Report

v. 2.0

Customer

Maverick

Smart Contract Audit

Maverick Phase 2

11th May 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	03.05.23	A. Zveryanskaya	Initial Draft
0.2	08.05.23	A. Zveryanskaya	Minor revision
1.0	08.05.23	A. Zveryanskaya	Release
1.1	11.05.23	A. Zveryanskaya	CVF-2 marked as Fixed, Client comment updated
1.2	11.05.23	A. Zveryanskaya	CVF-4 is downgraded, Client comment added
1.3	11.05.23	A. Zveryanskaya	CVF-7 is downgraded
2.0	11.05.23	A. Zveryanskaya	Release

2 Introduction

All modifications to this document are prohibited. Violators will be prosecuted to the full extent of the U.S. law.

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.

Following a formal product description, Maverick Protocol offers a new infrastructure for decentralized finance, built to facilitate the most liquid markets for traders, liquidity providers, DAO treasuries, and developers, powered by a revolutionary Automated Market Maker (AMM).

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

/			
	Distributor.sol	Poll.sol	PoolPositionBase.sol
	PoolPositionDynamic.sol	PoolPositionRouter.sol	PoolPositionStatic.sol
	RewardBase.sol	RewardOpen.sol	RewardPusher.sol
	RewardSingle.sol	RewardVote.sol	VoterToken.sol
factories/			
	PoolPosition AndRewardFactory.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 7 major, and a few less important issues. All identified Major issues have been fixed.

	Active	Fixed
Major	0	7
Moderate	6	7
Minor	29	23

Fixed 37 out of 72 issues

6 Major Issues

CVF-1. FIXED

- **Category** Flaw
- **Source** PoolPositionAndRewardFactory.sol

Description (POSSIBLY CRITICAL) The 'encodePacked' function simply concatenates arrays, so the same output may be produced for different inputs, resulting in collisions in 'hash' value. Note that it is not ensured in other files that the lengths of two arrays are the same.

Recommendation Consider using 'encode' function.

Client Comment *We removed the mapping that depends on this hash. There is actually no problem with a duplicate PoolPosition.*

```
113 bytes32 hash = keccak256(abi.encodePacked(pool, binIds, ratios));
```

```
130 bytes32 hash = keccak256(abi.encodePacked(pool, binIds, ratios));
```

CVF-2. FIXED

- **Category** Flaw
- **Source** PoolPositionBase.sol

Description This function allows multiple initialization.

Recommendation Consider explicitly forbidding such scenario.

Client Comment *Code was refactored to remove initializeContract.*

```
72 function initializeContract(IReward _voteReward) public virtual {
```

CVF-3. FIXED

- **Category** Suboptimal
- **Source** PoolPositionBase.sol

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

Recommendation Consider reading once and reusing.

```
114 params = new IPool.RemoveLiquidityParams[](binIds.length);
```

```
117 for (uint256 i = 1; i < binIds.length; i++) {
```

CVF-5. FIXED

- **Category** Suboptimal
- **Source** PoolPositionRouter.sol

Description In case "propertionD18" is greater that 1e18, this formula effectively decreases it to 1e18.

Recommendation Consider explicitly forbidding values greater than 1e18.

```
152 lpAmountStaked = proportionD18 == ONE ? balance : Math.min(balance,
    ↪ Math.mulDiv(proportionD18, balance, ONE));
```

CVF-6. FIXED

- **Category** Flaw
- **Source** PoolPositionDynamic.sol

Description There is no range check for these arrays.

Recommendation Consider explicitly requiring these arrays to be of the same length.

```
20 uint128[] memory _binIds,
uint128[] memory _ratios,
```

CVF-8. FIXED

- **Category** Suboptimal
- **Source** RewardOpen.sol

Description These constant names are useless. Also, one would have to change it in case the value will be changed.

Recommendation Consider using names that reflect the semantics of the constants, rather than their values.

```
13 uint256 constant FOUR_TENTHS = 4e17;
   uint256 constant SIX_TENTHS = 6e17;
   uint256 constant FOUR_YEARSD18 = (365 days) * 4 * 1e18;
```

CVF-9. FIXED

- **Category** Suboptimal
- **Source** RewardBase.sol

Description The value ”rewardData.length” is read from the storage multiple times.

Recommendation Consider reading once and reusing.

```
84 info = new RewardInfo[](rewardData.length);
   for (uint8 i = 1; i < rewardData.length; i++) {
```

7 Moderate Issues

CVF-4. INFO

- **Category** Flaw
- **Source** PoolPositionRouter.sol

Description The returned value is ignored.

Recommendation Consider explicitly checking that true was returned.

Client Comment *WETH9 can never return false.*

```
75 WETH9.transfer(recipient, value);
```

CVF-7. INFO

- **Category** Flaw
- **Source** Distributor.sol

Description The returned value is ignored here.

Recommendation Consider requiring the returned value to be true.

Client Comment *We control the mav contract and use the OZ ERC20 implementation which can only return true. No need to change this logic to do the check.*

```
40 mav.approve(address(reward), monthRewardAmount);
```

CVF-10. FIXED

- **Category** Flaw
- **Source** PoolPositionBase.sol

Description There is no length check for these arrays.

Recommendation Consider adding a check to ensure these arrays are of the same length.

```
40 uint128[] memory _binIds,
   uint128[] memory _ratios,
```

CVF-11. INFO

- **Category** Suboptimal
- **Source** PoolPositionRouter.sol

Description This code is executed even when ”payer” is not this contract.

```
74 WETH9.deposit{value: value}();
   WETH9.transfer(recipient, value);
```

CVF-12. FIXED

- **Category** Suboptimal
- **Source** PoolPositionDynamic.sol

Description Only the first element of this array is actually populated.

Recommendation Consider allocating an array of at most one element.

```
51 params = new IPool.RemoveLiquidityParams[](binIds.length);
```

CVF-13. FIXED

- **Category** Flaw
- **Source** RewardPusher.sol

Description The returned value is ignored.

Recommendation Consider explicitly checking that the returned value is true.

Client Comment *We refactored out this approve.*

```
23 base.approve(address(lpReward), amount);
```

CVF-14. FIXED

- **Category** Suboptimal
- **Source** RewardOpen.sol

Description The result of this calculation could be greater than one.

Recommendation Consider implementing some protection against such situation.

Client Comment *Added checks to fix (please confirm).*

```
65 uint256 rewardProportionToUser = FOUR_TENTHS + Math.mulDiv(
   ↪ SIX_TENTHS * lockDuration, proportionToLockD18, FOUR_YEARSD18)
   ↪ ;
```

CVF-15. FIXED

- **Category** Procedural
- **Source** RewardOpen.sol

Description Using compiler-enforced underflow protection to check business-level constraints is a bad practice. It makes the code error-prone and harder to read.

Recommendation Consider explicitly checking business-level constraints.

```
72 // will revert if proportion is above ONE
   amountToUser = Math.mulDiv(amount, ONE - proportionToLockD18, ONE);
```

CVF-16. INFO

- **Category** Flaw
- **Source** RewardOpen.sol

Description The returned value is ignored.

Recommendation Consider explicitly checking that the returned value is true or using the "SafeERC20" library.

Client Comment *We control and have deployed the base token contract. True is the only possible return value in the OZ implementation we use. No need to check.*

```
76 base.transfer(baseRecipient, amountToUser);
```

```
92 base.approve(address(ve), amountToLock);
```

```
107 base.approve(address(ve), amountToLock);
```

CVF-17. FIXED

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

Description This function is very dangerous as it allows the minter to burn tokens from arbitrary accounts without explicit approval.

Recommendation Consider limiting this ability or clearly describing why this function is actually safe.

Client Comment *Documenation to contract added.*

```
48 function burnAll(address account) public onlyMinter {
```

CVF-18. FIXED

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

Description This function allows a transferrer to transferring tokens from arbitrary addresses.

Recommendation Consider either limiting this ability or clearly describing why this ability is safe.

Client Comment *Documenation to contract added.*

```
72 function transferFrom(address from, address to, uint256 amount)
   ↪ public returns (bool) {
```

CVF-19. INFO

- **Category** Flaw
- **Source** Poll.sol

Description These functions could be called several times effectively changing previous values.

Recommendation Consider forbidding subsequent calls.

Client Comment *The factory is the only contract that can call these functions and the factory only calls these functions in the constructor. No need for the check.*

```
60 function attachGlobalVoteReward(IReward _globalVoteReward) external
   ↪ onlyFactory {
```

```
64 function attachRewardPusher(IRewardPusher _rewardPusher) public
   ↪ onlyFactory {
```

CVF-20. INFO

- **Category** Procedural
- **Source** Poll.sol

Description The previously set vote reward is not removed from trasferrers.

Client Comment *This funciton will only ever be called once.*

```
62 voterToken.addTransferer(address(_globalVoteReward));
```

8 Minor Issues

CVF-21. INFO

- **Category** Procedural
- **Source** PoolPositionAndRewardFactory.sol

Description We didn't review these files.

```
9  import {PoolPositionStaticDeployer} from "../factories/
       ↪ PoolPositionStaticDeployer.sol";
10 import {PoolPositionDynamicDeployer} from "../factories/
       ↪ PoolPositionDynamicDeployer.sol";
   import {RewardOpenDeployer} from "../factories/RewardOpenDeployer.
       ↪ sol";
   import {RewardVoteDeployer} from "../factories/RewardVoteDeployer.
       ↪ sol";
   import {IPoolPosition} from "../interfaces/IPoolPosition.sol";
   import {IPoolPositionAdmin} from "../interfaces/IPoolPositionAdmin.
       ↪ sol";
   import {IPoolPositionAndRewardFactory} from "../interfaces/
       ↪ IPoolPositionAndRewardFactory.sol";
   import {IVotingEscrow} from "../interfaces/IVotingEscrow.sol";
   import {IVault} from "../interfaces/IVault.sol";
   import {IReward} from "../interfaces/IReward.sol";
   import {IRewardPusher} from "../interfaces/IRewardPusher.sol";
20 import {IMav} from "../interfaces/IMav.sol";
   import {IPoll} from "../interfaces/IPoll.sol";
```

```
23 import {VeArtProxy} from "../VeArtProxy.sol";
   import {VotingEscrow} from "../VotingEscrow.sol";
   import {Mav} from "../Mav.sol";
```

CVF-22. FIXED

- **Category** Suboptimal
- **Source** PoolPositionAndRewardFactory.sol

Recommendation The key type for this mapping should be "IPoolPosition".

```
42 mapping(address => bool) public isPoolPosition;
```

CVF-23. FIXED

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

Recommendation The type for these mappings should be: mapping (IReward ⇒ IReward)

```
43 mapping(address => address) public voteRewardToLpReward;
   mapping(address => address) public lpRewardToVoteReward;
```

CVF-24. INFO

- **Category** Suboptimal
- **Source** PoolPositionAndRewardFactory.sol

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

```
50 error InvalidFeeProportion();
   error MustBeFactoryPool();
   error PoolPositionAlreadyDeployed();
```

CVF-25. FIXED

- **Category** Readability
- **Source** PoolPositionAndRewardFactory.sol

Recommendation "0.5e18" would be more readable.

```
60 uint256 public extractedFeeShareToVoters = 5e17;
```

CVF-26. FIXED

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

Description These functions should emit some events.

```
83 function updateFeeProportionSentOutOfPP(uint256 newValue) external
   ↪ onlyOwner {
```

```
87 function updateExtractedFeeShareToVoter(uint256 newValue) external
   ↪ onlyOwner {
```

```
91 function updateMiniumLockPeriod(uint256 newValue) external onlyOwner
   ↪  {
```

```
94 function addNewApprovedRewardToken(address rewardToken) external
   ↪ onlyOwner {
```

```
97 function deployMav(address mintTo) external onlyOwner {
```

CVF-27. FIXED

- **Category** Suboptimal
- **Source** PoolPositionAndRewardFactory.sol

Recommendation The ”ONE” constant should be used here.

```
84 if (newValue > 1e18) revert InvalidFeeProportion();
```

```
88 if (newValue > 1e18) revert InvalidFeeProportion();
```

CVF-28. FIXED

- **Category** Procedural
- **Source** PoolPositionAndRewardFactory.sol

Recommendation The hash generation logic should be extracted into a function.

Client Comment *Removed hash creation*

```
113 bytes32 hash = keccak256(abi.encodePacked(pool, binIds, ratios));
```

```
130 bytes32 hash = keccak256(abi.encodePacked(pool, binIds, ratios));
```

CVF-29. INFO

- **Category** Suboptimal
- **Source** PoolPositionAndRewardFactory.sol

Recommendation Double type conversion here wouldn't be necessary if "IPoolPostion" would inherit from "IERC20".

```
180 reward = RewardOpenDeployer.deployRewardOpen(IERC20(address(
    ↪ poolPosition)), this, ve, rewardPusher);
```

CVF-30. FIXED

- **Category** Procedural
- **Source** PoolPositionBase.sol

Description This compiler version requirement is inconsistent with other files.

Recommendation Consider specifying as "^0.8.0".

```
2 pragma solidity ^0.8.9;
```

CVF-31. INFO

- **Category** Procedural
- **Source** PoolPositionBase.sol

Description We didn't review these files.

```
17 import {IPoolPosition} from "./interfaces/IPoolPosition.sol";
   import {IVault} from "./interfaces/IVault.sol";
   import {IReward} from "./interfaces/IReward.sol";
20 import {IPoolPositionAndRewardFactory} from "./interfaces/
      ↪ IPoolPositionAndRewardFactory.sol";
   import {Utilities} from "./libraries/Utilities.sol";
```

CVF-32. INFO

- **Category** Suboptimal
- **Source** PoolPositionBase.sol

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

```
24 error InvalidBinIds();
   error BinIsMerged();
```

CVF-33. INFO

- **Category** Procedural
- **Source** PoolPositionBase.sol

Recommendation Brackets around multiplications are redundant.

Client Comment *Prettier adds these in.*

```
143 abTransfer(_voteReward, (tokenAOut * extractedFeeShareToVoters) /
       ↪ ONE, (tokenBOut * extractedFeeShareToVoters) / ONE, true);
```

CVF-34. FIXED

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

Recommendation The function name is confusing, as this function actually does perform burning inside.

Client Comment *Renamed to _ppBurn*

```
157 function _preBurn(address account, uint256 lpAmountToUnStake)
    ↪ internal checkBin checkpointLiquidity returns (IPool.
    ↪ RemoveLiquidityParams[] memory params) {
```

CVF-35. INFO

- **Category** Procedural
- **Source** PoolPositionStatic.sol

Description We didn't review these files.

```
 8 import {IPoolPositionAndRewardFactory} from "./interfaces/
    ↪ IPoolPositionAndRewardFactory.sol";
   import {IVault} from "./interfaces/IVault.sol";
10 import {IReward} from "./interfaces/IReward.sol";
```

CVF-36. INFO

- **Category** Suboptimal
- **Source** PoolPositionStatic.sol

Description No access level specified for these variables, so internal access will be used by default.

Recommendation Consider explicitly specifying an access level.

```
14 uint128[] sqrtPriceLowerEdges;
   uint128[] sqrtPriceUpperEdges;
   uint128[] checkpoints;
```

CVF-37. INFO

- **Category** Suboptimal
- **Source** PoolPositionStatic.sol

Description This code fills in-storage arrays with zero values.

Recommendation Consider using mappings instead of arrays for efficiency.

```
26 sqrtPriceLowerEdges = new uint128[](binIds.length);
   sqrtPriceUpperEdges = new uint128[](binIds.length);
```

CVF-38. INFO

- **Category** Procedural
- **Source** PoolPositionRouter.sol

Description We didn't review these files.

```
10 import "./interfaces/IPoolPosition.sol";
   import "./interfaces/IDistributor.sol";
   import "./interfaces/IReward.sol";
   import "./interfaces/IPoolPositionAndRewardFactory.sol";
   import "./interfaces/IPoolPositionRouter.sol";
   import "./interfaces/external/IWETH9.sol";
   import "./libraries/PoolPositionInspector.sol";
   import "./libraries/Multicall.sol";
```

CVF-39. FIXED

- **Category** Suboptimal
- **Source** PoolPositionRouter.sol

Recommendation Conversion to "IPosition" is redundant as "position" is already "IPosition".

```
37 routerTokenId = IPosition(position).mint(address(this));
```

CVF-40. FIXED

- **Category** Procedural
- **Source** PoolPositionDynamic.sol

Description This compiler version requirement is inconsistent with other files.

Recommendation Consider specifying as ”^0.8.0”.

```
2 pragma solidity ^0.8.9;
```

CVF-41. INFO

- **Category** Procedural
- **Source** PoolPositionDynamic.sol

Description We didn’t review these files.

```
8  import {IPoolPositionAndRewardFactory} from "./interfaces/
       ↪ IPoolPositionAndRewardFactory.sol";
   import {IVault} from "./interfaces/IVault.sol";
10 import {IReward} from "./interfaces/IReward.sol";
```

CVF-42. INFO

- **Category** Suboptimal
- **Source** PoolPositionDynamic.sol

Description No access level specified for these variables, so internal access will be used by default.

Recommendation Consider explicitly specifying an access level.

```
13 uint256 poolTickSpacing;
   uint256 checkpointA;
   uint256 checkpointB;
```

```
17 mapping(uint128 => uint256) activeBins;
```

CVF-43. INFO

- **Category** Procedural
- **Source** PoolPositionDynamic.sol

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

```
25 ) PoolPositionBase(_pool, _binIds, _ratios, factoryCount,
       ↪ _poolPositionAndRewardFactory, false, _protocolEscrow) {}
```

CVF-44. INFO

- **Category** Procedural
- **Source** Distributor.sol

Description We didn't review this file.

```
6 import {IReward} from "./interfaces/IReward.sol";
```

CVF-45. INFO

- **Category** Suboptimal
- **Source** Distributor.sol

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

```
 9 error InvalidMonthlyProportion();
10 error InvalidDisburement();
```

CVF-46. INFO

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

Recommendation Events are usually named via nouns, such as "Initialization", "MonthlyDistribution", etc.

```
11 event Initialize(IERC20 mav, uint256 startTimestamp, IReward reward)
       ↪ ;
   event SetMonthlyDisbursement(uint256 value);
   event Disburse(uint256 absoluteMonthNumber, uint256
       ↪ monthRewardAmount);
```

CVF-47. INFO

- **Category** Suboptimal
- **Source** Distributor.sol

Recommendation The "mav" and "reward" parameters should be indexed.

```
11 event Initialize(IERC20 mav, uint256 startTimestamp, IReward reward)
       ↪ ;
```

CVF-48. FIXED

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

Recommendation Here, the "ONE" constant should be used instead of "1e18" hardcoded value.

```
16 uint256 constant MAX_MONTHLY_PROPORTION = 1e18; // 100%
```

CVF-49. FIXED

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

Recommendation A better name would be "onlyPoll".

```
13 modifier checkCaller() {
```

CVF-50. INFO

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

Recommendation The type of the ”rewardTokenAddress” should be ”IERC20”.

```
11 constructor(IERC20 _stakingToken, IPoolPositionAndRewardFactory
   ↪ _rewardFactory, address poll, address rewardTokenAddress)
   ↪ RewardVote(_stakingToken, _rewardFactory, poll, address(0)) {
```

CVF-51. FIXED

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

Description This function should emit some event.

```
23 function setPusher(address _rewardPusherAddress) public onlyFactory
   ↪ {
```

CVF-52. INFO

- **Category** Procedural
- **Source** RewardPusher.sol

Description We didn’t review these files.

```
4 import {IReward} from "./interfaces/IReward.sol";
import {IRewardPusher} from "./interfaces/IRewardPusher.sol";
import {IPoolPositionAndRewardFactory} from "./interfaces/
   ↪ IPoolPositionAndRewardFactory.sol";
```

CVF-53. FIXED

- **Category** Suboptimal
- **Source** RewardPusher.sol

Recommendation This variable should be declared as immutable.

```
9 IERC20 public base;
```

CVF-54. FIXED

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

Recommendation The type of the "lpReward" argument should be "IReward".

```
20 function _push(address lpReward, address voteReward) internal {
```

```
27 function push(address lpReward, address voteReward) public {
```

```
31 function push(address lpReward) public {
```

CVF-55. FIXED

- **Category** Readability
- **Source** RewardOpen.sol

Recommendation "0.4e18" and "0.6e18" would be more readable.

```
13 uint256 constant FOUR_TENTHS = 4e17;
   uint256 constant SIX_TENTHS = 6e17;
```

CVF-56. INFO

- **Category** Suboptimal
- **Source** RewardOpen.sol

Recommendation This constant is redundant, as its value could be calculated as `1e18 - FOUR_TENTH`.

```
14 uint256 constant SIX_TENTHS = 6e17;
```

CVF-57. INFO

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

Recommendation Better names would be ”getRewards” and ”getReward” respectively.

```
42 function getReward(address recipient, uint8[] calldata
   ↪ rewardTokenIndices) external override {
```

```
49 function getRewardOne(address recipient, uint8 rewardTokenIndex)
   ↪ external override returns (uint256) {
```

CVF-58. INFO

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

Description This assignment should be done only when lockDuration >= minLockDuration

```
59 trueDuration = lockDuration;
```

CVF-59. FIXED

- **Category** Procedural
- **Source** VoterToken.sol

Description This compiler version requirement is inconsistent with other files.

Recommendation Consider specifying as ”^0.8.0”.

```
4 pragma solidity ^0.8.9;
```

CVF-60. INFO

- **Category** Procedural
- **Source** VoterToken.sol

Description We didn’t review this file.

```
7 import {IVoterToken} from "./interfaces/IVoterToken.sol";
```

CVF-61. FIXED

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

Recommendation A better name would be ”onlyTransferrer”.

```
32 modifier isTransferer() {
```

CVF-62. FIXED

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

Description These functions should emit some events.

```
42 function addTransferer(address transferer) public onlyMinter {
```

```
45 function removeTransferer(address transferer) public onlyMinter {
```

CVF-63. FIXED

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

Description This event is emitted even when ”amount” is zero.

Recommendation Consider not emitting events in such a case.

```
56 emit Transfer(account, address(0), amount);
```

CVF-64. FIXED

- **Category** Suboptimal
- **Source** VoterToken.sol

Description This variable is redundant.

Recommendation Consider removing it and using ”msg.sender” instead.

```
68 address owner = msg.sender;
```

CVF-65. INFO

- **Category** Procedural
- **Source** RewardBase.sol

Description We didn't review these files.

```
10 import {IReward} from "./interfaces/IReward.sol";
```

```
12 import {IPoolPositionAndRewardFactory} from "./interfaces/
   ↪ IPoolPositionAndRewardFactory.sol";
```

CVF-66. INFO

- **Category** Suboptimal
- **Source** RewardBase.sol

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

```
17 error DurationOutOfBounds();
```

```
20 error NotValidRewardToken();
```

```
22 error RewardTokenNotApproved();
   error StaleToken();
   error TokenNotStale();
   error RewardStillActive();
```

CVF-67. INFO

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

Recommendation The type of the "rewardTokenAddress" arguments should be "IERC20".

```
98  function earned(address account, address rewardTokenAddress)
       ↪ external view returns (uint256) {
```

```
149 function _checkAndAddRewardToken(address rewardTokenAddress)
       ↪ internal returns (uint8 rewardTokenIndex) {
```

```
208 function transferAndNotify(address rewardTokenAddress, uint256
       ↪ amount, uint256 duration) public nonReentrant {
```

```
219 function _notifyRewardAmount(address rewardTokenAddress, uint256
       ↪ amount, uint256 duration) internal {
```

CVF-68. INFO

- **Category** Procedural
- **Source** RewardBase.sol

Description We didn't review the "clip" function used here.

```
105 return data.rewards[account] + Math.mulDiv(balanceOf[account],
       ↪ MavMath.clip(data.rewardPerTokenStored + deltaRewardPerToken(
       ↪ data), data.userRewardPerTokenPaid[account]), ONE);
```

```
129 return Math.mulDiv(balanceOf[account], MavMath.clip(data.
       ↪ rewardPerTokenStored, data.userRewardPerTokenPaid[account]),
       ↪ ONE);
```

```
223 uint256 remainingRewards = MavMath.clip(data.rewardToken.balanceOf(
       ↪ address(this)), data.escrowedReward);
```

CVF-69. INFO

- **Category** Procedural
- **Source** Poll.sol

Description We didn't review these files.

```
9  import {IReward} from "./interfaces/IReward.sol";
10 import {IRewardPusher} from "./interfaces/IRewardPusher.sol";
   import {IPoll} from "./interfaces/IPoll.sol";
   import {IVotingEscrow} from "./interfaces/IVotingEscrow.sol";
   import {IVoterToken} from "./interfaces/IVoterToken.sol";
   import {IPoolPositionAndRewardFactory} from "./interfaces/
      ↪ IPoolPositionAndRewardFactory.sol";
```

CVF-70. FIXED

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

Description These functions should emit some events.

```
57 function addVotingRewardsContractToTokenPermissions(address
      ↪ rewardContract) public onlyFactory {
```

```
60 function attachGlobalVoteReward(IReward _globalVoteReward) external
      ↪ onlyFactory {
```

```
64 function attachRewardPusher(IRewardPusher _rewardPusher) public
      ↪ onlyFactory {
```

CVF-71. INFO

- **Category** Suboptimal
- **Source** Poll.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.

```
70 function vote(uint256 veTokenId, IReward[] memory voteTargets,
      ↪ uint256[] memory weights) external senderIsOwner(veTokenId) {
```

```
73 function _vote(uint256 veTokenId, IReward[] memory voteTargets,
      ↪ uint256[] memory weights) internal {
```

CVF-72. FIXED

- **Category** Suboptimal
- **Source** Poll.sol

Recommendation The conditional statement is redundant here. Just do: voteSuccessful = true;

```
100 if (!voteSuccessful) voteSuccessful = true;
```

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