Report

v. 2.0

Customer

Contango

Smart Contract Audit

Core V2. Part II

1th December 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	01.12.23	A. Zveryanskaya	Initial Draft
0.2	01.12.23	A. Zveryanskaya	Minor revision
1.0	01.12.23	A. Zveryanskaya	Release
1.1	01.12.23	A. Zveryanskaya	Original code link updated, project scope description added
2.0	01.12.23	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 is a unique decentralized market offering cPerps (Contango perps). Contango builds perps by automating a looping strategy, also known as recursive borrowing and lending.

3 Project scope

We were asked to review the new features of Core V2:

- Original Code
- Code with Fixes

Files:

core/		
	Maestro.sol	
libraries/extensions/		
	PositionIdExt.sol	
libraries/		
	DataTypes.sol	ERC20Lib.sol
moneymarkets/aave/		
	AaveMoneyMarket.sol	SparkMoneyMarket.sol
moneymarkets/compound/		
	CompoundMoneyMarket.sol	SonneMoneyMarket.sol
moneymarkets/morpho/		
	MorphoBlueMoneyMarket.sol	MorphoBlueReverseLookup.sol
utils/		
	SimpleSpotExecutor.sol	

The first part of the audit is available at the following link.

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 3 moderate, and a few less important issues.

	Info	Fixed
Moderate	0	3
Minor	5	8

Fixed 11 out of 16 issues

6 Moderate Issues

CVF-1. FIXED

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

Description This function doesn't allow specifying the minimum output amount.

Recommendation Consider adding such an ability.

```
17 +function executeSwap(IERC20 tokenToSell, IERC20 tokenToBuy, Swap
     ↪ calldata swap, address to) external returns (uint256 output) {
```

CVF-2. FIXED

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

Description Overflow is possible when converting types.

Recommendation Consider using safe type conversion.

```
41 +payload = Payload.wrap(bytes5(uint40(nextPayload++)));
```

CVF-3. FIXED

- **Category** Suboptimal
- **Source** SonneMoneyMarket.sol

Description Linear search doesn't scale.

Recommendation Consider implementing an ability to the caller to provide a hint for the index where to start searching or, even CToken to return.

```
85 +address[] memory allMarkets = comptroller.getAllMarkets();
   +for (uint256 i = 0; i < allMarkets.length; i++) {
```

7 Minor Issues

CVF-4. FIXED

- **Category** Procedural
- **Source** SimpleSpotExecutor.sol

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

Recommendation Consider specifying as "^0.8.0". Also relevant for: MorphoBlueReverseLookup.sol, MorphoBlueMoneyMarket.sol, CompoundMoneyMarket.sol, SonneMoneyMarket.sol, SparkMoneyMarket.sol.

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-5. FIXED

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

Recommendation The type for the "tokenToSell" and "tokenToBuy" parameters should be "IERC20".

```
15 +event SwapExecuted(address indexed tokenToSell, address indexed
     ↪ tokenToBuy, uint256 amountIn, uint256 amountOut);
```

CVF-6. INFO

- **Category** Procedural
- **Source** MorphoBlueReverseLookup.sol

Recommendation This interface should be moved into a separate file named "MorphoBlueReverseLookupEvents.sol".

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

```
10 +interface MorphoBlueReverseLookupEvents {
```

CVF-7. INFO

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

Recommendation Events are usually named via nouns, such as ”Market”.

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

```
12 +event MarketSet(Payload, Id);
```

CVF-8. INFO

- **Category** Procedural
- **Source** MorphoBlueReverseLookup.sol

Recommendation This interface should be moved into a separate file named ”MorphoBlueReverseLookupErrors.sol”.

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

```
16 +interface MorphoBlueReverseLookupErrors {
```

CVF-9. FIXED

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

Recommendation The type for this variable should be ”uint40”.

```
28 +uint256 public nextPayload = 1;
```

CVF-10. FIXED

- **Category** Readability
- **Source** MorphoBlueMoneyMarket.sol

Recommendation ”before before”

```
77 +morpho.accrueInterest(marketParams); // Accrue interest before
      ↪ before loading the market state
```

CVF-11. FIXED

- **Category** Procedural
- **Source** CompoundMoneyMarket.sol

Recommendation One of these two identical lines should be removed.

```
13 +using ERC20Lib for IERC20;
```

```
15 +using ERC20Lib for IERC20;
```

CVF-12. FIXED

- **Category** Procedural
- **Source** CompoundMoneyMarket.sol

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

```
105 +receive() external payable { }
```

CVF-13. FIXED

- **Category** Procedural
- **Source** SonneMoneyMarket.sol

Recommendation One of these two identical lines should be removed.

```
13 +using ERC20Lib for IERC20;
```

```
15 +using ERC20Lib for IERC20;
```

CVF-14. FIXED

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

Recommendation The value ”1e12” should be a named constant.

```
51 +    if (asset == usdc) balance /= 1e12;
```

```
58 +balance = asset == usdc ? (super.debtBalance(positionId, dai) + 1
   ↪ e12 - 1) / 1e12 : super.debtBalance(positionId, asset);
```

```
67 +        amount *= 1e12;
```

```
83 +    actualAmount = super._borrow(positionId, dai, amount * 1e12,
   ↪ address(this)) / 1e12;
```

```
100 +    super._repay(positionId, dai, actualAmount * 1e12, address(this
   ↪ ));
```

```
109 +        shares: super._withdraw(positionId, sDAI, sDAI.
   ↪ previewWithdraw(asset == usdc ? amount * 1e12 : amount),
   ↪ address(this)),
```

```
115 +        actualAmount /= 1e12;
```

```
139 +amount = isUsdc ? amount * 1e12 : amount;
```

```
156 +amount = metaParams.isUsdc ? amounts[0] / 1e12 : amounts[0];
```

CVF-15. INFO

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

Recommendation Consider introducing a new interface that inherits both, ”IERC20” and ”IERC20Permit”, and changing the type of the ”token” argument to this interface. This would make the code more clear and less error-prone.

Client Comment *We prefer to rely on the OZ interfaces.*

```
87 +function depositWithPermit(IERC20Permit token, EIP2098Permit
   ↪ calldata permit) public override returns (uint256) {
```

CVF-16. INFO

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

Recommendation Consider introducing a new interface that inherits both, ”IERC20” and ”IERC20Permit”, and changing the type of the ”tokenToSell” argument to this interface. This would make the code more clear and less error-prone.

Client Comment *We prefer to rely on the OZ interfaces.*

```
129 +function swapAndDepositWithPermit(IERC20 tokenToSell, IERC20
      ↪ tokenToDeposit, Swap calldata swap, EIP2098Permit calldata
      ↪ permit)
```

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