** Suboptimal
- **Source** Vault.sol

Description Copying the existing array every time is expensive.

Recommendation Consider using a more efficient data structure such as tree or heap.

```
525 for (uint256 i; i < length;) {
```

CVF-10. FIXED

- **Category** Suboptimal
- **Source** Vault.sol

Recommendation Complexity of this function is O(n^2). it could be reduced to O(n) by requiring both arrays to be sorted. Even more efficient check it possible if default rewards would be required to go first in the array.

```
735 for (uint256 i; i < length;) {
```

```
738     for (uint256 j; j < innerLength;) {
```

CVF-11. INFO

- **Category** Suboptimal
- **Source** RNGChainlinkV2.sol

Description It is quite unexpected, that the deployer has elevated privileges on a deployed contract, taking into account that there is separate "_owner" argument.

Recommendation Consider either giving elevated privileges only to the owner, or removing the "_owner" argument and using "msg.sender" as the owner.

```
allowedToCallRandomness[msg.sender] = true;
```

CVF-12. FIXED

- **Category** Unclear behavior
- **Source** RNGChainlinkV2.sol

Description There is not way to revoke call permit.

Recommendation Consider adding such ability.

```
allowedToCallRandomness[allowed] = true;
```

CVF-13. INFO

- **Category** Unclear behavior
- **Source** RNGChainlinkV2.sol

Description This makes probability of 1 two times higher than probability of any other number.

Recommendation Consider allowing zero as a valid random number. This would require changing the way how complete requests are distinguished.

```
randomNumbers[_internalRequestId] = (_randomNumber % type(uint256).max) + 1;
```

CVF-14. FIXED

- **Category** Unclear behavior
- **Source** TimeWeightedContribution-RewardProgram.sol

Recommendation Should be "<" to allow new period to start right after the current period.

```
if (_newPeriodStart <= periodFinish) revert RewardsOngoing();
```

CVF-15. FIXED

- **Category** Suboptimal
- **Source** DrawRewardProgram.sol

Description This call is performed in a loop and each invocation checks that the user is not banned.

Recommendation Consider performing this check once before the loop.

```
claimShare(account, epochs[i]);
```

CVF-16. INFO

- **Category** Suboptimal
- **Source** DrawRewardProgram.sol

Description Performing several token transfers in a single transaction is a bad idea as if once of the transfers will fail, all other transfers will be reverted as well.

Recommendation Consider counting what tokens a user is eligible to take, and allowing the user to take then one at once.

Client Comment *Will Not Fix. The logic right now is guarded by a single boolean flag on whether the user has claimed. Fixing this would require more bookkeeping on what each user is allowed to claim. i.e, gas++*

```
transferNftPrizes(epoch, numberOfPrizesPerWinner, prizeIndex, recipient);
```

```
token.safeTransfer(recipient, userReward);
```

CVF-17. FIXED

- **Category** Procedural
- **Source** DrawRewardProgram.sol

Recommendation This is not how one is supposed to use ERC-165 API. See specification for details: https://eips.ethereum.org/EIPS/eip-165#how-to-detect-if-a-contract-implements-erc-165

```
if (!myCollAdd.supportsInterface(0x80ac58cd)) revert NFTNotERC721();
```

CVF-18. INFO

- **Category** Suboptimal
- **Source** SortitionSumTreeFactory.sol

Description Supporting arbitrary number of children per node make code more complicated and significantly less efficient.

Recommendation Consider using a constant number of children instead, preferably a power of two.

```
uint K; // The maximum number of childs per node.
```

CVF-19. INFO

- **Category** Suboptimal
- **Source** SortitionSumTreeFactory.sol

Description Separate handling of the root node seems redundant.

Recommendation Consider inserting the first value as normal (non-parent) node #0. This would make the insertion logic simpler.

```
tree.nodes.push(0);
```

CVF-20. FIXED

- **Category** Suboptimal
- **Source** SortitionSumTreeFactory.sol

Recommendation Loop is not needed here. The start index could be calculated as(tree.nodes.length + tree.K - 2) / tree.K

```
// Find the start index.
for (uint i = 0; i < tree.nodes.length; i++) {
    if ((tree.K * i) + 1 >= tree.nodes.length) {
        startIndex = i;
        break;
    }
}
```

CVF-21. FIXED

- **Category** Procedural
- **Source** SortitionSumTreeFactory.sol

Description The value "tree.nodes.length" is read from the storage multiple times.

Recommendation Consider reading once and reusing.

```
for (uint i = 0; i < tree.nodes.length; i++) {
    if ((tree.K * i) + 1 >= tree.nodes.length) {
```

```
values = new uint[](loopStartIndex + _count > tree.nodes.length ?
   ↪ tree.nodes.length - loopStartIndex : _count);
```

```
for (uint j = loopStartIndex; j < tree.nodes.length; j++) {
```

CVF-22. FIXED

- **Category** Suboptimal
- **Source** SortitionSumTreeFactory.sol

Recommendation Performing this check on every loop iteration is suboptimal. Just loop up to "values.length" rather than "tree.nodes.length".

```
if (valuesIndex < _count) {
```

CVF-23. FIXED

- **Category** Suboptimal
- **Source** SortitionSumTreeFactory.sol

Description The value “tree.nodes.length” is read from the storage multiple times.

Recommendation Consider reading once and reusing.

```
while ((tree.K * treeIndex) + 1 < tree.nodes.length)  // While it still has children.
```

CVF-24. FIXED

- **Category** Suboptimal
- **Source** SortitionSumTreeFactory.sol

Description The value “tree.K” is read from the storage multiple times.

Recommendation Consider reading once and reusing.

```
for (uint i = 1; i <= tree.K; i++) { // Loop over children.
    uint nodeIndex = (tree.K * treeIndex) + i;
```

CVF-25. FIXED

- **Category** Suboptimal
- **Source** SortitionSumTreeFactory.sol

Description Nested loops are redundant here.

Recommendation Consider refactoring like this: uint k = tree.K; uint length = tree.nodes.length; uint index = 0; while (true) { uint value.= tree.nodes [index]; if (currentDrawnNumber >= value) { currentDrawnNumber -= value; index += 1; } else { uint nextIndex = k * index + 1; if (nextIndex >= length) break; index = nextIndex; } } ID = tree.nodeIndexesToIDs[index];

```
while ((tree.K * treeIndex) + 1 < tree.nodes.length)  // While it still has children.
    for (uint i = 1; i <= tree.K; i++) { // Loop over children.
```

CVF-26. FIXED

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

Description This allows adding the same factory more than once and emitting more then one event for it.

Recommendation Consider forbidding this.

```
68 factoryIsValid[validFactories[i]] = true;
```

CVF-27. FIXED

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

Description This allows adding the same implementation more than once and emitting more then one event for it.

Recommendation Consider forbidding this.

```
81 implementationStatus[validImplementations[i].implementationAddress]
     ↪ = status;
   emit ImplementationStatusUpdate(validImplementations[i].
     ↪ implementationAddress, isPausable, Status.ACTIVE);
```

CVF-28. FIXED

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

Description This condition is always true, as status.status is guaranteed to be Status.ACTIVE due to the “require” statement above.

```
95 if (status.status != Status.PAUSED) {
```

CVF-29. FIXED

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

Description This potentially allows unpausing implementation that are not paused, but are in some state different from ACTIVE, PAUSED, or KILLED. Currently there are no such states, but they could be added in the future version.

Recommendation Consider unpausing only when the current status is PAUSED.

```
if (status.status != Status.ACTIVE) {
    status.status = Status.ACTIVE;
```

CVF-30. FIXED

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

Description This function allows unpausing a paused implementation, but emits a different event.

Recommendation Consider disallowing adding an existing implementation.

```
function authorizeImplementation(address implementation, bool
   isPausable) external {
```

8 Moderate Issues

CVF-31. FIXED

- **Category** Procedural
- **Source** Vault.sol

Recommendation This is not how one is supposed to use ERC-165 API. See specification for details: https://eips.ethereum.org/EIPS/eip-165#how-to-detect-if-a-contract-implements-erc-165

```
94 if (!incentiveParams.nft.supportsInterface(0x80ac58cd)) revert
     ↪ NFTNotERC721(); // check if it supports ERC721
```

CVF-32. INFO

- **Category** Flaw
- **Source** Vault.sol

Description The reward programs array is not validated against the registry. Also it is not checked for duplicates.

Recommendation Consider properly validating the reward programs array.

```
104 rewardPrograms = _rewardPrograms;
```

CVF-33. FIXED

- **Category** Overflow/Underflow
- **Source** TimeWeightedContribution-RewardProgram.sol

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

Recommendation Consider using the “mulDiv” function.

```
288     + (lastRewardTime - _lastUpdateTime) * rewardRates[_rewardToken]
          ↪  * 1e18 / vaultTotalSupply;
```

```
298 return accountVaultBalance * (preCalculatedRewardPerToken -
     ↪ userRewardPerTokenPaid[_rewardToken][account])
    / (1e18) + rewards[_rewardToken][account];
```

CVF-34. FIXED

- **Category** Suboptimal
- **Source** DrawRewardProgram.sol

Description It is unclear where this check was done. Relying on constraints enforced elsewhere makes code more error prone.

Recommendation Consider adding an explicit assert here for this constraint.

```
// numberOfPrizesPerWinner has been checked to be <= numPrizes
// ensuring at least 1 winner
```

CVF-35. INFO

- **Category** Suboptimal
- **Source** SortitionSumTreeFactory.sol

Description This doesn't guarantee even distribution of probabilities for all the numbers in range 0..tree.nodes[0].

Recommendation Consider drawing another number in case _drawnNumber / tree.nodes[0] == MAX_DRAWN_NUMBER / tree.nodes[0]

Client Comment *We pass in an address generated by a keccak hash, so we can expect full 2^160 expressivity. As such, tree.nodes[0] is astronomically smaller than MAX_DRAWN_NUMBER. The concern here is that some numbers modulo tree.nodes[0] will have 1 more possible value in [0..2^160 - 1] than others. The effect of this is diluted to the point of being negligible because lim x-> infty [floor(x / k) + 1] / [floor(x / k)] = 1*

```
uint currentDrawnNumber = _drawnNumber % tree.nodes[0];
```

CVF-36. INFO

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

Description This function bans a user potentially many times if they are the owner of different LPs. Is this OK?

Client Comment *This is intentional. We are optimizing for repeated calls. Doing an SLOAD to check if a user is already banned would be pointless since that's as expensive as a SSTORE to the dirty (true) slot.*

```
/// @dev Ban user from staking and claiming rewards
userIsBanned[msg.sender] = true;
```

9 Minor Issues

CVF-37. INFO

- **Category** Procedural
- **Source** Collectionstaker.sol

Description We didn't review these files.

```
import {RNGChainlinkV2Interface} from "../rng/RNGChainlinkV2Interface.sol";
import {IValidator} from "../validators/IValidator.sol";
```

```
import {VaultETH} from "./VaultETH.sol";
import {VaultERC20} from "./VaultERC20.sol";
import {AdvancedVaultETH} from "./AdvancedVaultETH.sol";
import {AdvancedVaultERC20} from "./AdvancedVaultERC20.sol";
import {IRewardProgram} from "../interfaces/IRewardProgram.sol";
```

```
import {CreateVaultParams, AdvancedVaultParams} from "../interfaces/VaultTypes.sol";
```

```
} from "../interfaces/RewardProgramTypes.sol";
import {IVault} from "../interfaces/IVault.sol";
```

CVF-38. FIXED

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

Recommendation The name should be "CollectionStaker".

```
contract Collectionstaker is Ownable {
```

CVF-39. INFO

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

Recommendation The type of these variables should be more specific.

```
address public immutable vaultETHLogic;
address public immutable vaultERC20Logic;
address public immutable advancedVaultETHLogic;
address public immutable advancedVaultERC20Logic;
address public immutable timeWeightedContributionRewardProgramLogic;
address public immutable drawRewardProgramLogic;
```

CVF-40. FIXED

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

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

Client Comment *Recommended changes caused naming conflict lol. Opted for alternatives "VaultCreation", "RewardProgramCreation".*

```
event VaultCreated(address logic, IVault clone, bytes encodedParams);
```

```
event RewardProgramCreated(address logic, IRewardProgram clone, bytes encodedParams);
```

CVF-41. FIXED

- **Category** Suboptimal
- **Source** Collectionstaker.sol

Recommendation The “logic” and “clone” parameters should be indexed.

```
event VaultCreated(address logic, IVault clone, bytes encodedParams);
```

```
event RewardProgramCreated(address logic, IRewardProgram clone, bytes encodedParams);
```

CVF-42. INFO

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

Recommendation The type of these arguments should be more specific.

```
address _vaultETHLogic,
address _vaultERC20Logic,
address _advancedVaultETHLogic,
address _advancedVaultERC20Logic,
address _timeWeightedContributionRewardProgramLogic,
address _drawRewardProgramLogic
```

CVF-43. INFO

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

Recommendation This function should emit some event.

```
function setRNG(RNGChainlinkV2Interface _rngChainlink) external
  ↪ onlyOwner {
```

CVF-44. INFO

- **Category** Procedural
- **Source** Collectionstaker.sol

Recommendation These events should be emitted in the corresponding “createXXX” functions.

```
emit VaultCreated(template, vault, encodedParams);
```

```
emit VaultCreated(template, vault, encodedParams);
```

```
emit RewardProgramCreated(timeWeightedContributionRewardProgramLogic
  ↪ , rewardProgram, encodedParams);
```

```
emit RewardProgramCreated(drawRewardProgramLogic, rewardProgram,
  ↪ encodedParams);
```

CVF-45. INFO

- **Category** Procedural
- **Source** AdvancedVault.sol

Description We didn't review these files.

```
11 import {IRewardProgram} from "../interfaces/IRewardProgram.sol";
```

```
13 import {CreateVaultParams, AdvancedVaultParams} from "../interfaces/
     ↪ VaultTypes.sol";
```

CVF-46. FIXED

- **Category** Suboptimal
- **Source** AdvancedVault.sol

Description These numbers has nothing to do with ether.

Recommendation Consider using "WAD" or "1e18" instead.

```
30 * activePrice * (inertia / 1 ether) + (1 ether - inertia) / 1 ether
     ↪ * P.
```

```
32 * `inertia` must be no greater than 1 ether
```

```
80 * @return The linear penalty for a given price difference. Returns 1
     ↪  ether
```

```
93   if (diff <= _acceptableDifference) return 1 ether;
     return 1 ether
```

```
168               + (1 ether - inertia).fmul(amounts[i],
                    ↪ FixedPointMathLib.WAD);
```

CVF-47. INFO

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

Recommendation The key type should be more specific.

```
59 mapping(address => bool) public isStaked;
```

CVF-48. INFO

- **Category** Suboptimal
- **Source** AdvancedVault.sol

Description Encoding arguments into a byte sequence just to decode it inside the call looks suboptimal.

Recommendation Consider adding an internal version of “initialize” function into the “Vault” contract that accepts unpacked arguments.

```
70 super.initialize(abi.encode(incentiveParams, _rewardPrograms));
```

CVF-49. FIXED

- **Category** Suboptimal
- **Source** AdvancedVault.sol

Recommendation This could be simplified as: return (_maxPriceDeviation - diff).fdiv(_maxPriceDeviation - _acceptableDifference, FixedPointMathLib.WAD);

```
94 return 1 ether
       - (diff - _acceptableDifference).fdiv((_maxPriceDeviation - _acceptableDifference), FixedPointMathLib.WAD);
```

CVF-50. FIXED

- **Category** Procedural
- **Source** AdvancedVault.sol

Recommendation Inner brackets are redundant.

Client Comment *Implicitly fixed with #50.*

```
95 - (diff - _acceptableDifference).fdiv((_maxPriceDeviation - _acceptableDifference), FixedPointMathLib.WAD);
```

CVF-51. INFO

- **Category** Readability
- **Source** AdvancedVault.sol

Recommendation The meaning of these comments is unclear.

```
/// @dev This function call is non reentrant.
```

```
/// @dev This function call is non reentrant.
```

CVF-52. FIXED

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

Recommendation 1 ether should be a named constant.

Client Comment *Implicitly fixed with #47.*

```
+ (1 ether - inertia).fmul(amounts[i], FixedPointMathLib.WAD);
```

CVF-53. INFO

- **Category** Procedural
- **Source** Vault.sol

Description We didn't review these files.

```
import {IValidator} from "../validators/IValidator.sol";
```

```
import {IVault} from "../interfaces/IVault.sol";
import {IRewardProgram} from "../interfaces/IRewardProgram.sol";
import {ReentrancyGuard} from "../lib/ReentrancyGuard.sol";
import {CreateVaultParams, LPTokenInfo} from "../interfaces/
   ↪ VaultTypes.sol";
```

CVF-54. INFO

- **Category** Suboptimal
- **Source** Vault.sol

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

Client Comment *Genuinely don't see the value in this change. The fact that the access modifiers are different for the original mappings makes the change even more complicated.*

```
mapping(address => uint256) internal _balances;
```

```
mapping(address => bool) public userIsBanned;
```

CVF-55. INFO

- **Category** Procedural
- **Source** Vault.sol

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

```
error NFTNotERC721();
error NotTwoSidedLP();
error NotTradePool();
error PoolMismatch();
error TokenMismatch();
```

CVF-56. FIXED

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

Recommendation This value should be a named constant.

```
LOCK_TIME = 30 days;
```

CVF-57. INFO

- **Category** Readability
- **Source** Vault.sol

Recommendation Consider explicitly initializing loop counters with zero values for readability.

107
```
for (uint256 i; i < length;) {
```

125
```
for (uint256 i; i < length;) {
```

163
```
for (uint256 i; i < length;) {
```

226
```
for (uint256 i; i < length;) {
```

245
```
        uint256 j;
```

525
```
    for (uint256 i; i < length;) {
```

575
```
for (uint256 i; i < length;) {
```

680
```
    for (uint256 i; i < length;) {
```

701
```
    for (uint256 i; i < length;) {
```

735
```
for (uint256 i; i < length;) {
```

738
```
    for (uint256 j; j < innerLength;) {
```

CVF-58. FIXED

- **Category** Procedural
- **Source** Vault.sol

Recommendation This check should be done earlier.

133
```
if (rewardProgram.vault() != this) revert RewardProgramNotOwned(
  ↪ rewardProgram);
```

CVF-59. INFO

- **Category** Suboptimal
- **Source** Vault.sol

Recommendation It would be more efficient to merge these two arrays into a single array of structs with two fields. This would also make the length check unnecessary.

Client Comment *Once again, I find this change to be trivial. Also worth noting that it is a change to the function args, which could mean breaking changes to how the frontend calls the function.*

```
220 uint256[] memory tokenIds,
    bytes[][] calldata contexts
```

CVF-60. INFO

- **Category** Suboptimal
- **Source** Vault.sol

Description Shifting array in storage is expensive.

Recommendation Consider using a more efficient data structure, such as binary tree or heap.

```
270 while (j < innerLength) {
```

CVF-61. FIXED

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

Description Phantom overflow is possible here.

Recommendation Consider using a secure sqrt function.

Client Comment *Phantom overflow is addressed in the multiplication layer using Openzeppelin's Math.tryMul method (with built-in overflow flag).*

```
778 poolValue = Math.sqrt(amount0 * amount1);
```

CVF-62. INFO

- **Category** Procedural
- **Source** MonotonicIncreasingValidator.sol

Description We didn't review this file.

```
import {IValidator} from "./IValidator.sol";
```

CVF-63. INFO

- **Category** Procedural
- **Source** RNGChainlinkV2.sol

Description We didn't review these files.

```
import "../lib/Manageable.sol";
import "./RNGChainlinkV2Interface.sol";
```

CVF-64. INFO

- **Category** Suboptimal
- **Source** RNGChainlinkV2.sol

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

Client Comment *Similar reasons as #59 and#54.*

```
mapping(uint32 => uint256) internal randomNumbers;
```

```
mapping(uint32 => uint32) internal requestLockBlock;
```

CVF-65. FIXED

- **Category** Bad naming
- **Source** RNGChainlinkV2.sol

Recommendation Events are usually named via nouns, such as “KeyHash”, “SubscriptionId”, etc.

```
event KeyHashSet(bytes32 keyHash);
```

```
event SubscriptionIdSet(uint64 subscriptionId);
```

```
event VrfCoordinatorSet(VRFCoordinatorV2Interface indexed vrfCoordinator);
```

CVF-66. FIXED

- **Category** Suboptimal
- **Source** RNGChainlinkV2.sol

Description Here a value just written to the storage is read from the storage again.

Recommendation Consider refactoring as: uint32 _requestCounter = ++requestCounter;

```
requestCounter++;
uint32 _requestCounter = requestCounter;
```

CVF-67. FIXED

- **Category** Suboptimal
- **Source** RNGChainlinkV2.sol

Recommendation This variable is redundant. Use “requireId” instead.

```
uint32 _requestCounter = requestCounter;
```

CVF-68. FIXED

- **Category** Procedural
- **Source** TimeWeightedContribution-RewardProgram.sol

Recommendation Consider merging these “import” statements together.

```
import {TransferLib} from "../lib/TransferLib.sol";
```

```
import {IERC20Data} from "../lib/TransferLib.sol";
```

CVF-69. INFO

- **Category** Procedural
- **Source** TimeWeightedContribution-RewardProgram.sol

Description We didn’t review these files.

```
import {IValidator} from "../validators/IValidator.sol";
```

```
import {IRewardProgram} from "../interfaces/IRewardProgram.sol";
import {IVault} from "../interfaces/IVault.sol";
```

```
import {CreateRewardProgramParams, PrivilegedAddresses} from "../interfaces/RewardProgramTypes.sol";
```

CVF-70. FIXED

- **Category** Suboptimal
- **Source** TimeWeightedContribution-RewardProgram.sol

Recommendation This variable could be turned into a constant.

```
uint256 public LOCK_TIME;
```

CVF-71. INFO

- **Category** Suboptimal
- **Source** TimeWeightedContribution-RewardProgram.sol

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

```
mapping(IERC20 => bool) public rewardTokenValid;
```

```
mapping(IERC20 => uint256) public rewardRates;
```

```
mapping(IERC20 => uint256) public rewardPerTokenStored;
```

```
mapping(IERC20 => mapping(address => uint256)) public rewards;
mapping(IERC20 => mapping(address => uint256)) public userRewardPerTokenPaid;
```

CVF-72. INFO

- **Category** Documentation
- **Source** TimeWeightedContribution-RewardProgram.sol

Description The number format of the values in this mapping is unclear.

Recommendation Consider documenting.

```
mapping(IERC20 => uint256) public rewardRates;
```

CVF-73. INFO

- **Category** Suboptimal
- **Source** TimeWeightedContribution-RewardProgram.sol

Recommendation It would be more efficient to merge these mappings into a single mapping whose keys are reward tokens and user addresses, and values are structs encapsulating the values of the original mappings.

```
mapping(IERC20 => mapping(address => uint256)) public rewards;
mapping(IERC20 => mapping(address => uint256)) public
  ↪ userRewardPerTokenPaid;
```

CVF-74. INFO

- **Category** Procedural
- **Source** TimeWeightedContribution-RewardProgram.sol

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

```
error BadEndTime();
error LengthLimitExceeded();
error LengthMismatch();
error MissingExistingTokens();
error RepeatedToken();
error RewardsOngoing();
error RewardTokenMismatch();
error TooEarly();
error ZeroRewardRate();
error CallerNotDeployer();
error CallerNotVault();
```

CVF-75. INFO

- **Category** Readability
- **Source** TimeWeightedContribution-RewardProgram.sol

Recommendation Consider explicitly initializing “i” with zero value for readability.

```
for (uint256 i; i < rewardTokensLength;) {
```

```
    for (uint256 i; i < rewardTokensLength; ++i) {
```

CVF-76. FIXED

- **Category** Procedural
- **Source** TimeWeightedContribution-RewardProgram.sol

Description The expression “rewardToken[i]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
if (rewardTokenValid[_rewardTokens[i].tokenAddress]) revert
  ↪ RepeatedToken();
rewardTokenValid[_rewardTokens[i].tokenAddress] = true;
_rewardRate = _rewardTokens[i].amount / (incentiveParams.
  ↪ periodFinish - incentiveParams.startTime); // guaranteed
  ↪ endTime > startTime
```

```
rewardRates[_rewardTokens[i].tokenAddress] = _rewardRate;
```

```
_rewardTokenAddresses[i] = _rewardTokens[i].tokenAddress;
```

CVF-77. INFO

- **Category** Suboptimal
- **Source** TimeWeightedContribution-RewardProgram.sol

Description Using implicit underflow protection to ensure important business-level constraints is a bad practice, as it makes code harder to read and more error-prone.

Recommendation Consider using an explicit “require” statement to ensure that the period length is positive.

```
90 _rewardRate = _rewardTokens[i].amount / (incentiveParams.periodFinish - incentiveParams.startTime); // guaranteed endTime > startTime
```

CVF-78. FIXED

- **Category** Bad datatype
- **Source** TimeWeightedContribution-RewardProgram.sol

Recommendation This value should be a named constant.

```
100 LOCK_TIME = 30 days;
```

CVF-79. INFO

- **Category** Readability
- **Source** TimeWeightedContribution-RewardProgram.sol

Description Redefining the semantics of a variable is a bad practice that makes code harder to read.

Recommendation Consider using a separate variable to the reward rate.

```
157 // newReward = new reward rate
newReward /= (_newPeriodFinish - _newPeriodStart);
```

CVF-80. INFO

- **Category** Procedural
- **Source** TimeWeightedContribution-RewardProgram.sol

Description Performing several token transfers in a single transaction is a bad idea. If one of the transfers fails, other will also be reverted.

Recommendation Consider implementing an ability to sweep individual tokens separately.

```
rewardToken.safeTransfer(_deployer, rewardToken.balanceOf(address(
    ↪ this)));
```

CVF-81. INFO

- **Category** Procedural
- **Source** TimeWeightedContribution-RewardProgram.sol

Description Performing several token transfers in a single transaction is a bad idea. If one of the transfers fails, other will also be reverted.

Recommendation Consider implementing an ability to get individual tokens separately.

```
rewardToken.safeTransfer(account, reward);
```

CVF-82. INFO

- **Category** Suboptimal
- **Source** TimeWeightedContribution-RewardProgram.sol

Recommendation This expression could be simplified as: rewardPerToken(_rewardToken)

```
__rewardPerToken(_rewardToken, lastTimeRewardApplicable(),
    ↪ lastUpdateTime, vault.totalSupply())
```

CVF-83. INFO

- **Category** Bad datatype
- **Source** TimeWeightedContribution-RewardProgram.sol

Recommendation The value "1e18" should be a named constant.

```
288 + (lastRewardTime - _lastUpdateTime) * rewardRates[_rewardToken] * 1
      ↪ e18 / vaultTotalSupply;
```

```
299 / (1e18) + rewards[_rewardToken][account];
```

CVF-84. INFO

- **Category** Procedural
- **Source** TimeWeightedContribution-RewardProgram.sol

Recommendation Brackets are redundant.

```
299 / (1e18) + rewards[_rewardToken][account];
```

CVF-85. INFO

- **Category** Procedural
- **Source** TimeWeightedContribution-RewardProgram.sol

Description These two code fragments have much in common.

Recommendation Consider refactoring to avoid code duplication.

```
uint256 userVaultBalance =
    balanceIncreased ? vault.balanceOf(account) - amount : vault.balanceOf(account) + amount;
for (uint256 i; i < rewardTokensLength;) {
    IERC20 rewardToken = rewardTokens[i];

    /// @dev Allows us to avoid a pointless read to a freshly written
    /// storage slot if both time and user need to be updated.
    uint256 _rewardPerTokenStored;
    _rewardPerTokenStored =
        __rewardPerToken(rewardToken, newUpdateTime, oldUpdateTime, vaultTotalSupply);
    rewardPerTokenStored[rewardToken] = _rewardPerTokenStored;

    rewards[rewardToken][account] =
        __earned(account, rewardToken, userVaultBalance, _rewardPerTokenStored);
    userRewardPerTokenPaid[rewardToken][account] = _rewardPerTokenStored;

    unchecked {
        ++i;
    }
}
```

(352-368)

CVF-86. INFO

- **Category** Procedural
- **Source** DrawRewardProgram.sol

Recommendation Consider merging these two imports of the same file into one “import” statement.

```
import {TransferLib} from "../lib/TransferLib.sol";
```

```
import {NFTData, IERC20Data} from "../lib/TransferLib.sol";
```

CVF-87. INFO

- **Category** Procedural
- **Source** DrawRewardProgram.sol

Description We didn’t review this file.

```
import {IValidator} from "../validators/IValidator.sol";
```

```
import {IRewardProgram} from "../interfaces/IRewardProgram.sol";
import {IVault} from "../interfaces/IVault.sol";
import {ReentrancyGuard} from "../lib/ReentrancyGuard.sol";
```

```
import {RNGInterface} from "../rng/RNGInterface.sol";
```

```
} from "../interfaces/RewardProgramTypes.sol";
```

CVF-88. INFO

- **Category** Suboptimal
- **Source** DrawRewardProgram.sol

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

```
mapping(uint64 => PrizeSet) public epochPrizeSets;
mapping(uint64 => mapping(IERC20 => uint256)) public epochERC20PrizeAmounts;
mapping(uint64 => mapping(IERC721 => bool)) public epochERC721Collections;
```

```
mapping(uint64 => mapping(uint256 => NFTData)) public epochERC721PrizeIdsData;
mapping(uint64 => WinnerConfig) public epochWinnerConfigs;
```

```
mapping(uint64 => mapping(address => uint256)) public epochUserERC20NumWinnerEntries; // denominator is epochWinnerConfigs[epoch].numberOfDrawWinners
```

```
mapping(uint64 => mapping(address => uint256[])) public epochUserPrizeStartIndices;
```

```
mapping(uint64 => mapping(address => bool)) public isPrizeClaimable;
```

CVF-89. INFO

- **Category** Procedural
- **Source** DrawRewardProgram.sol

Recommendation The “epoch” parameters should be indexed.

```
event DrawOpen(uint64 epoch);
event DrawClose(uint64 epoch);
```

CVF-90. INFO

- **Category** Procedural
- **Source** DrawRewardProgram.sol

Recommendation Time parameter are usually not indexed.

```
uint256 indexed startTime,
uint256 indexed endTime
```

CVF-91. INFO

- **Category** Procedural
- **Source** DrawRewardProgram.sol

Recommendation The “user” parameter should be indexed.

```
event Claim(uint64 indexed epoch, address user);
```

CVF-92. INFO

- **Category** Suboptimal
- **Source** DrawRewardProgram.sol

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

```
error BadStartTime();
error IncorrectDrawStatus();
error NoClaimableShare();
error RNGRequestIncomplete();
```

CVF-93. INFO

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

Recommendation This value should be a named constant.

```
LOCK_TIME = 30 days;
```

CVF-94. INFO

- **Category** Procedural
- **Source** DrawRewardProgram.sol

Recommendation This check should be performed earlier.

```
if (block.timestamp < periodFinish + LOCK_TIME) revert TooEarly();
```

CVF-95. INFO

- **Category** Procedural
- **Source** DrawRewardProgram.sol

Recommendation This check should be done earlier.

```
if (drawStatus != DrawStatus.Closed) revert IncorrectDrawStatus();
```

CVF-96. INFO

- **Category** Suboptimal
- **Source** DrawRewardProgram.sol

Recommendation This conditional statement is redundant. It would be more efficient to always set "isPrizeClaimable" flag for the epoch and winner to true.

```
if (!isPrizeClaimable[_epoch][winner]) {
```

CVF-97. INFO

- **Category** Procedural
- **Source** DrawRewardProgram.sol

Recommendation Brackets are redundant here.

```
uint256 userReward = (totalReward * numerator) / denominator;
```

CVF-98. INFO

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

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

Recommendation Consider using the “mulDiv” function.

```
338 uint256 userReward = (totalReward * numerator) / denominator;
```

CVF-99. INFO

- **Category** Suboptimal
- **Source** DrawRewardProgram.sol

Recommendation Should be “<=”.

```
399 if (eventTimestamp < _drawStart) {
```

CVF-100. INFO

- **Category** Readability
- **Source** DrawRewardProgram.sol

Description The code below looks like it is always executed, while actually it is executed only when eventTimestamp >= _drawStart.

Recommendation Consider putting the rest of the function into explicit “else” branch.

```
404 }
```

CVF-101. INFO

- **Category** Readability
- **Source** DrawRewardProgram.sol

Description The code below looks like it is always executed, while actually it is executed only when numPrizes != 0

Recommendation Consider putting the rest of the function into explicit “else” branch.

```
438 }
```

CVF-102. INFO

- **Category** Suboptimal
- **Source** DrawRewardProgram.sol

Description Combining two separate use cases in one function is a bad practice.

Recommendation Consider splitting into two separate functions and extracting common parts into additional utility functions.

```
* @dev - workflow 1: call with non-zero drawStartTime to add a new epoch.
* @dev - workflow 2: call with zero drawStartTime to add prizes during the current epoch.
```

CVF-103. INFO

- **Category** Suboptimal
- **Source** SortitionSumTreeFactory.sol

Recommendation This could be simplified (and probably optimized) as: delete tree.stack; delete tree.nodes;

```
tree.stack = new uint[](0);
tree.nodes = new uint[](0);
```

CVF-104. INFO

- **Category** Suboptimal
- **Source** SortitionSumTreeFactory.sol

Recommendation The value for “hasMore“ could be calculated as: tree.nodes.length > loopStartIndex + _count

```
hasMore = true;
```

CVF-105. INFO

- **Category** Procedural
- **Source** SortitionSumTreeFactory.sol

Description The value “tree.K * treeIndex” is calculated on every loop iteration.

Recommendation Consider calculating once and reusing.

```
183 uint nodeIndex = (tree.K * treeIndex) + i;
```

CVF-106. INFO

- **Category** Readability
- **Source** SortitionSumTreeFactory.sol

Recommendation Consider putting the body of this loop into curly braces for readability.

```
181 while ((tree.K * treeIndex) + 1 < tree.nodes.length)  // While it still has children.
```

CVF-107. INFO

- **Category** Suboptimal
- **Source** SortitionSumTreeFactory.sol

Recommendation This check is redundant, as in case “tree.nodes.length” is zero, the value “tree.nodes[0]” is also zero.

```
211 if (tree.nodes.length == 0) {
        return 0;
    } else {
        return tree.nodes[0];
    }
```

CVF-108. INFO

- **Category** Suboptimal
- **Source** SortitionSumTreeFactory.sol

Description Supporting both, value increase and value decrease in one function is sub-optimal.

Recommendation Consider implementing two separate functions or at least two separate loops in one function.

```
function updateParents(SortitionSumTrees storage self, bytes32 _key,
    uint _treeIndex, bool _plusOrMinus, uint _value) private {
```

CVF-109. INFO

- **Category** Procedural
- **Source** SortitionSumTreeFactory.sol

Description The storage address of “tree.nodes[parentIndex]” is calculated several times here.

Recommendation Consider using the “++” and “–” or “+=” and “-=” operators.

```
tree.nodes[parentIndex] = _plusOrMinus ? tree.nodes[parentIndex] +
    _value : tree.nodes[parentIndex] - _value;
```

CVF-110. INFO

- **Category** Procedural
- **Source** TransferLib.sol

Description Defining top-level structs in a file named after a library makes it harder to navigate through code.

Recommendation Consider either moving the declarations into the library or moving them into a separate file.

```
struct NFTData {
```

```
struct IERC20Data {
```

CVF-111. INFO

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

Description In many cases it would be more efficient to pass an array of NTF IDs for each NFT address.

Recommendation Consider turning this field into an array.

```
uint256 nftID;
```

CVF-112. INFO

- **Category** Procedural
- **Source** Registry.sol

Description We didn’t review this file.

```
import {IRewardProgram} from "./interfaces/IRewardProgram.sol";
```

CVF-113. INFO

- **Category** Readability
- **Source** Registry.sol

Description Defining top-level types in a file named after a contract makes it harder to navigate through code.

Recommendation Consider moving the types declarations into the contract or moving them into a separate files.

```
enum Status {
```

```
struct ImplementationStatus {
```

CVF-114. INFO

- **Category** Procedural
- **Source** Registry.sol

Recommendation The “factory” parameters should be indexed.

```
event FactoryAddition(address factory);
event FactoryRemoval(address factory);
```

CVF-115. INFO

- **Category** Procedural
- **Source** Registry.sol

Recommendation The “implementation” parameter should be indexed.

```
event ImplementationStatusUpdate(address implementation, bool
  ↪ isPausable, Status status);
```

CVF-116. INFO

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

Description These events are emitted even if nothing actually changed.

```
emit FactoryAddition(factory);
```

```
emit FactoryRemoval(factory);
```

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