Auctus Finance (ACO) Process Quality Review

Score: 83%

This is an Auctos Finance ACO Process Quality Review completed on 26 October 2020. It was performed using the Process Review process (version 0.5) and is documented here. The review was performed by ShinkaRex of Caliburn Consulting. Check out our Telegram.

The final score of the review is 83%, a strong score. The breakdown of the scoring is in Scoring Appendix.

Summary of the Process

Very simply, the review looks for the following declarations from the developer's site. With these declarations, it is reasonable to trust the smart contracts.

1. **Here is my smart contract on the blockchain**
2. **You can see it matches a software repository used to develop the code**
3. **Here is the documentation that explains what my smart contract does**
4. **Here are the tests I ran to verify my smart contract**
5. **Here are the audit(s) performed to review my code by third party experts**

Disclaimer

This report is for informational purposes only and does not constitute investment advice of any kind, nor does it constitute an offer to provide investment advisory or other services. Nothing in this report shall be considered a solicitation or offer to buy or sell any security, future, option or other financial instrument or to offer or provide any investment advice or service to any person in any jurisdiction. Nothing contained in this report constitutes investment advice or offers any opinion with respect to the suitability of any security, and the views expressed in this report should not be taken as advice to buy, sell or hold any security. The information in this report should not be relied upon for the purpose of investing. In preparing the information contained in this report, we have not taken into account the investment needs, objectives and financial circumstances of any particular investor. This information has no regard to the specific investment objectives, financial situation and particular needs of any specific recipient of this information and investments discussed may not be suitable for all investors.

Any views expressed in this report by us were prepared based upon the information available to us at the time such views were written. Changed or additional information could cause such views to change. All information is subject to possible correction. Information may quickly become unreliable for various reasons, including changes in market conditions or economic circumstances.

Executing Code Verification

This section looks at the code deployed on the Mainnet that gets reviewed and its corresponding software repository. The document explaining these questions is here. This review will answer the questions;

1. Are the executing code address(s) readily available? (Y/N)
2. Is the code actively being used? (%)
3. Are the Contract(s) Verified/Verifiable? (Y/N)
4. Does the code match a tagged version in the code hosting platform? (%)
5. Is the software repository healthy? (%)

Are the executing code address(s) readily available? (Y/N)

> Answer: Yes

There are multiple addresses with the executing code addresses:

Factory docs.aco.finance	

https://docs.aco.finance/smart-contracts/flash-exercise
docs.aco.finance

https://docs.aco.finance/smart-contracts/writer
docs.aco.finance

This review only covers the contract ACOProxy.sol. This proxy is connected to ACOfactoryV2.sol, which is under address 0x64CDDD6A48E7783A2a3df8B6CBC1F93FE48f2276.

Is the code actively being used? (%)

Answer: 70%

Activity is 20 transactions a week, as indicated in the Appendix.

Percentage Score Guidance

100%	More than 10 transactions a day
70%	More than 10 transactions a week
40%	More than 10 transactions a month
10%	Less than 10 transactions a month
0%	No activity

Are the Contract(s) Verified/Verifiable? (Y/N)

Answer: Yes

0x176b98ab38d1aE8fF3F30bF07f9B93E26F559C17 is the Etherscan verified contract address of the Proxy.

0x64CDDD6A48E7783A2a3df8B6CBC1F93FE48f2276 is the address of ACOFactoryV2.sol, which is the address ran by the proxy.

Does the code match a tagged version on a code hosting platform? (%)

Answer: 100%

These contracts were easy to match. In ACO's contract documentation, they include a link to the code on the GitHub, along with the address.

Guidance:

100%	All code matches and Repository was clearly labelled
60 %	All code matches but no labelled repository. Repository was found manually
30%	Almost all code does match perfectly and repository was found manually
0%	Most matching Code could not be found

GitHub address : https://github.com/AuctusProject/aco

Deployed contracts in the following file;

ACODeployed.rar ACODeployed.rar - 8KB

Matching Repository: https://github.com/AuctusProject/aco/tree/master/smart-contracts/contracts

Is development software repository healthy? (%)

Answer: Yes

With 4 branches and 311 commits, this is a healthy project.

Documentation

This section looks at the software documentation. The document explaining these questions is here.

Required questions are;

1. Is there a whitepaper? (Y/N)
2. Are the basic application requirements documented? (Y/N)
3. Do the requirements fully (100%) cover the deployed contracts? (%)
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)
5. Is it possible to trace software requirements to the implementation in code (%)

Is there a whitepaper? (Y/N)

> Answer: Yes

Location: https://docs.aco.finance/

Their whitepaper is their documentation. You can find all details about their processes in there.

Are the basic application requirements documented? (Y/N)

> Answer: Yes

Location: https://docs.aco.finance/smart-contracts/factory

Do the requirements fully (100%) cover the deployed contracts? (%)

> Answer: 100%

The functions are well-documented, and categorized, within ACO's documentation.

Are there sufficiently detailed comments for all functions within the deployed contract code (%)

> Answer: 100%

Code examples are in the Appendix. As per the SLOC, there is 99% commenting to code.

How to improve this score

This score can improve by adding comments to the deployed code such that it comprehensively covers the code. For guidance, refer to the SecurEth Software Requirements.

Is it possible to trace requirements to the implementation in code (%)

> Answer: 60%

Although the functions are well-documented, there are no direct references to the code within the documentation.

Guidance:
100% - Clear explicit traceability between code and documentation at a requirement level for all code

60% - Clear association between code and documents via non explicit traceability
40% - Documentation lists all the functions and describes their functions
0% - No connection between documentation and code

How to improve this score

This score can improve by adding traceability from requirements to code such that it is clear where each requirement is coded. For reference, check the SecurEth guidelines on traceability.

Testing

This section looks at the software testing available. It is explained in this document. This section answers the following questions;

1. Full test suite (Covers all the deployed code) (%)
2. Code coverage (Covers all the deployed lines of code, or explains misses) (%)
3. Scripts and instructions to run the tests (Y/N)
4. Packaged with the deployed code (Y/N)
5. Report of the results (%)
6. Formal Verification test done (%)
7. Stress Testing environment (%)

Is there a Full test suite? (%)

> Answer: 100%

There is a testing suite available that covers the deployed code. However, there are no instructions provided on how to test the software yourself. Additionally there are no test addresses provided.

How to improve this score

This score can improve by adding tests to fully cover the code. Document what is covered by traceability or test results in the software repository.

Code coverage (Covers all the deployed lines of code, or explains misses) (%)

Answer: 50%

There is no indication of code coverage, and the audit had no documentation regarding test results.

Guidance:
100% - Documented full coverage
99-51% - Value of test coverage from documented results
50% - No indication of code coverage but clearly there is a reasonably complete set of tests
30% - Some tests evident but not complete
0% - No test for coverage seen

How to improve this score

This score can improve by adding tests achieving full code coverage. A clear report and scripts in the software repository will guarantee a high score.

Scripts and instructions to run the tests (Y/N)

Answer: No

No scripts and instructions are apparent in any of the documentation, or audit.

How to improve this score

Add the scripts to the repository and ensure they work. Ask an outsider to create the environment and run the tests. Improve the scripts and docs based on their feedback.

Packaged with the deployed code (Y/N)

Answer: Yes

Tests can be found in the /aco/tree/master/smart-contracts/test directory.

How to improve this score

Improving this score requires redeployment of the code, with the tests. This score gives credit to those who test their code before deployment and release them together. If a developer adds tests after deployment they can gain full points for all test elements except this one.

Report of the results (%)

Answer: 0%

There are no apparent reports of results.

How to improve this score

Add a report with the results. The test scripts should generate the report or elements of it.

Formal Verification test done (%)

Answer: 0%

There is no evidence of formal verification tests having been done.

Stress Testing environment (%)

Answer: 0%

There are no published addresses on any tests networks, and there is no documentation of the testing results.

Audits

> Answer: 90%

There was an audit done by OpenZeppelin. All issues were resolved or considered.

Guidance:

1. Multiple Audits performed before deployment and results public and implemented or not required (100%)
2. Single audit performed before deployment and results public and implemented or not required (90%)
3. Audit(s) performed after deployment and no changes required. Audit report is public. (70%)
4. No audit performed (20%)
5. Audit Performed after deployment, existence is public, report is not public and no improvements deployed OR smart contract address' not found, question 1 (0%)

Appendices

Author Details

The author of this review is Rex of Caliburn Consulting.

Email : rex@defisafety.com Twitter : @defisafety

I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

I was coaxed to go to EthDenver 2018 and there I started SecuEth.org with Bryant and Roman. We created guidelines on good processes for blockchain code development. We got EthFoundation funding to assist in their development.

Process Quality Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

Career wise I am a business development manager for an avionics supplier.

Scoring Appendix

Executing Code Appendix

Code Used Appendix

Example Code Appendix

```
pragma solidity ^0.6.6;

import "../libs/Address.sol";
import "../interfaces/IACOToken.sol";

/**
 * @title ACOFactory
 * @dev The contract is the implementation for the ACOProxy.
 */
contract ACOFactory {

    /**
     * @dev Emitted when the factory admin address has been changed.
     * @param previousFactoryAdmin Address of the previous factory admin.
     * @param newFactoryAdmin Address of the new factory admin.
     */
    event SetFactoryAdmin(address indexed previousFactoryAdmin, address inde

    /**
     * @dev Emitted when the ACO token implementation has been changed.
     * @param previousAcoTokenImplementation Address of the previous ACO tol
     * @param newAcoTokenImplementation Address of the new ACO token impleme
```

```
     */
    event SetAcoTokenImplementation(address indexed previousAcoTokenImpleme

    /**
     * @dev Emitted when the ACO fee has been changed.
     * @param previousAcoFee Value of the previous ACO fee.
     * @param newAcoFee Value of the new ACO fee.
     */
    event SetAcoFee(uint256 indexed previousAcoFee, uint256 indexed newAcoF

    /**
     * @dev Emitted when the ACO fee destination address has been changed.
     * @param previousAcoFeeDestination Address of the previous ACO fee des
     * @param newAcoFeeDestination Address of the new ACO fee destination.
     */
    event SetAcoFeeDestination(address indexed previousAcoFeeDestination, a

    /**
     * @dev Emitted when a new ACO token has been created.
     * @param underlying Address of the underlying asset (0x0 for Ethereum)
     * @param strikeAsset Address of the strike asset (0x0 for Ethereum).
     * @param isCall True if the type is CALL, false for PUT.
     * @param strikePrice The strike price with the strike asset precision.
     * @param expiryTime The UNIX time for the ACO token expiration.
     * @param acoToken Address of the new ACO token created.
     * @param acoTokenImplementation Address of the ACO token implementatio
     */
    event NewAcoToken(address indexed underlying, address indexed strikeAss

    /**
     * @dev The ACO fee value.
     * It is a percentage value (100000 is 100%).
     */
    uint256 public acoFee;

    /**
     * @dev The factory admin address.
     */
    address public factoryAdmin;

    /**
     * @dev The ACO token implementation address.
     */
    address public acoTokenImplementation;

    /**
     * @dev The ACO fee destination address.
     */
    address public acoFeeDestination;

    /**
     * @dev Modifier to check if the `msg.sender` is the factory admin.
     * Only factory admin address can execute.
     */
    modifier onlyFactoryAdmin() {
```

```
        require(msg.sender == factoryAdmin, "ACOFactory::onlyFactoryAdmin")
        _;
    }

    /**
     * @dev Function to initialize the contract.
     * It should be called through the `data` argument when creating the pr
     * It must be called only once. The `assert` is to guarantee that behav
     * @param _factoryAdmin Address of the factory admin.
     * @param _acoTokenImplementation Address of the ACO token implementati
     * @param _acoFee Value of the ACO fee.
     * @param _acoFeeDestination Address of the ACO fee destination.
     */
    function init(address _factoryAdmin, address _acoTokenImplementation, u
        require(factoryAdmin == address(0) && acoTokenImplementation == add

        _setFactoryAdmin(_factoryAdmin);
        _setAcoTokenImplementation(_acoTokenImplementation);
        _setAcoFee(_acoFee);
        _setAcoFeeDestination(_acoFeeDestination);
    }

    /**
     * @dev Function to guarantee that the contract will not receive ether.
     */
    receive() external payable virtual {
        revert();
    }

    /**
     * @dev Function to create a new ACO token.
     * It deploys a minimal proxy for the ACO token implementation address.
     * @param underlying Address of the underlying asset (0x0 for Ethereum)
     * @param strikeAsset Address of the strike asset (0x0 for Ethereum).
     * @param isCall Whether the ACO token is the Call type.
     * @param strikePrice The strike price with the strike asset precision.
     * @param expiryTime The UNIX time for the ACO token expiration.
     * @param maxExercisedAccounts The maximum number of accounts that can
     */
    function createAcoToken(
        address underlying,
        address strikeAsset,
        bool isCall,
        uint256 strikePrice,
        uint256 expiryTime,
        uint256 maxExercisedAccounts
    ) onlyFactoryAdmin external virtual returns(address) {
        address acoToken = _deployAcoToken(underlying, strikeAsset, isCall,
        emit NewAcoToken(underlying, strikeAsset, isCall, strikePrice, expi
        return acoToken;
    }
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	4	667	81	292	294	28

Comments to Code 292/ 294 = 99%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complexity
JavaScript	7	6558	1101	0	5547	59

Tests to Code 5547 / 294 = 1800%