Report

v. 1.0

Customer

Exactly

## Smart Contract Audit

# Exactly Protocol

5th November 2022

Report prepared by

ABDK Consulting

# Contents

# 1 Changelog

| # | Date | Author | Description |
|---|---|---|---|
| 0.1 | 04.11.22 | A. Zveryanskaya | Initial Draft |
| 0.2 | 04.11.22 | A. Zveryanskaya | Minor revision |
| 1.0 | 05.11.22 | 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:

- Original Repository
- Fix Repository

Files:

| | | | |
|---|---|---|---|
| **/** | | | |
| | Auditor.sol | ExactlyOracle.sol | InterestRateModel.sol |
| | MarketETHRouter.sol | Market.sol | |
| **utils/** | | | |
| | FixedLib.sol | | |

# 4 Methodology

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

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best 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 and that 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 is 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 that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check 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.

# 5 Our findings

We found 20 major, and a few less important issues. All identified Major issues have been fixed or otherwise addressed in collaboration with the client.

| | Active | Fixed |
|---|---|---|
| Major | 14 | 6 |
| Moderate | 7 | 2 |
| Minor | 43 | 29 |

Fixed 37 out of 101 issues

# 6 Major Issues

## CVF-2. INFO

- **Category** Flaw
- **Source** Market.sol

**Description** Inside the ”checkAllowance” call the borrower’s allowance is decreased by previewWithdraw(assets), while later floatingBorrowShares[borrower] is increased by previewBorrow(assets). The a borrower is not able to precisely control how much value could be taken from him by a sender.

**Recommendation** Consider denominate allowance in the same units that are taken from a borrower.

**Client Comment** *We believe the precision of control that a user can have is not 100% accurate but can still be useful for the purpose that it’s going to be used. From the protocol’s side, the allowance feature will only be used by the MarketETHRouter in the cases that users want to interact with ETH instead of WETH. By default, the web-app will suggest approving the maximum value possible, so this won’t bring issues. But, if the user wants to manually change this value, then the proportion between shares and assets will not considerably change in a short period of time, which we think won’t bring subtancial changes to what he previously allowed to be spent in the following minutes.*

```
106 checkAllowance(borrower, assets);
```

```
110 borrowShares = previewBorrow(assets);
```

```
120 floatingBorrowShares[borrower] += borrowShares;
```

## CVF-3. FIXED

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

**Description** There is no need to update the average.

**Recommendation** Just calculate the up to date value and use it, without writing it to the storage.

```
239 updateFloatingAssetsAverage();
```

```
314   updateFloatingAssetsAverage();
```

## CVF-4. FIXED

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

**Recommendation** This could be simplified and optimized as: while (packedMaturities > 0) { if (packedMaturities & 1 != 0) {...} packedMaturities »= 1; }

481
```
for (uint256 i = 0; i < 224; ) {
  if ((packedMaturities & (1 << i)) != 0) {
```

512
```
  unchecked {
    ++i;
  }
  if ((1 << i) > packedMaturities || maxAssets == 0) break;
```

550
```
for (uint256 i = 0; i < 224; ) {
  if ((packedMaturities & (1 << i)) != 0) {
```

566
```
  unchecked {
    ++i;
  }
  if ((1 << i) > packedMaturities) break;
```

729
730
```
for (uint256 i = 0; i < 224; ) {
  if ((packedMaturities & (1 << i)) != 0) {
```

741
```
  unchecked {
    ++i;
  }
  if ((1 << i) > packedMaturities) break;
```

## CVF-5. FIXED

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

**Description** The ”chargeTreasuryFee” function already updates the ”floatingAssets” variable, so this variable is updated twice here.

**Recommendation** Consider refactoring to avoid double updating.

**Client Comment** *This commit was added on top of a more updated version of the updateFloatingDebt(). We had to work on some refactors around the updateFloatingDebt() function since we discovered a bug where the floating share value decreased due to minting to the treasury in the middle of a deposit made by an external user.*

```
827 floatingAssets += chargeTreasuryFee(newDebt);
```

## CVF-6. INFO

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

**Description** There is no range check for the argument.

**Recommendation** Consider adding an appropriate check.

**Client Comment** *Since we started having lots of setters in the different contracts, it was difficult to find precise and fair min/max value checks for all of them. Besides, from a user's perspective, the level of risk is already there with core changes such as setting a new oracle, new IRM, upgrading markets. So these validations loss meaning. That's why in order to keep the protocol consistent we decided not to have them (it's important to highlight that all admin functions calls will first go through the timelock).*

```
939 function setBackupFeeRate(uint256 backupFeeRate_) public onlyRole(
      ↪ DEFAULT_ADMIN_ROLE) {
```

## CVF-7. INFO

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

**Description** There are no range checks for the arguments.

**Recommendation** Consider adding appropriate checks.

**Client Comment** *Answer is in CVF-5.*

```
948 function setDampSpeed(uint256 up, uint256 down) public onlyRole(
    ↪ DEFAULT_ADMIN_ROLE) {
```

## CVF-8. INFO

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

**Description** There is no range check for the argument.

**Recommendation** Consider adding an appropriate check.

**Client Comment** *Answer is in CVF-5.*

```
958 function setEarningsAccumulatorSmoothFactor(uint128
    ↪ earningsAccumulatorSmoothFactor_)
```

## CVF-9. INFO

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

**Description** There is no range check for the argument.

**Recommendation** Consider adding an appropriate check.

**Client Comment** *Answer is in CVF-5.*

```
976 function setMaxFuturePools(uint8 futurePools) public onlyRole(
    ↪ DEFAULT_ADMIN_ROLE) {
```

## CVF-10. INFO

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

**Description** There is no range check for the argument.

**Recommendation** Consider adding an appropriate check.

**Client Comment** *Answer is in CVF-5.*

```
984 function setPenaltyRate(uint256 penaltyRate_) public onlyRole(
    ↪ DEFAULT_ADMIN_ROLE) {
```

## CVF-11. INFO

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

**Description** There is no range check for the argument.

**Recommendation** Consider adding an appropriate check.

**Client Comment** *Answer is in CVF-5.*

```
992 function setReserveFactor(uint128 reserveFactor_) public onlyRole(
    ↪ DEFAULT_ADMIN_ROLE) {
```

## CVF-12. INFO

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

**Description** There is no range check for the ”treasuryFeeRate_” argument.

**Recommendation** Consider adding an appropriate check.

**Client Comment** *Answer is in CVF-5.*

```
1000 function setTreasury(address treasury_, uint256 treasuryFeeRate_)
     ↪ public onlyRole(DEFAULT_ADMIN_ROLE) {
```

## CVF-13. FIXED

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

**Description** The returned value is ignored.

**Recommendation** Consider reverting in case false was returned.

```
35 weth.approve(address(market), type(uint256).max);
```

## CVF-14. INFO

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

**Recommendation** It is a good practice to always round amount towards a protocol, i.e. against a user. In this case it means to round up, rather than down.

**Client Comment** *If we round up towards the protocol and against the user, there's this case were a user who only deposited in the pool and has that market checked as collateral, can't withdraw his whole position. We added a test to back up this scenario.*

```
115 sumDebtPlusEffects += withdrawAmount.mulDivDown(vars.oraclePrice,
    ↪ 10**decimals).mulWadDown(adjustFactor);
```

## CVF-15. INFO

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

**Description** The named of this function suggests that it is a checker, i.e. a function with not side effects nor return value, that either does nothing or reverts. However actually, this function may modify the state.

**Recommendation** Consider renaming.

**Client Comment** *We couldn't come up with a better name that fulfilled our expectations.*

```
130 function checkBorrow(Market market, address borrower) external {
```

## CVF-16. FIXED

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

**Description** Unlike other similar loops, this loop doesn't have a check to break in case there are no set bits left.

**Recommendation** Consider adding such check.

```
285 for (uint256 i = 0; i < marketCount; ) {
```

## CVF-17. INFO

- **Category** Suboptimal
- **Source** Auditor.sol

**Recommendation** It would be enough to do: if (assets > 0) return;

**Client Comment** *The handleBadDebt function is external and permissionless because we discovered that if some conditions are met, there's the possibility of a borrower still having positive collateral assets but 0 adjusted collateral, what would cause a revert in the liquidate transaction and that position wouldn't be able to be liquidated neither cleared off the debt. That's mainly the reason why the if is checking against the adjusted collateral and not only the assets amount. The protocol also counts on that function to directly invoke it if there's a borrower with bad debt that can't be liquidated.*

```
290 if (assets.mulDivDown(oracle.assetPrice(market), 10**m.decimals).
    ↪ mulWadDown(m.adjustFactor) > 0) return;
```

## CVF-18. INFO

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

**Description** There is not range check for the "adjustFactor" argument.

**Recommendation** Consider adding an appropriate check.

```
341 function setAdjustFactor(Market market, uint128 adjustFactor) public
    ↪  onlyRole(DEFAULT_ADMIN_ROLE) {
```

## CVF-19. FIXED

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

**Description** Incremental updates could accumulate errors.

**Recommendation** Consider doing like this: return scaleProportionally(position, position.principal + position.fee - amount);

```
131 position.principal -= principal;
    position.fee -= amount - principal;
```

## CVF-20. INFO

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

**Description** There is no explicit check that a price feed exists for the market.

**Recommendation** Consider adding such check.

**Client Comment** *We decided not to take actions since it's going to increase gas costs for all paths.*

```
34 (, int256 price, , uint256 updatedAt, ) = priceFeeds[market].
      ↪ latestRoundData();
```

## CVF-21. INFO

- **Category** Suboptimal
- **Source** InterestRateModel.sol

**Description** Using annual interest rates in smart contracts usually makes calculations more complicated and less precise.

**Recommendation** Consider using per-second rates instead and converting from an annual rate to the per-second rate off-chain.

**Client Comment** *The annual calculation is highly coupled around web app queries, maths calculations and code, so we feel it's not worth making changes around it.*

```
63 return fixedRate(utilizationBefore, utilizationAfter).mulDivDown(
      ↪ maturity - block.timestamp, 365 days);
```

# 7 Moderate Issues

## CVF-22. FIXED

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

**Description** he actual amount of shares subtracted from the borrower could be less than this value, in case this value is less than the total amount of the borrower's shares.

**Recommendation** Consider returning the actual subtracted amount.

```
139 borrowShares = previewRepay(assets);
```

## CVF-23. FIXED

- **Category** Flaw
- **Source** Market.sol

**Recommendation** The "previewRedeem" function should be used instead of "previewMint".

```
683 auditor.checkShortfall(this, owner, previewMint(shares));
```

```
694 auditor.checkShortfall(this, msg.sender, previewMint(shares));
```

```
709 auditor.checkShortfall(this, from, previewMint(shares));
```

## CVF-24. INFO

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

**Description** Overflow is possible here.

**Recommendation** Consider adding an explicit "require" statement to forbid having more than 256 markets.

**Client Comment** *We know the overflow is possible but we won't have more than 256 markets, it's also written in the @dev comment in natspec. We don't think is worth adding an extra check.*

```
329 index: uint8(marketList.length)
```

## CVF-25. INFO

- **Category** Suboptimal
- **Source** FixedLib.sol

**Description** This formula means that the yield for any ”amount” above ”memBackupSupplied” is zero.

**Recommendation** Consider just forbidding amounts greater than backup supplied.

**Client Comment** *We are aware of this characteristic and we consider it as ’free lunch’. We believe that the protocol should have as few prohibitions and caps as possible, at least from the smart contracts’ level, in order to help with the idea of a free and permisionless market/protocol. That’s why we also didn’t consider the idea of limiting borrows with a minimum amount required. We also plan to warn users who interact through the webpp, once they reach the max yield possible with a certain amount for a fixed deposit and finally we will be explicit about this in Exactly’s docs.*

```
25 yield = pool.unassignedEarnings.mulDivDown(Math.min(amount,
     ↪ memBackupSupplied), memBackupSupplied);
```

## CVF-26. INFO

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

**Description** It is not checked that ”maturity” fits into 32 bits.

**Recommendation** Consider adding such check. Alternatively, use the ”uint32” type for ”maturity”.

**Client Comment** *Requires a lot of implications in a lot of places in the code.*

```
158 if (encoded == 0) return maturity | (1 << 32);
```

```
167   return maturity | encoded | (1 << 32);
```

## CVF-27. INFO

- **Category** Flaw
- **Source** FixedLib.sol

**Description** If range > 256 - 32, this check could produce false positives, as it will check bits from the last 32, that will be wiped out anyway.

**Recommendation** Consider ignoring the last 32 bits in this check.

**Client Comment** *We will only support range <= 224.*

```
165 if (encoded >> (256 - range) != 0) revert MaturityOverflow();
```

## CVF-28. INFO

- **Category** Suboptimal
- **Source** FixedLib.sol

**Description** This uses linear search to find out the lowest bit set in a word.

**Recommendation** Consider using binary search: https://github.com/Uniswap/solidity-ib/blob/master/contracts/libraries/BitMath.sol#L44-L84

**Client Comment** *We're not going to optimize this. Refactoring this without being audited again is not a good idea, not a trivial change in the loop, too much extra complexity.*

```
188 while ((packed & 1) == 0 && packed != 0) {
```

## CVF-29. INFO

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

**Description** Underflow is possible here in case maturity < baseMaturity.

**Recommendation** Consider returning the original encoded maturity in such a case.

**Client Comment** *We'll not take any actions since this is a case where it would revert before. Maturity will not get up to that point like that.*

```
198 return encoded & ~(1 << (32 + ((maturity - baseMaturity) / INTERVAL)
    ↪ ));
```

## CVF-30. INFO

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

**Description** Overflow is possible when converting to ”int256”.

**Recommendation** Consider using safe conversion.

**Client Comment** *Won’t take any actions.*

```
89 int256((fixedMaxUtilization - utilizationBefore).divWadDown(
      ↪ fixedMaxUtilization - utilizationAfter)).lnWad()
```

```
110 int256((floatingMaxUtilization - utilizationBefore).divWadDown(
      ↪ floatingMaxUtilization - utilizationAfter))
```

# 8 Minor Issues

## CVF-31. INFO

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

**Recommendation** Specifying a particular compiler version makes it harder to upgrade to newer versions. Consider specifying as ”^0.8.0”. Also relevant for: MarketETHRouter.sol, Auditor.sol, FixedLib.sol, ExactlyOracle.sol, InterestRateModel.sol.

**Client Comment** *Questionable. Even slither consider it a bad practice not to set a fixed version.*

```
2 pragma solidity 0.8.16;
```

## CVF-32. INFO

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

**Description** We didn’t review these files.

```
5 import { FixedPointMathLib } from "solmate/src/utils/
    ↪ FixedPointMathLib.sol";
```

```
8 import { ERC4626, ERC20, SafeTransferLib } from "solmate/src/mixins/
    ↪ ERC4626.sol";
```

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

## CVF-33. INFO

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

**Description** It is unclear, whether the ”Pausable” and ”ERC4626” contracts are upgradeable.

**Client Comment** *For the ERC4626 we are using Solmate's solution since it's more gas efficient. Our deploy script already checks for storage collision. So let's say they change the storage layout and we decide to get the new version then our deploy script will prevent this from happening. However, if this happens (low chances) and we still want to apply the change we will probably end up writing a custom implementation and we are fine with it. We also considered using Open Zeppelin's implementation but we realized there's no much advantage in the tradeoff.*

```
14 contract Market is Initializable, AccessControlUpgradeable, Pausable
   ↪ , ERC4626 {
```

## CVF-34. INFO

- **Category** Documentation
- **Source** Market.sol

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

**Recommendation** Consider documenting.

**Client Comment** *We will add natspec for this clarification. We will work on this before deploying the protocol on Mainnet.*

```
26 mapping(uint256 => mapping(address => FixedLib.Position)) public
   ↪ fixedDepositPositions;
   mapping(uint256 => mapping(address => FixedLib.Position)) public
   ↪ fixedBorrowPositions;
```

## CVF-35. INFO

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

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

**Client Comment** *At this point it requires to many changes around other integrations and the improvement is not much.*

```
26 mapping(uint256 => mapping(address => FixedLib.Position)) public
      ↪ fixedDepositPositions;
   mapping(uint256 => mapping(address => FixedLib.Position)) public
      ↪ fixedBorrowPositions;
```

```
32 mapping(uint256 => FixedLib.Pool) public fixedPools;
```

## CVF-36. FIXED

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

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

```
28 mapping(address => uint256) public floatingBorrowShares;
```

```
30 mapping(address => uint256) public fixedBorrows;
   mapping(address => uint256) public fixedDeposits;
```

## CVF-37. INFO

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

**Description** The number format of these variables is unclear.

**Recommendation** Consider documenting.

**Client Comment** *We will add natspec for this clarification. We will work on this before deploying the protocol on Mainnet.*

```
38 uint256 public penaltyRate;
   uint256 public backupFeeRate;
40 uint256 public dampSpeedUp;
   uint256 public dampSpeedDown;
```

```
50 uint128 public earningsAccumulatorSmoothFactor;
   uint128 public reserveFactor;
```

```
57 uint256 public floatingUtilization;
```

```
60 uint256 public treasuryFeeRate;
```

## CVF-38. INFO

- **Category** Readability
- **Source** Market.sol

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

**Client Comment** *We consider 1e18 is already clear enough to be used as a hardcoded value.*

```
115 if (floatingBackupBorrowed + newFloatingDebt > floatingAssets.
       ↪ mulWadDown(1e18 - reserveFactor)) {
```

```
249   if (memFloatingBackupBorrowed + floatingDebt > floatingAssets.
         ↪ mulWadDown(1e18 - reserveFactor)) {
```

```
316     1e18 +
```

```
806 uint256 averageFactor = uint256(1e18 - (-int256(dampSpeedFactor * (
       ↪ block.timestamp - lastAverageUpdate))).expWad());
```

```
808   memFloatingAssetsAverage.mulWadDown(1e18 - averageFactor) +
```

```
874     (totalFloatingBorrowAssets() - floatingDebt).mulWadDown(1e18 -
           ↪ treasuryFeeRate);
```

## CVF-39. FIXED

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

**Description** This function should also return the actual amount of shares subtracted from the borrower, as this amount could be less than ”borrowShares”.

```
149 function refund(uint256 borrowShares, address borrower) external
       ↪ whenNotPaused returns (uint256 assets) {
```

## CVF-40. FIXED

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

**Description** This function should also return the actual amount of shares subtracted from the borrower, as this amount could be less than ”borrowShares”.

159
```
function noTransferRefund(uint256 borrowShares, address borrower)
    ↪ internal returns (uint256 assets) {
```

## CVF-41. FIXED

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

**Description** The expression ”fixedDepositPositions[maturity][receiver]” is calculated twice.

**Recommendation** Consider calculating once and reusing.

202
```
FixedLib.Position storage position = fixedDepositPositions[maturity
    ↪ ][receiver];
```

207
```
fixedDepositPositions[maturity][receiver] = FixedLib.Position(
    ↪ position.principal + assets, position.fee + fee);
```

## CVF-42. FIXED

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

**Recommendation** This assignment should be done after the check below.

248
```
floatingBackupBorrowed = memFloatingBackupBorrowed;
```

## CVF-43. FIXED

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

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

```
255 if (assetsOwed > maxAssets) revert Disagreement();
```

## CVF-44. INFO

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

**Description** The expression ”position.principle + position.fee” is calculated twice.

**Recommendation** Consider calculating once and reusing.

**Client Comment** *It’s a sum operation but we are reading the vars from memory so it’s not that expensive.*

```
310 if (positionAssets > position.principal + position.fee)
    ↪ positionAssets = position.principal + position.fee;
```

## CVF-45. FIXED

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

**Description** Here a redundant overflow check is performed for addition.

**Recommendation** Consider using bitwise OR instead: if (position.principal | position.fee == 0) {

```
349 if (position.principal + position.fee == 0) {
```

```
446 if (position.principal + position.fee == 0) {
```

## CVF-46. INFO

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

**Description** This function should also return the amount of borrow shares burned.

**Client Comment** *The fixed borrows are not accounted with shares denomination, so there's no point in only returning the subtracted floating borrow shares. In our opinion we can't return any better than repaidAssets.*

```
468 function liquidate(
```

## CVF-47. FIXED

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

**Recommendation** Bitwise AND would be more efficient.

```
479 uint256 baseMaturity = packedMaturities % (1 << 32);
548 uint256 baseMaturity = packedMaturities % (1 << 32);
726 uint256 baseMaturity = packedMaturities % (1 << 32);
```

## CVF-48. INFO

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

**Description** This loop uses linear search to find non-zero positions.

**Recommendation** Consider using faster approaches like binary search. Alternatively, consider allowing the liquidator to specify what maturities he wants to liquidate.

**Client Comment** *We're not going to optimize this. Refactoring this without being audited again is not a good idea, not a trivial change in the loop, too much extra complexity.*

```
481 for (uint256 i = 0; i < 224; ) {
550 for (uint256 i = 0; i < 224; ) {
729 for (uint256 i = 0; i < 224; ) {
```

## CVF-49. FIXED

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

**Recommendation** This could be simplified as: if (debt > maxAssets && maxAssets.mulDivDown(position, debt) == 0) maxAssets = 0;

**Client Comment** *Removed.*

```
506 if ((debt > maxAssets ? maxAssets.mulDivDown(position, debt) :
    ↪ maxAssets) == 0) maxAssets = 0;
```

## CVF-50. INFO

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

**Description** This function should return the amount of shares burned.

**Client Comment** *The seize function is only being called by another Market. As a downside we increase gas cost and bytecode size.*

```
596 function seize(
```

## CVF-51. INFO

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

**Description** This function should return the amount of shares burned.

**Client Comment** *The internalSeize function is internal so it's only being called inside the Market. As a downside we increase gas cost and bytecode size.*

```
612 function internalSeize(
```

## CVF-52. FIXED

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

**Recommendation** This assignment should be done after the check below.

```
641 floatingAssets = newFloatingAssets;
```

## CVF-53. INFO

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

**Recommendation** This function should be called before every update of the ”floatingAssets” variable.

**Client Comment** *When calculating the utilization rate for fixed borrows, an average of the floating assets is sent to the InterestRateModel. The reason behind this is to prevent a manipulation of a fixed borrow rate: a user could deposit a big amount in the floating pool to lower the utilization, ask for a very cheap fixed borrow and then withdraw the initially deposited amount. The feature is already working and covering this case only with the few floatingAssetsAverage updates that we have on the code, so it's not necessary to spend more gas in any other floatingAssets minimal update.*

```
802 function updateFloatingAssetsAverage() internal {
```

## CVF-54. 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.*

```
822 365 days
```

```
844 365 days
```

## CVF-55. FIXED

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

**Description** The name of this function looks like a checker name, which suggests that this function don't modify the state and don't return any value, but just revert in some cases and does nothing in the others. However actually, this function does modify the state.

**Recommendation** Consider renaming to "consumeAllowance".

908
```
function checkAllowance(address account, uint256 assets) internal {
```

## CVF-56. INFO

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

**Description** Here an underflow check added by compiler is used to enforce a business-evel constraint. This is a bad practice, as it makes code harder to read and more error-rone.

**Recommendation** Consider adding an explicit "require" statement to check that the allowance is sufficient.

**Client Comment** *We are fine with it, this is already how the withdraw works.*

912
```
if (allowed != type(uint256).max) allowance[account][msg.sender] =
    ↪ allowed - previewWithdraw(assets);
```

## CVF-57. INFO

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

**Description** Events are usually named via nouns.

**Recommendation** Consider removing "Set" from the names.

**Client Comment** *Open Zeppelin uses the same naming, at least we have consistency on them. We will not change this.*

```
1112 event BackupFeeRateSet(uint256 backupFeeRate);

1117 event DampSpeedSet(uint256 dampSpeedUp, uint256 dampSpeedDown);

1121 event EarningsAccumulatorSmoothFactorSet(uint256
        ↪ earningsAccumulatorSmoothFactor);

1125 event InterestRateModelSet(InterestRateModel indexed
        ↪ interestRateModel);

1129 event MaxFuturePoolsSet(uint256 maxFuturePools);

1133 event PenaltyRateSet(uint256 penaltyRate);

1137 event ReserveFactorSet(uint256 reserveFactor);

1142 event TreasurySet(address indexed treasury, uint256 treasuryFeeRate)
        ↪ ;
```

## CVF-58. INFO

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

**Recommendation** Including timestamp in an event is redundant, as each event is bound to a block, which has a timestamp anyway.

**Client Comment** *Json RPC getLogs doesn't return the timestamp. We are aware of this and we did that way to make it easier for log consumption.*

```
1152   uint256 timestamp,
```

```
1164 event FixedEarningsUpdate(uint256 timestamp, uint256 indexed
       ↪ maturity, uint256 unassignedEarnings);
```

```
1168 event AccumulatorAccrual(uint256 timestamp);
```

```
1173 event FloatingDebtUpdate(uint256 timestamp, uint256 utilization);
```

## CVF-59. INFO

- **Category** Suboptimal
- **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 either moving error declaration into the contract or moving them into a separate file named "errors.sol".

**Client Comment** *Disagree.*

```
1176 error AlreadyInitialized();
     error Disagreement();
     error InsufficientProtocolLiquidity();
     error NotAuditor();
1180 error SelfLiquidation();
     error ZeroWithdraw();
     error ZeroRepay();
```

## CVF-61. FIXED

- **Category** Readability
- **Source** MarketETHRouter.sol

**Description** These lines look like simple assignments, while actually they return values.

**Recommendation** Consider using ”return” statements instead for readability.

```
43 shares = market.deposit(msg.value, msg.sender);
```

```
47 shares = market.withdraw(assets, address(this), msg.sender);
```

```
51 borrowShares = market.borrow(assets, address(this), msg.sender);
```

## CVF-62. INFO

- **Category** Suboptimal
- **Source** MarketETHRouter.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 declaration into the contract or moving it into a separate file named ”errors.sol”.

**Client Comment** *Disagree.*

```
104 error NotFromWETH();
```

## CVF-63. INFO

- **Category** Procedural
- **Source** Auditor.sol

**Description** We didn’t review these files.

```
5 import { FixedPointMathLib } from "solmate/src/utils/
    ↪ FixedPointMathLib.sol";
```

## CVF-64. INFO

- **Category** Documentation
- **Source** Auditor.sol

**Description** The semantics of keys and values in this mapping is unclear.

**Recommendation** Consider documenting.

**Client Comment** *We will add natspec for this clarification. We will work on this before deploying the protocol on Mainnet.*

```
16 mapping(address => uint256) public accountMarkets;
```

## CVF-65. FIXED

- **Category** Suboptimal
- **Source** Auditor.sol

**Description** The expression ”1 « m.index” is calculated twice.

**Recommendation** Consider calculating once and reusing.

```
46 if ((marketMap & (1 << m.index)) != 0) return;
accountMarkets[msg.sender] = marketMap | (1 << m.index);
```

## CVF-66. FIXED

- **Category** Suboptimal
- **Source** Auditor.sol

**Description** The expression ”1 « m.index” is calculated twice.

**Recommendation** Consider calculating once and reusing.

```
70 if ((marketMap & (1 << m.index)) == 0) return;
accountMarkets[msg.sender] = marketMap & ~(1 << m.index);
```

## CVF-67. FIXED

- **Category** Readability
- **Source** Auditor.sol

**Recommendation** This could be optimized as: while (marketMap != 0) { if (marketMap & 1 != 0) {...) marketMap »= 1; }

```
92  for (uint256 i = 0; i < maxValue; ) {
      if ((marketMap & (1 << i)) != 0) {
```

```
119   unchecked {
120     ++i;
      }
      if ((1 << i) > marketMap) break;
```

```
189 for (uint256 i = 0; i < marketCount; ) {
190   if ((marketMap & (1 << i)) != 0) {
```

```
212   unchecked {
        ++i;
      }
      if ((1 << i) > marketMap) break;
```

```
285 for (uint256 i = 0; i < marketCount; ) {
      if ((marketMap & (1 << i)) != 0) {
```

```
293   unchecked {
        ++i;
      }
```

## CVF-68. INFO

- **Category** Suboptimal
- **Source** Auditor.sol

**Description** These loops use linear search to find bits that are set in a market map.

**Recommendation** Consider using more efficient approaches such as binary search.

**Client Comment** *We're not going to optimize this. Refactoring this without being audited again is not a good idea, not a trivial change in the loop, too much extra complexity.*

92
```
for (uint256 i = 0; i < maxValue; ) {
```

189
```
for (uint256 i = 0; i < marketCount; ) {
```

285
```
for (uint256 i = 0; i < marketCount; ) {
```

## CVF-69. INFO

- **Category** Suboptimal
- **Source** Auditor.sol

**Description** Division before multiplication here could lead to loss of precision.

**Recommendation** Consider doing all divisions at the very end like this: sumCollateral += vars.balance.mulDivDown (vars.oraclePrice * adjustFactor, 10**(decimals + 18));

**Client Comment** *In the checkLiquidation function we are also calculating the sumCollateral in the same way. However, in the checkLiquidation we are achieving this in two different steps, that's why if we optimize this only in the accountLiquidity function we can end up resulting in an inconsistency between two different values.*

105
```
sumCollateral += vars.balance.mulDivDown(vars.oraclePrice, 10**
    ↪ decimals).mulWadDown(adjustFactor);
```

## CVF-70. INFO

- **Category** Suboptimal
- **Source** Auditor.sol

**Recommendation** It would be more efficient to store 10**decimals value in the"MarketData" struct instead of just decimals.

**Client Comment** *Storing the base unit turned out to be more expensive, instead we worked on a smaller improvement.*

```
105 sumCollateral += vars.balance.mulDivDown(vars.oraclePrice, 10**
    ↪ decimals).mulWadDown(adjustFactor);
```

```
108 sumDebtPlusEffects += vars.borrowBalance.mulDivUp(vars.oraclePrice,
    ↪ 10**decimals).divWadUp(adjustFactor);
```

```
115     sumDebtPlusEffects += withdrawAmount.mulDivDown(vars.oraclePrice
        ↪ , 10**decimals).mulWadDown(adjustFactor);
```

## CVF-71. INFO

- **Category** Suboptimal
- **Source** Auditor.sol

**Description** Division before multiplication here could lead to loss of precision.

**Recommendation** Consider doing all divisions at the very end like this: sumDebtPlusEffects += vars.borrowBalance.mulDivUp (vars.oraclePrice * adjustFactor, 10**(decimals + 18));

**Client Comment** *In the checkLiquidation function we are also calculating the sumDebtPlusEffects in the same way. However, in the checkLiquidation we are achieving this in two different steps, that's why if we optimize this only in the accountLiquidity function we can end up resulting in an inconsistency between two different values.*

```
108 sumDebtPlusEffects += vars.borrowBalance.mulDivUp(vars.oraclePrice,
    ↪ 10**decimals).divWadUp(adjustFactor);
```

## CVF-72. INFO

- **Category** Procedural
- **Source** Auditor.sol

**Description** Division before multiplication here could lead to loss of precision.

**Recommendation** Consider doing all divisions at the very end like this: sumDebtPlusEffects += withdrawAmount.mulDivDown (vars.oraclePrice * adjustFactor, 10**(decimals + 18));

**Client Comment** *In the checkLiquidation function we are also calculating the sumDebtPlusEffects in the same way. However, in the checkLiquidation we are achieving this in two different steps, that's why if we optimize this only in the accountLiquidity function we can end up in an inconsistency resulting in two different values.*

```
115 sumDebtPlusEffects += withdrawAmount.mulDivDown(vars.oraclePrice,
        ↪ 10**decimals).mulWadDown(adjustFactor);
```

## CVF-73. FIXED

- **Category** Suboptimal
- **Source** Auditor.sol

**Description** This condition is always true. No need to check it, at least here.

**Recommendation** Consider removing this check or moving after the surrounding conditional statement.

```
143 // it should be impossible to break this invariant
    assert((accountMarkets[borrower] & (1 << m.index)) != 0);
```

## CVF-74. FIXED

- **Category** Suboptimal
- **Source** Auditor.sol

**Description** The value "10**decimals" is calculated twice.

**Recommendation** Consider calculating once and reusing.

```
203 uint256 value = debt.mulDivUp(m.price, 10**m.decimals);
```

```
207 value = collateral.mulDivDown(m.price, 10**m.decimals);
```

## CVF-75. FIXED

- **Category** Suboptimal
- **Source** Auditor.sol

**Description** Division before multiplication here could lead to loss of precision.

**Recommendation** Consider doing all divisions at the very end.

```
221 uint256 adjustFactor = usd.adjustedCollateral.divWadUp(usd.
       ↪ totalCollateral).mulWadUp(
      usd.totalDebt.divWadUp(usd.adjustedDebt)
    );
```

## CVF-76. INFO

- **Category** Suboptimal
- **Source** Auditor.sol

**Description** Division before multiplication here could lead to loss of precision.

**Recommendation** Consider doing all divisions at the very end.

```
224 uint256 closeFactor = (TARGET_HEALTH - usd.adjustedCollateral.
       ↪ divWadUp(usd.adjustedDebt)).divWadUp(
      TARGET_HEALTH - adjustFactor.mulWadDown(1e18 + memIncentive.
         ↪ liquidator + memIncentive.lenders)
    );
```

## CVF-77. FIXED

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

**Recommendation** This value should be a named constant.

```
234 maxLiquidatorAssets <
       ↪ 115792089237316195423570985008687907853269984665640564039457
       ↪ // type(uint256).max / WAD
```

## CVF-78. INFO

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

**Description** There are no range checks for these arguments.

**Recommendation** Consider adding appropriate checks.

**Client Comment** *Answer is in CVF-5.*

318
```
uint128 adjustFactor,
uint8 decimals
```

## CVF-79. INFO

- **Category** Suboptimal
- **Source** Auditor.sol

**Description** This event is emitted even if there is no actual change.

**Client Comment** *We don't consider worth changing it. Not a problem.*

345
```
emit AdjustFactorSet(market, adjustFactor);
```

355
```
emit LiquidationIncentiveSet(liquidationIncentive_);
```

362
```
emit OracleSet(oracle_);
```

## CVF-80. INFO

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

**Recommendation** Events are usually named via nouns, such as ”MarketListing” or just ”Market”, ”MarketEntering” etc.

**Client Comment** *Open Zeppelin uses the same naming, at least we have consistency on them. We will not change this.*

```
368 event MarketListed(Market market, uint8 decimals);
```

```
373 event MarketEntered(Market indexed market, address indexed account);
```

```
379 event MarketExited(Market indexed market, address indexed account);
```

```
384 event AdjustFactorSet(Market indexed market, uint256 adjustFactor);
```

```
388 event LiquidationIncentiveSet(LiquidationIncentive
    ↪ liquidationIncentive);
```

```
392 event OracleSet(ExactlyOracle oracle);
```

## CVF-81. FIXED

- **Category** Procedural
- **Source** Auditor.sol

**Recommendation** The ”market” parameter should be indexed.

```
368 event MarketListed(Market market, uint8 decimals);
```

## CVF-82. FIXED

- **Category** Procedural
- **Source** Auditor.sol

**Recommendation** The parameters should be indexed.

**Client Comment** *We will not index the LiquidationIncentive one.*

```
388 event LiquidationIncentiveSet(LiquidationIncentive
    ↪ liquidationIncentive);
```

```
392 event OracleSet(ExactlyOracle oracle);
```

## CVF-83. INFO

- **Category** Documentation
- **Source** Auditor.sol

**Description** The number format for these fields is unclear.

**Recommendation** Consider documenting.

**Client Comment** *We will add natspec for this clarification. We will work on this before deploying the protocol on Mainnet.*

```
395   uint128 liquidator;
      uint128 lenders;
```

```
402   uint256 oraclePrice;
```

```
406   uint128 adjustFactor;
```

```
422 uint256 price;
    uint128 adjustFactor;
```

## CVF-84. INFO

- **Category** Suboptimal
- **Source** Auditor.sol

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

**Recommendation** Consider either moving the declarations into the contract or moving them into separate files named ”errors.sol” and ”types.sol”.

**Client Comment** *Disagree.*

```
413 error AuditorMismatch();
    error InsufficientAccountLiquidity();
    error InsufficientShortfall();
    error MarketAlreadyListed();
    error MarketNotListed();
    error NotMarket();
    error RemainingDebt();
```

```
421 struct MarketVars {
      uint256 price;
      uint128 adjustFactor;
      uint8 decimals;
    }
```

```
427 struct LiquidityVars {
      uint256 totalDebt;
      uint256 totalCollateral;
430   uint256 adjustedDebt;
      uint256 adjustedCollateral;
      uint256 seizeAvailable;
    }
```

## CVF-85. INFO

- **Category** Procedural
- **Source** FixedLib.sol

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

```
5 import { FixedPointMathLib } from "solmate/src/utils/
    ↪ FixedPointMathLib.sol";
```

## CVF-86. FIXED

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

**Description** A public constant in a library doesn't make much sense.

**Recommendation** Consider declaring as internal.

```
10 uint256 public constant INTERVAL = 4 weeks;
```

## CVF-87. INFO

- **Category** Suboptimal
- **Source** FixedLib.sol

**Description** Here a returned value is used as a mutable variable. This is a bad practice that makes code harder to read.

**Recommendation** Consider using a separate variable instead.

```
27 yield -= backupFee;
```

## CVF-88. FIXED

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

**Description** The variable name is confusing, as it is not always equal to the original supply value.

**Recommendation** Consider renaming.

```
62 uint256 oldSupply = Math.max(borrowed, pool.supplied);
```

## CVF-89. FIXED

- **Category** Suboptimal
- **Source** FixedLib.sol

**Recommendation** This function would be simpler and more efficient if the cases "maturity > block.timestamp" and "maturity <= block.timestamp" would be treated separately like this:
if (lastAccrual == maturity) {...} else if (block.timestamp >= maturity) {...} else {...}

```
85 function accrueEarnings(Pool storage pool, uint256 maturity)
     ↪ internal returns (uint256 backupEarnings) {
```

## CVF-90. FIXED

- **Category** Suboptimal
- **Source** FixedLib.sol

**Recommendation** As "lastAccrual" may never exceed neither "maturity" nor "block.timestamp" it is safe here to use plain subtraction instead of the "secondsPre" function.

```
92 uint256 secondsSinceLastAccrue = secondsPre(lastAccrual, Math.min(
     ↪ maturity, block.timestamp));
```

```
94 uint256 secondsTotalToMaturity = secondsPre(lastAccrual, maturity);
```

## CVF-91. FIXED

- **Category** Suboptimal
- **Source** FixedLib.sol

**Recommendation** Bitwise AND would be more efficient.

```
160 uint256 baseMaturity = encoded % (1 << 32);
```

```
182 uint256 baseMaturity = encoded % (1 << 32);
```

## CVF-92. INFO

- **Category** Procedural
- **Source** FixedLib.sol

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

**Recommendation** Consider either moving the error definitions into the library or moving them into a separate file named ”errors.sol”.

**Client Comment** *Disagree.*

```
264 error MaturityOverflow();
    error UnmatchedPoolState(uint8 state, uint8 requiredState);
    error UnmatchedPoolStates(uint8 state, uint8 requiredState, uint8
       ↪ alternativeState);
```

## CVF-93. INFO

- **Category** Procedural
- **Source** ExactlyOracle.sol

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

```
5 import { AggregatorV2V3Interface } from "@chainlink/contracts/src/v0
     ↪ .8/interfaces/AggregatorV2V3Interface.sol";
```

## CVF-94. INFO

- **Category** Procedural
- **Source** ExactlyOracle.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 named ”errors.sol”.

**Client Comment** *Disagree.*

```
57 error InvalidPrice();
   error InvalidSource();
```

## CVF-95. INFO

- **Category** Procedural
- **Source** InterestRateModel.sol

**Description** We didn't review these files.

```
4 import { Math } from "@openzeppelin/contracts/utils/math/Math.sol";
  import { FixedPointMathLib } from "solmate/src/utils/
     ↪ FixedPointMathLib.sol";
```

## CVF-96. INFO

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

**Description** The number format of these variables is unclear.

**Recommendation** Consider documenting.

**Client Comment** *We will add natspec for this clarification. We will work on this before deploying the protocol on Mainnet.*

```
11 uint256 public immutable fixedCurveA;
   int256 public immutable fixedCurveB;
   uint256 public immutable fixedMaxUtilization;
```

```
15 uint256 public immutable floatingCurveA;
   int256 public immutable floatingCurveB;
   uint256 public immutable floatingMaxUtilization;
```

## CVF-97. INFO

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

**Description** The number format of these arguments is unclear.

**Recommendation** Consider documenting.

**Client Comment** *We will add natspec for this clarification. We will work on this before deploying the protocol on Mainnet.*

```
20 uint256 fixedCurveA_,
   int256 fixedCurveB_,
   uint256 fixedMaxUtilization_,
   uint256 floatingCurveA_,
   int256 floatingCurveB_,
   uint256 floatingMaxUtilization_
```

## CVF-98. FIXED

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

**Recommendation** Should be ”represented with 18 decimals”.

```
47 /// @return rate of the fee that the borrower will have to pay (
      ↪ represented with 1e18 decimals).
```

## CVF-99. FIXED

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

**Description** This should be calculated only if utilizationAfter <= 1e18.

```
58 uint256 utilizationBefore = borrowed.divWadDown(potentialAssets);
```

## CVF-100. INFO

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

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

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

```
63 return fixedRate(utilizationBefore, utilizationAfter).mulDivDown(
   ↪ maturity - block.timestamp, 365 days);
```

## CVF-101. FIXED

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

**Recommendation** The value ”2.5e9” should be a named constant.

```
85 utilizationAfter - utilizationBefore < 2.5e9
```

```
106 utilizationAfter - utilizationBefore < 2.5e9
```

## CVF-102. INFO

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

**Description** Underflow is possible when converting to ”uint256”.

**Recommendation** Consider using safe conversion.

**Client Comment** *Not worth adding a check.*

```
88 uint256(
```

```
109 uint256(
```

## CVF-103. INFO

- **Category** Procedural
- **Source** InterestRateModel.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 named ”errors.sol”.

**Client Comment** *Disagree*.

```
121 error AlreadyMatured();
    error UtilizationExceeded();
```

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