Report

v. 2.0

Customer

Contango

Smart Contract Audit

Core V2. Part III

21st February 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	30.01.24	A. Zveryanskaya	Initial Draft
0.2	30.01.24	A. Zveryanskaya	Minor revision
1.0	01.02.24	A. Zveryanskaya	Release
1.1	21.01.24	A. Zveryanskaya	Fix link updated
2.0	21.02.24	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. This is achieved using spot and money markets.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

/			
	CometMoneyMarket.sol	Comet ReverseLookup.sol	SiloMoneyMarket.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 major, and a few less important issues.
All identified Major issues have been fixed.

	Info	Fixed
Major	0	1
Minor	7	2

Fixed 3 out of 10 issues

6 Major Issues

CVF-1. FIXED

- **Category** Suboptimal
- **Source** CometReverseLookup.sol

Description The “nextPayload” storage variable is indirectly incremented here several times.

Recommendation Consider refactoring to avoid multiple reads and writes into the same storage variable.

Client Comment *There are at most 3 ”comets” on any given chain, but’s a fair point, so I’ll fix it for future reference.*

```
31 for (uint256 i; i < comets.length; i++) {
       _setComet(comets[i]);
```

7 Minor Issues

CVF-2. INFO

- **Category** Suboptimal
- **Source** SiloMoneyMarket.sol

Description Hardcoding mainnet addresses makes it harder to test code.

Recommendation Consider passing the addresses as constructor arguments and storing in immutable variables.

Client Comment *We test on forks, so this doesn't difficult our testing at all. It's unlikely we'll use this code outside Arbitrum, so like with the GranaryMM, we chose to hardcode the dependencies to avoid the overhead of config that would be necessary on the deploy scripts, as said config needs to be maintained and tested.*

```
19 ISiloLens public constant LENS = ISiloLens(0
      ↪ x07b94eB6AaD663c4eaf083fBb52928ff9A15BE47);
20 ISiloIncentivesController public constant INCENTIVES_CONTROLLER =
      ↪ ISiloIncentivesController(0
      ↪ xd592F705bDC8C1B439Bd4D665Ed99C4FaAd5A680);
   ISilo public constant WSTETH_SILO = ISilo(0
      ↪ xA8897b4552c075e884BDB8e7b704eB10DB29BF0D);
   IERC20 public constant WETH = IERC20(0
      ↪ x82aF49447D8a07e3bd95BD0d56f35241523fBab1);
   IERC20 public constant USDC = IERC20(0
      ↪ xFF970A61A04b1cA14834A43f5dE4533eBDDB5CC8);
```

CVF-3. INFO

- **Category** Procedural
- **Source** CometReverseLookup.sol

Description Specifying a particular compiler version in every file makes it harder migrating to newer versions.

Recommendation Consider specifying as "^0.8.0". Also relevant for: CometMoneyMarket.sol, SiloMoneyMarket.sol.

Client Comment *The curent pattern is to annotate 'pragma solidity ^0.8.4;' for the interfaces and 'pragma solidity 0.8.20;' for actual code, the reason being that we want to make sure that on deployment, we use the same compiler that we used for testing.*
It's worth noticing that this is a legacy decision from when we deployed with hardhat, so tha compiler config could change from dev to deploy, now that we do everything with foundry on the same repo we could change that, but it'd be a separate change that'd apply to the whole codebase.

```
2 pragma solidity 0.8.20;
```

CVF-4. INFO

- **Category** Procedural
- **Source** CometReverseLookup.sol

Description We didn't review this file.

Client Comment *Its ok, it's code generated by 'cast interface'*

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

CVF-5. INFO

- **Category** Procedural
- **Source** CometReverseLookup.sol

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

Client Comment *This is an artificial split so we don't have to repeat ourselves when testing, so we leave it on the same file so the code is easier to read/navigate, as from the business pov, it has no reason to be separate.*

```
10 interface CometReverseLookupEvents {
```

CVF-6. FIXED

- **Category** Procedural
- **Source** CometReverseLookup.sol

Recommendation The “comet” parameter should be indexed.

```
12 event CometSet(Payload payload, IComet comet);
```

CVF-7. INFO

- **Category** Procedural
- **Source** CometReverseLookup.sol

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

Client Comment *Same as CVF-5.*

```
16 interface CometReverseLookupErrors {
```

CVF-8. INFO

- **Category** Procedural
- **Source** CometMoneyMarket.sol

Description We didn’t review these files.

Client Comment *Its ok, it’s code generated by ‘cast interface‘.*

```
6 import "./dependencies/IComet.sol";
  import "./dependencies/ICometRewards.sol";
```

CVF-9. FIXED

- **Category** Suboptimal
- **Source** CometMoneyMarket.sol

Description Assignment in the middle of an expression makes code harder to read.

Recommendation Consider refactoring.

```
37 reverseLookup.comet(positionId.getPayload()).supply(asset,
   ↪ actualAmount = asset.transferOut(payer, address(this), amount)
   ↪ );
```

```
47 reverseLookup.comet(positionId.getPayload()).withdrawTo(to, asset,
   ↪ actualAmount = amount);
```

```
52 comet.supply(asset, actualAmount = asset.transferOut(payer, address(
   ↪ this), Math.min(amount, comet.borrowBalanceOf(address(this))))
   ↪ );
```

CVF-10. INFO

- **Category** Procedural
- **Source** SiloMoneyMarket.sol

Description We didn't review these files.

Client Comment *Silo.sol is code generated by 'cast interface' The MM_SILO constant is part of the deploy config that we chose to hardcode to avoid the same issues described for CVF-2.*

```
6 import "./Silo.sol";
```

```
10 import { MM_SILO } from "script/constants.sol";
```

ABDK
Consulting

About us

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

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting