Report

v. 1.0

Customer

Chronicle

Smart Contract Audit

Aggor

3rd June 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	03.06.24	A. Zveryanskaya	Initial Draft
0.2	03.06.24	A. Zveryanskaya	Minor revision
1.0	03.06.24	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.

Chronicle Protocol is a novel Oracle solution that has exclusively secured over $10B in assets for MakerDAO and its ecosystem since 2017. With a history of innovation, including the invention of the first Oracle on Ethereum, Chronicle Protocol continues to redefine Oracle networks. A blockchain-agnostic protocol, Chronicle overcomes the current limitations of transferring data on-chain by developing the first truly scalable, cost-efficient, decentralized, and verifiable Oracles, rewriting the rulebook on data transparency and accessibility.

3 Project scope

We were asked to review the Original Code in a private repository and related fixes.
The list of files:

/		
	Aggor.sol	
libs/		
	LibUniswapOracles.sol	LibMedian.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.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We provided Client with a few recommendations.

Moderate	Info	Fixed
	4	1

Fixed 1 out of 5 issues

6 Moderate Issues

CVF-1 INFO

- **Category** Suboptimal
- **Source** LibUniswapOracles.sol

Description This value in many cases could be precomputed.

Recommendation Consider passing it as an argument to allows the caller to precompute it.

Client Comment *Acknowledged. Leaving as is in favor of keeping the Uniswap TWAP logic fully contained in the 'LibUniswapOracle' library.*

```
35 uint128 amt = uint128(10 ** baseDecimals);
```

CVF-2 FIXED

- **Category** Documentation
- **Source** Aggor.sol

Description The semantics of the returned values is unclear.

Recommendation Consider giving descriptive names to the returned values and/or documenting.

Client Comment *Extensive documentation is provided for the return values in src/IAggor.sol which is inherited via the '@inheritdoc' pragma for public function. Additional documentation was added to the internal function.*

```
190 function _read() internal view returns (uint128, uint, Status memory
    ↪ ) {
```

CVF-3 INFO

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

Description This looks like a thing worth doing.

Client Comment *This Aggor implementation will be deployed with the Chainlink ETH/USD oracle on Ethereum mainnet. Chainlink does not use a proxy for this oracle allowing the reduction of code complexity in this case.*

285
```
// The only way to fully protect against malicious contract updates
   ↪ is
// via using a low-level staticcall with _manual_ returndata
   ↪ decoding!
```

CVF-4 INFO

- **Category** Suboptimal
- **Source** Aggor.sol

Recommendation This could be optimized using the underflow behavior. Note, that ”a <= b <= c” condition could be calculated like this: unchecked { return b - a <= c - a; } assuming that c >= a. In case b < a, the left part will underflow to a very large value that is guaranteed to exceed c - a. And if b >= a, this check just reduces to b <= c.

Client Comment *Acknowledged. Leaving as is in favor of code clarity.*

462
```
if (a > b) {
    return uint(b) * 1e18 >= agreementDistance * uint(a);
} else {
    return uint(a) * 1e18 >= agreementDistance * uint(b);
}
```

CVF-5 INFO

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

Description Overflow is possible during type conversion. Currently the code assumes that the ‘val‘ value is within the uint128 range, which results in the commented ‘assert‘ statement. We stress that such assumptions are bad practice, make the code error prone and more difficult for the audit.

Recommendation Consider using safe conversion irregarding of the assumptions or uncomment the assert statement.

Client Comment *The conversion is safe, indicated by the ‘assert(val <= type(uint128).max)‘ a couple lines above.*

263
```
return (true, uint128(val));
```

7 Recommendations

CVF-6 INFO

- **Category** Procedural
- **Source** LibMedian.sol

Recommendation Consider specifying as ”^0.8.0” unless there is something special regarding this particular version. Also relevant for: LibUniswapOracles.sol, Aggor.sol.

Client Comment *A Solidity version of ‘^0.8.0‘ would be wrong as custom errors were just introduced in ‘0.8.4‘. Explicitly defining the minimum version the contract was tested with is considered best practices.*

```
2 pragma solidity ^0.8.16;
```

CVF-7 INFO

- **Category** Procedural
- **Source** LibMedian.sol

Recommendation This comment should be removed.

Client Comment *Assert statements serve as additional documentation and provide useful information for auditors, integrators, and future developers. Furthermore, it enables more easily using formal verification tools in the future.*

```
16 // assert(sum <= 2 * type(uint128).max);
```

CVF-8 INFO

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

Recommendation The type for this argument should be ”IUniswapV3Pool”.

Client Comment *Acknowledged. Leaving as is as keeping public variables as type address instead of specific interface implementations allows external integrators to not have to import the exact same interface specification into their projects.*

```
26 address pool,
```

CVF-9 INFO

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

Recommendation The type for these arguments should be ”IERC20”.

Client Comment *See Issue 9 comment.*

```
27 address baseToken,
   address quoteToken,
```

CVF-10 FIXED

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

Description The semantics and the number format of the returned value is unclear.

Recommendation Consider giving a descriptive name to the returned value and/or documenting.

```
31 ) internal view returns (uint) {
```

CVF-11 INFO

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

Recommendation The argument type should be ”IUniswapV3Pool”.

Client Comment *See Issue 9 comment.*

```
47 function getOldestObservationSecondsAgo(address pool)
```

CVF-12 INFO

- **Category** Procedural
- **Source** Aggor.sol

Description We didn't review these files.

```
4  import {Auth} from "chronicle-std/auth/Auth.sol";
   import {IToll} from "chronicle-std/toll/IToll.sol";

7  import {IChronicle} from "chronicle-std/IChronicle.sol";

11 import {IERC20} from "forge-std/interfaces/IERC20.sol";

15 import {IAggor} from "./IAggor.sol";
```

CVF-13 FIXED

- **Category** Procedural
- **Source** Aggor.sol

Recommendation This phrase seems grammatically incorrect and should be rephrased.

```
25 * @dev While Chronicle oracle's␣normaly␣use␣the␣chronicle-std/Toll␣
      ↪ module␣for
   *␣access␣controlling␣the␣read␣functions␣this␣implementation␣uses␣a
   ␣␣␣␣␣␣
```

CVF-14 INFO

- **Category** Suboptimal
- **Source** Aggor.sol

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

Recommendation Consider treating all token amounts as integers.

Client Comment *This constraint is inherited from Uniswap's 'OracleLibrary' library.*

```
41 uint internal constant _MAX_UNISWAP_BASE_DECIMALS = 38;
```

CVF-15 INFO

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

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

Client Comment *See Issue 9 comment.*

```
61 address public immutable chronicle;
```

CVF-16 INFO

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

Recommendation The type for this variable should be ”IChainlinkAggregatorV3”.

Client Comment *See Issue 9 comment.*

```
63 address public immutable chainlink;
```

CVF-17 INFO

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

Recommendation The type for this variable should be ”IUniswapV3Pool”.

Client Comment *See Issue 9 comment.*

```
68 address public immutable uniswapPool;
```

CVF-18 INFO

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

Recommendation The type for these variables should be ”IERC20”.

Client Comment *See Issue 9 comment.*

```
70 address public immutable uniswapBaseToken;
```

```
72 address public immutable uniswapQuoteToken;
```

CVF-19 INFO

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

Recommendation The type for this argument should be ”IChronicle”.

Client Comment *See Issue 9 comment.*

```
99 address chronicle_,
```

CVF-20 INFO

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

Recommendation The type for this variable should be ”IChainlinkAggregatorV3”.

Client Comment *See Issue 9 comment.*

```
100 address chainlink_,
```

CVF-21 INFO

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

Recommendation The type for this argument should be ”IUniswapV3Pool”.

Client Comment *See Issue 9 comment.*

```
101 address uniswapPool_,
```

CVF-22 INFO

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

Recommendation The type for these arguments should be ”IERC20”.

Client Comment *See Issue 9 comment.*

```
102 address uniswapBaseToken_,
    address uniswapQuoteToken_,
```

CVF-23 INFO

- **Category** Suboptimal
- **Source** Aggor.sol

Recommendation This argument seems redundant, as its value could be derived from the ”uniswapBaseToken_” value.

Client Comment *See Issue 28 comment.*

```
104 uint8 uniswapBaseTokenDecimals_,
```

CVF-24 INFO

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

Recommendation The type for this argument should be ”IUniswapV3Pool”.

Client Comment *See Issue 9 comment.*

```
140 address uniswapPool_,
```

CVF-25 INFO

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

Recommendation The type for these arguments should be ”IERC20”.

Client Comment *See Issue 9 comment.*

```
141 address uniswapBaseToken_,
    address uniswapQuoteToken_,
```

CVF-26 INFO

- **Category** Procedural
- **Source** Aggor.sol

Recommendation This check should be performed earlier.

Client Comment *These checks are all executed during construction time. Optimizing in favor of reduced gas consumption in case of erroneus constructor arguments is unecessary.*

```
152 require(
        uniswapBaseToken_ != uniswapQuoteToken_,
        "Uniswap␣tokens␣must␣not␣be␣equal"
    );
```

CVF-27 INFO

- **Category** Suboptimal
- **Source** Aggor.sol

Recommendation This check makes the ”uniswapBaseTokenDecimals_” argument redundant, as its value could be derived from the ”uniswapBaseToken_”.value.

Client Comment *Acknowledged. Note that in future versions the constructor checks may be moved to the deployment script for which the decimals arguments would become necessary again.*

166
```
require(
    uniswapBaseTokenDecimals_ == IERC20(uniswapBaseToken_).decimals
        ↪ (),
    "Uniswap␣base␣token␣decimals␣mismatch"
);
```

CVF-28 INFO

- **Category** Readability
- **Source** Aggor.sol

Description The code below looks like it is always executed, while actually it is executed only when both oracles are not OK.

Recommendation Consider placing the rest of the function into an explicit ”else” branch.

Client Comment *The _read function uses a return-early approach as described in ‘docs/Aggor.md‘. Note further, that the code below is being executed only if both oracles are ok but not in agreement distance, not if both oracles are not ok.*

225
```
}
```

CVF-29 INFO

- **Category** Readability
- **Source** Aggor.sol

Description The code below looks like it is always executed, while actually it is executed only when TWAP isn't OK.

Recommendation Consider placing the rest of the function into an explicit "else" branch.

Client Comment *The _read function uses a return-early approach as described in 'docs/Aggor.md'. Note further that the code below is being executed only if both oracles are ok, not in agreement distance, and the TWAP cannot be read, not if only TWAP not being ok.*

```
231 }
```

CVF-30 INFO

- **Category** Procedural
- **Source** Aggor.sol

Recommendation These commented asserts should be either uncommented or removed.

Client Comment *Assert statements serve as additional documentation and provide useful information for auditors, integrators, and future developers. Furthermore, it enables more easily using formal verification tools in the future.*

```
250 // assert(val <= type(uint128).max);
    // assert(!ok || val != 0); // ok -> val != 0
    // assert(age <= block.timestamp);
```

```
260 // assert(_DECIMALS_CHRONICLES >= decimals).
```

CVF-31 INFO

- **Category** Procedural
- **Source** Aggor.sol

Description Part of this condition could be checked before ”isStale” was calculating.

Recommendation Consider refactoring to only calculate ”isStale” when this is actually needed.

Client Comment *This is an optimization that only triggers in case the Chainlink oracle's answer is out-of-bounds, ie invalid. Leaving as is due to only minimal runtime gas savings.*

```
311 isStale || answer <= 0 || uint(answer) > uint(type(uint128).max)
```

CVF-32 INFO

- **Category** Procedural
- **Source** Aggor.sol

Description These conversions looks weird.

Recommendation Consider refactoring.

```
362 answer = int(uint(val));
```

```
372 return int(uint(val));
```

CVF-33 INFO

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

Description The semantics of the returned values is unclear.

Recommendation Consider giving descriptive names to the returned values and/or documenting.

Client Comment *It is documented via the inherited documentation from 'IAggor.sol'.*

```
382 returns (uint, uint, Status memory)
```

CVF-34 INFO

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

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

```
396 require(agreementDistance_ <= 1e18);
```

```
463     return uint(b) * 1e18 >= agreementDistance * uint(a);
```

```
465     return uint(a) * 1e18 >= agreementDistance * uint(b);
```

CVF-35 INFO

- **Category** Suboptimal
- **Source** Aggor.sol

Recommendation Logging the old value is redundant, as it could be derived from the previous event.

Client Comment *Logging the redundant value makes offchain monitoring and manual inspection easier as previous events do not need to be searched for.*

```
400 msg.sender, agreementDistance, agreementDistance_
```

```
416 msg.sender, ageThreshold, ageThreshold_
```

CVF-36 INFO

- **Category** Procedural
- **Source** Aggor.sol

Recommendation This contract should be moved into a separate file and probably converted into a template.

```
475 contract Aggor_BASE_QUOTE_COUNTER is Aggor {
```

CVF-37 INFO

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

Recommendation The type for this argument should be ”IChronicle”.

Client Comment *See Issue 9 comment.*

490 `Aggor(`

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