Report

v. 3.0

Customer

Exactly

## Smart Contract Audit

# Exactly Protocol. Phase II

17th May 2023

Report prepared by

ABDK Consulting

# Contents

# 1 Changelog

| # | Date | Author | Description |
|---|---|---|---|
| 0.1 | 15.05.23 | A. Zveryanskaya | Initial Draft |
| 0.2 | 16.05.23 | A. Zveryanskaya | Minor revision |
| 1.0 | 16.05.23 | A. Zveryanskaya | Release |
| 1.1 | 16.05.23 | A. Zveryanskaya | CVF-5, 13 are downgraded |
| 2.0 | 16.05.23 | A. Zveryanskaya | Release |
| 2.1 | 17.05.23 | A. Zveryanskaya | Typos are fixed |
| 2.2 | 17.05.23 | A. Zveryanskaya | CVF-5 removed |
| 3.0 | 17.05.23 | 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.

Exactly is a decentralized, non-custodial and open-source protocol that provides an autonomous fixed and variable interest rate market enabling users to frictionlessly exchange the time value of their assets and completing the DeFi credit market.

# 3 Project scope

We were asked to review:

- New functionality as a diff to the code
- After-audit fixes

Files:

```
/
    Auditor.sol                 InterestRateModel.sol       Market.sol
    PriceFeedDouble.sol         RewardsController.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 13 major, and a few less important issues. All identified Major issues have been fixed.

| | Active | Fixed |
|---|---|---|
| Major | 0 | 13 |
| Moderate | 7 | 2 |
| Minor | 13 | 23 |

Fixed 38 out of 58 issues

# 6 Major Issues

## CVF-1. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “distribution[market]” is calculated multiple times.

**Recommendation** Consider calculating once and reusing.

```
52 +uint256 rewardsCount = distribution[market].availableRewardsCount;
```

```
54 +  update(account, market, distribution[market].availableRewards[r],
   ↪  ops);
```

## CVF-2. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “distribution[market]” is calculated several times.

**Recommendation** Consider calculating once and reusing.

```
68 +uint256 rewardsCount = distribution[market].availableRewardsCount;
```

```
70 +  ERC20 reward = distribution[market].availableRewards[r];
```

```
74 +      accountFixedBorrowShares(market, account, distribution[market
   ↪ ].rewards[reward].start)
```

## CVF-3. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “marketOps[i]” is calculated multiple times.

**Recommendation** Consider calculating once and reusing.

```
105 +Distribution storage dist = distribution[marketOps[i].market];
```

```
109 +    marketOps[i].market,
```

```
112 +      marketOps[i].market,
113 +      marketOps[i].operations,
```

```
123 +  for (uint256 o = 0; o < marketOps[i].operations.length; ) {
124 +    uint256 rewardAmount = dist.rewards[rewardsList[r]].accounts[
    ↪ msg.sender][marketOps[i].operations[o]].accrued;
```

```
127 +      dist.rewards[rewardsList[r]].accounts[msg.sender][marketOps[i
    ↪ ].operations[o]].accrued = 0;
```

## CVF-4. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The value “dist.availableRewardsCount” is read from the storage on every loop iteration.

**Recommendation** Consider reading once and reusing.

```
106 +for (uint128 r = 0; r < dist.availableRewardsCount; ) {
```

## CVF-6. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “dist.rewards[rewardsList[r]]’ is calculated multiple times.

**Recommendation** Consider calculating once and reusing.

```
124 +uint256 rewardAmount = dist.rewards[rewardsList[r]].accounts[msg.
    ↪ sender][marketOps[i].operations[o]].accrued;
```

```
127 +  dist.rewards[rewardsList[r]].accounts[msg.sender][marketOps[i].
    ↪ operations[o]].accrued = 0;
```

## CVF-7. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

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

**Recommendation** Consider calculating once and reusing.

```
256 +if (distribution[marketOps[i].market].availableRewardsCount == 0) {
```

```
264 +  marketOps[i].market,
265 +  marketOps[i].operations,
```

```
267 +  distribution[marketOps[i].market].rewards[reward].start
```

```
271 +  unclaimedRewards += distribution[marketOps[i].market]
```

```
283 +    AccountMarketOperation({ market: marketOps[i].market,
    ↪ accountOperations: ops })
```

## CVF-8. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “distribution[marketOps[i].market]” is calculated several times.

**Recommendation** Consider calculating once and reusing.

```
256 +if (distribution[marketOps[i].market].availableRewardsCount == 0) {
```

```
267 +  distribution[marketOps[i].market].rewards[reward].start
```

```
271 +  unclaimedRewards += distribution[marketOps[i].market]
```

## CVF-9. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “distribution[marketOps[i].market].rewards[reward]” is calculated multiple times.

**Recommendation** Consider calculating once and reusing.

```
267 +distribution[marketOps[i].market].rewards[reward].start
```

```
271 +unclaimedRewards += distribution[marketOps[i].market]
272 +.rewards[reward]
```

## CVF-10. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “rewardData.accounts[account]’ is calculated multiple times.

**Recommendation** Consider calculating once and reusing.

```
318 +uint256 accountIndex = rewardData.accounts[account][ops[i].
    ↪ operation].index;
```

```
326 +  rewardData.accounts[account][ops[i].operation].index = uint128(
    ↪ newAccountIndex);
```

```
329 +    rewardData.accounts[account][ops[i].operation].accrued +=
    ↪ uint128(rewardsAccrued);
```

## CVF-11. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “rewardData.accounts[account][ops[i].operation]” is calculated several times.

**Recommendation** Consider calculating once and reusing.

318
```
+uint256 accountIndex = rewardData.accounts[account][ops[i].
   ↪ operation].index;
```

326
```
+  rewardData.accounts[account][ops[i].operation].index = uint128(
   ↪ newAccountIndex);
```

329
```
+    rewardData.accounts[account][ops[i].operation].accrued +=
   ↪ uint128(rewardsAccrued);
```

## CVF-12. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “rewardData.accounts[account]” is calculated multiple times.

**Recommendation** Consider calculating once and reusing.

406
```
+rewardData.accounts[account][ops.accountOperations[o].operation].
   ↪ index,
```

## CVF-14. FIXED

- **Category** Overflow/Underflow
- **Source** RewardsController.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.

491
```
+uint256 distributionFactor = t.period > 0 ? rewardData.
   ↪ undistributedFactor.mulWadDown(target) / t.period : 0;
```

## CVF-15. FIXED

- **Category** Flaw
- **Source** RewardsController.sol

**Recommendation** This should be rounded up.

```
498 +lastUndistributed.mulWadDown(1e18 - exponential);
```

# 7 Moderate Issues

## CVF-13. INFO

- **Category** Flaw
- **Source** RewardsController.sol

**Description** This formula accumulates rounding errors. The more often rewards are updated, the less reward is accounted.

**Recommendation** Consider remembering the remainder from the last division by "baseUnit" and taking this remainder into account on a next division. Alternatively, calculate as: balance.mulDivDown(globalIndex, baseUnit) - balance.mulDivDown(accountIndex, baseUnit)

**Client Comment** *In this case, we believe that the benefits of the proposed fix do not justify the increased gas costs incurred for each transaction.*

```
428 +return balance.mulDivDown(globalIndex - accountIndex, baseUnit);
```

## CVF-16. FIXED

- **Category** Flaw
- **Source** RewardsController.sol

**Description** There is not explicit check that the market does exists.

**Recommendation** Consider adding such a check.

**Client Comment** *Added a @dev comment.*

```
48 +Market market = Market(msg.sender);
```

```
64 +Market market = Market(msg.sender);
```

## CVF-22. FIXED

- **Category** Flaw
- **Source** InterestRateModel.sol

**Recommendation** Division by zero is possible here when utilization = floatingMaxUtilization.

**Client Comment** *Utilization can never go higher than 1. Max utilization should always be higher than 1 too, we added checks at deployment time for this.*

```
120 +int256 r = int256(floatingCurveA.divWadDown(floatingMaxUtilization
    ↪ - utilization)) + floatingCurveB;
```

## CVF-17. INFO

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

**Description** Overflow is possible here.

**Recommendation** Consider using safe conversion.

**Client Comment** *Using uint32 for block.timestamp is still far from overflowing soon.*

```
311 +rewardData.lastUpdate = uint32(block.timestamp);
```

## CVF-18. INFO

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

**Description** Overflow is possible here.

**Recommendation** Consider using safe conversion.

**Client Comment** *There are early checks that make sure uint128 indexes won't overflow. Accrued rewards are calculated based on these indexes.*

```
329 +rewardData.accounts[account][ops[i].operation].accrued += uint128(
    ↪ rewardsAccrued);
```

## CVF-19. INFO

- **Category** Procedural
- **Source** RewardsController.sol

**Description** This loop doesn't scale, as it iterates all maturities from the start to the current time.

**Recommendation** Consider aggregating positions or refactoring this loop is some other way.

**Client Comment** *We'll probably look for other possible solutions in the mid-term.*

353
```
+for (uint256 maturity = firstMaturity; maturity <= maxMaturity; ) {
```

470
```
+  for (uint256 maturity = firstMaturity; maturity <= maxMaturity; )
   ↪  {
```

## CVF-20. INFO

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

**Description** Overflow is possible here.

**Recommendation** Consider using safe conversion.

**Client Comment** *Variables distributionFactor, releaseRate, and v.transitionFactor are derived from configuration values determined by the Exactly Team. newUndistributed and lastUndistributed are also denominated in amount of reward tokens, which are initially set by the Exactly Team. In this manner, Exactly will ensure that these values do not exceed 1000000e18-1e18 levels. Besides, it's programmatically impossible for v.utilization to go higher than 1e18.*

```
494 +  uint256 exponential = uint256((-int256(distributionFactor *
    ↪ deltaTime)).expWad());
```

```
502 +rewards = uint256(int256(releaseRate * deltaTime) - (int256(
    ↪ newUndistributed) - int256(lastUndistributed)));
```

```
506 +  lastUndistributed.mulWadDown(1e18 - uint256((-int256(
    ↪ distributionFactor * deltaTime)).expWad()));
507 +rewards = uint256(-(int256(newUndistributed) - int256(
    ↪ lastUndistributed)));
```

```
512 +  exponential = uint256((-int256(distributionFactor * deltaTime)).
    ↪ expWad());
```

```
520 +exponential = uint256((-int256(distributionFactor * (block.
    ↪ timestamp - t.end))).expWad());
```

```
522 +rewards = uint256(int256(releaseRate * deltaTime) - (int256(
    ↪ newUndistributed) - int256(lastUndistributed)));
```

```
537 +        (int256(v.utilization.divWadDown(1e18 - v.utilization)).
    ↪ lnWad() -
538 +          int256(v.transitionFactor.divWadDown(1e18 - v.
    ↪ transitionFactor)).lnWad())) / 1e18).expWad()
```

## CVF-21. INFO

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

**Description** Underflow is possible here.

**Recommendation** Consider using safe conversion.

**Client Comment** *Given the scenarios and logic in which each reward result is calculated, there's no possibility for them to underflow. We also count on fuzz testing and this kind of edge cases are constantly being monitored.*

```
502 +rewards = uint256(int256(releaseRate * deltaTime) - (int256(
       ↪ newUndistributed) - int256(lastUndistributed)));
```

```
507 +rewards = uint256(-(int256(newUndistributed) - int256(
       ↪ lastUndistributed)));
```

```
522 +rewards = uint256(int256(releaseRate * deltaTime) - (int256(
       ↪ newUndistributed) - int256(lastUndistributed)));
```

```
535 +    uint256(
```

## CVF-23. INFO

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

**Description** Over-/underflow is possible here.

**Recommendation** Consider using safe conversions.

**Client Comment** *We've made research about Chainlink's bad/invalid price answers and the standard value for these cases is 0. On top of this, for a negative Chainlink price to base casted as positive and not to revert, this price has to be equal to -1 and the priceFeedTwo value should be equal to only 1 unit. All other cases for negative prices will revert with an overflow when multiplied by priceFeedTwo's answer. Due to these reasons we decided not to take additional actions.*

```
28 +return int256(uint256(priceFeedOne.latestAnswer()).mulDivDown(
       ↪ uint256(priceFeedTwo.latestAnswer()), baseUnit));
```

# 8 Minor Issues

## CVF-25. FIXED

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

**Recommendation** The value “1e18” should be a named constant.

**Client Comment** *Added info in the comment above this definition.*

```
22 +uint256 public constant UTILIZATION_CAP = 1e18 - 1;
```

## CVF-28. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “claimedAmounts[r]” is calculated several times.

**Recommendation** Consider calculating once and reusing.

```
142 +if (claimedAmounts[r] > 0) {
143 +  rewardsList[r].safeTransfer(to, claimedAmounts[r]);
144 +  emit Claim(msg.sender, rewardsList[r], to, claimedAmounts[r]);
```

## CVF-29. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “distribution[market].rewards[reward].accounts[account][operation]’ is calculated twice.

**Recommendation** Consider calculating once and reusing.

```
199 +distribution[market].rewards[reward].accounts[account][operation].
    ↪ accrued,
200 +distribution[market].rewards[reward].accounts[account][operation].
    ↪ index
```

## CVF-30. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “distribution[market].rewards[reward] is calculated several times.

**Recommendation** Consider calculating once and reusing.

```
210 +distribution[market].rewards[reward].start,
211 +distribution[market].rewards[reward].end,
212 +distribution[market].rewards[reward].lastUpdate
```

## CVF-33. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression ops[i]” is calculated several times.

**Recommendation** Consider calculating once and reusing.

```
318 +uint256 accountIndex = rewardData.accounts[account][ops[i].
    ↪ operation].index;
```

```
320 +if (ops[i].operation) {
```

```
326 +   rewardData.accounts[account][ops[i].operation].index = uint128(
    ↪ newAccountIndex);
327 +   if (ops[i].balance != 0) {
328 +     uint256 rewardsAccrued = accountRewards(ops[i].balance,
    ↪ newAccountIndex, accountIndex, baseUnit);
329 +     rewardData.accounts[account][ops[i].operation].accrued +=
    ↪ uint128(rewardsAccrued);
330 +     emit Accrue(market, reward, account, ops[i].operation,
    ↪ accountIndex, newAccountIndex, rewardsAccrued);
```

## CVF-35. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “distribution[market].rewards[reward]” is calculated several times.

**Recommendation** Consider calculating one and reusing.

```
371 +distribution[market].rewards[reward].borrowIndex,
372 +distribution[market].rewards[reward].depositIndex,
373 +distribution[market].rewards[reward].lastUndistributed
```

## CVF-36. FIXED

- **Category** Documentation
- **Source** RewardsController.sol

**Recommendation** Consider explaining in a comment how it is possible for “lastUpdate” to be greater than “block.timestamp”.

```
393 +block.timestamp > lastUpdate ? block.timestamp - lastUpdate : 0
```

## CVF-38. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Recommendation** This could be simplified as: lastUndistributed.mulWadDown(exponential) + releaseRate.mulWadDown(1e18 - target).divWadDown(distributionFactor).mulWadDown(1e18 - exponential)

```
496 +lastUndistributed +
497 +releaseRate.mulWadDown(1e18 - target).divWadDown(distributionFactor
    ↪ ).mulWadDown(1e18 - exponential) -
498 +lastUndistributed.mulWadDown(1e18 - exponential);
```

```
514 +lastUndistributed +
515 +releaseRate.mulWadDown(1e18 - target).divWadDown(distributionFactor
    ↪ ).mulWadDown(1e18 - exponential) -
516 +lastUndistributed.mulWadDown(1e18 - exponential);
```

## CVF-39. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Recommendation** Here "x.mulWadDown(y).divWadDown(z)" could be simplified as "x.mulDivDown(y, z)".

```
497 +releaseRate.mulWadDown(1e18 - target).divWadDown(distributionFactor
    ↪ ).mulWadDown(1e18 - exponential) -
```

```
515 +releaseRate.mulWadDown(1e18 - target).divWadDown(distributionFactor
    ↪ ).mulWadDown(1e18 - exponential) -
```

## CVF-40. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Recommendation** Rounding should be up here for the whole formula to round down.

```
546 +1e18 - v.utilization.mulWadDown(1e18 - market.treasuryFeeRate())
```

## CVF-41. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Recommendation** The value is guaranteed to be "true" here.

```
590 +operation: ops[i],
```

## CVF-42. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Recommendation** The "operation" value is guaranteed to be "false" here.

```
594 +accountBalanceOps[i] = AccountOperation({ operation: ops[i],
    ↪ balance: market.balanceOf(account) });
```

## CVF-44. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “distribution[config[i].market]” is calculated several times.

**Recommendation** Consider calculating once and reusing.

```
626 +distribution[configs[i].market].availableRewards[
627 +  distribution[configs[i].market].availableRewardsCount
```

```
629 +distribution[configs[i].market].availableRewardsCount++;
630 +distribution[configs[i].market].baseUnit = 10 ** configs[i].market.
    ↪ decimals();
```

## CVF-45. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Recommendation** It would be more efficient to do increment when accessing the value.

```
627 +  distribution[configs[i].market].availableRewardsCount
```

```
629 +distribution[configs[i].market].availableRewardsCount++;
```

## CVF-46. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Recommendation** This could be simplified as; (configs[i].totalDistribution - released) / (configs[i].distributionPeriod - elapsed)

```
663 +rewardData.releaseRate = (configs[i].totalDistribution - released).
    ↪ mulWadDown(
664 +  1e18 / (configs[i].distributionPeriod - elapsed)
```

## CVF-47. FIXED

- **Category** Suboptimal
- **Source** RewardsController.sol

**Recommendation** This could be simplified as: configs[i].totalDistribution / configs[i].distributionPeriod

```
669 +rewardData.releaseRate = configs[i].totalDistribution.mulWadDown(1
       ↪ e18 / configs[i].distributionPeriod);
```

## CVF-48. FIXED

- **Category** Procedural
- **Source** RewardsController.sol

**Recommendation** This check should be performed earlier.

```
686 +if (configs[i].transitionFactor >= 1e18) revert InvalidConfig();
```

## CVF-49. FIXED

- **Category** Procedural
- **Source** RewardsController.sol

**Recommendation** This check should be performed earlier.

```
690 +if (configs[i].depositAllocationWeightFactor == 0) revert
       ↪ InvalidConfig();
```

## CVF-51. FIXED

- **Category** Documentation
- **Source** InterestRateModel.sol

**Recommendation** The word “gets” is confusing. “Returns” or “calculates“ would be better.

```
77 +/// @notice Gets the current annualized fixed rate to borrow with
      ↪ supply/demand values in the fixed rate pool and
```

## CVF-52. FIXED

- **Category** Documentation
- **Source** InterestRateModel.sol

**Description** The “utilization” returned value is not documented.

**Recommendation** Consider documenting.

87
```
+) external view returns (uint256 rate, uint256 utilization) {
```

## CVF-53. FIXED

- **Category** Documentation
- **Source** InterestRateModel.sol

**Description** The number format of the returned values is unclear.

**Recommendation** Consider documenting.

87
```
+) external view returns (uint256 rate, uint256 utilization) {
```

## CVF-55. FIXED

- **Category** Suboptimal
- **Source** Market.sol

**Description** The conditional operators here optimize very rare cases at cost of making most common cases more expensive.

**Recommendation** Consider removing these optimizations.

753
```
+if (msg.sender != to) memRewardsController.handleDeposit(to);
```

769
```
+if (from != to) memRewardsController.handleDeposit(to);
```

## CVF-57. FIXED

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

**Recommendation** This function should emit some event.

1050
```
+function setRewardsController(RewardsController rewardsController_)
   ↪  public onlyRole(DEFAULT_ADMIN_ROLE) {
```

## CVF-24. INFO

- **Category** Procedural
- **Source** RewardsController.sol

**Description** We didn't review this file.

```
10 +import { IPriceFeed } from "./utils/IPriceFeed.sol";
```

## CVF-26. INFO

- **Category** Suboptimal
- **Source** RewardsController.sol

**Recommendation** It would be more efficient to return a single array of structs with two fields, rather than two parallel arrays.

**Client Comment** *Due to ABI breaking change we prefer to avoid these kind of refactors.*

```
87 +function claimAll(address to) external returns (ERC20[] memory
   ↪ rewardsList, uint256[] memory claimedAmounts) {
```

## CVF-27. INFO

- **Category** Suboptimal
- **Source** RewardsController.sol

**Recommendation** It would be more efficient to return a single array of structs with two fields, rather than two parallel arrays.

**Client Comment** *Due to ABI breaking change we prefer to avoid these kind of refactors.*

```
101 +) public returns (ERC20[] memory, uint256[] memory claimedAmounts)
    ↪ {
```

## CVF-31. INFO

- **Category** Suboptimal
- **Source** RewardsController.sol

**Recommendation** A bit mask would be more efficient that a boolean array.

**Client Comment** *Although the approach can improve performance, it will decrease readability and at this point, it could also result in a breaking change in the ABI.*

```
227 +bool[] memory ops = new bool[](2);
228 +ops[0] = true;
229 +ops[1] = false;
```

## CVF-32. INFO

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression “distribution[market]” is calculated twice.

**Recommendation** Consider calculating once and reusing.

**Client Comment** *We tried this and the gas costs went up, we don’t find it worth.*

```
298 +uint256 baseUnit = distribution[market].baseUnit;
299 +RewardData storage rewardData = distribution[market].rewards[reward
       ↪ ];
```

## CVF-34. INFO

- **Category** Procedural
- **Source** RewardsController.sol

**Description** The returned values actually don’t have names.

**Recommendation** Consider naming them.

**Client Comment** *We don’t usually explicitly name the return value when it’s not necessary. We aren’t consistent with this rule on other parts of the code.*

```
366 +/// @return borrowIndex The index for the floating and fixed borrow
       ↪  operation.
367 +/// @return depositIndex The index for the floating deposit
       ↪ operation.
368 +/// @return lastUndistributed The last amount of undistributed
       ↪ rewards.
```

## CVF-37. INFO

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The third branch has much in common with the first to branches.

**Recommendation** Consider refactoring like this: if (lastUpdate < t.end) { // Calculate rewards for the period from lastUpdate till min(block.timestamp, end) } if (block.timestamp > t.end) { // Calculate rewards for the period from max(lastUpdate, t.end) till block>timestamp }

**Client Comment** *After working on this simplification, gas costs actually went up, that's why we are going to keep the previous approach. We appreciate the suggestion though, it relies on less code and looks more tidy.*

```
492 +if (block.timestamp <= t.end) {
```

```
503 +} else if (rewardData.lastUpdate > t.end) {
```

```
508 +} else {
```

## CVF-43. INFO

- **Category** Suboptimal
- **Source** RewardsController.sol

**Description** The expression "rewardEnabled[config[i]]" is calculated twice.

**Recommendation** Consider calculating once and reusing.

**Client Comment** *The expression is read initially and subsequently accessed for writing.*

```
619 +if (rewardEnabled[configs[i].reward] == false) {
```

```
621 +   rewardEnabled[configs[i].reward] = true;
```

## CVF-50. INFO

- **Category** Procedural
- **Source** RewardsController.sol

**Description** Declaring top level errors in a file named after a contract makes it harder to navigate through the code.

**Recommendation** Consider either moving the error declarations into the contract or moving them into a separate file.

**Client Comment** *Disagree.*

```
852 +error IndexOverflow();
853 +error InvalidConfig();
```

## CVF-54. INFO

- **Category** Procedural
- **Source** Market.sol

**Recommendation** Token symbols are usually in CAPITAL LETTERS.

**Client Comment** *We prefer to keep the 'exa' word in lowercase. Other protocols also follow this same rule (i.e 'wstETH', Beefy's 'moo' LP tokens).*

```
116 +symbol = string.concat("exa", assetSymbol);
```

## CVF-56. INFO

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

**Recommendation** The value "365 days' should be a named constant.

**Client Comment** *Disagree. We think it's already clear that represents 1 year.*

```
885 +interestRateModel.floatingRate(floatingUtilization).mulDivDown(
      ↪ block.timestamp - lastFloatingDebtUpdate, 365 days)
```

```
903 +interestRateModel.floatingRate(floatingUtilization).mulDivDown(
      ↪ block.timestamp - lastFloatingDebtUpdate, 365 days)
```

## CVF-58. INFO

- **Category** Procedural
- **Source** Market.sol

**Description** Declaring top level errors in a file named after a contract makes it harder to navigate through the code.

**Recommendation** Consider moving the declarations into the contract or moving them into a separate file.

**Client Comment** *Disagree.*

```
1284 +error ZeroBorrow();
1285 +error ZeroDeposit();
```

```
1287 +error ZeroWithdraw();
```

## CVF-59. INFO

- **Category** Procedural
- **Source** PriceFeedDouble.sol

**Description** We didn't review this file.

```
5 +import { IPriceFeed } from "./utils/IPriceFeed.sol";
```

ABDK
Consulting

## About us

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

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 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