Report

v. 2.0

Customer

Contango

Smart Contract Audit

Core V2

3rd October 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	02.10.23	A. Zveryanskaya	Initial Draft
0.2	02.10.23	A. Zveryanskaya	Minor revision
1.0	02.10.23	A. Zveryanskaya	Release
1.1	03.10.23	A. Zveryanskaya	Introduction updated
1.2	03.10.23	A. Zveryanskaya	Fix link updated
2.0	03.10.23	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.

Contango offers perps by automating a looping strategy (recursive borrowing and lending) via flash loans.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

core/			
	Contango.sol	FeeManager.sol	Maestro.sol
	OrderManager.sol	OrderManager Arbitrum.sol	OrderManager Optimism.sol
	PositionNFT.sol	ReferralManager.sol	Vault.sol
libraries/			
	Arrays.sol	BitFlags.sol	DataTypes.sol
	ERC20Lib.sol	Errors.sol	MathLib.sol
	Roles.sol	Validations.sol	
libraries/extensions/			
	PositionIdExt.sol		
moneymarkets/			
	BaseMoneyMarket.sol	ImmutableBeacon Proxy.sol	UnderlyingPosition Factory.sol
	Upgradeable BeaconWithOwner.sol		
moneymarkets/aave/			
	AaveMoneyMarket.sol		
moneymarkets/exactly/			
	ExactlyMoneyMarket.sol	Exactly ReverseLookup.sol	

moneymarkets/interfaces/			
	IFlashBorrowProvider.sol	IMoneyMarket.sol	IUnderlyingPosition Factory.sol
oracle/			
	AaveOracle.sol		
utils/			
	SpotExecutor.sol		

4 Methodology

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

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 1 critical, 10 major, and a few less important issues. All identified Critical and Major issues have been fixed.

Issues	Info	Fixed
Severity: Critical	0	1
Major	0	10
Moderate	24	2
Minor	112	14

Fixed 27 out of 163 issues

6 Critical Issues

CVF-1. FIXED

- **Category** Flaw
- **Source** ERC20Lib.sol

Description The ”to” argument is ignored in this case.

Recommendation Consider explicitly requiring ”to == this”.

```
34 nativeToken.deposit{ value: amount }();
```

7 Major Issues

CVF-2. FIXED

- **Category** Flaw
- **Source** Contango.sol

Description This callback function doesn't verify that it was called by the proper flash loan provider and that the original flash loan requestor is this contract. Both statements should be properly verified.

209
```
function completeOpenFromFlashLoan(
```

350
```
function completeClose(address, /* initiator */ address repayTo,
    ↪ address asset, uint256 amount, uint256 fee, bytes calldata
    ↪ params)
```

CVF-5. FIXED

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

Description The "tmpResult" variable should be cleared after this assignment.

127
```
result = tmpResult;
```

CVF-6. FIXED

- **Category** Suboptimal
- **Source** PositionIdExt.sol

Description This shifts the value twice: first by 128 bits to the left, and then by 248 bits to the right.

Recommendation Consider refactoring to shift only once.

14
```
mm = MoneyMarket.wrap(uint8(bytes1(raw << 128)));
```

28
```
return MoneyMarket.wrap(uint8(bytes1(PositionId.unwrap(positionId)
    ↪ << 128)));
```

CVF-7. FIXED

- **Category** Suboptimal
- **Source** PositionIdExt.sol

Description This shifts the value twice: first by 136 bits to the left, and then by 224 bits to the right.

Recommendation Consider refactoring to shift only once.

```
15 expiry = (uint32(bytes4(raw << 136)));
```

```
32 return (uint32(bytes4(PositionId.unwrap(positionId) << 136)));
```

CVF-8. FIXED

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

Description This line should be surrounded with ”unchecked {}” to properly handle the ”type(int256).min” value.

```
11 return value < 0 ? uint256(-value) : 0;
```

CVF-10. FIXED

- **Category** Procedural
- **Source** ERC20Lib.sol

Description Usage of the ”transfer” function is discouraged.

Recommendation Consider using ”call” instead.

```
32 payable(to).transfer(amount);
```

CVF-13. FIXED

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

Description Conversion like: uint256(-x), where "x" is a "int256" value should be surrounded by an "unchecked" block to properly support "type(int256).min" value.

376
```
    repaymentAmount -= Math.min(uint256(-cb.cashflow), quoteBalance)
       ↪ ;
```

383
```
uint256 uCashflow = uint256(-cb.cashflow);
```

453
```
uint256 borrowAmount = cashflowInBase ? execParams.swapAmount :
   ↪ uint256(-tradeParams.cashflow);
```

533
```
uint256 uCashflow = ERC20Lib.transferOut(_cashflowToken, address(
   ↪ this), address(vault), uint256(-cb.cashflow));
```

CVF-14. FIXED

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

Description In case swap.input is positive or equals to "type(int256).min", this line will lead to a failed assertion.

Recommendation Consider explicitly reverting on positive values with meaningful error message. Also, consider properly supporting the "type(int256).min" value.

Client Comment *Can't be positive by design, but will get changed to read current balance before swap to avoid potential dust issues.*

491
```
if ((-swap.input).toUint256() > requiredBase) revert
   ↪ InsufficientBaseOnOpen(requiredBase, -swap.input);
```

CVF-21. FIXED

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

Description There is no range check for the argument.

Recommendation Consider adding an appropriate check.

```
85 function setGasMultiplier(uint64 _gasMultiplier) external override
   ↪ onlyRole(DEFAULT_ADMIN_ROLE) {
```

CVF-22. FIXED

- **Category** Suboptimal
- **Source** OrderManager.sol

Description Using the ”abs” function here is suboptimal, as ”trade_.cashflow” is already known to be negative.

Recommendation Consider just negating the value.

Client Comment *abs() is used to take advantage of the unchecked and already audited code of OZ lib as doing the unchecked here would make the code less readable (same on other related CVFs).*

```
190 if (trade_.cashflow < 0) _withdraw(cashflowToken, order.owner,
    ↪ trade_.cashflow.abs() - keeperReward, order.owner);
```

8 Moderate Issues

CVF-3. INFO

- **Category** Suboptimal
- **Source** SpotExecutor.sol

Description The "unit" argument is used only when calculating the "price" value, which could actually be calculated by the caller using "input", "output", and "unit". Consider removing the "unit" argument and the "price" returned value.

```
15 function executeSwap(IERC20 tokenToSell, IERC20 tokenToBuy, Currency
   ↪  inputCcy, uint256 unit, ExecutionParams memory execParams)
```

CVF-4. INFO

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

Description Consider implementing the ERC-3156 flash loan API instead of a custom API.

Client Comment *These are not the same thing. ERC-3156 is just to handle flash loan whereas this impl is to handle flash borrows, e.g. borrow first and provide collateral later in the same tx as supported by aave v3.*

```
7 interface IFlashBorrowProvider is IERC165 {
```

CVF-9. INFO

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

Description These checks try to replicate checks usually performed by an NFT smart contract. This is a bad practice, as different NFT smart contracts may implement different checks. Consider removing these checks and leaving them to particular NFT smart contracts.

```
8 if (!positionNFT.isApprovedForAll(onBehalfOf, msg.sender)) revert
   ↪ Unauthorised(msg.sender);
```

```
13 if (positionOwner != msg.sender && !positionNFT.isApprovedForAll(
   ↪ positionOwner, msg.sender)) revert Unauthorised(msg.sender);
```

CVF-12. INFO

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

Description This should round up, as inputs are usually rounded this way.

274
```
uint256 maxQuoteInput = expectedBaseOutput.mulDiv(tradeParams.
    ↪ limitPrice, _instrument.baseUnit);
```

CVF-16. INFO

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

Description This formula doesn't allow signing several identical permits.

Recommendation Consider hashing in a counter.

Client Comment *The permit passed is immediately consumed by executing the deposit, so we don't think there's a need for multiple identical one.*

75
```
nonce: uint256(keccak256(abi.encode(sender, token, permit.amount,
    ↪ permit.deadline))),
```

CVF-17. FIXED

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

Description This should be surrounded with "unchecked{}" to support "type(int256).min".

147
```
amount = uint256(-trade_.cashflow);
```

CVF-18. INFO

- **Category** Suboptimal
- **Source** OrderManagerOptimism.sol

Description A token transfer may consume much more than 30k gas.

Recommendation Consider either using per-token estimations, or limiting the gas amount to be used by token transfers.

Client Comment *The idea here is to guesstimate the 2 transfer that will occur after this gas cost is calculated, so we're happy with an approx average estimate, since the real incentive for the bots should come out of the multiplier on the whole gas estimation, so we can always adjust there based on observation.*

```
17 // 21000 min tx gas (starting gasStart value) + gas used so far + 60
      ↪ k for the 2 ERC20 transfers
```

CVF-19. INFO

- **Category** Suboptimal
- **Source** OrderManagerArbitrum.sol

Description A token transfer may consume much more than 30k gas.

Recommendation Consider either using per-token estimations, or limiting the gas amount to be used by token transfers.

Client Comment *The idea here is to guesstimate the 2 transfer that will occur after this gas cost is calculated, so we're happy with an approx average estimate, since the real incentive for the bots should come out of the multiplier on the whole gas estimation, so we can always adjust there based on observation.*

```
16 // 21000 min tx gas (starting gasStart value) + gas used so far + 60
      ↪ k for the 2 ERC20 transfers
```

CVF-20. INFO

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

Description In ERC-20, the decimals property is used by UI to render token amounts in a human-readable way. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts as integers.

```
70 nativeTokenUnit = 10 ** _nativeToken.decimals();
```

CVF-23. INFO

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

Description Overflow is possible when converting types.

Recommendation Consider using safe conversion.

```
22 input = ERC20Lib.myBalanceI(tokenToSell) - int256(execParams.
   ↪ swapAmount);
   output = int256(ERC20Lib.transferBalance(tokenToBuy, msg.sender));
```

CVF-24. INFO

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

Description Using different price directions depending on what what side the base currency is at, makes it harder to use the returned price.

Recommendation Consider always using the same direction.

Client Comment *This is about having the price always calculated in quote terms from an instrument perspective, so, in fact, making it usable in any situation.*

```
25 price = inputCcy == Currency.Base ? output.abs().mulDiv(unit, input.
   ↪ abs()) : input.abs().mulDiv(unit, output.abs());
```

CVF-25. INFO

- **Category** Suboptimal
- **Source** IFlashBorrowProvider.sol

Description For a callback with such signature it is non-trivial to verify that the call is eligible.

Recommendation Consider passing the original caller address as a callback argument.

Client Comment *This reposibility is with the FlashBorrowProvider implementation, not on the callback itself. e.g. AaveMoneyMarket.executeOperation(). On the callback we just check if it's the same contract we asked the flashBorrow to be executed.*

```
26 function(IERC20, uint256, bytes memory) external returns (bytes
     ↪ memory) callback
```

CVF-26. INFO

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

Description This function adds new markets from the auditor and updates the existing ones, but doesn't remove markets removed from the auditor.

Recommendation Consider either fixing this, or clearly explaining why this is fine.

```
23 function _update(IAuditor _auditor) private {
```

CVF-27. INFO

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

Description The ”ISOLATION_MODE” flag is used per position, while the ”UserUseReserveAsCollateral” flag is set per asset, thus this flag would affect even those positions that don’t have ”ISOLATION_MODE” flag.

Recommendation Consider either fixing this or clearly explaining why this is fine.

Client Comment *The ISOLATION_MODE flag is an asset config on the underlying money market (Aave in this case) and is sent when the position is created and packed witht he positionId for efficiency. What we do here when setting the reserve as collateral flag on Aave is also at the position level, which will be the context loaded (position/account address) but executing the AaveMoneyMarket via its beacon proxy implementation. So when we set the config on Aave, we’re doing so as the position account itself.*

```
60 if (isBitSet(positionId.getFlags(), ISOLATION_MODE)) POOL.
      ↪ setUserUseReserveAsCollateral(address(asset), true);
```

CVF-28. INFO

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

Description This function doesn’t follow ERC-165 recommendations.

Recommendation Consider properly implementing the standard.

Client Comment *Unclear which recommendation we’re not following, as we’re using the implementation described as ”pure” here https://eips.ethereum.org/EIPS/eip-165#implementation*

```
148 function supportsInterface(bytes4 interfaceId) public pure override(
      ↪ BaseMoneyMarket, IERC165) returns (bool) {
```

CVF-29. INFO

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

Description This function doesn't follow ERC-165 recommendations.

Recommendation Consider properly implementing the standard.

Client Comment *Unclear which recommendation we're not following, as we're using the implementation described as "pure" here https://eips.ethereum.org/EIPS/eip-165#implementation*

```
45 function supportsInterface(bytes4 interfaceId) external pure virtual
   ↪ override returns (bool) {
```

CVF-30. INFO

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

Description This allows overwriting an existing money market. Consider either forbidding this or clearly explaining why this is fine.

```
21 moneyMarkets[imm.moneyMarketId()] = imm;
```

CVF-31. INFO

- **Category** Unclear behavior
- **Source** ERC20Lib.sol

Description This shortcut violates several requirements: 1. It doesn't check nor reduce allowance 2. It doesn't call transfer hooks 3. It doesn't check the balance is sufficient 4. It doesn't emit events Also, it optimizes a rare case by making

Recommendation the most common case more expensive. Consider removing this shortcut.

Client Comment *This is by design, when it's a transfer to self or no amount at all we just skip it. It's easier to do it then put these validations everywhere.*

```
19 if (payer == to || amount == 0) return amount;
```

```
27 if (payer == to || amount == 0) return amount;
```

CVF-32. FIXED

- **Category** Unclear behavior
- **Source** ERC20Lib.sol

Description This allows using ether that was at the contract's balance before the call.

Recommendation Consider explicitly requiring "amount == msg.value".

Client Comment *Superseded by CVF-1.*

```
34 nativeToken.deposit{ value: amount }();
```

CVF-33. INFO

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

Description In case the cash flow currency is Base, this should be tradeParams.quantity - tradeParams.cashFlow.

Client Comment *When opening with base cashflow, the flashLoanAmount should be exactly the swapAmount, because for cashflow positive, we'll just grab the base and lend and for negative cashflow, we'll borrow, swap and transfer, in this case the swapAmount should already take that into account.*

```
167 : execParams.swapAmount;
```

CVF-34. INFO

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

Description This check isn't enough to validate a flash loan callback, as a malicious actor may request a flash loan with fake "params" and pass this contract as a callback, so in general "params" cannot be trusted, unless the callback verifies that the flash loan was originally requested by this contract. Usually, in order to make such verification simpler, flash loan implementations either use "msg.sender" as the callback address, or pass the original "msg.sender" as a callback argument. However, here neither option is implemented. Consider performing proper callback verification.

Client Comment *When a flash borrow is used, it is initiated via a money market, which will then have the checks you mention, e.g. AaveMoneyMarket, using the positionId account as context (loaded via a becon proxy). The check at this level (Contango.sol) is limiting the callback call from coming from the positionId account.*

```
205 if (address(_moneyMarket(cb.positionId)) != msg.sender) revert
        ↪ NotFlashBorrowProvider(msg.sender);
```

CVF-35. INFO

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

Description Underflow is possible when converting type.

Recommendation Consider using safe conversion.

```
607     cost = uint256(_trade.cashflow).mulDiv((-_trade.swap.input).
            ↪ toUint256(), _trade.swap.output.toUint256()) + borrowed;
```

```
612 cost = uint256(_trade.cashflow) + borrowed - repaid;
```

CVF-36. INFO

- **Category** Procedural
- **Source** Contango.sol

Description As base=0 has a special meaning, there should be an explicit check to ensure ”base” isn’t zero.

Client Comment *No special meaning, this is just checking that there is nothing registered under the instruments[symbol] mapping.*

```
702 if (instruments[symbol].base != IERC20(address(0))) revert
        ↪ InstrumentAlreadyExists(symbol);
```

```
705 instruments[symbol].base = base;
```

CVF-37. INFO

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

Description There is no check to ensure that the sum of referrer fee, trader fee, and the protocol fee equals the ”fee” argument.

Recommendation Consider adding such a check to protect from inconsistencies in the referral manager.

Client Comment *The ReferralManager does have a check on setRewardsAndRebates() line 21. On the FeeManager itself there’s no checks since it will just fail without funds if trying to over transfer.*

```
55     if (feeDistribution.referrer > 0) vault.depositTo(base,
           ↪ feeDistribution.referrerAddress, feeDistribution.referrer)
           ↪ ;
```

```
57     if (feeDistribution.trader > 0) vault.depositTo(base, trader,
           ↪ feeDistribution.trader);
```

```
61 vault.depositTo(base, treasury, protocolFee);
```

CVF-38. INFO

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

Description Overflow is possible here.

Recommendation Consider using safe conversion.

Client Comment *int256(params.cashflow) can't overflow because params.cashflow is an int128. The following .toUint256() reverts if value is negative.*

```
230 _deposit(cashflowToken, int256(params.cashflow).toUint256());
```

CVF-39. INFO

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

Description As owner=0 has a special meaning, there should be an explicit "require" statement to ensure "owner" is not zero.

Client Comment *No special meaning, this is just checking that there is nothing registered under the orders[orderId] mapping.*

```
124 function _place(OrderParams calldata params, address owner) internal
    ↪ returns (OrderId orderId) {
```

9 Minor Issues

CVF-40. INFO

- **Category** Procedural
- **Source** SpotExecutor.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-41. INFO

- **Category** Procedural
- **Source** SpotExecutor.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-42. FIXED

- **Category** Procedural
- **Source** SpotExecutor.sol

Description We didn't review this file.

```
8 import "../interfaces/IContango.sol";
```

CVF-43. INFO

- **Category** Procedural
- **Source** AaveOracle.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-44. INFO

- **Category** Procedural
- **Source** AaveOracle.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as ”^0.8.0”.

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-45. FIXED

- **Category** Procedural
- **Source** AaveOracle.sol

Description We didn’t review this file.

```
10 import "../interfaces/IOracle.sol";
```

CVF-46. INFO

- **Category** Procedural
- **Source** IFlashBorrowProvider.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: MIT
```

CVF-47. INFO

- **Category** Procedural
- **Source** IFlashBorrowProvider.sol

Recommendation Consider specifying as ”^0.8.0” unless there is something special about this particular version.

Client Comment *0.8.4 needed to support custom errors, so we applied on all interfaces.*

```
2 pragma solidity ^0.8.4;
```

CVF-48. INFO

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

Description Supporting ERC-165 doesn’t seem mandatory for implementations of this interface.

Recommendation Consider not inheriting this interface from ”IERC165” and leave this for implementations.

```
7 interface IFlashBorrowProvider is IERC165 {
```

CVF-49. INFO

- **Category** Procedural
- **Source** IMoneyMarket.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: MIT
```

CVF-50. INFO

- **Category** Procedural
- **Source** IMoneyMarket.sol

Recommendation Consider specifying as "^0.8.0" unless there is something special about this particular version.

Client Comment *0.8.4 needed to support custom errors, so we applied on all interfaces.*

```
2 pragma solidity ^0.8.4;
```

CVF-51. INFO

- **Category** Documentation
- **Source** IMoneyMarket.sol

Description The semantics of this function is unclear.

Recommendation Consider documenting.

```
11 function NEEDS_ACCOUNT() external view returns (bool);
```

CVF-52. INFO

- **Category** Procedural
- **Source** IMoneyMarket.sol

Description This functions shouldn't be in this interface, as it is implementation-specific.

Client Comment *While each impl will initialise differently, they will all need some sort of intialising at the generic level, so this function has to be here.*

```
15 function initialise(PositionId positionId, IERC20 collateralAsset,
   ↪ IERC20 debtAsset) external;
```

CVF-53. INFO

- **Category** Procedural
- **Source** IUnderlyingPositionFactory.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: MIT
```

CVF-54. INFO

- **Category** Procedural
- **Source** IUnderlyingPositionFactory.sol

Recommendation Consider specifying as "^0.8.0" unless there is something special about this particular version.

Client Comment *0.8.4 needed to support custom errors, so we applied on all interfaces.*

```
2 pragma solidity ^0.8.4;
```

CVF-55. INFO

- **Category** Procedural
- **Source** IUnderlyingPositionFactory.sol

Description This interface should be moved into a separate file named "IUnderlyingPositionFactoryEvents.sol".

Client Comment *They are in the same file by choice, same as IContango, etc*

```
6 interface IUnderlyingPositionFactoryEvents {
```

CVF-56. INFO

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

Description Events are usually named via nouns, such as "UnderlyingPosition" or "MoneyMarket".

Client Comment *We believe events should be actions that happened in the past, hence this naming.*

```
8 event UnderlyingPositionCreated(address indexed account, PositionId
    ↪ indexed positionId);
  event MoneyMarketRegistered(MoneyMarket indexed mm, IMoneyMarket
    ↪ indexed moneyMarket);
```

CVF-57. INFO

- **Category** Procedural
- **Source** ExactlyMoneyMarket.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-58. INFO

- **Category** Procedural
- **Source** ExactlyMoneyMarket.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-59. FIXED

- **Category** Procedural
- **Source** ExactlyMoneyMarket.sol

Description We didn't review this file.

```
8 import "./dependencies/IExactlyRewardsController.sol";
```

CVF-60. INFO

- **Category** Suboptimal
- **Source** ExactlyMoneyMarket.sol

Description It would be more efficient to swap the left and right parts of this condition.

```
67 if (market.previewRepay(amount) == 0 || debt == 0) return 0;
```

CVF-61. INFO

- **Category** Procedural
- **Source** ExactlyReverseLookup.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-62. INFO

- **Category** Procedural
- **Source** ExactlyReverseLookup.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as ”^0.8.0”.

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-63. FIXED

- **Category** Procedural
- **Source** ExactlyReverseLookup.sol

Description We didn’t review this file.

```
4 import "./dependencies/IAuditor.sol";
```

CVF-64. INFO

- **Category** Procedural
- **Source** ExactlyReverseLookup.sol

Description This function should emit some event.

```
19 function update() external {
```

CVF-65. INFO

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

Description Despite the name, this function returns a single money market.

Recommendation Consider renaming.

```
33 function markets(IERC20 asset) external view returns (IMarket market
   ↪ ) {
```

CVF-66. INFO

- **Category** Procedural
- **Source** AaveMoneyMarket.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-67. INFO

- **Category** Procedural
- **Source** AaveMoneyMarket.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-68. FIXED

- **Category** Procedural
- **Source** AaveMoneyMarket.sol

Description We didn't review this file.

```
8 import "./dependencies/IAaveRewardsController.sol";
```

CVF-69. INFO

- **Category** Procedural
- **Source** AaveMoneyMarket.sol

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

Recommendation Consider either moving the constant definitions into the contract or moving them into a separate file.

Client Comment *We think this is fine since it's in the file that is related to said config while also being easy to import and use elsewhere.*

```
16 uint256 constant E_MODE = 0;
   uint256 constant ISOLATION_MODE = 1;
```

CVF-70. INFO

- **Category** Procedural
- **Source** BaseMoneyMarket.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-71. INFO

- **Category** Procedural
- **Source** BaseMoneyMarket.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as ”^0.8.0”.

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-72. FIXED

- **Category** Procedural
- **Source** BaseMoneyMarket.sol

Description We didn’t review this file.

```
4 import "../interfaces/IContango.sol";
```

CVF-73. INFO

- **Category** Procedural
- **Source** ImmutableBeaconProxy.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: MIT
```

CVF-74. INFO

- **Category** Procedural
- **Source** ImmutableBeaconProxy.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as ”^0.8.0”.

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-75. INFO

- **Category** Procedural
- **Source** UnderlyingPositionFactory.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-76. INFO

- **Category** Procedural
- **Source** UnderlyingPositionFactory.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-77. INFO

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

Description Despite the name, this function not always creates a new position, but in some cases could return an existing position.

Recommendation Consider renaming to reflect this fact.

Client Comment *That is not correct, this always load a money market on the positionId context and if the money market needs a new contract to be created to isolate the positions, it goes and create it. Some money markets are able to individualise positions with just an id and no need for a different contract/address per position to maitain isolation.*

```
25 function createUnderlyingPosition(PositionId positionId) external
     ↪ override onlyRole(CONTANGO_ROLE) returns (IMoneyMarket imm) {
```

CVF-78. INFO

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

Description Despite the name, this function not always creates a new money market, but in some cases could return an existing money market.

Recommendation Consider renaming to reflect this fact.

Client Comment *That is not correct, this always load a money market on the positionId context and if the money market needs a new contract to be created to isolate the positions, it goes and load it with the position address by calculating it with the positionId. Some money markets are able to individualise positions with just an id and no need for a different contract/address per position to maitain isolation.*

```
35 function moneyMarket(PositionId positionId) external view override
     ↪ returns (IMoneyMarket imm) {
```

CVF-79. INFO

- **Category** Procedural
- **Source** UpgradeableBeaconWithOwner.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: MIT
```

CVF-80. INFO

- **Category** Procedural
- **Source** UpgradeableBeaconWithOwner.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-81. INFO

- **Category** Procedural
- **Source** PositionIdExt.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-82. INFO

- **Category** Procedural
- **Source** PositionIdExt.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-83. INFO

- **Category** Procedural
- **Source** Errors.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: MIT
```

CVF-84. INFO

- **Category** Procedural
- **Source** Errors.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-85. INFO

- **Category** Procedural
- **Source** MathLib.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos.*

```
1 //SPDX-License-Identifier: MIT
```

CVF-86. INFO

- **Category** Procedural
- **Source** MathLib.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-87. INFO

- **Category** Procedural
- **Source** Roles.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: MIT
```

CVF-88. INFO

- **Category** Procedural
- **Source** Roles.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-89. INFO

- **Category** Procedural
- **Source** Validations.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-90. INFO

- **Category** Procedural
- **Source** Validations.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-91. INFO

- **Category** Procedural
- **Source** Arrays.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: MIT
```

CVF-92. INFO

- **Category** Procedural
- **Source** Arrays.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-93. INFO

- **Category** Procedural
- **Source** BitFlags.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: MIT
```

CVF-94. INFO

- **Category** Procedural
- **Source** BitFlags.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-95. INFO

- **Category** Suboptimal
- **Source** BitFlags.sol

Description Here two shifts are performed: one explicit, and anther implicit, when converting uint8 to bytes1.

Recommendation Consider refactoring to use only one shift: bytes1 mask = bytes1(0×01) « bit;

```
8 bytes1 mask = bytes1(uint8(1 << bit));
```

CVF-96. INFO

- **Category** Procedural
- **Source** DataTypes.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-97. INFO

- **Category** Procedural
- **Source** DataTypes.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-98. INFO

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

Description The names of this type looks like a named of a smart contract.

Recommendation Consider renaming.

```
26 type MoneyMarket is uint8;
```

CVF-99. INFO

- **Category** Procedural
- **Source** ERC20Lib.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: MIT
```

CVF-100. INFO

- **Category** Procedural
- **Source** ERC20Lib.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-101. FIXED

- **Category** Procedural
- **Source** ERC20Lib.sol

Description We didn't review this file.

```
6 import "../dependencies/IWETH9.sol";
```

CVF-102. INFO

- **Category** Documentation
- **Source** ERC20Lib.sol

Description The semantics of the returned value is unclear.

Recommendation Consider giving a descriptive name to the returned value and/or describing in a documentation comment.

```
16 function transferOut(IERC20 token, address payer, address to,
   ↪ uint256 amount) internal returns (uint256) {
```

```
24 function transferOut(IERC20 token, address payer, address to,
   ↪ uint256 amount, IWETH9 nativeToken) internal returns (uint256)
   ↪  {
```

```
42 function _transferOut(IERC20 token, address payer, address to,
   ↪ uint256 amount) internal returns (uint256) {
```

CVF-103. INFO

- **Category** Suboptimal
- **Source** ERC20Lib.sol

Description This delegation is suboptimal.

Recommendation Consider directly calling ”token.safeTransfer”.

Client Comment *Superseded by refactor of CVF-1.*

```
48 return transferBalance(token, to, IWETH9(address(0)));
```

CVF-104. INFO

- **Category** Suboptimal
- **Source** ERC20Lib.sol

Description This delegation is suboptimal.

Recommendation Consider directly calling ”token.safeTransfer”.

Client Comment *Superseded by refactor of CVF-1.*

```
53 return balance > 0 ? transferOut(token, address(this), to, balance,
   ↪ nativeToken) : 0;
```

CVF-105. INFO

- **Category** Suboptimal
- **Source** ERC20Lib.sol

Description This function performs up to 4 external calls, which seems to much.

Recommendation Consider simplifying.

Client Comment *Not sure what can be simplified since we want a behaviour similar to the old OZ safeApprove()*

```
64 function approveIfNecessary(IERC20 asset, address spender) internal
   ↪ {
```

CVF-106. INFO

- **Category** Procedural
- **Source** PositionNFT.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-107. INFO

- **Category** Procedural
- **Source** PositionNFT.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-108. INFO

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

Description The key type for this mapping should be "IContango".

Client Comment *It can't be, because this mapping is supposed to hold any contract that we want to have contango permissions suce as Contango itself, Maestro and OrderManager.*

```
16 mapping(address contractAddr => bool enabled) public
     ↪ contangoContracts;
```

CVF-109. INFO

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

Description The type of the ”timelock” argument should be more specific.

```
18 constructor(address timelock) ERC721("Contango Position", "CTGP") {
```

CVF-110. INFO

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

Description The type of the ”contractAddr” should be ”IContango”.

Client Comment *It can't be, because this mapping is supposed to hold any contract that we want to have contango permissions suce as Contango itself, Maestro and OrderManager.*

```
46 function setContangoContract(address contractAddr, bool enabled)
     ↪ external onlyRole(DEFAULT_ADMIN_ROLE) {
```

CVF-111. INFO

- **Category** Procedural
- **Source** PositionNFT.sol

Description This function should emit some event.

```
46 function setContangoContract(address contractAddr, bool enabled)
     ↪ external onlyRole(DEFAULT_ADMIN_ROLE) {
```

CVF-112. INFO

- **Category** Procedural
- **Source** Contango.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-113. INFO

- **Category** Procedural
- **Source** Contango.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as ”^0.8.0”.

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-114. FIXED

- **Category** Procedural
- **Source** Contango.sol

Description We didn’t review this file.

```
11 import "../interfaces/IContango.sol";
```

CVF-115. INFO

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

Description The argument type should be more specific.

Client Comment *Superseded by CVF-109.*

```
82 function initialize(address timelock) public initializer {
```

CVF-116. INFO

- **Category** Suboptimal
- **Source** Contango.sol

Description This check should be performed earlier.

```
100 if (address(instrument_.base) == address(0)) revert
    ↪ InvalidInstrument(symbol);
```

CVF-117. INFO

- **Category** Suboptimal
- **Source** Contango.sol

Description This assignment should be performed only when tradeParams.quantity > 0.

```
129 address owner = onBehalfOf;
```

CVF-118. INFO

- **Category** Documentation
- **Source** Contango.sol

Description It is unclear what these scenarios really are.

Client Comment *All of these scenarios are present in a public spreadsheet that is linked in the tests AbstractPositionLifeCycle.ft.t.sol in line 6 https://docs.google.com/spreadsheets/d/1uLRNJOn3uy2PR5H2QJ-X8unBRVCu1Ra51ojMjylPH90/edit#gid=0*
Will add to the README.

```
421 // Scenarios 19, 30: _guaranteedBaseCashflowRemoval
```

CVF-119. INFO

- **Category** Suboptimal
- **Source** Contango.sol

Description This argument is redundant. Just allow the caller to conditionally call the "_executeSwap" function.

```
472 bool shouldRun, // externalised condition
```

CVF-120. INFO

- **Category** Procedural
- **Source** Contango.sol

Description This approach for verifying callback is inefficient and unreliable, as it is prone to reentrancy attacks.

Recommendation Consider modifying the flash loan API to make verification easier.

```
507 callbackHash = keccak256(data);
```

```
513 if (keccak256(data) != callbackHash) revert UnexpectedCallback();
```

CVF-121. INFO

- **Category** Suboptimal
- **Source** Contango.sol

Description The third argument is redundant.

Recommendation Consider removing it.

Client Comment *Superseded by refactor of CVF-1.*

```
522 deposited = ERC20Lib.transferBalance(token, address(vault), IWETH9(
    ↪ address(0)));
```

CVF-122. INFO

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

Description The ”shouldRun” argument is redundant. Just allow the caller to conditionally call this function.

```
530 function _handleNegativeCashflow(bool shouldRun, FlashLoanCallback
    ↪ memory cb) private {
```

CVF-123. INFO

- **Category** Procedural
- **Source** Contango.sol

Description These functions looks like pure encoder/decoder, while actually they do modify the contract’s state.

Recommendation Consider either making then pure or renaming.

```
688 function encodeTrade(Trade memory _trade) internal returns (bytes
    ↪ memory data) {
```

```
693 function decodeTrade(bytes memory data) internal returns (Trade
    ↪ memory) {
```

CVF-124. INFO

- **Category** Procedural
- **Source** Contango.sol

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

Recommendation Consider either moving the function definitions into the contract or moving them into a separate file.

Client Comment *We think this is fine since it's used only on this file and we can use as a lib without having to declare one.*

```
751 function cashflowToken(Contango.FlashLoanCallback memory cb) pure
      ↪ returns (IERC20) {
```

```
755 function encodeFlashLoanCallback(Contango.FlashLoanCallback memory
      ↪ cb) pure returns (bytes memory) {
```

```
759 function decodeFlashLoanCallback(bytes memory data) pure returns (
      ↪ Contango.FlashLoanCallback memory) {
```

CVF-125. INFO

- **Category** Procedural
- **Source** FeeManager.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-126. INFO

- **Category** Procedural
- **Source** FeeManager.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-127. FIXED

- **Category** Procedural
- **Source** FeeManager.sol

Description We didn't review these files.

```
7  import "../interfaces/IContango.sol";
   import "../interfaces/IFeeManager.sol";
   import "../interfaces/IReferralManager.sol";
10 import "../interfaces/IVault.sol";
```

CVF-128. INFO

- **Category** Documentation
- **Source** FeeManager.sol

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

Client Comment *We think it's pretty clear and self explanatory with the modifier.*

```
75 function _authorizeUpgrade(address newImplementation) internal
      ↪ override onlyRole(DEFAULT_ADMIN_ROLE) { }
```

CVF-129. INFO

- **Category** Procedural
- **Source** Maestro.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-130. INFO

- **Category** Procedural
- **Source** Maestro.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as ”^0.8.0”.

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-131. FIXED

- **Category** Procedural
- **Source** Maestro.sol

Description We didn't review these files.

```
9  import { IPermit2 } from "../dependencies/Uniswap.sol";
```

```
11 import "../interfaces/IMaestro.sol";
   import "../interfaces/IContango.sol";
   import "../interfaces/IOrderManager.sol";
   import "../interfaces/IVault.sol";
   import "../libraries/Errors.sol";
   import "../libraries/Validations.sol";
```

CVF-132. INFO

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

Description The type of this variable should be more specific.

Client Comment *Superseded by CVF-109.*

```
27 address public immutable timelock;
```

CVF-133. INFO

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

Description The type of the "_timelock" argument should be more specific.

Client Comment *Superseded by CVF-109.*

```
35 constructor(address _timelock, IContango _contango, IOrderManager
     ↪ _orderManager, IVault _vault, IPermit2 _permit2) {
```

CVF-134. INFO

- **Category** Suboptimal
- **Source** Maestro.sol

Description This check seems redundant, as it tries to predict whether the subsequent token transfer will succeed.

Recommendation Consider leaving this to the token contract and removing this check.

Client Comment *We want to revert early if that's the case.*

```
278 if (cashflow > 0 && int256(permit.amount) < cashflow) revert
    ↪ InsufficientPermitAmount(uint256(cashflow), permit.amount);
```

CVF-135. INFO

- **Category** Procedural
- **Source** OrderManagerOptimism.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-136. INFO

- **Category** Procedural
- **Source** OrderManagerOptimism.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-137. INFO

- **Category** Documentation
- **Source** OrderManagerOptimism.sol

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

Client Comment *We think it's pretty clear and self explanatory that the constructor is only there to satisfy the contract it extends from.*

```
11 constructor(IContango _contango, IWETH9 _nativeToken) OrderManager(
       ↪ _contango, _nativeToken) { }
```

CVF-138. INFO

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

Description The value "60_000" should be a named constant.

Client Comment *The comment above this line explains what it is.*

```
18 uint256 l2GasSpent = gasStart - gasleft() + 60_000;
```

CVF-139. INFO

- **Category** Procedural
- **Source** OrderManagerArbitrum.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-140. INFO

- **Category** Procedural
- **Source** OrderManagerArbitrum.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-141. FIXED

- **Category** Procedural
- **Source** OrderManagerArbitrum.sol

Description We didn't review this file.

```
4 import "../dependencies/ArbGasInfo.sol";
```

CVF-142. INFO

- **Category** Documentation
- **Source** OrderManagerArbitrum.sol

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

Client Comment *We think it's pretty clear and self explanatory that the constructor is only there to satisfy the contract it extends from.*

```
11 constructor(IContango _contango, IWETH9 _nativeToken) OrderManager(
   ↪ _contango, _nativeToken) { }
```

CVF-143. INFO

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

Description The value ”60_000” should be a named constant.

Client Comment *The comment above this line explains what it is.*

```
17 uint256 gasSpent = gasStart - gasleft() + 60_000;
```

CVF-144. INFO

- **Category** Procedural
- **Source** ReferralManager.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-145. INFO

- **Category** Procedural
- **Source** ReferralManager.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as ”^0.8.0”.

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-146. FIXED

- **Category** Procedural
- **Source** ReferralManager.sol

Description We didn't review this file.

```
6 import "../interfaces/IReferralManager.sol";
```

CVF-147. INFO

- **Category** Documentation
- **Source** ReferralManager.sol

Description The number format of these variables is unclear.

Recommendation Consider documenting.

Client Comment *It is explained in the IReferralManager.setRewardsAndRebates() natspec but we'll also add in this contract*

```
11 uint256 public referrerRewardPercentage;
   uint256 public traderRebatePercentage;
```

CVF-148. INFO

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

Description The argument type should be more specific.

Client Comment *Superseded by CVF-109.*

```
17 constructor(address timelock) {
```

CVF-149. INFO

- **Category** Suboptimal
- **Source** ReferralManager.sol

Description This assignment should be performed only if the referrer address is zero.

Client Comment *That's not correct, this sets the full fee amount to the protocol, and then it gets overridden only if there are referrers.*

```
58 feeDistribution.protocol = amount;
```

CVF-150. INFO

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

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-151. INFO

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

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-152. FIXED

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

Description We didn't review these files.

```
9  import "../interfaces/IVault.sol";
10 import "../libraries/ERC20Lib.sol";
```

CVF-153. INFO

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

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

```
29 mapping(IERC20 token => bool supported) public override
      ↪ isTokenSupported;
30 mapping(address owner => mapping(IERC20 token => uint256 balance))
      ↪ private _accountBalances;
   mapping(IERC20 token => uint256 balance) private _totalBalances;
```

CVF-154. INFO

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

Description This function should emit some event.

```
46 function setTokenSupport(IERC20 token, bool isSupported) external
      ↪ onlyRole(DEFAULT_ADMIN_ROLE) {
```

CVF-155. INFO

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

Description Usually, a function that accepts tokens from the caller, either expects the tokens to already be at the contract's balance, or uses the "transferFrom" function to transfer tokens from the caller. However, this function combines both

Client Comment *This is by design.*

```
58 function _deposit(IERC20 token, address payer, address account,
   ↪ uint256 amount) private returns (uint256) {
```

CVF-156. INFO

- **Category** Documentation
- **Source** Vault.sol

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

Client Comment *We think it's pretty clear and self explanatory with the modifier.*

```
112 function _authorizeUpgrade(address newImplementation) internal
   ↪ override onlyRole(DEFAULT_ADMIN_ROLE) { }
```

CVF-157. INFO

- **Category** Procedural
- **Source** OrderManager.sol

Description Different files have different licenses.

Recommendation Consider making them consistent.

Client Comment *Rationale is: - our own specific implementation code is BUSL - our own interfaces and generic code are MIT - dependencies have whatever license they already had on their own repos*

```
1 //SPDX-License-Identifier: BUSL-1.1
```

CVF-158. INFO

- **Category** Procedural
- **Source** OrderManager.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0".

Client Comment *Set as 0.8.20 to ensure the compiler versions taken on tests and deployment are always the same. The idea being that interfaces have ^0.8.4 therefore not blocking any downstream dependency on upgrading and if any of these need upgrade we would need to redeploy anyway.*

```
2 pragma solidity 0.8.20;
```

CVF-159. FIXED

- **Category** Procedural
- **Source** OrderManager.sol

Description We didn't review these files.

```
14 import "../interfaces/IOrderManager.sol";
   import "../interfaces/IContango.sol";
   import "../interfaces/IVault.sol";
   import "../interfaces/IOracle.sol";
```

CVF-160. INFO

- **Category** Documentation
- **Source** OrderManager.sol

Description The number format of these fields is unclear.

Recommendation Consider documenting.

```
32 uint128 limitPrice;
   uint128 tolerance;
```

CVF-161. INFO

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

Description The type of the ”timelock” argument should be more specific.

Client Comment *Superseded by CVF-109.*

```
73 function initialize(address timelock, uint64 _gasMultiplier, uint64
   ↪ _gasTip, IOracle _oracle) public initializer {
```

CVF-162. INFO

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

Description The initial ”gasStart” value should be a named constant.

```
77 gasStart = 21_000;
```

CVF-163. INFO

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

Description The value ”1e4” should be a named constant.

Client Comment *Superseded by CVF-160.*

```
209 if (order.orderType == OrderType.StopLoss) order.limitPrice =
    ↪ uint128(order.limitPrice.mulDiv(1e4 - order.tolerance, 1e4));
```

CVF-164. INFO

- **Category** Documentation
- **Source** OrderManager.sol

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

Client Comment *We think it's pretty clear and self explanatory with the modifier.*

```
248 function _authorizeUpgrade(address newImplementation) internal
    ↪ override onlyRole(DEFAULT_ADMIN_ROLE) { }
```

CVF-165. INFO

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

Description The hardcoded values here should be named constants.

```
270 uint256 gasSpent = gasStart - gasleft() + 16 * msg.data.length + 60
    ↪ _000;
```

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